

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Domen Štangar

Zgoščevanje redkih (LR) tabel

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: prof. doc. dr. Boštjan Slivnik

Ljubljana, 2010



Št. naloge: 00489/2009

Datum: 15.11.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **DOMEN ŠTANGAR**

Naslov: **ZGOŠČEVANJE REDKIH (LR) TABEL
SPARSE (LR) TABLE COMPRESSION**

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Raziščite metode za učinkovito predstavitev redkih tabel, ki so del sintaksnih analizatorjev vrste LR. Programsko realizirajte nekaj najboljših metod za stiskanje redkih tabel in eksperimentalno ugotovite njihovo učinkovitost.

Mentor:

B. Slivnik
doc. dr. Boštjan Slivnik



Dekan:

Franco Solina
prof. dr. Franc Solina

Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavlanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Besedilo je oblikovano z urejevalnikom besedil $\text{\LaTeX}$.

original izdane teme diplomskega dela

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani Domen Štangar,

z vpisno številko 63000116,

sem avtor diplomskega dela z naslovom:

Zgoščevanje redkih (LR) tabel

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. doc. dr. Boštjana Slivnika
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI"

V Ljubljani, dne 14. 5. 2010

Podpis avtorja:

Kazalo

Povzetek	1
Abstract	2
1 Redke tabele	3
1.1 Uvod v problematiko	3
1.2 Uvod v sintaksni analizator vrste LR	4
2 Metode zgoščevanja	7
2.1 Metode zgoščevanja matrik	7
2.2 Metode zgoščevanja LR tabel	10
2.3 Metoda barvanja grafov (GCS)	10
2.4 Metoda eliminacije (LES)	16
2.5 Metoda odmika (RDS)	20
2.6 Metoda zamika začetne in zadnje vrednosti (SDS)	30
2.7 Metoda vrstica, stolpec (RCS)	30
2.8 Metoda izogibanja ničnih vrednosti (SZS)	32
2.9 Primerjava metod pri zgoščevanju LR tabel	33
3 Implementacija sintaksnega analizatorja	35
3.1 Sintaksni analizator Bison	35
3.1.1 Algoritem zgoščevanja tabel pri Bisonu	36
3.1.2 Algoritem preverjanja sintakse	40
3.2 Analiza patenta za zgoščevanje LR tabel	42
4 Sklep	44
A Priloga	45
Seznam slik	45

Seznam tabel	47
Seznam algoritmov	48
Literatura	49
B izvorna koda zgoščevanje redke tabele	50
C izvorna koda analize in uporabe <i>Bison</i> tabele	71
D izvorna koda uporabe patenta	76

Seznam uporabljenih kratic in simbolov

Bison: Odprtokodna rešitev, v kateri najdemo sintaksni generator in analizator vrste LALR (Look-Ahead LR).

Check tabela: Tabela pri metodi RDS, v kateri preverjamo, veljavnost elementa.

Data table: Tabela pri metodi RDS, v kateri imamo vrednosti elementov.

Displacement tabela: Tabela pri metodi RDS, v kateri imamo odmike za vrstico ali stolpec.

Eli: Odprtokodna rešitev, v kateri imamo sintaksni generator in analizator vrste LR. Ima tudi možnost uporabe starejših metod poleg trenutno edine uporabljene metode za zgoščanje LR tabel.

GCS: Graph coloring scheme - Metoda barvanja grafov

LES: Line elimination scheme - Metoda eliminacije

LR: (Left to right) - Sintaksna analiza, ki bere vhod od leve proti desni ter deterministično sestavi skrajno desno izpeljavo v nasprotnem vrstnem redu.

Matlab: Popularno programsko orodje, v katerem se tudi uporabljajo redke tabele

RCS: Row column scheme - Metoda vrstica, stolpec

RDS: Row displacement scheme - Metoda odmika

SDS: Significant distance scheme - Metoda zamika začetne in zadnje vrednosti

Sigmap: Tabela pri metodi GCS in LES, v kateri preverjamo, ali imamo za določeno vrstico ali stolpec veljavno vrednost.

SZS: Suppressed zero scheme - Metoda izogibanja ničnih vrednosti

Povzetek

Diplomska naloga se ukvarja z algoritmi za zgoščanje redkih tabel. Redka tabela predstavlja tabelo, v kateri imamo veliko elementov, njihova vrednost je nepomembna, vendar nam kljub temu zasedajo prostor. Problematika redkih tabel se pojavlja v LR tabelah, tabelah velikih dimenzij in pri matrikah velikega reda, v katerih je veliko vrednosti 0.

LR tabele se uporabljajo pri sintaksni analizi in lahko zasedejo veliko pomnilnika, hkrati pa je vsebina tabele skorajda prazna, zato potrebujemo algoritme, ki zgoščujejo tabele. Ob vsakem izvajanju ali prevajanju programov potrebujemo dostop do tabele. Če teh tabel ne bi zgoščevali, bi za samo preverjanje sintakse potrebovali več pomnilnika.

Diplomska naloga v prvem delu opisuje praktične primere zgoščevanja redkih tabel in podrobno opisuje algoritme posameznih metod. Tako obravnavamo metode barvanja grafov, eliminacije, odmika vrstic in metodo vrstica-stolpec. Na koncu prvega dela je narejena primerjava rezultatov na primeru LR tabel. Iz primerjave dobimo trenutno najbolj pogosto uporabljeno metodo za zgoščevanje redkih tabel.

V drugem delu je predstavljena uporaba zgoščenih LR tabel pri sintaksnem analizatorju *Bison* in implementacijo sintaksnega analizatorja z uporabo LR tabel, ki jih generira generator *Bison*. Implementiran je tudi v delu [2] objavljeni patent na področju zgoščanja LR tabel.

V prilogah so navedene izvirne kode za vse metode, ki so bile obravnavane v nalogi, prav tako pa tudi implementacija sintaksnega analizatorja.

Ključne besede:

redke tabele, LR tabele, sintaksna analiza, redke matrike, zgoščevanje redkih tabel

Abstract

This diploma thesis deals with algorithms for compressing sparse tables. Sparse table represents a table in which we have many items whose value is irrelevant, but it nevertheless take up space. The problem of sparse tables occurs in LR tables, tables of large dimensions and high order matrices in which we have high number of values 0.

The LR table is used for syntactic analysis and can take up lot of memory, while the contents of the table is almost empty, so we need algorithms that compress those tables. At any execution or translation of a program we need access to the table. If these tables would not be compressed, we would need more memory just to check the syntax. In the first part of the paper we describe practical examples of sparse tables and we describe in detail the algorithms of each method. We describe methods like graph coloring, line elimination, row displacement and row column scheme.

In second part we present the use of compressed LR tables and implement a syntactic analyzer using LR tables generated by the *Bison*. We also implement a patent which is also published in compressing LR table. In the annexes you can find the source code for all methods that we have discussed in this paper and the implementation of the syntactical analyzer.

Key words:

sparse tables, LR tables, syntax analysis, sparse matrix, compressing sparse tables

Poglavje 1

Redke tabele

1.1 Uvod v problematiko

V diplomski nalogi se ukvarjamo s problematiko zgoščanja LR tabel. LR tabele dobimo iz sintaksnih generatorjev in so del sintaksnih analizatorjev, zato jih potrebujemo za sintakšno analizo. Običajno so te tabele večjih dimenzij in vsebujejo zelo malo elementov s pomembno vrednostjo. Taki tabeli rečemo redka tabela.

Redko tabelo si lahko predstavljamo kot veliko tabelo, ki ima veliko število stolpcev in vrstic, vendar ima večina vsebine tabele nično vrednost. V tabeli z določenimi dimenzijami celice zasedajo pomnilnik, ne glede na to, ali potrebujemo vrednost določene celice, zato se nam tako tabelo splača zgoščevati.

Lahko si predstavljamo, da potrebujemo tabelo, v kateri imamo vse izdelke in vse stranke, ki te izdelke kupujejo. Če vrstice tabele predstavljajo stranke, stolpci izdelke, in imamo 10.000 izdelkov in 5000 strank, potrebujemo za to tabelo 50.000.000 celic. Glede na to, da vse stranke ne kupujejo vseh izdelkov, si lahko predstavljamo, da bo tabela večinoma prazna.

Še en primer redke tabele so matrike. Matriki kot sta spodnja ali zgornja trikotna, predstavljata primer redkih tabel. Ko delamo z matrikami velikega reda, lahko prihranimo veliko pomnilnika.

Redke tabele lahko definiramo tudi kot tabele, za katere poznamo zapis na način, ki zaseda manj pomnilnika, kot ga v osnovni obliki. Taka definicija redke table je zapisana v delu [4]. Za novi zapis je pomemben čas, ki ga potrebujemo, da zapis uredimo, oziroma kakšna je dodatna cena za novi zapis, glede na pomnilnik ali čas dostopa do določene celice.

Vsekakor bi radi, da je čas gradnje novega zapisa čim krajši, da nam končni zapis zaseda čim manj pomnilnika in da imamo pri tem dostop čim krajši,

vendar kot običajno, vse hkrati ni možno. Imamo algoritme, ki imajo kratek dostopni čas do elementov, vendar se nam podaljša čas grajenja tega zapisa. Ločimo dinamično ali statično dodajanje elementov. Pri statičnem dodajanju elementov imamo vrednosti določene že vnaprej in si zato lahko vzamemo več časa, kot pri dinamičnem dodajanju elementov. Statično dodajanje elementov je torej bolj kvalitetno in tudi bolj dostopno, kot dinamično dodajanje. V diplomski nalogi se bomo osredotočili na statični način dodajanja elementov in kakovost določenih metod za zgoščanje LR tabel.

1.2 Uvod v sintaksni analizator vrste LR

Prevajalnik je računalniški program, ki prevede izvorno kodo programske opreme, napisano v višjenivojskem programskem jeziku, v vmesno kodo programske opreme ali strojni jezik. Programske opreme prevajalnikov vsebujejo več faz oziroma stopenj, po katerih poteka prevod. Prvo od faz imenujemo leksikalna analiza. Naloga leksikalne analize je, da vhodne znake združi v večje pomenske enote, kot so imena spremenljivk, ključne besede in številčne konstante. Tem enotam pravimo osnovni simboli in jih uporablja sintaksna analiza. To je postopek, s katerim ugotovimo sintaksno zgradbo nekega niza simbolov (glede na gramatiko jezika). Sintaksni analizator vrste LR se v praksi uporablja najbolj pogosto in predstavlja najpomembnejšo vrsto desne analize. Sintaksni analizator vrste LR je skladovni pretvornik, ki izračuna desno sled izpeljave vhodnega niza in jo izpiše v obratnem vrstnem redu.

LR sintaksni analizator je sestavljen iz vhoda, izhoda, sklada, pogona in dveh tabel (LR tabela). Kako bo sintaksni analizator deloval, določata tabeli prehodov in odločitev. Analizator bere po en simbol naenkrat. Katero operacijo bo izbral, se odloči glede na trenutno stanje in vhod, ter na podlagi odločitvene tabele in tabele prehodov. Odločitvena tabela lahko vsebuje pomik, prevedbo, sprejem ali napako. Če je operacija prevedba, potem na stanje analizatorja vpliva tudi tabela prehodov. Ta vsebuje prehode, ki jih uporabimo glede na trenutno stanje sklada.

Na velikost teh dveh tabel vpliva gramatika, zato je treba omeniti še vmesne in končne simbole, ki so del gramatike jezika in hkrati vplivajo na velikost teh dveh tabel. V primeru (slika 1.1) so simboli **id**, '+', '*', '(', ')', '\$' končni simboli, Izraz, Termin, Faktor pa so vmesni simboli.

Primer delovanja analizatorja je viden v tabeli 1.2, pri vhodu **id+id*id\$**. Vhod si lahko predstavljamo kot niz znakov, s katerim preverjamo ali je sintaksa pravilna. *Id* v našem primeru predstavlja neko spremenljivko, ki jo uporabljamo

v programu.

Pri preverjanju sintakse uporabljamo tabelo 1.1, brez katere sintaksni analizator ne more delovati. V našem primeru je gramatika zelo osnovna, LR tabeli se povečujeta, glede na število vmesnih in končnih simbolov, zato je zgoščevanje te tabele smiselno.

Slika 1.1: Gramatika

1. $Izraz \rightarrow Izraz + Termin$
2. $Izraz \rightarrow Termin$
3. $Termin \rightarrow Termin * Faktor$
4. $Termin \rightarrow Faktor$
5. $Faktor \rightarrow (Izraz)$
6. $Faktor \rightarrow \mathbf{id}$

Tabela 1.1: LR Tabela

Stanje	Odločitvena tabela						Tabela prehodov		
	id	+	*	(	)	\$	Izraz	Termin	Faktor
0	Po5			Po4			1	2	3
1		Po6				Spr			
2		p2	Po7		p2	p2			
3		p4	p4		p4	p4			
4	Po5			Po4			8	2	3
5		p6	p6		p6	p6			
6	Po5			Po4				9	3
7	Po5			Po4					10
8		Po6			Po11				
9		p1	Po7		p1	p1			
10		p3	p3		p3	p3			
11		p5	p5		p5	p5			

Tabela 1.2: Delovanje LR analizatorja

Korak	Sklad	Simboli	Vhod	Pripomba
1	0		id+id \$	Pomik v stanje 5
2	0 5	id	+id *id\$	Prevedba po 6 (<i>Faktor</i> → id)
3	0 3	Faktor	+id *id\$	Prevedba po 4 (<i>Termin</i> → <i>Faktor</i>)
4	0 2	Termin	+id *id\$	Prevedba po 2 (<i>Izraz</i> → <i>Termin</i>)
5	0 1	Izraz	+id *id\$	Pomik v stanje 6
6	0 1 6	Izraz+	id *id\$	Pomik v stanje 5
7	0 1 6 5	Izraz+ id	*id \$	Prevedba po 6 (<i>Faktor</i> → id)
8	0 1 6 3	Izraz+Faktor	*id \$	Prevedba po 4 (<i>Termin</i> → <i>Faktor</i>)
9	0 1 6 9	Izraz+Termin	*id \$	Pomik v stanje 7
10	0 1 6 9 7	Izraz+Termin*	id \$	Pomik v stanje 5
11	0 1 6 9 7 5	Izraz+Termin* id	\$	Prevedba po 6 (<i>Faktor</i> → id)
12	0 1 6 9 7 10	Izraz+Termin*Faktor	\$	Prevedba po 3 (<i>Termin</i> → <i>Termin</i> * <i>Faktor</i>)
13	0 1 6 9	Izraz+Termin	\$	Prevedba po 1 (<i>Izraz</i> → <i>Izraz</i> + <i>Termin</i>)
14	0 1	Izraz	\$	Sprejem

Poglavje 2

Metode zgoščevanja

V tem poglavju bomo predstavili splošne metode, ki se uporabljajo pri zgoščevanju redkih tabel. V našem primeru je to zgoščevanje LR tabel ali matrik. Glede na to, da je problematika podobna, se metode za zgoščevanje lahko uporabljajo v obeh primerih, le da lahko pri LR tabelah izkoriščamo določene lastnosti, ki jih matrike nimajo.

2.1 Metode zgoščevanja matrik

Pri matrikah nas zanima hitro grajenje zgoščene oblike in hiter čas dostopa do elementov. Tako obstajajo različne rešitve:

- Slovar po ključu (stolpec, vrstica) (*Dictionary of Keys*)
- Seznam seznamov (*List of Lists*)
- Koordiniran seznam (*Coordinate List*)
- Yale format matrike (*Yale Sparse Matrix*)

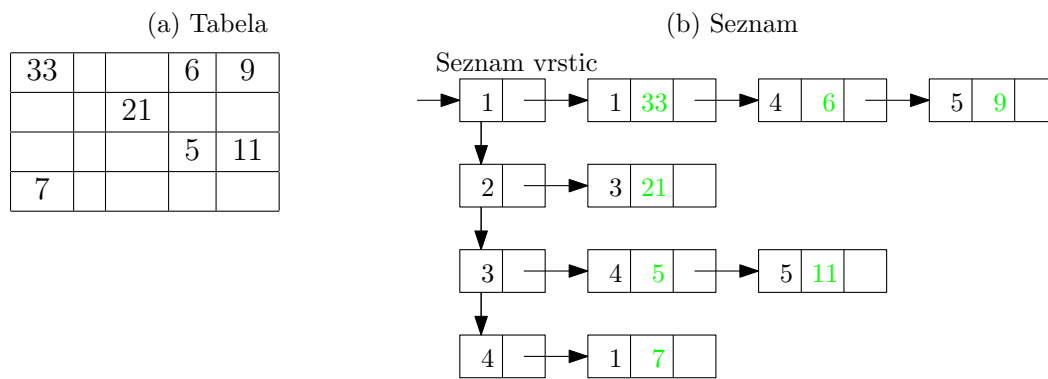
Slovar po ključu stolpec, vrstica

V slovar preslikamo vse pare (vrstica, stolpec), ki imajo vrednost drugačno od nič. Pri iskanju elementa, ki ga ni v slovarju, moramo pregledati celotni slovar. To je osnovna oblika zgoščevanja, vendar ta način ne omogoča konstantnega dostopa do elementov.

Seznam seznamov

To je seznam vrstic, od katerih vsaka vsebuje kazalec na nov seznam vrstice. Ta vsebuje indeks stolpca in vrednost celice stolpca. Rezultat je seznam seznamov, s katerim lahko dostopamo do redke tabele, s tem zapisom pa lahko prihranimo nezasedeni prostor, vendar čas dostopa ni konstanten.

Slika 2.1: Seznam seznamov



Koordiniran seznam

To obliko uporablja tudi matematični program *Matlab*. Kako *Matlab* uporablja ta zapis, je podrobno opisano v delu [4]. Podobno metodo uporabljamo tudi pri zgoščanju LR tabel.

Matlab uporablja tri tabele za zapis ne-ničnih elementov. Za primer smo uporabili vhodno tabelo (2.1a). Prva tabela vsebuje vse ne-nične elemente (tabela 2.1b) in si elementi sledijo glede na stolpec. Druga tabela vsebuje vrstico, na kateri naj bi bil ta element (tabela 2.1c), tretja tabela pa vsebuje indekse za začetek stolpca, s katerim dobimo indeks na prvo tabelo (tabela 2.1d). S tem načinom zapisa potrebujemo prostor za $2 * nnz + n$ elementov. Nnz predstavlja število neničnih elementov, n pa število stolpcev.

Podobno metodo uporabljamo tudi pri zgoščevanju LR tabel, ki je podrobno opisana v poglavju 2.7. Razlika med metodama je, v smeri, ki jo uporabljamo za preverjanje veljavnosti elementov - tako, kot si sledijo v vrstici, ali kot si sledijo v smeri stolpca.

Tabela 2.1: Primer za *Matlab*(a) Vhodna
tabela

2			
1	3	5	
		4	6

(b) Nenični elementi

2	1	3	5	4	6
---	---	---	---	---	---

(c) Vrstica elementa

0	1	1	1	2	2
---	---	---	---	---	---

(d) Indeks prvega elementa
v stolpcu

0	2	3	5
---	---	---	---

Yale format matrike

To je zelo uporabna metoda, ki se tudi uporablja za zgoščevanje LR tabel. Opis bomo nadaljevali v poglavju 2.5. Razlika med zapisom LR tabel ali matrik je v tem, kako si elementi sledijo v tabeli. Pri matrikah se običajno uporablja stolpec, pri LR tabelah se uporabljajo vrstice.

2.2 Metode zgoščevanja LR tabel

Metode za uporabo pri zgoščevanju LR tabel so bile podrobno analizirane v delu [3, 1].

Tako imamo metode:

- Metoda barvanja grafov (*Graph Coloring Scheme*)
- Metoda eliminacije (*Line Elimination Scheme*)
- Metoda odmika (*Row Displacement Scheme*)
- Metoda zamika začetne in zadnje vrednosti (*Significant Distance Scheme*)
- Metoda vrstica, stolpec (*Row Column Scheme*)
- Metoda izogibanja ničnih vrednosti (*Suppressed Zero Scheme*)

2.3 Metoda barvanja grafov (GCS)

V tej metodi poskušamo združiti vrstice ali stolpce tako, da čimbolj zapolnimo prazne prostore. Kot primer imamo tabelo (2.2a), kjer poskušamo združiti vrstice, ki nimajo vrednosti v istih stolpcih. Ko to združevanje končamo, dobimo novo tabelo, ki je bolj zgoščena, in novo preslikavo vrstic (tabela 2.2b). Nova preslikava vrstic preusmeri staro vrstico v novo vrstico, s katero najdemo vrednosti določene vrstice. Ko so vrstice preslikane, dobimo bolj zgoščeno obliko tabele z novimi dimenzijami (2.2c). S tem končamo zgoščevanje po vrsticah. Isto lahko naredimo tudi z zgoščevanjem stolpcev in dobimo končno zgoščeno tabelo 2.2e. Da lahko preverjamo veljavnosti elementov, potrebujemo tudi bitno polje *sigmap* v velikosti tabele pred zgoščanjem, tako da ne moremo dobiti neveljavne vrednosti.

Metoda se imenuje metoda barvanja grafov zato, ker iskanje vrstic, ki jih lahko združimo, poteka po principu barvanja grafov. Vsaka vrstica predstavlja vozlišče v grafu, kjer poskušamo združiti vozlišča s tem, da uporabimo čim manj barvila in s tem zmanjšamo dimenzije tabele.

Zgoščevanje se začne z iskanjem sosednjih vrstic, ki jih kasneje uporabimo pri barvanju. Pri tem štejemo sosednje vrstice, katerih število pomaga pri vrstnem redu barvanja. Sosednja vrstica je tista vrstica, ki ima različne vrednosti na istih stolpcih. To nam pomaga pri barvanju, da sosednji vrstici ne moreta biti v isti barvi. Tista vrstica, ki ima največ sosednjih vrstic, bo bolj verjetno v svoji barvi, zato vrstice, ki jih imajo največ, barvamo najprej. Vsaka barva

predstavlja vrstico ali stolpec v novi tabeli.

Rezultat tega zgoščevanja je tabela, ki ima veliko manj praznega prostora. Kot slabost te metode je treba omeniti, da potrebujemo bitno polje (*sigmap*, algoritem 2.2), s katerim preverjamo, ali imamo v določeni vrstici ali stolpcu nenično vrednost. Brez bitnega polja ne moremo preverjati veljavnosti določenega elementa.

Tabela 2.2: Primer metode GCS

(a) Primer tabele

1			
	2		
		3	
		5	4

(b) Columnmap

1
1
1
2

(c) Nova tabela po zgoščevanju vrstic

1	2	3	
		5	4

(d) Rowmap

1	2	3	1
---	---	---	---

(e) Končna oblika nove tabele

1	2	3
4		5

Algoritem

V prvem delu algoritma določimo vse sosednje vrstice za vsako vrstico posebej, kar prikazuje algoritem 2.3. Nato sortiramo vrstice po številu sosednjih vrstic in sledi še barvanje teh vrstic. Barvanje poteka po algoritmu 2.4. Začnemo s prvo barvo in poskušamo barvati z že obstoječimi barvami. Če ne najdemo prostora z obstoječimi barvami, moramo dodati novo barvo. Kakovost rezultata zgoščevanja je odvisen od števila barv, ki jih potrebujemo, da lahko združimo vrstice.

Algorithm 2.1 GCS branje tabele

Izhod: Vrednost elementa na določenem mestu**Vhod:** $sigmap[visina][sirina]$ (Bitno polje za preverjanje veljavnosti elementov)**Vhod:** $novatabela[][]$ (Zgoščena oblike tabele)**Vhod:** $rowmap[visina]$, $columnmap[sirina]$ (Preslikava vrstic in stolpcev v novo tabelo)**Vhod:** i, j (Vrstica in stolpec zahtevanega elementa) **if** $sigmap[i, j] = 1$ **then** **return** $novatabela[rowmap[i], columnmap[j]]$ **else** **return** 0 **end if**

Algorithm 2.2 Generiranje tabele Sigmap

Izhod: $sigmap[visina_{tabela}][sirina_{tabela}]$ (Bitno polje za preverjanje veljavnosti elementov)**Vhod:** $vhodnatabela[visina_{tabela}][sirina_{tabela}]$, $sirina_{tabela}$, $visina_{tabela}$ **for** $i = 0$ **to** $visina_{tabela}$ **do** **for** $j = 0$ **to** $sirina_{tabela}$ **do** **if** $vhodnatabela[i, j] \neq 0$ **then** $sigmap[i, j] \leftarrow 1$ **else** $sigmap[i, j] \leftarrow 0$ **end if** **end for** **end for**

Algorithm 2.3 GCS iskanje in dodajanje sosednjih vrstic

Vhod: $vhodnatabela[visina_{tabele}][sirina_{tabele}]$, $sirina_{tabele}$, $visina_{tabele}$
Izhod: $sosednje_{vrstice}[visina_{tabele}][]$ (Sosednje vrstice)
Izhod: $stevilososedov_{vrstice}[visina_{tabele}]$ (Število sosednjih vrstic posamezne vrstice)
Izhod: $sortirane_{vrstice}[visina_{tabele}]$ (Vsebuje sortirane indekse vrstic po ključu števila sosednjih vrstic)
for $i = 0$ **to** $visina_{tabele}$ **do**
 for $j = 0$ **to** i **do**
 $jesosednjavrstica \leftarrow 1$
 for $x = 0$ **to** $sirina_{tabele}$ **do**
 if $vhodnatabela[j][x] \neq 0$ **and** $vhodnatabela[i][x] \neq 0$ **and**
 $vhodnatabela[i][x] \neq vhodnatabela[j][x]$ **then**
 $jesosednjavrstica \leftarrow 0$
 end if
 end for
 if $jesosednjavrstica = 1$ **then**
 $sosednje_{vrstice}[i][stevilososedov_{vrstice}[i]] \leftarrow j$
 $stevilososedov_{vrstice}[i] \leftarrow stevilososedov_{vrstice}[i] + 1$
 $sosednje_{vrstice}[j][stevilososedov_{vrstice}[j]] \leftarrow i$
 $stevilososedov_{vrstice}[j] \leftarrow stevilososedov_{vrstice}[j] + 1$
 end if
 end for
end for
 $sortirane_{vrstice}[] \leftarrow \text{Qsort}(stevilososedov_{vrstice})$ {Sortiramo vrstice po številu sosednjih vrstic}

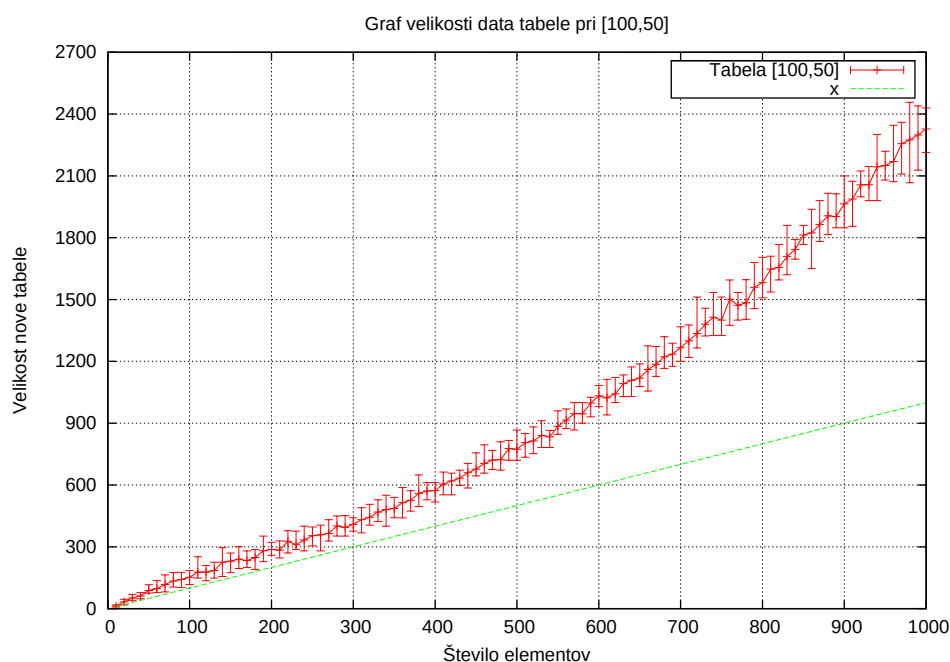
Algorithm 2.4 GCS barvanje grafov (zgoščevanje)

Vhod: $sirina_{tabela}$, $visina_{tabela}$, $vhodnatabela[visina_{tabela}][sirina_{tabela}]$
Vhod: $sosednje_{vrstice}[visina_{tabela}][]$
Vhod: $stevilososedov_{vrstice}[visina_{tabela}]$
Vhod: $sortirane_{vrstice}[visina_{tabela}]$
Izhod: $ntable[][]$ (Nova tabela, zgoščena po vrsticah)
Izhod: $barva_{vrstice}$ Preusmeritve v novo vrstico pri branju jo imenujemo *rowmap*
for $i = 0$ **to** $visina_{tabela}$ **do**
 $barva_{vrstice}[i] \leftarrow -1$
end for
 $maks_{barva} \leftarrow 0$
for $i = 0$ **to** $visina_{tabela}$ **do**
 $trenutna_{vrstica} \leftarrow sortirane_{vrstice}[i]$
 for $trenutna_{barva} = 0$ **to** $maks_{barva} + 1$ **do**
 $zeimamobarvo \leftarrow 0$
 for $j = 0$ **to** $stevilososedov_{vrstice}[trenutna_{vrstica}]$ **do**
 if $trenutna_{barva} = barva_{vrstice}[trenutna_{vrstica}]$ **then**
 $zeimamobarvo \leftarrow 1$ {Sosednja vrstica s to barvo že obstaja}
 KONEC FOR ZANKE
 end if
 end for
 if $zeimamobarvo = 0$ **then**
 $barva_{vrstice}[trenutna_{vrstica}] \leftarrow trenutna_{barva}$ {Našli smo barvo}
 end if
 end for
 if $zeimamobarvo \neq 0$ **then**
 $maks_{barva} \leftarrow maks_{barva} + 1$ {Nismo barvali, povečamo število barv}
 $barva_{vrstice}[trenutna_{vrstica}] \leftarrow maks_{barva}$
 end if
end for
 Naredimo novo tabelo ($ntable$), ki ima dimenzije $[maks_{barva}][sirina_{tabela}]$.
for $i = 0$ **to** $visina_{tabela}$ **do**
 for $j = 0$ **to** $sirina_{tabela}$ **do**
 $ntable[barva_{vrstice}[i]][j] \leftarrow vhodnatabela[i][j]$ {Vpišemo vrednosti v novo obliko}
 end for
end for

Analiza algoritma

Velikost vseh podatkov, ki jih potrebujemo za dostop do zgoščene oblike, se povečuje linearno glede na dimenzije tabele. Pomanjkljivost metode je, da nima možnosti prepoznave ponavljajočih se vrednosti, kot to počneta metoda eliminacije vrstic in metoda odmika vrstic. Velika slabost te metode je, da potrebujemo bitno polje, v katerem imamo zapisano veljavnost določene celice. Zgoščena oblika nove tabele ima še vedno določene prazne prostore, ki jih nismo mogli združiti. Iz slike 2.2 je razvidno, kakšen je presežek praznega prostora, glede na število elementov, ki je v tabeli. Imamo tabelo dimenzije 100*50, kar nam prinese pri polni zasedenosti 5000 elementov. Na sliki je tudi razvidno, da imamo pri dvajset odstotni zasedenosti tabele, še dvakrat toliko presežka, kot je elementov v tabeli.

Slika 2.2: GCS graf



2.4 Metoda eliminacije (LES)

Metoda deluje na principu eliminacije enakovrednih vrstic in stolpcev. To počnemo izmenično. Najprej izločimo vrstico, nato stolpec. Ko eliminiramo vse, kar lahko, dobimo novo tabelo, ki ima drugačne dimenzije kot tabela, s katero smo začeli. Za vrstice ali stolpce, ki so bili eliminirani, si zapišemo, v katerem koraku so bili eliminirani in jih tako ločimo med seboj. Za tiste vrstice ali stolpce, ki jih ne moremo eliminirati, zapišemo, da so bili eliminirani v zadnjem koraku. Tako dobimo štiri nove tabele, dve za branje vrstic in dve za branje stolpcev.

Branje tabel poteka po algoritmu 2.5. Tako imamo tabeli dr in dc , ki predstavljata korak eliminacije, in tabeli r in c , v katerih so zapisane vrednosti eliminiranih vrstic, indeksi na nove vrstice ali stolpce v novi tabeli (primer je v tabeli 2.3). Ker gre v tem primeru za preusmerjanje v novo tabelo, nastane, kot pri metodi barvanja grafov, problem preverjanja veljavnosti elementov. Zato potrebujemo še bitno polje za preverjanje veljavnosti. Zgoščevanje poteka po algoritmu 2.6. Pri algoritmu za zgoščevanje tabel smo izpustili funkciji, ki preverjata ali lahko eliminiramo določen stolpec ali vrstico. Funkciji $f_ali_lahko_eliminiramo_vrstico()$ in $f_ali_lahko_eliminiramo_stolpec()$ preverjata, ali imamo iste vrednosti na vseh elementih vrstice ali stolpca. Pri tem moramo paziti, da ne upoštevamo že eliminiranih vrstic ali stolpcev. Ob zaključku algoritma smo izpustili prepis neeliminiranih vrstic in stolpcev v bolj zgoščeno obliko.

Tabela 2.3: LES primer

(a) Tabela	(b) r	(c) dr	(d) c	(e) dc																																													
<table> <tr><td></td><td></td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td>14</td></tr> <tr><td></td><td>19</td><td></td><td></td><td></td></tr> <tr><td></td><td></td><td></td><td></td><td></td></tr> <tr><td>45</td><td>81</td><td></td><td></td><td>70</td></tr> </table>										14		19									45	81			70	<table> <tr><td>0</td></tr> <tr><td>14</td></tr> <tr><td>19</td></tr> <tr><td>0</td></tr> <tr><td>70</td></tr> </table>	0	14	19	0	70	<table> <tr><td>1</td></tr> <tr><td>3</td></tr> <tr><td>5</td></tr> <tr><td>7</td></tr> <tr><td>9</td></tr> </table>	1	3	5	7	9	<table> <tr><td>45</td><td>81</td><td>0</td><td>0</td><td>1</td></tr> </table>	45	81	0	0	1	<table> <tr><td>2</td><td>6</td><td>4</td><td>13</td><td>13</td></tr> </table>	2	6	4	13	13
				14																																													
	19																																																
45	81			70																																													
0																																																	
14																																																	
19																																																	
0																																																	
70																																																	
1																																																	
3																																																	
5																																																	
7																																																	
9																																																	
45	81	0	0	1																																													
2	6	4	13	13																																													

Algorithm 2.5 LES branje tabele

Izhod: Vrednost elementa na določenem mestu

Vhod: $\text{sigmap}[\text{visina}][\text{sirina}]$ (Bitno polje za preverjanje veljavnosti elementov)

Vhod: i, j (Vrstica in stolpec zahtevanega elementa)

Vhod: $\text{dr}[\text{visina}], \text{r}[\text{visina}]$ (Vrstice)

Vhod: $\text{dc}[\text{sirina}], \text{c}[\text{sirina}]$ (Stolpci)

Vhod: $\text{novatabela}[][]$ (Nova tabela, v kateri so elementi, ki jih nismo mogli zgostiti)

```

if  $\text{sigmap}[i][j] = 1$  then
  if  $\text{dr}[i] < \text{dc}[j]$  then
    return  $\text{r}[i]$ 
  else
    if  $\text{dr}[i] > \text{dc}[j]$  then
      return  $\text{c}[j]$ 
    else
      return  $\text{novatabela}[\text{r}[i]][\text{c}[j]]$ 
    end if
  end if
end if

```

Algorithm 2.6 LES zgoščevanje

Vhod: $vhodna_{tabela}[visina][sirina], sirina, visina$ **Izhod:** $nova_{tabela}[], dr[sirina], dc[visina], r[sirina], c[visina]$ $dr_{vrstice}[sirina] \leftarrow -1, r_{vrstice}[sirina] \leftarrow -1$ $dc_{vrstice}[visina] \leftarrow -1, c_{vrstice}[visina] \leftarrow -1$ $scan \leftarrow 0$ **repeat** $dodali_vrstico \leftarrow 0, dodali_stolpec \leftarrow 0$ **for** $j = 0$ **to** $visina$ **do****if** $f_{ali_lahko_eliminiramo_vrstico}(j)$ **then**

Preverimo ali ima vrstica po celi dolžini iste vrednosti

 $dr[j] \leftarrow scan, r[j] \leftarrow vrednost_{vrstice}$ $dodali_vrstico \leftarrow 1$

KONEC FOR ZANKE

end if**end for** $scan \leftarrow scan + 1$ {koncali vrstico}**for** $j = 0$ **to** $sirina$ **do****if** $f_{ali_lahko_eliminiramo_stolpec}(j)$ **then**

Preverimo ali ima vrstica po celem stolpcu iste vrednosti

 $dc[j] \leftarrow scan, c[j] \leftarrow vrednost_{stolpca}$ $dodali_stolpec \leftarrow 1$

KONEC FOR ZANKE

end if**end for** $scan \leftarrow scan + 1$ {koncali stolpec}**until** $dodali_stolpec \neq 0$ **or** $dodali_vrstica \neq 0$ $scan \leftarrow scan + 1$

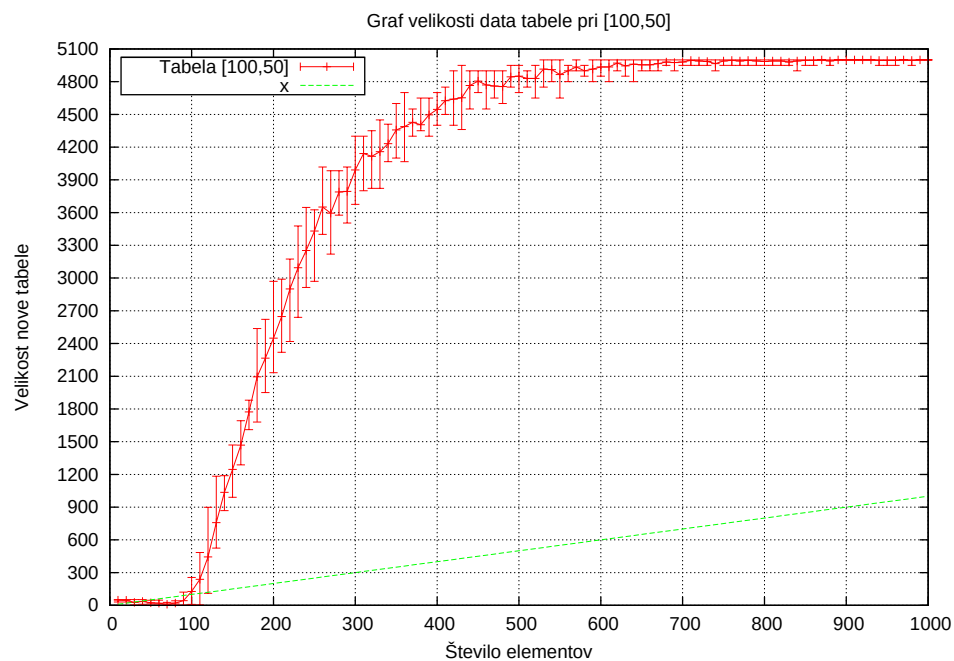
Določimo indekse vrsticam, stolpcem, ki niso bili eliminirani

for $j = 0$ **to** $visina$ **do****if** $dr[j] = -1$ **then** $r[j] \leftarrow indeks_vrstice_v_novi_tabeli, dr[j] \leftarrow scan$ {Določimo novo vrstico v novi tabeli}**end if****end for****for** $j = 0$ **to** $sirina$ **do****if** $dc[j] = -1$ **then** $c[j] \leftarrow indeks_stolpca_v_novi_tabeli, dc[j] \leftarrow scan$ {Določimo nov stolpec v novi tabeli}**end if****end for**Vse neeliminirane vrstice in stolpce prepišemo v novo tabelo ($nova_{tabela}[][]$)

Analiza algoritma

Algoritem deluje učinkovito, dokler še imamo veliko praznih vrstic oz. stolpcev (slika 2.3). Ko nam zmanjka vrstic, ki bi jih lahko eliminirali, ostane samo tabela v originalni velikosti in prostora ne pridobimo.

Slika 2.3: LES graf



2.5 Metoda odmika (RDS)

Za lažje razumevanje delovanja metode moramo razumeti, kako je tabela predstavljena v računalniku. Tabela, ki ima dve dimenziji, je v računalniku predstavljena kot enodimenzionalno polje. Da je to mogoče, moramo poznati dimenzije tabele. Tabela 2.4 predstavlja v pomnilniku 5×10 celic. Prva vrstica predstavlja prvih 10 elementov, druga vrstica naslednjih 10 elementov in tako dalje.

Metoda, ki je predstavljena v delu [5], deluje na principu odmika vrstic. Vrstice shranjujemo tako, kot so običajno predstavljene dvodimenzionalne tabele v računalniku, se pravi linearno, vendar v našem primeru nimajo konstantnega odmika. S takim načinom predstavitve tabele lahko izkoristimo prostor, ki ga imajo na pol prazne vrstice. To pomeni, da se vrstice lahko prekrivajo. Kot primer lahko vzamemo vrstico z enim elementom. Lahko jo vrinemo kamorkoli v seznam, kjer je prosto mesto. Ker se vrstice lahko prekrivajo, potrebujemo še dodatno tabelo, v kateri je zapisano, za katero vrstico je veljavna vrednost. Tabele v sliki 2.5 predstavljajo novo zgoščeno obliko tabele, ki je predstavljena v sliki 2.4. Novo obliko predstavljajo tri nove tabele - zgoščena oblika, kjer imamo vrednosti elementov, odmik ali *displacement* tabela, v kateri imamo odmike za vsako vrstico posebej in *check* tabela, s katero lahko preverjamo veljavnost elementa.

Branje teh tabel poteka po algoritmu 2.7. Iščemo vrednost tabele v določeni vrstici in stolpcu. Začnemo z *displacement* tabelo, v kateri dobimo začetni odmik za določeno vrstico, in temu odmiku prištejemo številko stolpca, ki nas zanima. Tako dobimo indeks v *data* tabeli. Da je ta vrednost veljavna, moramo preveriti tudi to, ali je vrednost veljavna za pravo vrstico (*check* tabela).

Tabela 2.4: Vhodna tabela

[illegible]

Tabela 2.5: Nova predstavitev

	1	2	3	4	5	6	7	8	9	10	11	12	13	odmik
vrstica														
1				87					14		62			3
2	13	99					66	67						0
3			65											-6
4					28					11				0
5														0

<i>data</i> tabela	13	99	65	87	28		66	67	14	11	62		
<i>check</i> tabela	1	1	2	0	3		1	1	0	3	0		

Tabele, ki jih potrebujemo so:

- *data* tabela, ki vsebuje elemente,
- *displacement* tabela, ki vsebuje indeks za določeno vrstico,
- *check* tabela, ki vsebuje vrednost vrstice (stolpec), pri kateri je vrednost veljavna.

Algorithm 2.7 RDS branje tabele

Izhod: Vrednost elementa na določenem mestu

Vhod: *vrstica*, *stolpec* (Zahtevan element)

Vhod: *displace*[], *data*[], *check*[]

Vhod: *stolpec*, *vrstica* (Vrstica in stolpec zahtevanega elementa)

$indeks \leftarrow \text{displace}[\text{vrstica}] + \text{stolpec}$

if $indeks \geq 0$ and $indeks \leq \text{length}(\text{data} [])$ **then**

if $\text{check} [indeks] = \text{vrstica}$ **then**

return $\text{data}[indeks]$

end if

end if

return 0

Obstajajo tudi različne variacije zgoščevanja. Vrstice lahko zapisujemo, kot je prikazano v zgornjem primeru, ali pa elemente zapisujemo, kot si sledijo v stolpcih. Končni rezultat je različen in je lahko v določenih primerih zapis krajši. Razlika je tudi v različnem načinu preverjanja veljavnosti elementov.

V zgornjem primeru imamo številko vrstice, lahko pa bi imeli številko stolpca. Razlog za uporabo številke stolpca je, da nam dovoljuje ponavljanje celotne vrstice. Torej imamo dve vrstici, ki imata iste vrednosti in imata tako isti odmik. Pri tem je treba paziti, da se dve vrstici, ki nimata istih vrednosti, ne prekrivata s številko stolpca.

Prednost tega algoritma je, da ima možnost ponavljanja celotne vrstice in da je časovna zahtevnost konstantna $O(1)$.

Algoritem zgoščevanja tabel

Cilj samega algoritma za zgoščevanje je poiskati odmike vrstic tako, da imamo zgoščeno (*data*) tabelo v velikosti števila neničnih elementov. Iskanje teh rešitev je *NP* problem, ker nam iskanje najboljšega vrstnega reda vrstic predstavlja problem v velikosti $N!$.

V delu [5] je predstavljena rešitev, ki se imenuje metoda prvega primernege prostora s padajočim številom elementov (*first-fit decreasing*), v nadaljevanju metoda prvega primernege. To pomeni, da vrstice, ki imajo največje število elementov, najprej dodamo v seznam dodajanja. Posledica tega je, da dajemo možnost manjšim vrsticam, da se vrinejo v vrstice z večjim številom elementov. Algoritem, ki ga predstavljamo, ni identičen algoritmu, ki je bil predstavljen v delu [5], ampak ima dodano še preverjanje širine vrstice, ki nam pomaga v primeru, ko imamo vrstice z istim številom elementov.

Sam algoritem za zgoščevanje zaradi enostavnosti lahko razdelimo na tri dele. V prvem delu določimo število elementov in širino vrstice (algoritem 2.8). V drugem delu sortiramo vrstice po številu neničnih elementov in po dolžini vrstice (algoritem 2.9). Dolžina vrstice naj bi se začela s prvim neničnim elementom, do zadnjega neničnega elementa. Rezultat sortiranja uporabimo pri zadnjem delu, tako da dodajamo vrstice, ki so bile sortirane po prejšnjem algoritmu (algoritem 2.10). Razlog za sortiranje vrstic je v tem, da vrstice, ki imajo več elementov, dodamo najprej, s tem pa lahko pridobimo prostor za krajše vrstice.

Časovno zahtevnost algoritmov lahko razdelimo na 3 dele. V nadaljevanju bo n predstavljal red tabele. Časovno zahtevnost algoritma 2.9, 2.10 lahko definiramo kot $O(n^2)$. Za algoritem 2.10 težje določimo, koliko korakov bo potrebno, da najdemo prostor za vrstico. Zato lahko rečemo, da za iskanje prostega mesta potrebujemo od 1 do n korakov za vsako vrstico. Iz tega lahko sklepamo, da bo algoritem potreboval $O(n)$ v najboljšem primeru ali $O(n^2)$ v najslabšem primeru.

Algorithm 2.8 RDS štetje elementov in določitev dolžine vrstic

Vhod: $tabela[visina_{tabela}][sirina_{tabela}]$ (Tabela, ki jo zgoščamo)

Izhod: $stevilo_elementov[visina_{tabela}], sirina_vrstice[visina_{tabela}]$

```

for  $i = 0$  to  $visina_{tabela}$  do
   $zacetek\_vrstice, konec\_vrstice \leftarrow -1$ 
   $temp\_stevilo\_elementov \leftarrow 0$ 
  for  $j = 0$  to  $sirina_{tabela}$  do
    if  $tabela[i][j] \neq 0$  then
      if  $zacetek\_vrstice < 0$  then
         $zacetek\_vrstice, konec\_vrstice \leftarrow j$  {Našli začetek vrstice}
      else {Že našel začetek vrstice}
         $konec\_vrstice \leftarrow j$ 
      end if
       $temp\_stevilo\_elementov \leftarrow temp\_stevilo\_elementov + 1$ 
    end if
  end for
   $stevilo\_elementov[i] \leftarrow temp\_stevilo\_elementov$ 
   $sirina\_vrstice[i] \leftarrow (konec\_vrstice + 1) - zacetek\_vrstice$ 
end for

```

Algorithm 2.9 RDS sortiranje vrstic

Vhod: $stevilo_elementov[visina_{tabele}]$ **Vhod:** $sirina_vrstice[visina_{tabele}]$ **Izhod:** $sortirane_vrstice[visina_{tabele}]$ Sortirana tabela z indeksi vrstic $count \leftarrow 0$ {Število vrstic, dodanih v tabelo}**for** $i = 0$ **to** $visina_{tabele}$ **do** $pozicija \leftarrow count - 1$ {Pozicija v tabeli} $x \leftarrow stevilo_elementov[sortirane_vrstice[pozicija]]$ $y \leftarrow stevilo_elementov[i]$ {Sortiramo po številu elementov}**while** $pozicija \geq 0$ **and** $x < y$ **do** $pozicija \leftarrow pozicija - 1$ **end while** $j \leftarrow sirina_vrstice[sortirane_vrstice[pozicija]]$ $k \leftarrow sirina_vrstice[i]$ {Sortiramo po širini vrstic}**while** $pozicija \geq 0$ **and** $x = y$ **and** $j < k$ **do** $pozicija \leftarrow pozicija - 1$ **end while****for** $j = count - 1$ **to** $position$ **do** $sortirane_vrstice[j + 1] \leftarrow sortirane_vrstice[j]$ **end for** $sortirane_vrstice[pozicija + 1] \leftarrow i$ $count \leftarrow count + 1$ **end for**

Algorithm 2.10 RDS zgoščevanje

Vhod: $tabela[visina_{tabele}][sirina_{tabele}]$, $sortirane_vrstice[visina_{tabele}]$ **Vhod:** $sirina_vrstice[visina_{tabele}]$, $zacetek_vrstice[visina_{tabele}]$ **Izhod:** $tabele\ displance[], data[], check[]$

```

for  $i = 0$  to  $visina_{tabele}$  do
   $vrstica \leftarrow sortirane_vrstice[i]$  {Indeks vrstice (sortirane)}
   $length \leftarrow sirina_vrstice[vrstica]$ 
   $place\_at \leftarrow -1$ 
  repeat
     $ok \leftarrow 1$ 
     $place\_at \leftarrow place\_at + 1$ 
    for  $k = 0$  to  $length$  do
       $j \leftarrow k + zacetek_vrstice[vrstica]$ 
      if  $tabela[vrstica][j] \neq 0$  and  $data[place\_at + k] \neq 0$  then
         $ok \leftarrow 0$  {Ni mogoče dodati vrstice na tem indeksu}
      end if
    end for
  until  $ok \neq 1$  {Dodajamo vrstico}
   $displace[vrstica] \leftarrow place\_at - zacetek_vrstice[vrstica]$ 
  for  $k = 0$  to  $length$  do
     $j \leftarrow k + zacetek_vrstice[vrstica]$ 
    if  $tabela[vrstica][j] \neq 0$  then
       $data[place\_at + k] \leftarrow tabela[vrstica][j]$ 
       $check[place\_at + k] \leftarrow vrstica$ 
    end if
  end for
end for

```

Analiza algoritma

Da algoritem prvega primernege deluje učinkovito, mora imeti tabela določene lastnosti.

Imamo tabelo dimenzije $m * m$, v kateri imamo n neničnih elementov, in definiramo funkcijo $n(l)$, ki predstavlja število vseh neničnih elementov v vrsticah z več kot l elementov.

Izrek, ki je omenjen v delu [5], pravi, da če imamo tabelo, ki ima lastnost harmoničnega padanja:

Definicija. H : Za vse l , $n(l) \leq n/(l+1)$,
potem velja, da bodo vsi elementi prišli v tabelo v velikosti n .

Dokaz. Predpostavljamo, da ima ta vrstica vsaj 1 nenični element. Po definiciji je H število prejšnjih neničnih elementov največ n/l . Kar predstavlja, da imamo zasedenega največ l prostora od n . Torej velja, da pridejo vsi nenični elementi v tabelo dolžine n . $\square$

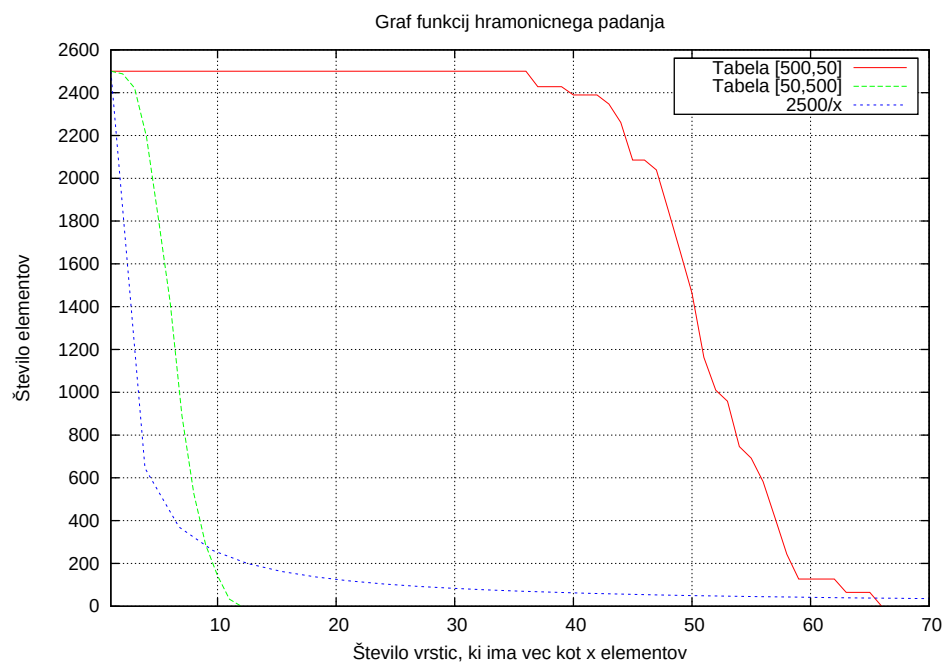
To tudi pomeni, da če se vrednost $n(l)/n$ hitro zmanjšuje, s tem, ko se l povečuje, potem lahko rečemo, da bo metoda prvega primernege delovala učinkovito. Iz te lastnosti lahko tudi sklepamo, da bo končna velikost podatkov $2 * n + m$. Lastnost harmoničnega padanja preprosto pomeni, da imamo polovico vrstic samo z enim neničnim elementom.

Imamo dva primera tabel - tabeli v velikosti $[500, 50]$ in $[50, 500]$. Vsebujeta 2500 naključno generiranih elementov. Za obe tabeli imamo dve funkciji. Zelena črta predstavlja funkcijo $n(l)$, polni črti predstavljata harmonično padanje $n/(l+1)$. Iz grafa je razvidno, da ima tabela z več vrsticami kot stolpci, zelo odmaknjeno funkcijo od harmoničnega padanja. Posledica tega je, da ima zgoščena oblika tabele zelo veliko praznih prostorov, ker se vrstice z manj elementi ne morejo vrniti v prazne prostore. Iz tega grafa je razvidna tudi zasedenost same vrstice.

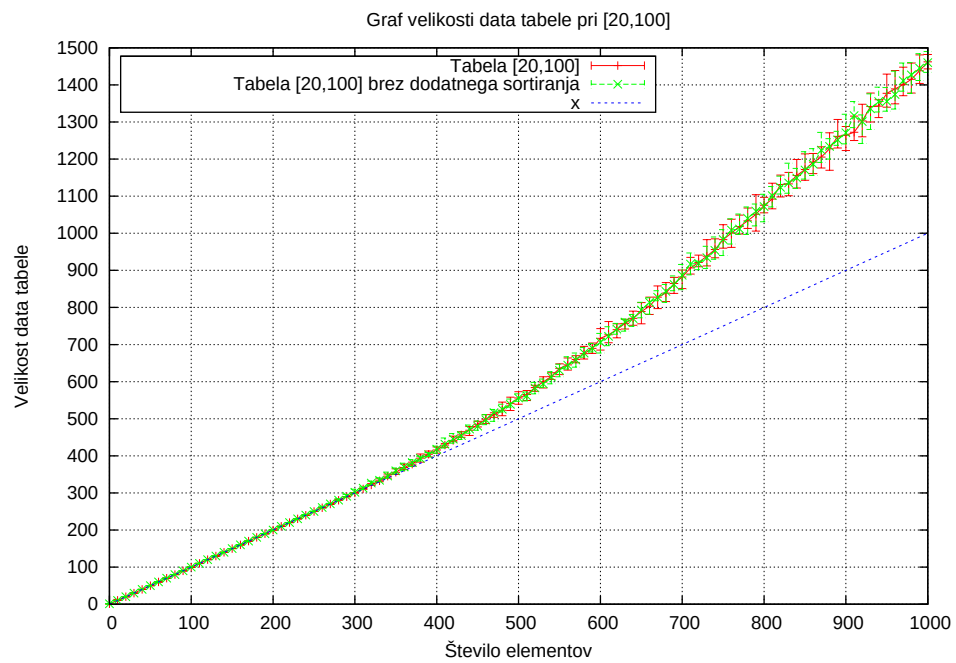
Primer za to, do katere gostote elementov je ta metoda še primerna, je v najboljšem primeru $2 * n + m$ elementov. Če imamo tabelo velikosti $[50, 50]$, ki predstavlja 2500 elementov, nam v najboljšem primeru, ko je ta zapis še smislen, predstavlja 1225 število neničnih elementov. Teoretična zgornja meja, ko je ta zapis še smiselno uporabiti, bi bila 49-odstotna zasedenost, vendar v praksi to ni vedno mogoče. Glede na to, da v našem primeru uporabljamo odmik vrstice, ne moremo pričakovati, da bomo imeli dobre rezultate, če bomo število vrstic zmanjševali. Če imamo več stolpcev kot vrstic, se s povečanjem gostote zmanjša možnost, da bi se vrstice prekrivale. To je vidno v sliki 2.6, v kateri imamo 100 stolpcev in 50 vrstic, in v sliki 2.5, v kateri imamo

20 stolpcev in 100 vrstic. Če naredimo primerjavo velikosti *data* tabele, je več kot očitno, da se v prvem primeru veliko hitreje povečuje velikost *data* tabele. Iz slike 2.5 je tudi razvidna razlika, če uporabljamo še dodatno sortiranje po širini vrstice. Ta nam pomaga v primeru, ko imamo veliko vrstic z enakim številom elementov in v tem primeru daje za odtenek boljše rezultate.

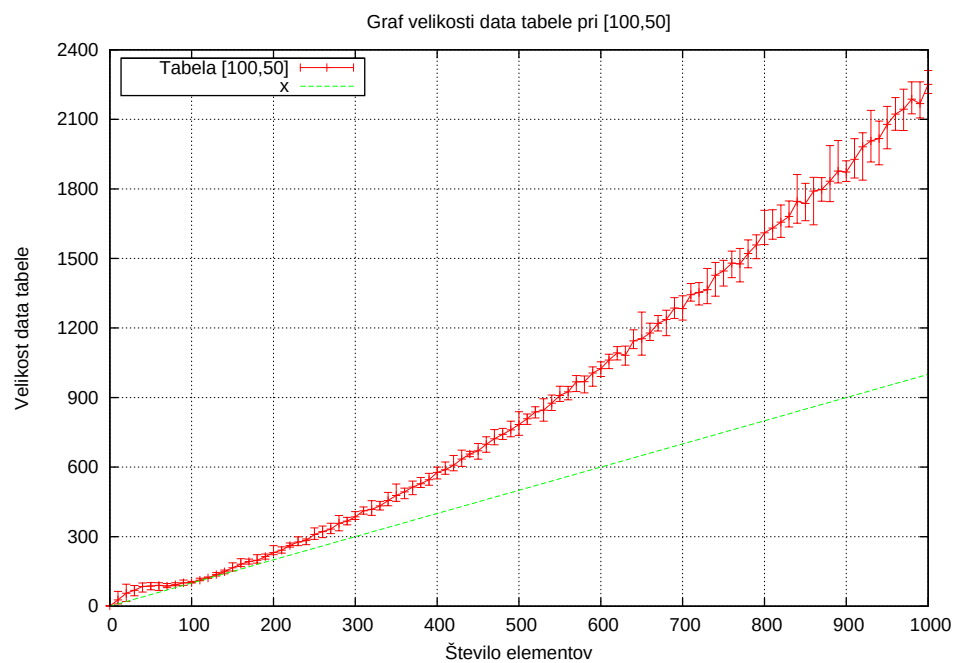
Slika 2.4: Harmonično padanje



Slika 2.5: RDS graf tabele [20,100]



Slika 2.6: RDS graf tabele [100,50]



2.6 Metoda zamika začetne in zadnje vrednosti (SDS)

Metoda deluje tako, da si začnemo zapisovati vrstico od prve nenične vrednosti, do zadnje nenične vrednosti. Tako dobimo dve novi tabeli *first* in *last*, ki predstavljata začetni in končni indeks vsake vrstice, s katero se začnejo nenične vrednosti. Branje tabele poteka po algoritmu 2.11. Ker metoda ne daje dobrih rezultatov, je ne obravnavamo podrobno.

Algorithm 2.11 SDS branje tabele

Izhod: Vrednost elementa na določenem mestu

Vhod: i, j (Vrstica in stolpec zahtevanega elementa)

Vhod: $first[visina]$, $last[visina]$, $rowpointer[visina]$

if $(j < first[i])$ **or** $(j > last[i])$ **then**

return 0

else

return $value[rowpointer[i]+j]$

end if

2.7 Metoda vrstica, stolpec (RCS)

Metoda zapisuje nenične elemente tako, kot si sledijo v vrstici ali stolpcu (ista metoda, drugačen način zapisa). Tako imamo tri tabele. V eni so vse nenične vrednosti, v drugi številke vrstice ali stolpca, za katero vrednost je veljavna nenična vrednost, in tretjo, v katero si zapisujemo začetek neke vrstice ali stolpca. Za primer imamo v poglavju 2.1 že koordiniran seznam. Slabost te metode je, da nimamo konstantnega dostopnega časa. Algoritem za branje RCS tabel je 2.12 in zgoščanje poteka po algoritmu 2.13.

Algorithm 2.12 RCS branje tabele

Izhod: Vrednost elementa na določenem mestu**Vhod:** *row_pointer[visina], column_index[], vrednosti[]***Vhod:** *i, j* (Vrstica in stolpec zahtevanega elementa)

```

for tmp=row_pointer[i] to row_pointer[i + 1] do
  if column_index[tmp] = j then
    return vrednosti[tmp]
  end if
end for
return 0

```

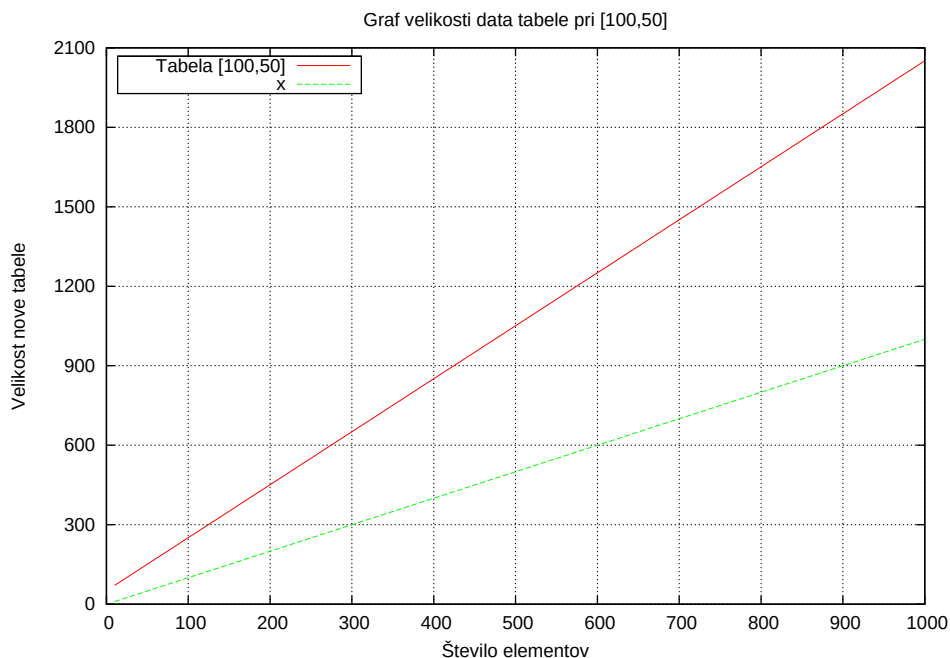
Algorithm 2.13 RCS zgoščevanje

Izhod: *row_pointer[visina], column_index[], vrednosti[]* Zgoščena oblika tabele**Vhod:** *tabela[visina][sirina], visina, sirina**stevilo_vrednosti* $\leftarrow$ 0*row_pointer*[0] $\leftarrow$ 0 {Prva vrstica se začne s prvim elementom}*stevilo_vrstic* $\leftarrow$ 1**for** *i* = 0 **to** *visina* **do** **for** *j* = 0 **to** *sirina* **do** **if** *tabela*[*i*][*j*] $\neq$ 0 **then** $column_index[stevilo_vrednosti] \leftarrow tabela[i][j]$ $stevilo_vrednosti \leftarrow stevilo_vrednosti + 1$ **end if** **end for** $row_pointer[stevilo_vrstic] \leftarrow stevilo_vrednosti$ $stevilo_vrstic \leftarrow stevilo_vrstic + 1$ **end for**

Analiza algoritma

Kot je bilo že v poglavju 2.1 omenjeno, tabela narašča z velikostjo neničnih elementov. Tako imamo tabelo v velikosti $2 * nnz + n$, kjer nnz predstavlja število neničnih elementov, in n , ki predstavlja število vrstic.

Slika 2.7: RCS graf



2.8 Metoda izogibanja ničnih vrednosti (SZS)

Metoda deluje zelo podobno kot metoda RCS. Razlika je v tem, da v tabelo zapisujemo še nenične elemente. Tako izgubimo tabelo, ki preverja veljavnost vrednosti za določen stolpec. Poleg tega, da si zapisujemo nenične elemente, dodamo še poseben zapis, v katerem zapisujemo začetek elementov z vrednostjo nič in število elementov. Bistvene prednosti v tem zapisu ni. Kot pri metodi RCS je tudi tu slabost nekonstanten čas dostopa.

2.9 Primerjava metod pri zgoščevanju LR tabel

Prva primerjava metod je v delu [3], dotakne se vseh metod in primerja njihovo učinkovitost na različnih programskih jezikih. Analiza je vidna v tabeli 2.8. Faktor, v tabeli predstavlja ulomek:

$$\frac{\text{prostor aktualne metode}}{\text{prostor najboljše metode}}$$

To pomeni, da metoda, ki zaseda najmanj prostora, dobi faktor 1, tako iz tabele vidimo, katera metoda v povprečju zaseda najmanj pomnilnika. Izkaže se, da je najboljša metoda GCS.

Naredili smo tudi analizo na lastnih LR tabelah. LR tabelo smo pridobili iz sintaksnega analizatorja *Bison*, za programski jezik *Pascal*, drugo tabelo pa iz sintaksnega generatorja *Eli*, za programski jezik *ALGOL60*. Vzeli smo samo odločitveno tabelo, ker tako najlažje naredimo primerjavo. Ločili smo tudi rezultate samega RDS algoritma s podporo ponavljanja vrstic in brez.

Med metodami smo primerjali, kateri podatki so potrebni za dostop in neničelne vrednosti same nove tabele. Za obe smo uporabili velikost podatkovnega tipa *celega števila*. Čeprav bi lahko podatke za dostop do tabele omejili na velikost manjšega podatkovnega tipa, nam v novodobnih računalniških sistemih to ne prinese nobene prednosti.

Tabela za programski jezik *ALGOL60* je pridobljena s sintaksnim generatorjem *Eli*[9]. Dobimo odločitveno tabelo v dimenzijah $56 * 119$, ki nam v nezgoščeni obliki predstavlja 6664 elementov. Tabela 2.6 predstavlja, koliko smo z določeno metodo zgostili odločitveno tabelo. Tudi v tem primeru je najboljša metoda GCS.

Tabela za programski jezik *Pascal* je pridobljena s sintaksnim generatorjem *Bison*[8]. Dobimo odločitveno tabelo v dimenzijah $67 * 410$, ki nam v nezgoščeni obliki predstavlja 27470 elementov. Tabela 2.7 predstavlja zgostitev kot v prejšnjem primeru, vendar nam v tem primeru najboljše rezultate vrne metoda RDS, z možnostjo ponavljanja vrstic.

Razlika med sintaksnimi generatorji in med različnimi jeziki se kaže v načinu zapisovanja LR tabel. *Bison* za delovanje analizatorja potrebuje odločitveno tabelo in tabelo prehodov pri analizatorju, *Eli* uporablja odločitveno tabelo za terminale in odločitveno tabelo za neterminale. Prednost algoritma RDS je v tem, da generator *Bison* generira tabele, v katerih je veliko ponovitev istih vrednosti v stanjih (vrstice).

Rezultate smo pridobili z izvirno kodo v dodatku B.

Tabela 2.6: Zgoščevanje odločitvene tabele za Algol60

	Podatki za dostop	Podatki nove tabele	Seštevek	Odstotek zgostitve
RDS	1269+119	1269	2657	39.9 %
RDS pon.	856+119	856	1831	27.5 %
RCS	866+120	866	1852	27.8 %
LES	119*2+56*2	56*87	3917	58.8 %
GCS	56+119+208(<i>sigmap</i>)	34*31	1437	21.56 %

Tabela 2.7: Zgoščevanje odločitvene tabele za Pascal

	Podatki za dostop	Podatki nove tabele	Seštevek	Odstotek zgostitve
RDS	1043+410	1043	2496	9.09 %
RDS pon.	241+410	241	892	3.25 %
RCS	769+411	769	1949	7.09 %
LES	410*2+67*2	63*13	1773	6.45 %
GCS	67+410+858(<i>sigmap</i>)	58*13	2089	7.6 %

Tabela 2.8: Primerjava metod pri [3]

Metoda	GCS	LES	RDS	SDS	LES&GCS	LES&RDS	RCS	SZS
Faktor	1.04	2.01	1.54	1.63	1.09	1.11	1.93	1.42

Poglavje 3

Implementacija sintaksnega analizatorja

V tem poglavju bomo predstavili delovanje samega LR analizatorja in realizacije določenih LR analizatorjev, ki se uporabljajo. Imamo novejšje analizatorje, ki uporabljajo za analizo eno odločitveno tabelo in tabelo prehodov, obstajajo pa tudi starejši analizatorji, ki imajo dve odločitveni tabeli eno za končne simbole in drugo za vmesne simbole. V tem primeru zgoščujemo dve tabeli. Tako pridemo do najbolj pogosto uporabljane metode RDS, ki nam hkrati omogoča zgoščevanje tabele prehodov in odločitvene tabele. Od starejših metod je najboljša metoda GCS, vendar ni primerna za uporabo pri sodobnih računalniških sistemih, kjer je dostop do bitnih polj drag.

Tako imamo analizatorje kot so [10, 8, 7, 9], ki uporabljajo prirejeno metodo RDS, in malo starejše analizatorje [9] *Cola*, ki uporabljajo metode, opisane v prejšnjem poglavju. Metoda RCS je nadgrajena toliko, da nam daje veliko boljše rezultate kot ostali algoritmi zato, ker imamo poleg možnosti ponavljanja vrstic še zgoščene tabele.

Sodobni analizatorji so zelo podrobno opisani v delu [6], odkoder prihaja tudi večina rešitev, ki se trenutno najbolj uporabljajo.

V tem poglavju bomo opisali delovanje same sintaksne analize na določenem vhodu na analizatorju *Bison*, opisali pa bomo tudi patent za zgoščevanje LR tabel.

3.1 Sintaksni analizator Bison

Sintaksni analizator *Bison* je odprtokodna rešitev in je vrste *LALR* (Look-Ahead LR). V tem delu predstavljamo delovanje tega analizatorja. Z uporabo

gramatike iz slike 1.1 *Bison* generira naslednjo tabelo 3.1. Za delovanje analizatorja *Bison* določi celoštevilčno vrednost vsakemu simbolu (končnim in vmesnim) tabela 3.2. *Bison* naredi tudi posebne simbole, ki se uporabljajo interno. Ti simboli so *\$end*, *error*, *\$undefined*, *\$accept*. Predstavljajo konec stavka, napako, nedefiniran simbol ali sprejem stavka. Delovanje *Bison* analizatorja lahko opišemo tudi kot skladovni avtomat na sliki 3.1, kjer polna črta predstavlja *pomik*, črtkana pa *prevedbo*.

3.1.1 Algoritem zgoščevanja tabel pri Bisonu

Algoritem zgoščevanja tabel poteka z metodo *RDS*, z dodano možnostjo ponavljanja vrstic. Poleg ponavljanja vrstic imamo še tabelo, ki vsebuje privzete vrednosti za vsako vrstico, ali stolpec, in tako zasedajo manj prostora v *data* tabeli. Zapis vmesnih in končnih simbolov v *data* se razlikuje, glede na to, ali zapisujemo končne ali vmesne simbole. Končne simbole zapisujemo tako, kot si sledijo v vrsticah, vmesne pa zapisujemo, kot si sledijo v stolpcih. Razlog za tak zapis je v privzeti vrednosti za določeno vrstico pri končnih simbolih in privzeti vrednosti za določen stolpec pri vmesnih simbolih. S tem načinom zapisa ne potrebujemo dveh tabel, ampak lahko združimo dve tabeli v eno in s tem tudi prihranimo prostor.

Pri končnih simbolih *Bison* shranjuje pomik kot pozitivno število in prevedbo kot negativno število, tako da za shranjevanje te tabele potrebujemo predznačeno celo število. Pred zgostitvijo tabele, se definira privzete vrednosti, za vsako vrstico v odločitveni tabeli, ki jih izločimo (tabela 3.3). Privzete vrednosti v odločitveni tabeli so vedno prevedbe. V tabeli za privzete vrednosti je tudi preverjanje napake, in sicer tako, da vrednost 0 predstavlja napako.

Kot je bilo omenjeno, tabela prehodov *Bison* shranjuje vrednosti, kot si sledijo v stolpcih oz. po vmesnih simbolih. Razlog za to je v privzeti vrednosti, ki je vezana na stolpec, in se ponavlja v določenem stolpcu. Privzete vrednosti se izloči tako, kot v odločitveni tabeli, s tem pa prihranimo prostor (tabela 3.4). Ko so tabele s privzetimi vrednostmi narejene, se zgostijo. Tako iz odločitvene tabele in tabele prehodov dobimo *displacement* tabeli (tabeli 3.5 in 3.6), *data* (tabela 3.7) ter *check* (tabela 3.8) tabelo.

Za delovanje analizatorja potrebujemo simbol, ki ga dobimo po operaciji prevedba. Tako tabela vsebuje simbol, ki ga dobimo po prevedbi po določenem pravilu (tabela 3.9). Pri isti operaciji potrebujemo tudi število simbolov, ki smo jih reducirali za določeno pravilo. Tako dobimo še tabelo 3.10.

Tabela 3.1: Odločitvena tabela in tabela prehodov

Stanje	Odločitvena tabela						Tabela prehodov		
	id	(	)	*	+	\$end	Izraz	Termin	Faktor
0	Po1	Po2					3	4	5
1	p6	p6	p6	p6	p6	p6			
2	Po1	Po2					6	4	5
3					Po8	Po7			
4	p2	p2	p2	Po9	p2	p2			
5	p4	p4	p4	p4	p4	p4			
6			Po10		Po8				
7	spr	spr	spr	spr	spr	spr			
8	Po1	Po2						11	5
9	Po1	Po2							12
10	p1	p1	p1	p1	p1	p1			
11	p1	p1	p1	Po9	p1	p1			
12	p3	p3	p3	p3	p3	p3			

Tabela 3.2: Simboli v Bisonu

Simbol	Vrednost
\$end	0
error	1
\$undefined	2
id	3
+	4
*	5
(	6
)	7
\$accept	8
Izraz	9
Termin	10
Faktor	11

Tabela 3.3: Privzeta akcija za določeno stanje

Stanje	0	1	2	3	4	5	6	7	8	9	10	11	12
privzeta vrednost	0	7	0	0	3	5	0	1	0	0	6	2	4

Tabela 3.4: Privzeto stanje za vsak vmesni simbol

Vmesni simbol	accept\$	Izraz	Termin	Faktor
privzeta vrednost	-1	3	4	5

Tabela 3.5: *Displacement* tabela za vsa stanja

Stanje	0	1	2	3	4	5	6	7	8	9	10	11	12
indeks	-1	-5	-1	0	-4	-5	2	-5	-1	-1	-5	-4	-5

Tabela 3.6: *Displacement* tabela za vse vmesne simbole

Vmesni simbol	accept\$	Izraz	Termin	Faktor
privzeta vrednost	-5	1	3	-2

Tabela 3.7: *Data* tabela

Indeks	0	1	2	3	4	5	6	7	8	9	10	11
vrednost	7	9	1	6	8	2	8	12	0	10	0	1

Tabela 3.8: *Check* tabela

Indeks	0	1	2	3	4	5	6	7	8	9	10	11
vrednost	0	5	3	2	4	6	4	9	-1	7	-1	8

Tabela 3.9: Simboli po pravilih

Indeks	0	1	2	3	4	5	6	7
vrednost	0	8	9	9	10	10	11	11

Tabela 3.10: Število simbolov v pravilu

Indeks	0	1	2	3	4	5	6	7	8	9
vrednost	0	2	3	1	3	5	0	1	0	0

[illegible]

3.1.2 Algoritem preverjanja sintakse

V tem delu predstavljamo algoritem delovanja sintaksnega analizatorja. Za delovanje samega analizatorja potrebujemo tabele, ki so bile že predstavljene:

- *yyTable* tabela 3.7,
- *yyCheck* tabela 3.8,
- *yyPact* tabela 3.5,
- *yyDefact* tabela 3.3,
- *yyPgoto* tabela 3.4,
- *yyr1* tabela 3.9,
- *yyr2* tabela 3.10.

Za delovanje potrebujemo tudi določene konstante, ki jih potrebujemo za pravilno delovanje:

- konstanto *yyPact_ninf*, ki predstavlja maksimalno negativno število, in nam pove, ali imamo sploh kako vrednost v tabeli za določeno stanje,
- konstanto *yyLast*, ki predstavlja maksimalni indeks pri *yyTabeli*,
- konstanto *yyNtokens*, s katero lahko ločimo vmesne in končne simbole,
- konstanto *yyFinal*, ki predstavlja stanje, v katerem ugotovimo pravilno sintakso.

Sam algoritem smo razdelili na dva dela. V prvem delu algoritma predstavljamo delovanje glede na posamezne operacije, v drugem delu pa predstavljamo, kako beremo tabele, da dobimo operacijo za določeno stanje. Algoritem se izvaja, dokler ima kakšno stanje v skladu ali dokler ne pridemo do stanja, kjer je vhod veljaven.

Algoritem v delu, kjer imamo različne operacije, je viden v algoritmu 3.1, v katerem predstavljamo razliko med operacijama prevedba in pomik. Pri operaciji pomik, potisnemo novo stanje na sklad in povečamo trenutni vhod za en znak. Pri operaciji prevedba, se nam zgodi malo več stvari. Najprej se nam zmanjša sklad, za število simbolov, glede na pravilo, ki ga reduciramo. Tako lahko preberemo stanje, ki ga potrebujemo za branje tabele prehodov. Kot je razvidno iz algoritma, dobimo novo stanje, ki ga dobimo iz zgoščene tabele ali

iz privzete tabele za določen vmesni simbol.

Imamo tudi algoritem, s katerim dobimo naslednjo operacijo (algoritem 3.2).

Izvedba kode je v dodatku C.

Algorithm 3.1 Bison sintaksna analiza

```

Vhod: yyPact[yyLast], yycheck[yyLast], yyTable[]
  stack[0]  $\leftarrow$  0
  dolzina  $\leftarrow$  dolzina + 1
  while dolzina > 0 do
    Pridobimo naslednjo operacijo, algoritem 3.2
    if akcija = Pomik then
      if novo_stanje = yyFinal then
        Dobili smo veljavno sintakso
      end if
      stack[dolzina]  $\leftarrow$  novo_stanje {Potisnemo novo stanje na sklad}
      dolzina  $\leftarrow$  dolzina + 1
      Trenuten vhod povečamo za en znak
    else
      if akcija = Prevedba then
        dolzina = dolzina - yyr2[pravilo] {S sklada snamemo yyr2[pravilo]
        simbolov}
        NovSimbol  $\leftarrow$  yyr1[pravilo]
        PrejsnjeStanje  $\leftarrow$  stack[dolzina]
        Nidx  $\leftarrow$  yyPgoto[NovSimbol - yyNtokens] + PrejsnjeStanje
        if 0  $\leq$  Nidx and Nidx  $\leq$  yyLast and yycheck[Nidx] ==
        PrejsnjeStanje then
          stack[dolzina]  $\leftarrow$  yyTable[Nidx]
        else
          stack[dolzina]  $\leftarrow$  yyDefgoto[NovSimbol - yyNtokens]
          dolzina  $\leftarrow$  dolzina + 1
        end if
      else
        if akcija = Napaka then
          Napaka v sintaksi
        end if
      end if
    end if
  end while

```

Algorithm 3.2 *Bison*, naslednja operacija

```

Vhod: yypact[yyLast], yycheck[yyLast], yytable[]
trenutno_stanje  $\leftarrow$  stack[dolzina]
if yypact[trenutno_stanje] = yyPact_ninf then
    akcija  $\leftarrow$  Prevedba
    pravilo  $\leftarrow$  yydefact[trenutno_stanje]
else
    znak  $\leftarrow$  Naslednji znak na vhodu
    yyn  $\leftarrow$  yyPact[trenutno_stanje] + znak
    if yyn < 0 or yyLAST < yyn or yyCheck[yyn]  $\neq$  znak then
        akcija  $\leftarrow$  Prevedba
        pravilo  $\leftarrow$  yyDefact[trenutno_stanje]
    else
        if yyTable[yyn] <= 0 then
            if yyTable[yyn] = yyPact_ninf or yyTable[yyn] = 0 then
                akcija  $\leftarrow$  Napaka
            else
                akcija  $\leftarrow$  Prevedba
                pravilo  $\leftarrow$  -yyTable[yyn]
            end if
        else
            akcija  $\leftarrow$  Pomik
            novo_stanje  $\leftarrow$  yyTable[yyn]
        end if
    end if
end if

```

3.2 Analiza patenta za zgoščevanje LR tabel

V tem poglavju opisujemo zgoščevanje LR tabel po patentu [2]. Patent predstavlja metodo za zgoščanje LR tabel. Primer, ki ga predstavlja, vsebuje dve tabeli, eno za končne simbole in drugo za vmesne simbole. Metodo smo primerjali z obstoječimi metodami, ki smo jih predstavili. Direktne primerjave z analizatorjem *Bison* ne moremo izvesti, ker ne uporabljamo istih tabel. Algoritem patenta uporablja podobne metode, kot jih uporablja sedanja različica *Bison*. Uporablja metodo z uporabo privzete vrednosti, za vsako stanje posebej (vrstice), nato permutira stolpce tako, da so karseda podobni zgornji trikotni obliki (matrike). To je tudi edina bistvena razlika, glede na sintaksni analizator *Bison*. Bistvene

prednosti s tem načinom pri *Bison*-u ne bi pridobili, ker je glavna prednost metode RDS pred ostalimi algoritmi to, da imamo čim več podobnih vrstic. Pri izvedbi patenta smo našli napako na sliki št. 12, v kateri so napačne vrednosti in napačna velikost. Izvedba patenta je v dodatku D.

Poglavje 4

Sklep

V diplomski nalogi je bila implementirana večina uspešnih metod za zgoščevanje LR tabel, z uporabo teh tabel pa je bil implementiran tudi sam LR analizator. Implementiran je bil patent, ki je trenutno edini objavljen patent za zgoščevanje LR tabel in z uporabo zgoščenih tabel tudi sam LR analizator.

Zgoščevanje redkih tabel so raziskovali že, odkar obstajajo programski jeziki, kompleksnost rešitev pa se je večala s procesno močjo računalnikov. V sedanjem času nismo toliko omejeni s pomnilnikom, zato so metode, ki jih sedaj uporabljamo, drugačne, kot so bile. To je bil tudi razlog za težave s pridobivanjem gradiva, ker se metode, ki so se uporabljale včasih, ne uporabljajo več.

Metode, ki so bile predstavljene, so se uporabljale za zgoščevanje LR tabel, in sicer za vmesne in končne simbole. Ko se je sam generator LR tabel spremenil toliko, da smo dobili tabele prehodov in odločitev, je s tem metoda RDS pridobila toliko prednosti pred ostalimi metodami, da se uporablja samo še ta.

Dodatek A

Priloga

Slike

1.1	Gramatika	5
2.1	Seznam seznamov	8
2.2	GCS graf	15
2.3	LES graf	19
2.4	Harmonično padanje	27
2.5	RDS graf tabele [20,100]	28
2.6	RDS graf tabele [100,50]	29
2.7	RCS graf	32
3.1	Bison skladovni avtomat	39

Tabele

1.1	LR Tabela	6
1.2	Delovanje LR analizatorja	6
2.1	Primer za <i>Matlab</i>	9
2.2	Primer metode GCS	11
2.3	LES primer	16
2.4	Vhodna tabela	20
2.5	Nova predstavitev	21
2.6	Zgoščevanje odločitvene tabele za Algol60	34
2.7	Zgoščevanje odločitvene tabele za Pascal	34
2.8	Primerjava metod pri [3]	34
3.1	Odločitvena tabela in tabela prehodov	37
3.2	Simboli v Bisonu	37
3.3	Privzeta akcija za določeno stanje	37
3.4	Privzeto stanje za vsak vmesni simbol	38
3.5	<i>Displacement</i> tabela za vsa stanja	38
3.6	<i>Displacement</i> tabela za vse vmesne simbole	38
3.7	<i>Data</i> tabela	38
3.8	<i>Check</i> tabela	38
3.9	Simboli po pravilih	38
3.10	Število simbolov v pravilu	38

Seznam algoritmov

2.1	GCS branje tabele	12
2.2	Generiranje tabele Sigmap	12
2.3	GCS iskanje in dodajanje sosednjih vrstic	13
2.4	GCS barvanje grafov (zgoščevanje)	14
2.5	LES branje tabele	17
2.6	LES zgoščevanje	18
2.7	RDS branje tabele	21
2.8	RDS štetje elementov in določitev dolžine vrstic	23
2.9	RDS sortiranje vrstic	24
2.10	RDS zgoščevanje	25
2.11	SDS branje tabele	30
2.12	RCS branje tabele	31
2.13	RCS zgoščevanje	31
3.1	Bison sintaksna analiza	41
3.2	<i>Bison</i> , naslednja operacija	42

- [1] P. Charles, “A practical method for constructing efficient LALR(k) parsers with automatic error recovery,” Dr. dizertacija, Univerza New York, Maj 1991. Dostopno na: <http://jikes.sourceforge.net/documents/thesis.pdf>
- [2] G. A. Charles, Philippe Gerard Fisher Jr. A method for lr (left-right) table compression. Dostopno na: <http://www.freepatentsonline.com/EP0313973.html>
- [3] P. Dencker, K. Dürre, and J. Heuft, “Optimization of parser tables for portable compilers,” *ACM Trans. Program. Lang. Syst.*, st. 6, str. 546–572, Oktober 1984. Dostopno na: <http://doi.acm.org/10.1145/1780.1802>
- [4] J. R. Gilbert, C. Moler, and R. Schreiber, “Sparse matrices in matlab: Design and implementation,” 1997.
- [5] R. E. Tarjan and A. C.-C. Yao, “Storing a sparse table,” *Commun. ACM*, st. 22, no. 11, str. 606–611, 1979.
- [6] A. V. A. . M. S. L. . R. S. . J. D. Ullman, *Compilers: Principles, Techniques, and Tools: International Edition*. Pearson Education, 2007.
- [7] Beaver. Dostopno na: <http://beaver.sourceforge.net/>
- [8] Bison. Dostopno na: <http://www.gnu.org/software/bison/>
- [9] Eli. Dostopno na: <http://eli-project.sourceforge.net/>
- [10] Sablecc. Dostopno na: <http://sablecc.org/>

Dodatek B

izvorna koda zgoščevanje redke tabele

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>

#include <fcntl.h>
#include <sys/stat.h>

//int width=15;
//int height=10;
//#define WW 15
//#define WH 10
#define DGOSTOTA 0.1
//#define WW width
//#define WH height

struct dyn_table{
    int w;
    int h;
    int *data;
};

struct RDS_data{
    struct dyn_table displace_rows;
    int *data;
    int *check_data;
    int data_len;
    int end_data;
    int method;
};

int rt_data(struct dyn_table *tbl,int y,int x){
    if(y>=tbl->h||y<0) abort();
    if(x>=tbl->w||x<0) abort();
    return tbl->data[(tbl->w*y)+x];
};

int set_data(struct dyn_table *tbl,int y,int x,int val){
    if(y>=tbl->h||y<0) abort();
    if(x>=tbl->w||x<0) abort();
    tbl->data[(tbl->w*y)+x]=val;
    return 0;
};

void free_dyn_table(struct dyn_table *tbl){
    if(tbl->data)
        free(tbl->data);
    tbl->data=NULL;
};
```

```

    tbl->w=tbl->h=0;
};

int alloc_dyn_table(struct dyn_table *tbl,int w,int h){
    free_dyn_table(tbl);
    tbl->data=malloc(sizeof(int)*w*h);
    memset(tbl->data,0,sizeof(int)*w*h);
    tbl->w=w;
    tbl->h=h;
};

int set_dyn_table(struct dyn_table *tbl){
    memset(tbl->data,0xfe,sizeof(int)*tbl->w*tbl->h);
};

void print_data_RDS(struct RDS_data *pck_data);

void alloc_data_RDS(struct RDS_data *pck,int size){
    if(!pck) return;
    int old_data_size=pck->data_len;
    if(pck->data){
        pck->data=realloc(pck->data,size);
        memset(((char *)pck->data)+old_data_size,0,size-old_data_size);
    }else{
        pck->data=malloc(size);
        memset(pck->data,0,size);
    };
    if(pck->check_data){
        pck->check_data=realloc(pck->check_data,size);
        memset(((char *)pck->check_data)+old_data_size,0xfe,size-old_data_size);
    }else{
        pck->check_data=malloc(size);
        memset(pck->check_data,0xfe,size);
    };
    pck->data_len=size;
};

void reset_data_RDS(struct RDS_data *pck_data,int row_count){
    if(pck_data->data){
        bzero(pck_data->data,pck_data->data_len);
    };
    if(pck_data->check_data){
        //bzero(pck_data->check_data,pck_data->data_len);
        memset(pck_data->check_data,0xfe,pck_data->data_len);
    };
    //bzero(pck_data->displace_rows,sizeof(pck_data->displace_rows));
    alloc_dyn_table(&pck_data->displace_rows,row_count,1);
    set_dyn_table(&pck_data->displace_rows);
    pck_data->end_data=0;
};

void free_data_RDS(struct RDS_data *pck_data){
    if(pck_data->data){
        free(pck_data->data);
        pck_data->data=NULL;
    };
    if(pck_data->check_data){
        free(pck_data->check_data);
        pck_data->check_data=NULL;
    };
    free_dyn_table(&pck_data->displace_rows);
};

struct RCS_data{
    int *values;
    int *column_index;
    int values_size;
    int number_of_values_in;
    int *row_ptr;
    int row_size;
    int number_of_rows;

    int method;
};

void reset_data_RCS(struct RCS_data *pck_data){
    if(pck_data->values){
        bzero(pck_data->values,pck_data->values_size);
    };
    if(pck_data->column_index){

```

```

    bzero(pck_data->column_index, pck_data->values_size);
};
if(pck_data->row_ptr){
    bzero(pck_data->row_ptr, pck_data->row_size);
};
pck_data->number_of_values_in=0;
pck_data->number_of_rows=0;
};

void alloc_data_RCS(struct RCS_data *pck, int nnz_len, int row_count){
    if(!pck) return;
    if(pck->values_size!=nnz_len){
        int old_data_size=pck->values_size;
        if(pck->values){
            pck->values=realloc(pck->values, nnz_len);
            memset(((char *)pck->values)+old_data_size, 0, nnz_len-old_data_size);
        }else{
            pck->values=malloc(nnz_len);
            memset(pck->values, 0, nnz_len);
        };
        if(pck->column_index){
            pck->column_index=realloc(pck->column_index, nnz_len);
            memset(((char *)pck->column_index)+old_data_size, 0, nnz_len-old_data_size);
        }else{
            pck->column_index=malloc(nnz_len);
            memset(pck->column_index, 0, nnz_len);
        };
        pck->values_size=nnz_len;
    };

    if(pck->row_size!=row_count){
        int old_data_size=pck->row_size;
        if(pck->row_ptr){
            pck->row_ptr=realloc(pck->row_ptr, row_count);
            memset(((char *)pck->row_ptr)+old_data_size, 0, row_count-old_data_size);
        }else{
            pck->row_ptr=malloc(row_count);
            memset(pck->row_ptr, 0, row_count);
        };
        pck->row_size=row_count;
    };
};

void free_RCS_DATA(struct RCS_data *pck_data){
    if(pck_data->values){
        free(pck_data->values);
        pck_data->values=NULL;
    };
    if(pck_data->column_index){
        free(pck_data->column_index);
        pck_data->column_index=NULL;
    };
    if(pck_data->row_ptr){
        free(pck_data->row_ptr);
        pck_data->row_ptr=NULL;
    };
};

struct LES_data{
    int *r_ptr, *c_ptr;
    int *dr_ptr, *dc_ptr;
    //need extra sigma
    int r_size, c_size;
    //int in_r, in_c;
    struct dyn_table new_table;
};

void reset_data_LES(struct LES_data *pck_data){
    if(pck_data->r_ptr){
        bzero(pck_data->r_ptr, pck_data->r_size);
    };
    if(pck_data->c_ptr){
        bzero(pck_data->c_ptr, pck_data->c_size);
    };
    if(pck_data->dr_ptr){
        bzero(pck_data->dr_ptr, pck_data->r_size);
    };
    if(pck_data->dc_ptr){
        bzero(pck_data->dc_ptr, pck_data->c_size);
    };
};

```

```

    };
};

void alloc_data_LES(struct LES__data *pck,int r_size,int c_size){
    int old_data_size;
    old_data_size=pck->r_size;
    if(old_data_size!=r_size){
        if(pck->r_ptr){
            pck->r_ptr=realloc(pck->r_ptr,r_size);
            memset(((char *)pck->r_ptr)+old_data_size,0,r_size-old_data_size);
        }else{
            pck->r_ptr=malloc(r_size);
            memset(pck->r_ptr,0,r_size);
        };

        if(pck->dr_ptr){
            pck->dr_ptr=realloc(pck->dr_ptr,r_size);
            memset(((char *)pck->dr_ptr)+old_data_size,0,r_size-old_data_size);
        }else{
            pck->dr_ptr=malloc(r_size);
            memset(pck->dr_ptr,0,r_size);
        };
        pck->r_size=r_size;
    }

    old_data_size=pck->c_size;
    if(old_data_size!=c_size){
        if(pck->c_ptr){
            pck->c_ptr=realloc(pck->c_ptr,c_size);
            memset(((char *)pck->c_ptr)+old_data_size,0,c_size-old_data_size);
        }else{
            pck->c_ptr=malloc(c_size);
            memset(pck->c_ptr,0,c_size);
        };

        if(pck->dc_ptr){
            pck->dc_ptr=realloc(pck->dc_ptr,c_size);
            memset(((char *)pck->dc_ptr)+old_data_size,0,c_size-old_data_size);
        }else{
            pck->dc_ptr=malloc(c_size);
            memset(pck->dc_ptr,0,c_size);
        };
        pck->c_size=c_size;
    };
};

struct GCS__data{
    struct dyn_table col_info;
    struct dyn_table row_info;
    struct dyn_table table;
};

void free_GCS(struct GCS__data *pck){
    free_dyn_table(&pck->col_info);
    free_dyn_table(&pck->row_info);
    free_dyn_table(&pck->table);
};

int get_remove_defaults(struct dyn_table * table,struct dyn_table * default_reductions){
    int k,i,j;
    struct dyn_table temp_table={0};
    int count=0;

    if(default_reductions->w==0||default_reductions->h==0)
        alloc_dyn_table(default_reductions,table->h,1);
    alloc_dyn_table(&temp_table,table->w,2);

    for(k=0;k<table->h;k++){
        int _cnt=0;

        for(i=0;i<table->w;i++){
            set_data(&temp_table,0,i,0);
            set_data(&temp_table,1,i,0);
        };

        for(i=0;i<table->w;i++){
            int cur_reduction=rt_data(table,k,i);
            if(cur_reduction!=0){
                int index=-1;

```

```

    for (j=0;j<_cnt;j++){
        int tmp_val=rt_data(&temp_table,0,j);
        if (tmp_val!=0){
            if (tmp_val==cur_reduction){
                index=j;
                set_data(&temp_table,1,j,rt_data(&temp_table,1,j)+1);
                break;
            }
        };
    };
    if (index<0){
        index=_cnt;
        set_data(&temp_table,0,j,cur_reduction);
        set_data(&temp_table,1,j,rt_data(&temp_table,1,j)+1);
        _cnt++;
    };
};

int idx_max=0;
int max_val=rt_data(&temp_table,1,0);

for (i=1;i<_cnt;i++){
    if (max_val<rt_data(&temp_table,1,i)){
        idx_max=i;
        max_val=rt_data(&temp_table,1,i);
    };
};
set_data(default_reductions,0,k,rt_data(&temp_table,0,idx_max));

//printf("vrstica k=%d max_val=%d val=%d\n",k,rt_data(&temp_table,1,idx_max),rt_data(&temp_table,0,idx_max));
};

//remove default reductions from data
for (k=0;k<table->h;k++){
    int def_reduction=rt_data(default_reductions,0,k);
    if (def_reduction)
        for (i=0;i<table->w;i++){
            if (rt_data(table,k,i)==def_reduction){
                //printf("k=%d i=%d def_reduction=%d val=%d\n",k,i,def_reduction,rt_data(table,k,i));
                set_data(table,k,i,0);
                count++;
            };
        };
};
free_dyn_table(&temp_table);
return count;
};

void print_table(struct dyn_table *table){
    int i,j;
    for (i=0;i<table->h;i++){
        printf("%d. ",i);
        for (j=0;j<table->w;j++){
            printf("%2d ",rt_data(table,i,j));
        }
        printf("\n");
    }
};

void load_data_from_file(char *f_name,struct dyn_table *table){
    char buffer[1024];
    unsigned int in_buff=0;
    int err;
    int fd;
    char *endstr;
    char *tmp_end_str;
    unsigned int lines_count=0;
    int tbl_w=0,tbl_h=0;
    fd=open(f_name,O_RDONLY);
    if (fd<0)
        return;
    do{
        err=read(fd,buffer+in_buff,1023-in_buff);
        if (err<0){
            printf("error reading\n");

```

```

        break;
    };
    in_buff+=err;
    buffer[in_buff]=0;
    if((endstr=strchr(buffer, '\n'))){
    {
        char *start_buff=buffer;
        unsigned int row_cnt=0;
        while(endstr>start_buff&&(( tmp_end_str=strchr(start_buff, ' ') )&&endstr>=
            tmp_end_str)|| ( ( tmp_end_str=strchr(start_buff, '\n') )&&endstr>=tmp_end_str))
        ){
            *tmp_end_str=0;
            if(lines_count==0){
                if(row_cnt==0){
                    tbl_h=atoi(start_buff);
                }
                else{
                    tbl_w=atoi(start_buff);
                    printf("Got table w=%d h=%d\n",tbl_w,tbl_h);
                    if(table->w||table->h)
                        abort();
                    alloc_dyn_table(table,tbl_w,tbl_h);
                }
            }
            else{
                int tmpval=atoi(start_buff);
                if(tmpval){
                    //printf("GOT tmpval=%d ss=%s ln=%d row=%d\n",tmpval,start_buff,lines_count,
                        row_cnt);
                    set_data(table,lines_count-1,row_cnt,tmpval);
                }
                //else
                //set_data(table,lines_count,row_cnt,0);
            }
            start_buff=tmp_end_str+1;
            row_cnt++;
        };
        lines_count++;
    }
    endstr++;
    in_buff-=endstr-buffer;
    memmove(buffer,endstr,in_buff);
    //printf("removing size =%d\n",endstr-buffer);
}
else{
    printf("err\n");
    break;
};
}while(in_buff);
close(fd);
};

void fill_table(struct dyn_table * table,int count){
    //int count;
    int i,j,c;
    //bzero(table,WM*WH*sizeof(int));
    if(count<=0)
        count=1;
    //printf("Dodal %d elementov\n",count);
#ifdef 0
    for(i=0;i<table->w;i++)
        for(j=0;j<table->h;j++)
            set_data(table,j,i,i);
    return;
#endif
#ifdef 0
    for(i=0;i<table->w;i++)
        for(j=0;j<table->h;j++)
            set_data(table,j,i,j);
    j=0;
    for(i=0;i<table->w;i++)
        set_data(table,j,i,rand()%100);
    return;
#endif
    for(c=0;c<count;c++){
        do{
            i=rand()%table->w;
            j=rand()%table->h;
        }while(rt_data(table,j,i)!=0);
        //table[j][i]=(rand()%99)+1;
    }
}

```

```

        set_data(table, j, i, (rand() % 99) + 1);
    };
};

void check_rows(struct dyn_table * table, struct dyn_table * rows_data) {[WH][4]
    int col, row;
    int start, end, count;
    for (row=0; row<table->h; row++){
        end=start=-1;
        count=0;
        for (col=0; col<table->w; col++){
            if (rt_data(table, row, col) != 0) {
                //printf("%d, %d\n", row, col);
                if (start < 0) {
                    start=col;
                    end=start+1;
                }
                else {
                    end=col+1;
                }
                count++;
            }
        };
        set_data(rows_data, 0, row, count);
        set_data(rows_data, 1, row, end-start);
        set_data(rows_data, 2, row, start);
        set_data(rows_data, 3, row, end);
        //rows_data[row][1]=end-start;
        //rows_data[row][2]=start;
        //rows_data[row][3]=end;
        //printf("row=%d start=%d end=%d count=%d\n", row, start, end, count);
    };
};

int sort_rows(struct dyn_table * rows_data, struct dyn_table * row_list, int metoda) {[WH][4] [WH]
    int count=0;
    int i, j;
    int pos;
    for (i=0; i<rows_data->w; i++){
        int rowcount=rt_data(rows_data, 0, i);
        //printf("row %d len=%d\n", i, rowcount);
        pos=count-1;
        if (rowcount>0) {
            while (pos>=0 && rt_data(rows_data, 0, rt_data(row_list, 0, pos)) < rowcount)
                pos--;
            if (!(metoda&2))
                while (pos>=0 && rowcount==rt_data(rows_data, 0, rt_data(row_list, 0, pos)) && rt_data(
                    rows_data, 1, rt_data(row_list, 0, pos)) < rt_data(rows_data, 1, i))
                    pos--;
            for (j=count-1; j>pos; j--){
                set_data(row_list, 0, j+1, rt_data(row_list, 0, j));
            };
            //row_list[pos+1]=i;
            set_data(row_list, 0, pos+1, i);
            count++;
        }
        else {
            set_data(row_list, 0, pos+1, i);
            count++;
        }
    };
    //for (i=0; i<count; i++)
    //    printf("%d count=%d tally=%d\n", row_list[i], rows_data[row_list[i]][0], rows_data[
        row_list[i]][1]);
    return rows_data->w;
    //return count;
};

int match_row(struct dyn_table * table, struct dyn_table * rows_data, struct dyn_table *
    row_list, int srow, int crow)
{
    if (rt_data(rows_data, 0, srow) > 0 && rt_data(rows_data, 0, srow) == rt_data(rows_data, 0, crow) &&
        rt_data(rows_data, 1, srow) == rt_data(rows_data, 1, crow)
        ) { //rt_data(rows_data, 2, srow) == rt_data(rows_data, 2, crow) && rt_data(rows_data, 3, srow)
            == rt_data(rows_data, 3, crow)
        }
    int c;
    for (c=rt_data(rows_data, 2, srow); c<rt_data(rows_data, 2, srow)+rt_data(rows_data, 1, srow);
        c++){

```

```

        if(rt_data(table,srow,c)!=rt_data(table,crow,c))
            return 0;
    };
    //for (c=rt_data(rows_data,2,srow);c<rt_data(rows_data,2,srow)+rt_data(rows_data,1,srow);c++)
    //    printf("c=%d v1=%d v2=%d\n",c,rt_data(table,srow,c),rt_data(table,crow,c));
    //printf("found match c=%d r=%d\n",crow,srow);
    return 1;
};
return 0;
};

int pack_row_RDS(struct dyn_table * table,struct dyn_table * rows_data,struct RDS_data *
pck_data,int row_idx){
    int displace_idx=rt_data(rows_data,2,row_idx);
    int i;
    int place_at=0;
    int ok;

    if(rt_data(rows_data,1,row_idx)<=0) return -999;
    for(;;place_at++){
        int k;
        //printf("place at=%d idx=%d\n",place_at,row_idx);
        ok=1;
        if((place_at+rt_data(rows_data,1,row_idx)+1)*sizeof(int)>=pck_data->data_len){
            int data_len=pck_data->data_len*2;
            if(data_len<=(rt_data(rows_data,1,row_idx)*sizeof(int)))
                data_len+=(rt_data(rows_data,1,row_idx)*sizeof(int)*sizeof(int))*2;
            alloc_data_RDS(pck_data,data_len);
        };

        for(k=0;k<rt_data(rows_data,1,row_idx)&&ok;k++){
            #if 0
                if(pck_data->data[place_at+k]!=0){
                    //printf("place_at=%d k=%d data=%d\n",place_at,k,pck_data->data[place_at+k]);
                    ok=0;
                    break;
                };
            #else
                if(rt_data(table,row_idx,rt_data(rows_data,2,row_idx)+k)&&pck_data->data[place_at+k]
                    !=0){
                    //printf("place_at=%d k=%d data=%d\n",place_at,k,pck_data->data[place_at+k]);
                    ok=0;
                    break;
                };
            #endif
        };
        if(pck_data->method&1)
            for(k=0;k<table->h&&ok;k++){//lines added
                if(rt_data(&pck_data->displace_rows,0,k)==0xfefefefe)//not defined yet
                    continue;
                //printf("k=%d displace_rows,0,k)=%d place_at-(rows_data,2,row_idx)=%d\n",k,
                    rt_data(&pck_data->displace_rows,0,k),place_at-rt_data(rows_data,2,row_idx));
                if(rt_data(&pck_data->displace_rows,0,k)==place_at-rt_data(rows_data,2,row_idx)){
                    //printf("SAME INDEX k=%d %d\n",rt_data(&pck_data->displace_rows,0,k),place_at-
                        rt_data(rows_data,2,row_idx));
                    //printf("Same index \n");
                    ok=0;
                    break;
                };
            };
        if(ok){
            //printf("found place at %d tally=%d\n",place_at,rows_data[row_idx][1]);
            for(k=0;k<rt_data(rows_data,1,row_idx);k++){
                if(rt_data(table,row_idx,rt_data(rows_data,2,row_idx)+k)!=0){
                    pck_data->data[place_at+k]=rt_data(table,row_idx,rt_data(rows_data,2,row_idx)+k);
                    //printf("writing data idx=%d d=%d\n",place_at+k,pck_data->data[place_at+k]);
                    //pck_data->check_data[place_at+k]=rows_data[row_idx][2]+k;
                    if(pck_data->method&1)
                        pck_data->check_data[place_at+k]=rt_data(rows_data,2,row_idx)+k;
                    else
                        pck_data->check_data[place_at+k]=row_idx;
                };
            };
            if(place_at+rt_data(rows_data,1,row_idx)>pck_data->end_data)
                pck_data->end_data=place_at+rt_data(rows_data,1,row_idx);
        }
        return place_at-rt_data(rows_data,2,row_idx);
    };
}

```

```

    };
};

void add_tables_RDS(struct dyn_table * table, struct dyn_table * rows_data, struct dyn_table
    * row_list, int row_count, struct RDS__data * pck_data){
    int cnt;

    for (cnt=0; cnt<row_count; cnt++){
        int idx, j, found=0;
        int found_idx;
        int table_index=rt_data(row_list, 0, cnt);

        if (pck_data->method&1){
            for (j=0; j<cnt; j++){
                int tmpval=rt_data(row_list, 0, j);
                //printf("table_index=%d row_list=%d\n", table_index, tmpval);
                if (table_index!=tmpval&&match_row(table, rows_data, row_list, table_index, tmpval)){
                    //printf("FOund=%d %d\n", tmpval, table_index);
                    found++;
                    found_idx=rt_data(&pck_data->displace_rows, 0, tmpval);
                    break;
                };
                //if (table_index!=row_list[j]){//&&match_row(table, rows_data, row_list, table_index,
                //    row_list[j])
                //    found++;
            };
        };

        if (rt_data(rows_data, 0, table_index)>0){
            //printf("adding row %d\n", table_index);
            if (found){
                //printf("!!! vrstica se ponavlja idx=%d cnt=%d\n", table_index, cnt);
                found=found_idx;
                //abort();
            }
            else{
                found=pack_row_RDS(table, rows_data, pck_data, table_index);
            };
            set_data(&pck_data->displace_rows, 0, table_index, found);
            //printf("idx=%d found=%d\n", table_index, found);
        }
        else{
            //empty row
            //set_data(&pck_data->displace_rows, 0, table_index, 100000);
            set_data(&pck_data->displace_rows, 0, table_index, 0xfefefefe);
        };
    };
};

int check_table(struct dyn_table * table /*[WH][WW]*/, struct RDS__data * pck_data){
    int i, j;
    for (i=0; i<table->h; i++){
        for (j=0; j<table->w; j++){
            int val=0;
            int idx;
            int chk_dat;
            idx=rt_data(&pck_data->displace_rows, 0, i)+j;
            if (idx>=0&&idx<pck_data->end_data){
                //check_data
                chk_dat=pck_data->check_data[idx];
                if (pck_data->method&1){
                    if (chk_dat==j)//j
                        val=pck_data->data[idx];
                }
                else{
                    if (chk_dat==i)//j
                        val=pck_data->data[idx];
                };
            };

            if (val!=rt_data(table, i, j)){
                printf("False data val=%d tab=%d chk_dat=%d i=%d, j=%d idx=%d\n", val, rt_data(table,
                    i, j), chk_dat, i, j, idx);
                return -1;
            };
        };
    };
    return 0;
};

```

```

void print_data_RDS(struct RDS__data *pck_data){
    int i;
    int tmp;

    printf("DISPLACE INDEX\n");
    for(i=0;i<pck_data->displace_rows.w;i++){
        tmp=rt_data(&pck_data->displace_rows,0,i);
        if(tmp<0xfffff)
            printf("NA ");
        else
            printf("%2d ",rt_data(&pck_data->displace_rows,0,i));
    };
    printf("\n");

    printf("DATA TABLE\n");
    for(i=0;i<pck_data->end_data;i++){
        printf("%2d ",pck_data->data[i]);
    };
    printf("\n");
    printf("CHECK TABLE\n");
    for(i=0;i<pck_data->end_data;i++){
        if(pck_data->check_data[i]<0)
            printf("NA ");
        else
            printf("%2d ",pck_data->check_data[i]);
    };
    printf("\n");
};

void add_tables_RCS(struct dyn_table * table,struct RCS__data *pck_data){
    int i,j;
    pck_data->row_ptr[0]=0;
    pck_data->number_of_rows=1;
    for(j=0;j<table->h;j++){
        for(i=0;i<table->w;i++){
            int tmp_val=rt_data(table,j,i);
            if(tmp_val){
                if(pck_data->values_size<pck_data->number_of_values_in*sizeof(int)+sizeof(int)){
                    //printf("allocing new values\n");
                    alloc_data_RCS(pck_data,pck_data->values_size*2,pck_data->row_size);
                };
                *(pck_data->values+pck_data->number_of_values_in)=tmp_val;
                pck_data->column_index[pck_data->number_of_values_in]=i;
                //printf("Adding value %d i=%d, j=%d number_of_values_in=%d\n",tmp_val,i,j,pck_data->number_of_values_in);
                pck_data->number_of_values_in++;
            };
        };
        //printf("number_of_rows=%d in_values=%d val=%d\n",pck_data->number_of_rows,pck_data->number_of_values_in,pck_data->row_ptr[1]);
        *(pck_data->row_ptr+pck_data->number_of_rows)=pck_data->number_of_values_in;
        pck_data->number_of_rows++;
    };
};

int return_RCS_val(struct RCS__data *pck_data,int x,int y){
    int i;
    if(y>pck_data->number_of_rows){
        printf("err req\n");
        return 0;
    };
    for(i=pck_data->row_ptr[y];i<pck_data->row_ptr[y+1];i++){
        //printf("i=%d\n",i);
        //printf("pck_data->column_index[i]=%d\n",pck_data->column_index[i]);
        if(pck_data->column_index[i]==x)
            return pck_data->values[i];
    };
    return 0;
};

void test_RCS_data(struct dyn_table * table,struct RCS__data *pck_data){
    int i,j;
    for(i=0;i<table->h;i++){
        for(j=0;j<table->w;j++){
            int tmp_val;
            tmp_val=return_RCS_val(pck_data,j,i);
            if(tmp_val!=rt_data(table,i,j)){
                printf("result false tmp_val=%d data=%d i=%d,j=%d\n",tmp_val,rt_data(table,i,j),i,j);
                abort();
            };
        };
    };
};

```

```

    };
};

int LES_scan_line(struct dyn_table * table,int line,int *r_value,struct dyn_table *
    removed_column){
    unsigned int i,parsed_line=0;
    int value=0;
    for(i=0;i<table->w;i++){
        if(rt_data(removed_column,0,i))
            continue;
        parsed_line++;
        if(rt_data(table,line,i)){
            if(value==0){
                value=rt_data(table,line,i);
            }
            else{
                if(value!=rt_data(table,line,i))
                    return 0;
            }
        }
    };
};
if(parsed_line<=1)
    return 0;
if(value==0)
    return -1;
*r_value=value;
return 1;
};

int LES_scan_col(struct dyn_table * table,int column,int *r_value,struct dyn_table *
    removed_rows){
    unsigned int i,parsed_col=0;
    int value=0;
    for(i=0;i<table->h;i++){
        if(rt_data(removed_rows,0,i))
            continue;
        parsed_col++;
        if(rt_data(table,i,column)){
            if(value==0){
                value=rt_data(table,i,column);
            }
            else{
                if(value!=rt_data(table,i,column))
                    return 0;
            }
        }
    };
};
if(parsed_col<=1)
    return 0;
if(value==0)
    return -1;
*r_value=value;
return 1;
};

void add_tables_LES(struct dyn_table * table,struct LES_data *pck_data){
    struct dyn_table row_info={0};
    struct dyn_table col_info={0};

    int col_added,row_added;
    unsigned int i,j;
    int value,ret;
    unsigned int scan=0;

    alloc_dyn_table(&row_info,table->h,1);
    alloc_dyn_table(&col_info,table->w,1);

    do{
        row_added=col_added=0;
        scan++;
        for(i=0;i<table->h;i++){
            if(rt_data(&row_info,0,i))//already did that row
                continue;
            ret=LES_scan_line(table,i,&value,&col_info);
            //printf("line i=%d ret=%d value=%d\n",i,ret,value);
            if(ret==-1){
                //empty line
                //printf("empty line\n");
                set_data(&row_info,0,i,1);//line removed
            }
        }
    }while(scan<table->h);
}

```

```

        pck_data->dr_ptr[i]=scan;
        pck_data->r_ptr[i]=0;
        row_added++;
        break;
    }
    else
    if (ret){
        set_data(&row_info,0,i,1); //line removed
        pck_data->dr_ptr[i]=scan;
        pck_data->r_ptr[i]=value;
        row_added++;
        break;
    }
};

scan++;
for (i=0;i<table->w;i++){
    if (rt_data(&col_info,0,i)) //already did that row
        continue;
    ret=LES_scan_col(table,i,&value,&row_info);
    //printf("colu i=%d ret=%d value=%d\n",i,ret,value);
    if (ret==-1){
        //empty line
        //printf("empty column\n");
        set_data(&col_info,0,i,1); //column removed
        pck_data->dc_ptr[i]=scan;
        pck_data->c_ptr[i]=0;
        col_added++;
        break;
    }
    else
    if (ret){
        set_data(&col_info,0,i,1); //column removed
        pck_data->dc_ptr[i]=scan;
        pck_data->c_ptr[i]=value;
        col_added++;
        break;
    }
};

//printf("row_added=%d col_added=%d scan=%d\n",row_added,col_added,scan);

}while(row_added || col_added);
scan++;
int new_w=0,new_h=0;
for (i=0;i<table->h;i++){ //count height for new table
    if (!rt_data(&row_info,0,i)){
        //printf("new scan row=%d new_h=%d\n",i,new_h);
        pck_data->dr_ptr[i]=scan;
        pck_data->r_ptr[i]=new_h;
        new_h++;
    }
};
for (i=0;i<table->w;i++){ //count height for new table
    if (!rt_data(&col_info,0,i)){
        //printf("new scan col=%d new_w=%d\n",i,new_w);
        pck_data->dc_ptr[i]=scan;
        pck_data->c_ptr[i]=new_w;
        new_w++;
    }
};
//printf("new table is %d,%d\n",new_w,new_h);
if (new_w&&new_h){
    alloc_dyn_table(&pck_data->new_table,new_h,new_w);

    new_h=0;
    for (i=0;i<table->w;i++){
        if (!rt_data(&col_info,0,i)){
            new_w=0;
            for (j=0;j<table->h;j++){
                if (!rt_data(&row_info,0,j)){
                    int val=rt_data(table,j,i);
                    //printf("new_w=%d,new_h=%d val=%d\n",new_w,new_h,val);
                    set_data(&pck_data->new_table,new_h,new_w,val);
                    new_w++;
                }
            }
            new_h++;
        }
    }
};
};
};

```

```

    free_dyn_table(&row_info);
    free_dyn_table(&col_info);
};

void test_LES_data(struct dyn_table * table, struct LES_data *pck_data){
    unsigned int i, j;
#ifdef 0
    //print_table(table);
    for(i=0; i<table->h; i++){
        printf("i=%d r=%d dr=%d\n", i, pck_data->r_ptr[i], pck_data->dr_ptr[i]);
    };
    for(i=0; i<table->w; i++){
        printf("i=%d c=%d dc=%d\n", i, pck_data->c_ptr[i], pck_data->dc_ptr[i]);
    };
    print_table(&pck_data->new_table);
#endif
    for(i=0; i<table->h; i++){
        for(j=0; j<table->w; j++){
            int val=rt_data(table, i, j);
            if(val){ //as sigmap
                //printf("pck_data->dr_ptr[i]=%d pck_data->dc_ptr[j]=%d i=%d, j=%d\n", pck_data->
                    dr_ptr[i], pck_data->dc_ptr[j], i, j);
                if(pck_data->dr_ptr[i]<pck_data->dc_ptr[j]){
                    if(val!=pck_data->r_ptr[i]){
                        printf("here its value =%d val=%d\n", pck_data->r_ptr[i], val);
                        abort();
                    };
                }
                else if(pck_data->dr_ptr[i]>pck_data->dc_ptr[j]){
                    if(pck_data->c_ptr[j]!=val){
                        printf("here its value2 =%d val=%d\n", pck_data->c_ptr[j], val);
                        abort();
                    };
                }
                else{
                    int vall=rt_data(&pck_data->new_table, pck_data->c_ptr[j], pck_data->r_ptr[i]);
                    if(val!=vall){
                        printf("table value pck_data->r_ptr[i]=%d c_ptr[j]=%d val=%d vall=%d\n",
                            pck_data->r_ptr[i], pck_data->c_ptr[j], val, vall);
                        abort();
                    };
                };
            };
        };
    };
};

struct adj{
    int max_size;
    int cur_elem;
    unsigned int *ptr;
    int color;
    unsigned int collision_count;
};

int line_compare(struct dyn_table * table, int l1, int l2){
    unsigned int i;
    if(l1==l2)
        abort();
    for(i=0; i<table->w; i++){
        int val=rt_data(table, l1, i);
        int vall=rt_data(table, l2, i);
        if(val&&vall&&val!=vall)
            return 0;
    };
    return 1;
};

int row_compare(struct dyn_table * table, int l1, int l2){
    unsigned int i;
    if(l1==l2)
        abort();
    for(i=0; i<table->h; i++){
        int val=rt_data(table, i, l1);
        int vall=rt_data(table, i, l2);
        if(val&&vall&&val!=vall)
            return 0;
    };
    return 1;
};

```

```

void check_adj_data_size(struct adj *adj, int new_size){
    if(new_size==0)
        new_size=1024;
    if(new_size>adj->max_size){
        if(adj->max_size){
            adj->ptr=malloc(new_size);
            memset(adj->ptr,0,new_size);
        }
        else{
            adj->ptr=realloc(adj->ptr,new_size);
            memset(((char *)adj->ptr)+adj->max_size,0,new_size-adj->max_size);
        }
        adj->max_size=new_size;
    };
};

void free_row_data(struct adj *data, int size){
    unsigned int i;
    for(i=0;i<size;i++){
        if(data[i].ptr)
            free(data[i].ptr);
    };
    free(data);
};

void qsort_adj( struct adj *graph, struct dyn_table * table, int l, int r )
{
    int i,j,pivot;
    i = l;
    j = r;
    pivot = graph[rt_data(table,0,(l+r)/2)].collision_count;
    do {
        while( graph[rt_data(table,0,i)].collision_count > pivot ) i++;
        while( graph[rt_data(table,0,j)].collision_count < pivot ) j--;

        if( i <= j ) {
            //struct adj help;
            int help_idx;
            //help = graph+i;
            //memcpy(&help, graph+i, sizeof(struct adj));
            //memmove(graph+i, graph+j, sizeof(struct adj));
            //memcpy(graph+j, &help, sizeof(struct adj));
            //graph[i] = graph[j];
            //graph[j] = help;
            help_idx=rt_data(table,0,i);
            set_data(table,0,i,rt_data(table,0,j));
            set_data(table,0,j,help_idx);
            i++;
            j--;
        };
    } while( i <= j );

    if( l < j ) qsort_adj( graph, table, l, j );
    if( i < r ) qsort_adj( graph, table, i, r );
}

struct adj *get_row_data(struct dyn_table * table, struct dyn_table *sorted_index){
    unsigned int i,j;
    struct adj *data;

    data=malloc(sizeof(struct adj)*table->h);
    if(!data)
        abort();
    memset(data,0,sizeof(struct adj)*table->h);
    alloc_dyn_table(sorted_index, table->h,1);
    for(i=0;i<table->h;i++){
        check_adj_data_size(data+i, sizeof(int)*table->h);
        data[i].color=-1;
        set_data(sorted_index,0,i,i);
    };

    for(i=0;i<table->h;i++){
        for(j=0;j<i;j++){//table->h
            //if(i==j)
            //continue;
            if(!line.compare(table,i,j)){
                //printf("i=%d adding j=%d\n",i,j);
                if(!data[i].max_size||data[i].max_size<data[i].cur_elem+sizeof(int)){

```

```

        abort();
        //check_adj_data_size(data+i, data[i].max_size*2);
    };
    data[i].ptr[data[i].cur_elem]=j;
    data[i].cur_elem++;
    data[i].collision_count++;

    data[j].ptr[data[j].cur_elem]=i;
    data[j].cur_elem++;
    data[j].collision_count++;
};
};
};

qsort_adj(data, sorted_index, 0, table->h-1);
/*for(i=0; i<table->h; i++){
    int idx=rt_data(sorted_index, 0, i);
    printf("data i=%d xa=%d idx=%d\n", idx, data[idx].collision_count, i);
};*/
return data;
};

unsigned int color_rows(struct adj *data, struct dyn_table *sorted_index, unsigned int size)
{
    unsigned int max_color=0;
    unsigned int i, j;
    for(i=0; i<size; i++){
        int idx=rt_data(sorted_index, 0, i);
        int collision;
        unsigned int cur_color;
        //printf("!!! data[idx].cur_elem=%d\n", data[idx].cur_elem);
        for(cur_color=0; cur_color<=max_color; cur_color++){
            collision=0;
            for(j=0; j<data[idx].cur_elem; j++){
                int new_idx=data[idx].ptr[j];
                //printf("!!! j=%d idx=%d\n", j, new_idx);
                if(data[new_idx].color==cur_color){
                    //printf("collision inc color and color it idx=%d with=%d cur_color=%d\n", idx,
                        new_idx, cur_color);
                    collision++;
                    break;
                }
                else{
                };
            };
        };
        if(!collision){
            //printf("we got color=%d for idx=%d\n", max_color, idx);
            data[idx].color=cur_color;
            break;
        };
    };
    if(collision){
        max_color++;
        //printf("collision coloring it idx=%d max_color=%d\n", idx, max_color);
        data[idx].color=max_color;
    };
};
return max_color;
};

void fill_temp_table(struct dyn_table * table, struct dyn_table * new_table, struct adj *
    data, struct GCS_data *pck_data){
    int i;
    alloc_dyn_table(&pck_data->row_info, table->h, 1);
    for(i=0; i<table->h; i++){
        int new_idx=data[i].color;
        if(new_idx>=0){
            int j;
            set_data(&pck_data->row_info, 0, i, new_idx);
            for(j=0; j<table->w; j++){
                int val=rt_data(table, i, j);
                if(val)
                    set_data(new_table, new_idx, j, val);
            };
        }
        else{
            set_data(&pck_data->row_info, 0, i, -1);
        };
    };
};
};

```

```

};

struct adj *get_col_data(struct dyn_table * table, struct dyn_table *sorted_index){
    unsigned int i, j;
    struct adj *data;

    data=malloc(sizeof(struct adj)*table->w);
    if (!data)
        abort();
    memset(data, 0, sizeof(struct adj)*table->w);
    alloc_dyn_table(sorted_index, table->w, 1);
    for (i=0; i<table->w; i++){
        check_adj_data_size(data+i, sizeof(int)*table->w);
        data[i].color=-1;
        set_data(sorted_index, 0, i, i);
    };

    for (i=0; i<table->w; i++){
        for (j=0; j<i; j++){//table->h
            //if (i==j)
            //continue;
            if (!row_compare(table, i, j)){
                //printf("i=%d adding j=%d\n", i, j);
                if (!data[i].max_size || data[i].max_size<data[i].cur_elem+sizeof(int)){
                    abort();
                    //check_adj_data_size(data+i, data[i].max_size*2);
                };
                data[i].ptr[data[i].cur_elem]=j;
                data[i].cur_elem++;
                data[i].collision_count++;

                data[j].ptr[data[j].cur_elem]=i;
                data[j].cur_elem++;
                data[j].collision_count++;
            };
        };
    };

    qsort_adj(data, sorted_index, 0, table->w-1);
    /*for (i=0; i<table->h; i++){
        int idx=rt_data(sorted_index, 0, i);
        printf("data i=%d xa=%d idx=%d\n", idx, data[idx].collision_count, i);
    };*/
    return data;
};

void fill_final_table(struct dyn_table * table, struct adj *data, struct GCS_data *pck_data)
{
    int i;
    alloc_dyn_table(&pck_data->col_info, table->w, 1);
    for (i=0; i<table->w; i++){
        int new_idx=data[i].color;
        if (new_idx>=0){
            int j;
            set_data(&pck_data->col_info, 0, i, new_idx);
            for (j=0; j<table->h; j++){
                int val=rt_data(table, j, i);
                if (val)
                    set_data(&pck_data->table, j, new_idx, val);
            };
        }
        else{
            set_data(&pck_data->row_info, 0, i, -1);
        };
    };
};

void add_tables_GCS(struct dyn_table * table, struct GCS_data *pck_data){
    struct dyn_table col_info={0};
    unsigned int i, j;
    struct adj *adj_info;
    struct dyn_table temp_table={0};
    unsigned temp_table_len;

    //alloc_dyn_table(&row_info, table->h, 1);
    adj_info=get_row_data(table, &col_info);
    temp_table_len=color_rows(adj_info, &col_info, table->h);
    alloc_dyn_table(&temp_table, table->w, temp_table_len+1);

```

```

    fill_temp_table(table,&temp_table,adj_info,pck_data);
    free_dyn_table(&col_info);
    free_row_data(adj_info,table->h);

#ifdef 0
    printf("after row coloring\n");
    print_table(&temp_table);
#endif

    adj_info=get_col_data(&temp_table,&col_info);
    temp_table.len=color_rows(adj_info,&col_info,temp_table.w);
    alloc_dyn_table(&pck_data->table,temp_table.len+1,temp_table.h);
    fill_final_table(&temp_table,adj_info,pck_data);

#ifdef 0
    printf("after column coloring\n");
    print_table(&pck_data->table);
#endif

    free_dyn_table(&col_info);
    free_row_data(adj_info,temp_table.w);
    free_dyn_table(&temp_table);

};

void test_GCS_data(struct dyn_table * table,struct GCS_data *pck_data){
    unsigned int i,j;
    for(i=0;i<table->h;i++){
        for(j=0;j<table->w;j++){
            int val=rt_data(table,i,j);
            if(val){//as sigmap
                int r_idx,c_idx;
                int vall;
                r_idx=rt_data(&pck_data->row_info,0,i);
                c_idx=rt_data(&pck_data->col_info,0,j);
                vall=rt_data(&pck_data->table,r_idx,c_idx);
                if(vall!=val){
                    printf("r_idx=%d c_idx=%d val=%d vall=%d\n",r_idx,c_idx,val,vall);
                    abort();
                }
            }
        }
    }
};

int main(){
    //int table [WH] [WW];
    //int table_data [WH] [4];
    //int row_list [WH];
    int width=15;
    int height=10;
    struct dyn_table table={0};
    struct dyn_table table_data={0};
    struct dyn_table row_list={0};

    int row_count;
    int rep;
    int element_count;

    srand(time(0));

#ifdef REPORT
#define WANTRDS
#define WANTRDS1
//#define WANTGCS
//#define WANTLES
//#define WANTRCS
    int fd_report;

    fd_report=open("report.txt",O_CREAT|O_WRONLY|O_TRUNC,S_IRWXU);

    //for (width=50; width<=500; width+=50)
    //for (height=100; height<=500; height+=50)
    width=100;
    height=50;
#ifdef WANTRDS
#ifdef WANTRDS1
    //width=500;
    //height=50;

```

```

        width=500;
        height=50;
#ifdef WANTS
        struct RDS__data packed={0};
        packed.method=1;
        alloc_dyn_table(&table_data, table.h,4);
        alloc_data_RDS(&packed,1024);
        reset_data_RDS(&packed, table.h);
#ifdef WANT_RDS1
        int fd1, fd2, fd3;
        char writestr[256];
        // alloc_dyn_table(&row_list, table.h,1);
        fd1=open("report1.txt", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
        fd2=open("report2.txt", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
        fd3=open("report3.txt", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
#endif
#endif
#ifdef WANT_GCS
        struct GCS__data packed_GCS={0};
#endif
#ifdef WANT_LES
        struct LES__data packed_LES={0};
        alloc_data_LES(&packed_LES, height*sizeof(int), width*sizeof(int));
#endif
#ifdef WANT_RCS
        struct RCS__data packed_RCS={0};
        alloc_data_RCS(&packed_RCS, 1024, (height+1)*sizeof(int));
#endif
// alloc_data_RDS(&packed, 1024);
#ifdef WANT_RDS1
        element_count=2500;
#else
        for(element_count=10; element_count <=1000; element_count+=10)
#endif
        {
            int repeat;
            int result=0;
            int min_val;
            int max_val;
            // element_count=(int)(DGOSTOTA*(float)(WW*WH));
            // printf("Element count = %d\n", element_count);

            //memset(packed.displace_rows, 0, sizeof(packed.displace_rows));

            //for(rep=0; rep < 1000; rep++){
#ifdef WANT_RDS1
            for(repeat=0; repeat < 10; repeat++)
            {
                int tmp_result;
                alloc_dyn_table(&table, width, height);
                alloc_dyn_table(&table_data, height, 4);
                alloc_dyn_table(&row_list, height, 1);
                fill_table(&table, element_count);
#ifdef WANT_RDS
                reset_data_RDS(&packed, height);
                check_rows(&table, &table_data);
                row_count=sort_rows(&table_data, &row_list, packed.method);
                // if (height==500)
#ifdef WANT_RDS1
                {
                    int cnt, cnt1;
                    int dada;
                    int data=0;

                    do{
                        cnt1=0;
                        for(cnt=0; cnt<table.h; cnt++){
                            int tmp;
                            int table_index=rt_data(&row_list, 0, cnt);
                            tmp=rt_data(&table_data, 0, table_index);
                            if(tmp>data)
                                cnt1+=tmp;
                        }
                        sprintf(writestr, "%d %d\n", data+1, cnt1); // ((double)element_count (double)
                        write(fd1, writestr, strlen(writestr));
                        sprintf(writestr, "{%d,%f}", data, ((double)cnt1)/((double)element_count); //
                        write(fd3, writestr, strlen(writestr));
                        sprintf(writestr, "%d,%f}", data, ((double)element_count)/((double)(data+1));
                        write(fd2, writestr, strlen(writestr));

```

```

        data++;
    }while(cnt1>0);
};
#endif
add_tables_RDS(&table,&table_data,&row_list,row_count,&packed);
if(check_table(&table,&packed)){
    //check failed
    printf("BUG CHECK FAILED\n");
    //print_table(&table);
    //print_data_RDS(&packed);
    //break;
    abort();
}
tmp_result=packed.end_data;//-element_count
#endif
#ifdef WANT_GCS
add_tables_GCS(&table,&packed_GCS);
test_GCS_data(&table,&packed_GCS);

tmp_result=packed_GCS.table.w*packed_GCS.table.h;//-element_count

free_GCS(&packed_GCS);
#endif
#ifdef WANT_LES
reset_data_LES(&packed_LES);
add_tables_LES(&table,&packed_LES);
test_LES_data(&table,&packed_LES);
tmp_result=packed_LES.new_table.w*packed_LES.new_table.h;
#endif
#ifdef WANT_RCS
reset_data_RCS(&packed_RCS);
add_tables_RCS(&table,&packed_RCS);
test_RCS_data(&table,&packed_RCS);
//printf("reduced size data len=%d row-size=%d / total len=%d\n",packed_RCS.
    number_of_values_in,packed_RCS.number_of_rows,table.w*table.h);
tmp_result=packed_RCS.number_of_values_in*2+packed_RCS.number_of_rows;
#endif
result+=tmp_result;
if(repeat==0){
    min_val=max_val=tmp_result;
}
else{
    if(tmp_result<min_val)
        min_val=tmp_result;
    if(tmp_result>max_val)
        max_val=tmp_result;
};
//printf("packed.end_data=%d\n",packed.end_data);
};
printf("avg=%f min_val=%d maxval=%d element_count=%d\n",(float)result/(float)repeat,
    min_val,max_val,element_count);
//};
//sprintf(writestr,"%d,%d,%f",width,height,(double)result/5.0);
//sprintf(writestr,"%d,%d,%d",width,height,result);
//sprintf(writestr,"%d,%d",width,result);
sprintf(writestr,"%d %f %d %d\n",element_count,(float)result/(float)repeat,min_val,
    max_val);
write(fd_report,writestr,strlen(writestr));
free_dyn_table(&table);
free_dyn_table(&table_data);
free_dyn_table(&row_list);

};
#else
//element_count=(int)(DGOSTOTA*(float)(width*WH));
#define WANT_RDS
//define WANT_RDS1
#define WANT_LES
#define WANT_GCS
#define WANT_RCS

width=5;
height=5;
element_count=5;
printf("Element count = %d\n",element_count);

//memset(packed.displace_rows,0,sizeof(packed.displace_rows));

//for(rep=0;rep<1000;rep++){

```

```

#if 1
//load_data_from_file("table.t1",&table);
load_data_from_file("pascal.t1",&table);
printf("table.w=%d table.h=%d total_size=%d\n",table.w,table.h,table.w*table.h);
#endif
if 0
struct dyn_table def_val={0};
printf("Removed values=%d\n",get_remove_defaults(&table,&def_val));
//print_table(&table);
#endif
#else
alloc_dyn_table(&table,width,height);
fill_table(&table,element_count);
#endif
#ifdef WANT_RDS
struct RDS__data packed={0};
packed.method=0;
alloc_dyn_table(&table_data,table.h,4);
alloc_dyn_table(&row_list,table.h,1);
alloc_data_RDS(&packed,1024);
reset_data_RDS(&packed,table.h);

check_rows(&table,&table_data);
row_count=sort_rows(&table_data,&row_list,packed.method);

add_tables_RDS(&table,&table_data,&row_list,row_count,&packed);
if(check_table(&table,&packed)){
//check failed
printf("BUG CHECK FAILED\n");
print_table(&table);
print_data_RDS(&packed);
//break;
abort();
}
else{
#ifdef REPORT
sprintf(writestr,"%d",packed.end_data);
write(fd_report,writestr,strlen(writestr));
#endif
};
//print_data_RDS(&packed);
printf("RDS1 reduced size data len=%d total len=%d\n",packed.end_data*2+packed.
displace_rows.w,table.w*table.h);
printf("RDS1 total data_l=%d check_l=%d displace=%d\n",packed.end_data,packed.end_data,
packed.displace_rows.w);
packed.method=1;
reset_data_RDS(&packed,table.h);
check_rows(&table,&table_data);
row_count=sort_rows(&table_data,&row_list,packed.method);

add_tables_RDS(&table,&table_data,&row_list,row_count,&packed);
if(check_table(&table,&packed)){
//check failed
printf("BUG CHECK FAILED\n");
print_table(&table);
print_data_RDS(&packed);
//break;
abort();
}
printf("RDS2 reduced size data len=%d total len=%d\n",packed.end_data*2+packed.
displace_rows.w,table.w*table.h);
printf("RDS2 total data_l=%d check_l=%d displace=%d\n",packed.end_data,packed.end_data,
packed.displace_rows.w);
#endif
endif
if 0
print_table(&table);
endif
//};

#ifdef WANT_RCS
struct RCS__data packed_RCS={0};
alloc_data_RCS(&packed_RCS,1024,(table.h+1)*sizeof(int));
if 0
struct dyn_table def_val={0};
printf("RCS Removed values=%d\n",get_remove_defaults(&table,&def_val));
//print_table(&table);
endif
add_tables_RCS(&table,&packed_RCS);
test_RCS_data(&table,&packed_RCS);

```

```

printf("RCS reduced size data len=%d / total len=%d \n",packed_RCS.number_of_values_in
*2+packed_RCS.number_of_rows,table.w*table.h);
printf("RCS values_l=%d check_l=%d rows_l=%d\n",packed_RCS.number_of_values_in ,
packed_RCS.number_of_values_in ,packed_RCS.number_of_rows);
#endif

//printf("full matrix len %d,%d=%d data_size=%d\n",WW,WH,WH*WW,WH*WW*sizeof(int));

#ifdef WANT_LES
struct LES_data packed_LES={0};
alloc_data_LES(&packed_LES,table.h*sizeof(int),table.w*sizeof(int));
add_tables_LES(&table,&packed_LES);
test_LES_data(&table,&packed_LES);
int les_row_size=(packed_LES.r_size/sizeof(int));
int les_col_size=(packed_LES.c_size/sizeof(int));
printf("LES reduced size data len=%d\n",les_row_size*2+les_col_size*2+packed_LES.
new_table.w*packed_LES.new_table.h);
printf("LES row_data=%d col_size=%d newtable_w=%d newtable_h=%d\n",les_row_size ,
les_col_size ,packed_LES.new_table.w,packed_LES.new_table.h);
#endif

#ifdef WANT_GCS
struct GCS_data packed_GCS={0};
//alloc_data_GCS(&packed_GCS,table.h*sizeof(int),table.w*sizeof(int));
add_tables_GCS(&table,&packed_GCS);
test_GCS_data(&table,&packed_GCS);
int gcs_data_size=packed_GCS.col_info.w+packed_GCS.row_info.w;
int gcs_bit_field=((table.w*table.h)/(sizeof(int)*8));
int gcs_new_data_len=packed_GCS.table.w*packed_GCS.table.h;
printf("GCS reduced size data len=%d,%d,%d bitf=%d new table len=%d,%d,%d \n",packed_GCS
.col_info.w,packed_GCS.row_info.w,gcs_data_size,gcs_bit_field,packed_GCS.table.w,
packed_GCS.table.h,gcs_new_data_len);
printf("GCS TOTAL=%d\n",gcs_data_size+gcs_bit_field+gcs_new_data_len);
free_GCS(&packed_GCS);
#endif

#endif

free_dyn_table(&table);
free_dyn_table(&table_data);
free_dyn_table(&row_list);
return 0;
};

```

Dodatek C

izvorna koda analize in uporabe *Bison* tabele

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#if defined (__STDC__) || defined (__cplusplus)
    typedef signed char yysigned_char;
#else
    typedef short int yysigned_char;
#endif

#ifdef YYTYPE_UINT8
    typedef YYTYPE_UINT8 yytype_uint8;
#else
    typedef unsigned char yytype_uint8;
#endif
#ifdef YYTYPE_INT8
    typedef YYTYPE_INT8 yytype_int8;
#elif (defined __STDC__ || defined __C99_FUNC__ \
      || defined __cplusplus || defined _MSC_VER)
    typedef signed char yytype_int8;
#else
    typedef short int yytype_int8;
#endif

#define YYDEBUG 1

/* YYFINAL -- State number of the termination state. */
#define YYFINAL 7
/* YYLAST -- Last index in YYTABLE. */
#define YYLAST 11

/* YYNTOKENS -- Number of terminals. */
#define YYNTOKENS 8
/* YYNNTS -- Number of nonterminals. */
#define YYNNTS 4
/* YYNRULES -- Number of rules. */
#define YYNRULES 7
/* YYNSTATES -- Number of states. */
#define YYNSTATES 13

#ifdef YYDEBUG || YYERROR_VERBOSE || YYTOKEN_TABLE
/* YYTNAME[SYMBOL-NUM] -- String name of the symbol SYMBOL-NUM.
   First, the terminals, then, starting at YYNTOKENS, nonterminals. */
static const char *const yytname[] =
{
    "$end", "error", "$undefined", "id", "'+'", "'*'", "'('", "')'",
    "$accept", "exp", "term", "factor", 0
};
#endif
```

```

/* YYDEFACT[STATE-NAME] — Default rule to reduce with in state
   STATE-NAME when YYTABLE doesn't specify something else to do. Zero
   means the default is an error. */

/* YYR1[YYN] — Symbol number of symbol that rule YYN derives. */
static const yytype_uint8 yyr1[] =
{
    0,      8,      9,      9,      10,      10,      11,      11
};

/* YYR2[YYN] — Number of symbols composing right hand side of rule YYN. */
static const yytype_uint8 yyr2[] =
{
    0,      2,      3,      1,      3,      1,      3,      1
};

static const yytype_uint8 yydefact[] =
{
    0,      7,      0,      0,      3,      5,      0,      1,      0,      0,
    6,      2,      4
};

/* YYDEFGOTO[NTERM-TERM]. */
static const yytype_int8 yydefgoto[] =
{
    -1,      3,      4,      5
};

/* YYPACT[STATE-NAME] — Index in YYTABLE of the portion describing
   STATE-NAME. */
#define YYPACT_NINF -5
static const yytype_int8 yypact[] =
{
    -1,      -5,      -1,      0,      -4,      -5,      2,      -5,      -1,      -1,
    -5,      -4,      -5
};

/* YYPGOTO[NTERM-TERM]. */
static const yytype_int8 yypgoto[] =
{
    -5,      1,      3,      -2
};

/* YYTABLE[YYPACT[STATE-NAME]]. What to do in state STATE-NAME. If
   positive, shift that token. If negative, reduce the rule which
   number is the opposite. If zero, do what YYDEFACT says.
   If YYTABLE_NINF, syntax error. */
#define YYTABLE_NINF -1
static const yytype_uint8 yytable[] =
{
    7,      9,      1,      6,      8,      2,      8,      12,      0,      10,
    0,      11
};

static const yytype_int8 yycheck[] =
{
    0,      5,      3,      2,      4,      6,      4,      9,      -1,      7,
    -1,      8
};

struct dyn_table{
    int w;
    int h;
    int *data;
};

int rt_data(struct dyn_table *tbl,int y,int x){
    if(y>=tbl->h||y<0) abort();
    if(x>=tbl->w||x<0) abort();
    return tbl->data[(tbl->w*y)+x];
};

int set_data(struct dyn_table *tbl,int y,int x,int val){
    if(y>=tbl->h||y<0) abort();
    if(x>=tbl->w||x<0) abort();
    tbl->data[(tbl->w*y)+x]=val;
    return 0;
};

void free_dyn_table(struct dyn_table *tbl){

```

```

    if(tbl->data)
        free(tbl->data);
    tbl->data=NULL;
    tbl->w=tbl->h=0;
};

int alloc_dyn_table(struct dyn_table *tbl,int w,int h){
    free_dyn_table(tbl);
    tbl->data=malloc(sizeof(int)*w*h);
    memset(tbl->data,0,sizeof(int)*w*h);
    tbl->w=w;
    tbl->h=h;
};

int set_dyn_table(struct dyn_table *tbl){
    memset(tbl->data,0xfe,sizeof(int)*tbl->w*tbl->h);
};

void main(){
    int sym,state;
    for(state=0;state<YYNTOKENS;state++){
        printf("%s ",yytname[state]);
        printf("\n");
    }
    for(state=0;state<YYNSTATES;state++){
        printf("state %2d ",state);
        for(sym=0;sym<YYNTOKENS;sym++){
            if(yypact[state]==YYPACT_NINF){
                if(yydefact[state]==0)
                    printf(" e ");
                else
                    printf(" r ");
            }
            else{
                int yyn;
                yyn=yypact[state]+sym;
                if(yyn<0 || YYLAST<yyn || yycheck[yyn]!=sym){
                    if(yydefact[state]==0)
                        printf(" E ");
                    else
                        printf(" D ");
                }
                else{
                    printf("%2d ",yytable[yyn]);
                }
            }
        }
        printf("\n");
    }

    for(state=YYNTOKENS;state<YYNTOKENS+YYNNTS;state++){
        printf("%d %s ",state,yytname[state]);
        printf("\n");
    }

    for(state=0;state<YYNSTATES;state++){
        printf("state %2d ",state);
        for(sym=0;sym<YYNNTS;sym++){
            int yystate;
            yystate=yypgoto[sym]+state;
            if(0<=yystate&&yystate<=YYLAST&&yycheck[yystate]==state){
                printf("%2d ",yytable[yystate]);
            }
            else{
                printf("%2d ",yydefgoto[sym]);
            }
        }
        printf("\n");
    }

    {
        //LR PARSER BEGIN
        struct dyn_table input={0};
        struct dyn_table stack={0};
        int stack_size;
        int error=0;
        int ccc=0;
        int curr_state;
        int curr_input=0;
        alloc_dyn_table(&input,4,1);
        alloc_dyn_table(&stack,256,1);
        set_data(&input,0,0,3);//id
    }
}

```

```

set_data(&input,0,1,5);/*
set_data(&input,0,2,3);/*id
set_data(&input,0,3,0);/*+

set_data(&stack,0,0,0);
stack_size=1; /*start index
while(!error&&stack_size>0&&ccc<15){
    ccc++;
    int action=-1;

    curr_state=rt_data(&stack,0,stack_size-1);
    {
        int i;
        printf("stack size=%d ",stack_size);
        for(i=0;i<stack_size;i++){
            printf("%d ",rt_data(&stack,0,i));
            printf("\n");
        };
        printf("curr_state=%d\n",curr_state);

        int rule_num=0;
        if(yyvspact[curr_state]==YYPACT_NINF){
            action=3;
            printf("YYPACT_NINF default action yyact[curr_state]=%d YYACT_NINF=%d\n",yyvspact[
                curr_state],YYPACT_NINF);
            //default action
        }
        else{
            int yyn;
            int next_input;
            if(curr_input<input.w){
                next_input=rt_data(&input,0,curr_input);
            }
            else{
                printf("no input curr_inp=%d w=%d\n",curr_input,input.w);
                if(stack_size>=1)
                    stack_size-=1;
            }

            yyn=yyvspact[curr_state]+next_input;
            if (yyn < 0 || YYLAST < yyn || yycheck[yyn] != next_input){
                action=3; //default action
            }
            else{
                if(yytable[yyn]<=0){
                    int value=yytable[yyn];
                    if (value == 0 || value == YYTABLE_NINF){
                        printf("got error\n");
                        error=1;
                        break;
                    };
                    printf("reduce action value=%d\n",-value);
                    rule_num=-value;
                    action=2;
                }
                else{
                    action=1; //shift
                    rule_num=yytable[yyn];
                }
            };
        };
    };

    if(action==3){
        rule_num=yydefact[curr_state];
        action=2;
    };

    if(action==1){
        printf("shift to state %2d\n",rule_num);
        if(rule_num==YYFINAL){
            printf("input accepted\n");
            break;
        };
        set_data(&stack,0,stack_size++,rule_num); //state
        curr_input++;
    }
    else
    if(action==2){
        printf("reduce by rule %d\n",rule_num);
        stack_size-=yyr2[rule_num];
        int old_state=rt_data(&stack,0,stack_size-1);

```

```

    int sym;
    int sym_fix;

    sym=yyl1[rule_num];
    sym_fix=sym-YYNTOKENS;
    if(sym<YYNTOKENS){
        printf("err\n");
        break;
    };
    int yystate=yypgoto[sym_fix]+old_state;
    printf("REDUCE BY sym=%d sym_fix=%d yystate=%d old_state=%d ycheck[yystate]=%d\n"
        ,sym,sym_fix,yystate,old_state,ycheck[yystate]);
    if (0 <= yystate && yystate <= YYLAST && ycheck[yystate] == old_state){
        printf("1-%2d\n",yytable[yystate]);
        set_data(&stack,0,stack_size++,yytable[yystate]); //state
        //set_data(&stack,0,stack_size++,yyl1[rule_num]); //symbol value
    }
    else{
        printf("2-%2d\n",yydefgoto[sym_fix]);
        //if(yydefgoto[sym_fix]){
        set_data(&stack,0,stack_size++,yydefgoto[sym_fix]); //state
        //set_data(&stack,0,stack_size++,yyl1[rule_num]); //symbol value
        //}
        //else{
        //    printf("WHAT\n");
        //    abort();
        //    set_data(&stack,0,stack_size-1,yyl1[rule_num]); //sym_fix
        //};
    };

}
//else
//if(action==3){
//    //default action
//    printf("default action\n");
//    if(yydefact[curr_state]==0){
//        printf("error\n");
//        error=1;
//    }
//    else{
//        };
//};
};
};
};

```

Dodatek D

izvorna koda uporabe patenta

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifdef REPORT
#include <fcntl.h>
#include <sys/stat.h>
#endif

int width=15;
int height=10;
//define WW 15
//define WH 10
#define DGOSTOTA 0.1
#define WW width
#define WH height

struct dyn_table{
    int w;
    int h;
    int *data;
};

struct packed_data{
    //int displace_rows [WH];
    struct dyn_table displace_rows;
    int *data;
    int *check_data;
    int data_len;
    int end_data;
    int end_check;
};

struct LR_parser_state{
    //int curr_state;
    int curr_input;
    int curr_table; //term table or nonterm
    //int start_index;
    struct dyn_table stack;
    int stack_size;
};

int match_row(struct dyn_table * table, struct dyn_table * rows_data, struct dyn_table *
    row_list, int srow, int crow);

int rt_data(struct dyn_table *tbl, int y, int x){
    if(y>=tbl->h||y<0) abort();
    if(x>=tbl->w||x<0) abort();
    return tbl->data[(tbl->w*y)+x];
};

int set_data(struct dyn_table *tbl, int y, int x, int val){
    if(y>=tbl->h||y<0) abort();
    if(x>=tbl->w||x<0) abort();
}
```

```

    tbl->data[(tbl->w*y)+x]=val;
    return 0;
};

void free_dyn_table(struct dyn_table *tbl){
    if(tbl->data)
        free(tbl->data);
    tbl->data=NULL;
    tbl->w=tbl->h=0;
};

int alloc_dyn_table(struct dyn_table *tbl,int w,int h){
    free_dyn_table(tbl);
    tbl->data=malloc(sizeof(int)*w*h);
    memset(tbl->data,0,sizeof(int)*w*h);
    tbl->w=w;
    tbl->h=h;
};

int set_dyn_table(struct dyn_table *tbl){
    memset(tbl->data,0xfe,sizeof(int)*tbl->w*tbl->h);
};

void print_data(struct packed_data *pck_data);

void alloc_data(struct packed_data *pck,int size){
    if(!pck) return;
    int old_data_size=pck->data_len;
    if(pck->data){
        pck->data=realloc(pck->data,size);
        memset(((char *)pck->data)+old_data_size,0,size-old_data_size);
    }else{
        pck->data=malloc(size);
        memset(pck->data,0,size);
    };
    if(pck->check_data){
        pck->check_data=realloc(pck->check_data,size);
        memset(((char *)pck->check_data)+old_data_size,0xfe,size-old_data_size);
    }else{
        pck->check_data=malloc(size);
        memset(pck->check_data,0xfe,size);
    };
    pck->data_len=size;
};

void reset_data(struct packed_data *pck_data){
    if(pck_data->data){
        bzero(pck_data->data,pck_data->data_len);
    };
    if(pck_data->check_data){
        //bzero(pck_data->check_data,pck_data->data_len);
        memset(pck_data->check_data,0xfe,pck_data->data_len);
    };
    //bzero(pck_data->displace_rows,sizeof(pck_data->displace_rows));
    alloc_dyn_table(&pck_data->displace_rows,WH,1);
    set_dyn_table(&pck_data->displace_rows);
    pck_data->end_data=0;
    pck_data->end_check=0;
};

void print_table(struct dyn_table *table){
    int i,j;
    for (i=0;i<table->h;i++){
        {
            for(j=0;j<table->w;j++){
                {
                    printf("%2d ",rt_data(table,i,j));
                }
                printf("\n");
            }
        }
    };
};

void fill_table(struct dyn_table * table,int *temp_table,int count){
    //int count;
    int c;
    for(c=0;c<count;c++){
        //printf("a=%d b=%d c=%d\n",c/table->w,c%table->w,temp_table[c]);
        set_data(table,c/table->w,c%table->w,temp_table[c]);
    };
};

```

```

    //};
};

void check_rows(struct dyn_table * table, struct dyn_table * rows_data){//[WH][4]
    int col, row;
    int start, end, count;
    for (row=0; row<table->h; row++){
        end=start=-1;
        count=0;
        for (col=0; col<table->w; col++){
            if (rt_data(table, row, col)!=0){
                //printf("%d,%d\n", row, col);
                if (start<0){
                    start=col;
                    end=start+1;
                }
                else{
                    end=col+1;
                }
                count++;
            }
        };
        set_data(rows_data, 0, row, count);
        set_data(rows_data, 1, row, end-start);
        set_data(rows_data, 2, row, start);
        set_data(rows_data, 3, row, end);
        //printf("setting row_data=%d c=%d tal=%d start=%d end=%d\n", row, count, end-start, start
        , end);
    };
};

void check_columns(struct dyn_table * table, struct dyn_table * rows_data, struct dyn_table
* column_data){//, struct dyn_table * row_list
    int ii, col, row;
    int count;
    struct dyn_table temp_table={0};
    int matches_count=0;

    alloc_dyn_table(column_data, table->w, 2);
    alloc_dyn_table(&temp_table, table->w, 2);

    for (col=0; col<table->h; col++)
        //for (ii=0; ii<table->h; ii++)
        {
            for (row=0; row<table->h; row++)
                if (row!=col){
                    int matched=0;
                    int k;
                    for (k=0; k<matches_count; k++){
                        int tmpval=rt_data(&temp_table, 0, k);
                        if (tmpval==col || row==tmpval)
                            matched++;
                    };

                    if (!matched && match_row(table, rows_data, NULL, col, row)){
                        //printf("found match row=%d col=%d\n", row, col);
                        set_data(&temp_table, 0, matches_count, row);
                        set_data(&temp_table, 0, matches_count, col);
                        matches_count++;
                    };
                };
        };

    for (col=0; col<table->w; col++){
        int k;
        int first_h=-1;
        count=0;
        //match for repeat
        for (row=0; row<table->h; row++)
            //for (ii=0; ii<table->h; ii++)
            {
                int found=0;
                //row=rt_data(row_list, 0, ii);
                //printf("row=%d ii=%d\n", row, ii);

                if (row<0) continue;
                for (k=0; k<matches_count; k++){
                    int tmpval=rt_data(&temp_table, 0, k);
                    int tmpval1=rt_data(&temp_table, 1, k);

```

```

        if (row==tmpval){
            //printf("MATCH found match tmpval=%d row=%d col=%d\n",tmpval+1,row+1,col+1);
            found++;
        };
    };
    if (!found && rt_data(table,row,col)!=0){
        //printf("%d,%d count=%d\n",row,col,count);
        //if (first_h<0)
            first_h=row;
        //first_h=ii;
        count++;
    };
};
set_data(column_data,0,col,count);
set_data(column_data,1,col,first_h);
//printf("cl\n",row,start,end,count);
};

free_dyn_table(&temp_table);

};

void get_terminal_defaults(struct dyn_table * table, struct dyn_table * default_reductions)
{
    int k,i,j;
    struct dyn_table temp_table={0};
    int count=0;

    alloc_dyn_table(&temp_table,table->w,2);

    for (k=0;k<table->h;k++){
        int _cnt=0;

        for (i=0;i<table->w;i++){
            set_data(&temp_table,0,i,0);
            set_data(&temp_table,1,i,0);
        };

        for (i=0;i<table->w;i++){
            int cur_reduction=rt_data(table,k,i);
            if (cur_reduction<0){
                int index=-1;

                for (j=0;j<_cnt;j++){
                    int tmp_val=rt_data(&temp_table,0,j);
                    if (tmp_val<0){
                        if (tmp_val==cur_reduction){
                            index=j;
                            set_data(&temp_table,1,j,rt_data(&temp_table,1,j)+1);
                            break;
                        };
                    };
                };
                if (index<0){
                    index=_cnt;
                    set_data(&temp_table,0,j,cur_reduction);
                    set_data(&temp_table,1,j,rt_data(&temp_table,1,j)+1);
                    _cnt++;
                };
            };
        };

        int idx_max=0;
        int max_val=rt_data(&temp_table,1,0);

        for (i=1;i<_cnt;i++){
            if (max_val<rt_data(&temp_table,1,i)){
                idx_max=i;
                max_val=rt_data(&temp_table,1,i);
            };
        };
        set_data(default_reductions,0,k,rt_data(&temp_table,0,idx_max));

        //printf("vrstica k=%d max_val=%d val=%d\n",k,rt_data(&temp_table,1,idx_max),rt_data(&
            temp_table,0,idx_max));
    };

    //remove default reductions fromd data
    for (k=0;k<table->h;k++){

```

```

    int def_reduction=rt_data(default_reductions,0,k);
    for(i=0;i<table->w;i++){
        if(rt_data(table,k,i)==def_reduction){
            set_data(table,k,i,0);
        };
    };
};
free_dyn_table(&temp_table);
};

void get_non_terminal_defaults(struct dyn_table * table,struct dyn_table *
    default_reductions){
    int k,i,j;
    struct dyn_table temp_table={0};
    int count=0;

    alloc_dyn_table(&temp_table,table->h,2);

    for(k=0;k<table->w;k++){
        int _cnt=0;

        for(i=0;i<table->h;i++){
            set_data(&temp_table,0,i,0);
            set_data(&temp_table,1,i,0);
        };

        for(i=0;i<table->h;i++){
            int cur_action=rt_data(table,i,k);
            if(cur_action!=0){
                int index=-1;

                for(j=0;j<_cnt;j++){
                    int tmp_val=rt_data(&temp_table,0,j);
                    if(tmp_val==cur_action){
                        int val=rt_data(&temp_table,1,j);
                        index=j;
                        set_data(&temp_table,1,j,val+1);//bug+1
                        break;
                    };
                };
                if(index<0){
                    int val=rt_data(&temp_table,1,j);
                    index=_cnt;
                    set_data(&temp_table,0,j,cur_action);
                    set_data(&temp_table,1,j,val+1);
                    _cnt++;
                };
            };
        };

        //for(i=0;i<_cnt;i++){
        // printf("i=%d val1=%d val2=%d\n",i,rt_data(&temp_table,0,i),rt_data(&temp_table,1,i
        ));
        //};

        int idx_max=0;
        int max_val=rt_data(&temp_table,1,0);

        for(i=1;i<_cnt;i++){
            if(max_val<rt_data(&temp_table,1,i)){
                idx_max=i;
                max_val=rt_data(&temp_table,1,i);
            };
        };
        set_data(default_reductions,0,k,rt_data(&temp_table,0,idx_max));

        //printf("vrstica k=%d max_val=%d val=%d _cnt=%d\n",k,rt_data(&temp_table,1,idx_max),
        rt_data(&temp_table,0,idx_max),_cnt);
    };

    //remove default reductions fromd data
    for(k=0;k<table->w;k++){
        int def_action=rt_data(default_reductions,0,k);
        for(i=0;i<table->h;i++){
            if(rt_data(table,i,k)==def_action){
                set_data(table,i,k,0);
            };
        };
    };
};
free_dyn_table(&temp_table);

```

```

};

void grow_terminal_table(struct dyn_table * table, struct dyn_table * default_reductions,
                        int rules_count){
    struct dyn_table temp_table={0};

    int i,j;
    int old_w=table->w;
    int old_h=table->h;
    alloc_dyn_table(&temp_table, table->w, table->h);
    memcpy(temp_table.data, table->data, sizeof(int)*table->w*table->h);

    free_dyn_table(table);
    alloc_dyn_table(table, old_w+1, old_h);

    for(i=0; i<table->h; i++){
        int def_red=rt_data(default_reductions, 0, i);
        if(def_red<0){
            set_data(table, i, 0, def_red);
        }
        else{
            set_data(table, i, 0, rules_count+2); //this is special status error
        };
    };

    for(i=0; i<old_w; i++){
        for(j=0; j<old_h; j++){
            set_data(table, j, i+1, rt_data(&temp_table, j, i));
        };
    };

    free_dyn_table(&temp_table);
};

void reorder_fields(struct dyn_table * table, struct dyn_table * column_data, struct
dyn_table * terminal_translate, int startx, struct dyn_table * reorder_table){
    int i, pos, j, k;
    //struct dyn_table temp_table={0};
    struct dyn_table copy_table={0};
    int count=0;

    alloc_dyn_table(reorder_table, column_data->w, 1);
    printf("REORDER CALLED\n");
    //sort first
    for(i=startx; i<column_data->w; i++){
        int rowcount=rt_data(column_data, 0, i);
        pos=count-1;
        while(pos>=0&&rt_data(column_data, 0, rt_data(reorder_table, 0, pos)) < rowcount)
            pos--;
        while(pos>=0&& rowcount==rt_data(column_data, 0, rt_data(reorder_table, 0, pos)) &&
            rt_data(column_data, 1, rt_data(reorder_table, 0, pos)) < rt_data(column_data, 1, i))
            //while(pos>=0&& rowcount==rt_data(column_data, 0, rt_data(&temp_table, 0, pos)) &&
            //rt_data(column_data, 1, rt_data(&temp_table, 0, pos)) > rt_data(column_data, 1, i))
            pos--;
        for(j=count-1; j>pos; j--){
            set_data(reorder_table, 0, j+1, rt_data(reorder_table, 0, j));
        };
        //row_list[pos+1]=i;
        set_data(reorder_table, 0, pos+1, i);
        count++;
    };
    /*for(i=0; i<count; i++){
        printf("(%d,%d,%d) ", rt_data(reorder_table, 0, i), rt_data(column_data, 0, rt_data(
            reorder_table, 0, i)), rt_data(column_data, 1, rt_data(reorder_table, 0, i)));
    };
    printf("\n"); */

    alloc_dyn_table(&copy_table, table->w, table->h);
    for(k=0; k<count; k++){
        int column_index=rt_data(reorder_table, 0, k);
        //printf("copy k=%d column_index=%d table->h=%d\n", k, column_index, table->h);
        for(i=0; i<table->h; i++){
            set_data(&copy_table, i, k+startx, rt_data(table, i, column_index));
        };
        //set_data(terminal_translate, 0, column_index,
    };
    if(startx)
        for(i=0; i<table->h; i++){
            set_data(&copy_table, i, 0, rt_data(table, i, 0));
        };
    //print_table(&copy_table);

```

```

//printf("end print\n");
memcpy(table->data, copy_table.data, sizeof(int)*table->w*table->h);
//free_dyn_table(&temp_table);
free_dyn_table(&copy_table);
};

void reorder_nonterm_default_table(struct dyn_table * reorder_table, struct dyn_table *
    default_table){
    struct dyn_table temp_table={0};
    int i;

    alloc_dyn_table(&temp_table, default_table->w, 1);

    for(i=0; i<default_table->w; i++){
        int reorder_index=rt_data(reorder_table, 0, i);
        //printf("reorder index=%d\n", reorder_index);
        set_data(&temp_table, 0, i, rt_data(default_table, 0, reorder_index));
    };

    memcpy(default_table->data, temp_table.data, sizeof(int)*default_table->w*default_table->h);
    free_dyn_table(&temp_table);
};

int sort_rows(struct dyn_table * rows_data, struct dyn_table * row_list, int flags){//[WH
    ][4] [WH]
    int count=0;
    int i, j;
    int pos;
    for(i=0; i<rows_data->w; i++){
        set_data(row_list, 0, i, -1);
    }
    for(i=0; i<rows_data->w; i++){
        int rowcount=rt_data(rows_data, 0, i);
        //printf("row %d len=%d\n", i, rowcount);
        if(rowcount>0){
            pos=count-1;
            while(pos>=0&&rt_data(rows_data, 0, rt_data(row_list, 0, pos)) < rowcount)
                pos--;
            //while(pos>=0&& rowcount==rt_data(rows_data, 0, rt_data(row_list, 0, pos)) && rt_data(
            //    rows_data, 1, rt_data(row_list, 0, pos)) < rt_data(rows_data, 1, i))
            if(flags)
                while(pos>=0&& rowcount==rt_data(rows_data, 0, rt_data(row_list, 0, pos)) && rt_data(
                    rows_data, 3, rt_data(row_list, 0, pos)) < rt_data(rows_data, 3, i))
                    pos--;
            //printf("setting row=%d to pos=%d len=%d alt=%d\n", i, pos, rowcount, rt_data(rows_data
            //    , 3, i));
            for(j=count-1; j>pos; j--){
                set_data(row_list, 0, j+1, rt_data(row_list, 0, j));
            };
            //row_list[pos+1]=i;
            set_data(row_list, 0, pos+1, i);
            count++;
        };
    };
    //printf("sort_rows called\n");
    //print_table(row_list);
    for(i=0; i<count; i++)
        printf("%d count=%d tally=%d\n", row_list[i], rows_data[row_list[i]][0], rows_data[
            row_list[i]][1]);
    return count;
};

int match_row(struct dyn_table * table, struct dyn_table * rows_data, struct dyn_table *
    row_list, int srow, int crow){
    //printf("match_row row=%d srow=%d\n", crow, srow);
    if(rt_data(rows_data, 0, srow)>0&&rt_data(rows_data, 0, srow)==rt_data(rows_data, 0, crow)&&
        rt_data(rows_data, 1, srow)==rt_data(rows_data, 1, crow)){
        int c;
        for(c=rt_data(rows_data, 2, srow); c<rt_data(rows_data, 2, srow)+rt_data(rows_data, 1, srow);
            c++){
            if(rt_data(table, srow, c)!=rt_data(table, crow, c))
                return 0;
        };
        //printf("rows_data[srow][2] =%d size=%d c=%d srow=%d crow=%d\n", rows_data[srow][2],
        //    rows_data[srow][1], c, srow, crow);
        //printf("table[srow][c]=%d table[crow][c]=%d\n", table[srow][c], table[crow][c]);
        //printf("ROW MATCHES crow=%d srow=%d\n", crow, srow);
        return 1;
    }
}
return 0;

```

```

};

int pack_row(struct dyn_table * table, struct dyn_table * rows_data /*[WH][4]*/, struct
packed_data *pck_data, int row_idx, int lines_added){
    int displace_idx=rt_data(rows_data, 2, row_idx);
    int i;
    int place_at=0;
    int ok;

    if(rt_data(rows_data, 1, row_idx)<=0){
        abort();
        return -1;
    };
    for(;; place_at++){
        int k;
        //printf("place at=%d idx=%d\n", place_at, row_idx);
        ok=1;
        if((place_at+table->w+1)*sizeof(int)>=pck_data->data_len){
            //printf("pck_data->data_len=%d\n", pck_data->data_len);
            int data_len=pck_data->data_len*2;
            if(data_len<=(+table->w*sizeof(int)))
                data_len+=(+table->w*sizeof(int))*2;
            alloc_data(pck_data, data_len);
        };

        for(k=0;k<table->w&&ok;k++){//rt_data(rows_data, 1, row_idx)
# if 0
            if(pck_data->data[place_at+k]!=0){
                //printf("place_at=%d k=%d data=%d\n", place_at, k, pck_data->data[place_at+k]);
                ok=0;
                break;
            };
# else
            //printf("row_idx=%d k=%d place_at=%d data=%d loc=%d\n", row_idx, k, place_at, rt_data(
                table, row_idx, k), pck_data->data[place_at+k]);
            if(rt_data(table, row_idx, k)&&pck_data->data[place_at+k]){//rt_data(rows_data, 2,
                row_idx)+
                //printf("place_at=%d k=%d data=%d\n", place_at, k, pck_data->data[place_at+k]);
                ok=0;
                break;
            };
# endif
        };
    };
# if 1
    for(k=0;k<table->h&&ok;k++){//lines_added
        //printf("lines_added=%d k=%d\n", lines_added, k);
        //printf("rt_data(&pck_data->displace_rows, 0, k)=%d place_at=rt_data(rows_data, 2,
            row_idx)=%d\n", rt_data(&pck_data->displace_rows, 0, k), place_at=rt_data(rows_data,
            2, row_idx));
        if(rt_data(&pck_data->displace_rows, 0, k)==place_at){
            //printf("SAME INDEX k=%d %d\n", rt_data(&pck_data->displace_rows, 0, k), place_at=
                rt_data(rows_data, 2, row_idx));
            //printf("Same index \n");
            ok=0;
            break;
        };
    };
# endif
    if(ok){
        //printf("found place at %d tally=%d\n", place_at, rt_data(rows_data, 1, row_idx));
        int ww=0;
        int wc=0;
        for(k=0;k<table->w;k++){//rt_data(rows_data, 1, row_idx)
            if(rt_data(table, row_idx, k)!=0){//rt_data(rows_data, 2, row_idx)+
                pck_data->data[place_at+k]=rt_data(table, row_idx, k);//rt_data(rows_data, 2,
                    row_idx)+
                //printf("writing data idx=%d d=%d\n", place_at+k, pck_data->data[place_at+k]);
                //pck_data->check_data[place_at+k]=row_idx;
                pck_data->check_data[place_at+k]=k;//rt_data(rows_data, 2, row_idx)+
                ww=k;
            }
            else{
                wc=k;
                if(pck_data->check_data[place_at+k]<0)
                    pck_data->check_data[place_at+k]=0;
                //pck_data->check_data[place_at+k]=0;
            };
        };
        //if(place_at+rt_data(rows_data, 1, row_idx)+k>pck_data->end_data)
        //    pck_data->end_data=place_at+rt_data(rows_data, 1, row_idx)+k;
    }
}

```

```

        if(place_at+ww>pck_data->end_data)
            pck_data->end_data=place_at+ww;
        if(place_at+wc>pck_data->end_check)
            pck_data->end_check=place_at+wc;

        return place_at; // -rt_data(rows_data,2,row_idx)
    };
};

void add_tables(struct dyn_table * table, struct dyn_table * rows_data, struct dyn_table *
    row_list, int row_count, struct packed_data * pck_data){
    int cnt;
    int lines_added=0;

    for(cnt=0; cnt<row_count; cnt++){
        int idx, j, found=0;
        int table_index=rt_data(row_list, 0, cnt);

        //printf("adding row =%d\n", table_index);
        for(j=0; j<cnt; j++){
            //printf("table_index=%d rt_data(row_list, 0, j)=%d\n", table_index, rt_data(row_list, 0,
            j));
            //table_index!=rt_data(row_list, 0, j)
            if(match_row(table, rows_data, row_list, table_index, rt_data(row_list, 0, j))){ //row_list
                [j] //
                found++;
                break;
            }
        };

        if(rt_data(rows_data, 0, table_index)>0){
            //printf("adding row %d\n", table_index);
            if(found){
                //printf("vrstica se ponavlja idx=%d\n", table_index);
                //found=table_index;
                found=rt_data(&pck_data->displace_rows, 0, rt_data(row_list, 0, j));
            }
            else{
                found=pack_row(table, rows_data, pck_data, table_index, lines_added);
                lines_added++;
            };
            //printf("adding row %d found=%d\n", table_index+1, found+1);
            set_data(&pck_data->displace_rows, 0, table_index, found);
            //printf("idx=%d found=%d\n", table_index, found);
        };
    };
};

int pack_row_nonterm(struct dyn_table * table, struct dyn_table * rows_data /*[WH][4]*/,
    struct packed_data * pck_data, int row_idx, int lines_added, struct packed_data *
    term_table){
    int displace_idx=rt_data(rows_data, 2, row_idx);
    int i;
    int place_at=0;
    int ok;

    //if(rt_data(rows_data, 1, row_idx)<=0){
    //    abort();
    //    return -1;
    //};
    for(; place_at++){
        int k;
        //printf("place at=%d idx=%d\n", place_at, row_idx);
        ok=1;
        if((place_at+table->w+1)*sizeof(int)>=pck_data->data_len){
            //print_data(pck_data);
            int data_len=pck_data->data_len*2;
            if(data_len<=(+table->w*sizeof(int)))
                data_len+=(+table->w*sizeof(int)*sizeof(int))*2;
            alloc_data(pck_data, data_len);
        };

        if(pck_data->data[place_at])
            ok=0;

        for(k=0; k<table->w&&ok; k++){ //rt_data(rows_data, 1, row_idx)
            //printf("row_idx=%d k=%d place_at=%d data=%d loc=%d\n", row_idx, k, place_at, rt_data(
            table, row_idx, k), pck_data->data[place_at+k]);

```

```

        if(rt_data(table, row_idx, k) && pck_data->data[place_at+k+1]){ //rt_data(rows_data, 2,
            row_idx)+
            //printf("place_at=%d k=%d data=%d\n", place_at, k, pck_data->data[place_at+k]);
            ok=0;
            break;
        };
    };
    if(ok){
        int ww=0;
        int wc=0;
        //printf("row=%d found place at %d tally=%d\n", row_idx, place_at, rt_data(rows_data, 1,
            row_idx));
        pck_data->data[place_at]=rt_data(&term_table->displace_rows, 0, row_idx)+1; //rt_data(
            rows_data, 2, row_idx)+
        //printf("Setting to term index to %d\n", pck_data->data[place_at]);
        pck_data->check_data[place_at]=0; //rt_data(rows_data, 2, row_idx)+
        for(k=0; k<table->w; k++){ //rt_data(rows_data, 1, row_idx)
            if(rt_data(table, row_idx, k)!=0){ //rt_data(rows_data, 2, row_idx)+
                pck_data->data[place_at+k+1]=rt_data(table, row_idx, k); //rt_data(rows_data, 2,
                    row_idx)+
                pck_data->check_data[place_at+k+1]=k+1; //rt_data(rows_data, 2, row_idx)+
                //printf("writing data place_at=%d idx=%d d=%d chk=%d\n", place_at, k, pck_data->
                    data[place_at+k+1], pck_data->check_data[place_at+k+1]);
                //pck_data->check_data[place_at+k]=row_idx;
                ww=k;
            }
            else{
                if(pck_data->check_data[place_at+k+1]<0)
                    pck_data->check_data[place_at+k+1]=0;
                wc=k;
            }
        };
        if(place_at+ww+1>pck_data->end_data)
            pck_data->end_data=place_at+ww+1;
        if(place_at+wc+1>pck_data->end_check)
            pck_data->end_check=place_at+wc+1;
        //printf("Sending enddata=%d place_at=%d k=%d\n", pck_data->end_data, place_at, k);
        set_data(&pck_data->displace_rows, 0, row_idx, place_at);
        //if(place_at+rt_data(rows_data, 1, row_idx)>pck_data->end_data)
        //    pck_data->end_data=place_at+rt_data(rows_data, 1, row_idx);

        return place_at; //rt_data(rows_data, 2, row_idx)
    };
};

void fix_nonterm_default_table(struct packed_data *nonterm_packed, struct packed_data *
    term_packed, struct dyn_table *def_table, int rules_cnt){
    int i;
    for(i=0; i<def_table->w; i++){
        int value=rt_data(def_table, 0, i)-1;
        //printf("Got value=%d\n", value);
        if(value>0){
            int newvalue=rt_data(&nonterm_packed->displace_rows, 0, value);
            //printf("new value=%d\n", newvalue+rules_cnt);
            set_data(def_table, 0, i, newvalue+rules_cnt);
        }
    };
};

void add_tables_nonterm(struct dyn_table *table, struct dyn_table *rows_data, struct
    dyn_table *row_list, int row_count, struct packed_data *pck_data, struct packed_data *
    term_table, int rules_count){
    int cnt;
    int lines_added=0;

    for(cnt=0; cnt<row_count; cnt++){
        int idx, j, found=0;
        int table_index=rt_data(row_list, 0, cnt);

        found=pack_row_nonterm(table, rows_data, pck_data, table_index, lines_added, term_table);
        lines_added++;
        //printf("adding row %d found=%d\n", table_index+1, found+1);
        //set_data(&pck_data->displace_rows, 0, table_index, found);
        //printf("idx=%d found=%d\n", table_index, found);
    };
    //add unadded rows
    printf("ADDING EMPTY ROWS\n");
    for(cnt=0; cnt<table->h; cnt++){
        int found=0;

```

```

//printf("cnt=%d\n", cnt);
for (lines_added=0; lines_added<table->h; lines_added++){
    if (rt_data(row_list, 0, lines_added)==cnt){
        //printf("line =%d in list \n", cnt);
        found++;
    };
};
if (found)
    continue;
pack_row_nonterm (table, rows_data, pck_data, cnt, 0, term_table);
//printf("adding row %d found=%d\n", cnt+1, found+1);
};

//translate all data
printf("translate data\n");
for (cnt=0; cnt<pck_data->end_data; cnt++){
    int found=0;
    //printf("cnt=%d\n", cnt);
    for (lines_added=0; lines_added<table->h; lines_added++){
        if (rt_data(&pck_data->displace_rows, 0, lines_added)==cnt){
            //printf("line =%d in list \n", cnt);
            found++;
        };
    };
    if (found)
        continue;
    int data=pck_data->data[cnt];
    if (data>0){
        int tdata;
        tdata=rt_data(&pck_data->displace_rows, 0, data-1)+rules_count;
        pck_data->data[cnt]=tdata;
        //printf("Setting data=%d tdata=%d\n", data, tdata);
    }
    else{
        pck_data->data[cnt]=-data;
    };
};

};

};

void fix_terminal_table(struct packed_data *terminal_data, struct packed_data *
    nonterminal_data, int rules_cnt){
    int i;

    for (i=0; i<terminal_data->end_data; i++){
        int value=terminal_data->data[i];
        if (value>rules_cnt){
            int to=nonterminal_data->end_data+rules_cnt+(value-rules_cnt)-1;
            //printf("convert error condition value=%d i=%d to=%d\n", value, i, to);
            terminal_data->data[i]=to;
        }
        else
            if (value>0){
                int to=rt_data(&nonterminal_data->displace_rows, 0, value-1)+rules_cnt;
                //printf("value=%d setto=%d\n", value, rt_data(&nonterminal_data->displace_rows, 0,
                    value-1)+rules_cnt);
                terminal_data->data[i]=to;
            }
            else if (value<0){
                //int to=rt_data(&nonterminal_data->displace_rows, 0, (-value)-1)+rules_cnt;
                int to=-value;
                //printf("unk stuff value=%d to=%d\n", value, to);
                terminal_data->data[i]=to;
            };
    };
};

};

int check_table(struct dyn_table * table, struct packed_data *pck_data){
    int i, j;
    for (i=0; i<table->h; i++){
        for (j=0; j<table->w; j++){
            int val=0;
            int chk=0;
            int idx;
            idx=rt_data(&pck_data->displace_rows, 0, i)+j;
            if (idx>=0&&idx<pck_data->end_data){
                //check_data
                chk=pck_data->check_data[idx];
            }
        }
    }
};

```

```

        if(chk==j)//j
            val=pck_data->data[idx];
    };

    if(val!=rt_data(table,i,j)){
        printf("False data val=%d chk=%d tab=%d i=%d,j=%d idx=%d displace=%d\n",val,chk,
            rt_data(table,i,j),i,j,idx,rt_data(&pck_data->displace_rows,0,i));
        return -1;
    };
};
};
return 0;
};

void print_data(struct packed_data *pck_data){
    int i;
    int tmp;

    printf("DISPLACE INDEX\n");
    for(i=0;i<WH;i++){
        tmp=rt_data(&pck_data->displace_rows,0,i);
        if(tmp<-0xffffffff)
            printf("NA ");
        else
            printf("%2d ",rt_data(&pck_data->displace_rows,0,i));
    };
    printf("\n");

    printf("DATA TABLE\n");
    for(i=0;i<pck_data->end_data;i++){
        printf("%2d ",pck_data->data[i]);
    };
    printf("\n");
    printf("CHECK TABLE\n");
    for(i=0;i<pck_data->end_check;i++){
        if(pck_data->check_data[i]<0)
            printf("NA ");
        else
            printf("%2d ",pck_data->check_data[i]);
    };
    printf("\n");
};

int main(){
    //int table [WH][WW];
    //int table_data [WH][4];
    //int row_list [WH];
    struct dyn_table table={0};
    struct dyn_table table_data={0};
    struct dyn_table row_list={0};
    struct dyn_table column_list={0};
    struct dyn_table translate_terminal={0};
    struct dyn_table terminal_defaults={0};
    struct dyn_table nonterminal_defaults={0};
    struct dyn_table nonterminal_reordertable={0};
    struct dyn_table terminal_reordertable={0};

    int row_count;
    int rep;
    int element_count;

    struct packed_data packed;
    struct packed_data nonterm_packed;

    srand(time(0));
    bzero((void *) &packed,sizeof(struct packed_data));
    bzero((void *) &nonterm_packed,sizeof(struct packed_data));
    alloc_data(&packed,1024);
    alloc_data(&nonterm_packed,1024);
    width=6;
    height=12;
#define TERMINAL_COUNT 6
#define NON_TERMINAL_COUNT 5
#define STATES_COUNT 12
    int term_table[STATES_COUNT*TERMINAL_COUNT]={
        5,6,0,3,0,0,
        0,0,0,7,0,STATES_COUNT+1,
        5,6,0,0,0,0,
        0,0,-1,-1,9,-1,
        0,6,-8,-8,-8,-8,

```

```

    5,6,0,3,0,0,
    5,6,0,0,0,0,
    0,0,-2,-2,9,-2,
    5,6,0,0,0,0,
    0,0,12,7,0,0,
    0,0,-3,-3,9,-3,
    0,6,-10,-10,-10,-10
};

int nonterm_table[STATES_COUNT*NON_TERMINAL_COUNT]={
    2,4,-4,-6,-7,
    0,0,0,0,0,
    0,8,-4,-6,-7,
    0,0,0,0,0,
    0,0,0,0,-9,
    10,4,-4,-6,-7,
    0,11,-4,-6,-7,
    0,0,0,0,0,
    0,0,-5,-6,-7,
    0,0,0,0,0,
    0,0,0,0,0,
    0,0,0,0,-11
};
#define RULES_COUNT 12

alloc_dyn_table(&table,WW,WH);
alloc_dyn_table(&table_data,WH,4);
alloc_dyn_table(&row_list,WH,1);
//alloc_dyn_table(&column_list,WW,1);
alloc_dyn_table(&terminal_defaults,STATES_COUNT,1);
alloc_dyn_table(&translate_terminal,TERMINAL_COUNT,1);

//memset(packed.displace_rows,0,sizeof(packed.displace_rows));

//for(rep=0;rep<1000;rep++){
reset_data(&packed);
//fill_table(&table,element_count);
fill_table(&table,term_table,STATES_COUNT*TERMINAL_COUNT);
//check_rows(&table,&table_data);
//check_columns(&table,&table_data,&column_list);
get_terminal_defaults(&table,&terminal_defaults);
//print_table(&terminal_defaults);
//printf("terminal defaults\n");
grow_terminal_table(&table,&terminal_defaults,RULES_COUNT);
check_rows(&table,&table_data);
row_count=sort_rows(&table_data,&row_list,0);
check_columns(&table,&table_data,&column_list);//,&row_list
//print_table(&table);
//printf("after check\n");
reorder_fields(&table,&column_list,&translate_terminal,1,&terminal_reordertable);
//print_table(&table);
//printf("after reorder\n");
check_rows(&table,&table_data);
//print_table(&table);
//printf("after reorder\n");
//row_count=sort_rows(&table_data,&row_list);
add_tables(&table,&table_data,&row_list,row_count,&packed);
//if(check_table(&table,&packed)){
//    //check failed
//    printf("BUG CHECK FAILED\n");
//    //print_table(&table);
//    print_data(&packed);
//    //break;
//}
//else{

//};
//print_table(&table);
print_data(&packed);

printf("DOING NONTERM TABLE\n");
free_dyn_table(&table);
free_dyn_table(&table_data);
free_dyn_table(&row_list);
free_dyn_table(&column_list);

alloc_dyn_table(&table,NON_TERMINAL_COUNT,STATES_COUNT);
alloc_dyn_table(&table_data,STATES_COUNT,4);
alloc_dyn_table(&row_list,STATES_COUNT,1);

```

```

alloc_dyn_table(&nonterminal_defaults, NON_TERMINAL_COUNT, 1);

fill_table(&table, nonterm_table, STATES_COUNT * NON_TERMINAL_COUNT);
//print_table(&table);
get_nonterminal_defaults(&table, &nonterminal_defaults);
//print_table(&nonterminal_defaults);
//printf("nonterminal_defaults defaults\n");
//print_table(&table);
//printf("DOING SORT\n");
check_rows(&table, &table_data);
row_count = sort_rows(&table_data, &row_list, 1);
check_columns(&table, &table_data, &column_list); //, &row_list
//print_table(&table);
//printf("after SORT\n");
reorder_fields(&table, &column_list, &translate_terminal, 0, &nonterminal_reordertable);
reorder_nonterm_default_table(&nonterminal_reordertable, &nonterminal_defaults);
//print_table(&table);
check_rows(&table, &table_data);
row_count = sort_rows(&table_data, &row_list, 1);
reset_data(&nonterm_packed);
print_table(&table);
//printf("BEFORE PACK\n");
add_tables_nonterm(&table, &table_data, &row_list, row_count, &nonterm_packed, &packed,
    RULES_COUNT);
fix_nonterm_default_table(&nonterm_packed, &packed, &nonterminal_defaults, RULES_COUNT);
print_data(&nonterm_packed);
printf("fixing term data\n");
print_data(&packed);
fix_terminal_table(&packed, &nonterm_packed, RULES_COUNT);
printf("fixed term data\n");
print_data(&packed);
//LR_PARSER_BEGIN
    struct LR_parser_state LRp = {0};
    struct dyn_table input = {0};
    int reduce_table[RULES_COUNT] = {1, 2, 3, 1, 3, 1, 1, 1, 2, 3, 4};
    int reduce_symb_table[RULES_COUNT] = {4, 4, 4, 2, 2, 3, 3, 5, 5, 1, 1};
    char *rules_strings[] = {"E::=T", "E::=+T", "E::=E+T", "T::=F", "T::=T*F", "F::=V", "F::=R", "V
        ::=id", "V::=idR", "R::=(E)", "R::=(E)R"};
    char *term_strings[] = {"(", "*", "+", "id", ")", "eof"}; //sorted
    char *nonterm_strings[] = {"R", "T", "F", "E", "V"}; //sorted
    int error = 0;
    int ccc = 0;
    alloc_dyn_table(&LRp.stack, 128, 1); //stack size
    alloc_dyn_table(&input, 6, 1);
    set_data(&input, 0, 0, 4); //id
    set_data(&input, 0, 1, 2); // *
    set_data(&input, 0, 2, 4); // +
    set_data(&input, 0, 3, 3); // +
    set_data(&input, 0, 4, 4); //id
    set_data(&input, 0, 5, 6); //eof
    set_data(&LRp.stack, 0, 0, 19 - RULES_COUNT);
    set_data(&LRp.stack, 0, 1, 0);
    LRp.stack_size = 2; //start index
    LRp.curr_table = 0;
    while(!error && LRp.stack_size > 0){ //LRp.curr_input < input.w[LRp.curr_input]
        int next_input = 0;
        int curr_state = -1;
        /*{
            int i;
            printf("stack size=%d\n", LRp.stack_size);
            for(i=0; i<LRp.stack_size; i++)
                printf("%d ", rt_data(&LRp.stack, 0, i));
            printf("\n");
        }*/
        curr_state = rt_data(&LRp.stack, 0, LRp.stack_size - 2);
        //printf("Current state=%d stack_size=%d cur_table=%d\n", curr_state, LRp.stack_size,
            LRp.curr_table);
        if(LRp.curr_table){ //current_symbol < 0
            int current_symbol = rt_data(&LRp.stack, 0, LRp.stack_size - 1);
            //int non_term_index = packed_data[curr_state];
            int non_term_index = curr_state;
            //printf("NONTERM GOT CURRENT SYMBOL %d index=%d\n", current_symbol, non_term_index)
                ;
            if(non_term_index < 0 || non_term_index > nonterm_packed.end_data){
                printf("overflow\n");
                error = 1;
                break;
            }
        }
        if(non_term_index + (-current_symbol) > nonterm_packed.end_data){

```

```

        error=1;
        printf("overflow2  nonterm_packed.end.data=%d\n", nonterm_packed.end.data);
        break;
    };
    int symbol=-(current_symbol);
    int rule_reduce=0;
    //printf("!!!! nonterm_packed.check_data[non_term_index+symbol]=%d value=%d symbol
    =%d def_val=%d\n", nonterm_packed.check_data[non_term_index+symbol],
    nonterm_packed.data[non_term_index+symbol], symbol, nonterm_packed.data[
    non_term_index]);
    if(nonterm_packed.check_data[non_term_index+symbol]==symbol){
        rule_reduce=nonterm_packed.data[non_term_index+symbol];
        if(rule_reduce<RULES.COUNT)
            rule_reduce=rule_reduce;
        //printf("NONTERM GOT SPECIFIC REDUCE f=%d\n", rule_reduce);
    }
    else{
        rule_reduce=rt_data(&nonterminal_defaults,0,symbol-1);
        //printf("NONTERM GOT DEFAULT REDUCE f=%d current_symbol=%s\n", rule_reduce,
        nonterm_strings[symbol-1]);
    };
    if(rule_reduce==0){
        error=1;
        break;
    }
    else
        if(rule_reduce>0){
            //int new_state=nonterm_packed.data[rule_reduce-RULES.COUNT];
            printf("NONTERMI GOTO idx=%d\n", rule_reduce-RULES.COUNT);
            //printf("SHIFT check ok new new value=%d\n", newvalue1);
            set_data(&LRp.stack,0,LRp.stack.size++,rule_reduce-RULES.COUNT); //state,
            new_state-1
            set_data(&LRp.stack,0,LRp.stack.size++,0); //symbol
            LRp.curr_table=0; //switch table
        }
        else{
            int rule_number=-(rule_reduce+1);
            int reduced_symbol=reduce_symb_table[rule_number];
            //LRp.stack.size-=reduce_table[rule_number]*2;
            //printf("NONTERM REDUCE curr_state=%d new_symbol=%d rule=%s\n", curr_state,-
            reduced_symbol, rules_strings[rule_number]);
            printf("NONTERMI REDUCE rule=%s new_symbol=%d\n", rules_strings[rule_number],-
            reduced_symbol);
            set_data(&LRp.stack,0,LRp.stack.size-1,-reduced_symbol);
        };
    };
}
else{
    if(LRp.curr_input<input.w){
        next_input=rt_data(&input,0,LRp.curr_input);
    }
    else{
        printf("no input curr_inp=%d w=%d\n",LRp.curr_input,input.w);
        if(LRp.stack.size>=2)
            LRp.stack.size-=2;
    }
    int next_index=nonterm_packed.data[curr_state]-1; //+next_input
    int value=packed.data[next_index+next_input];
    //printf("next_input=%d check=%d symbol=%s value=%d index=%d\n", next_input, packed.
    check_data[next_index+next_input], term_strings[next_input-1], value, next_index);
    ;
    if(next_input==packed.check_data[next_index+next_input]){
        //int new_value=value-RULES.COUNT;
        //printf("FOUND NEXT SHIFT\n");
        //int newvalue1=nonterm_packed.data[new_value]-1;
        printf("TERMINAL SHIFT symbol=%s newstate_value=%d\n", term_strings[next_input
        -1],value);
        if((value-RULES.COUNT)>=nonterm_packed.end.data){
            int diff=(value-RULES.COUNT)-nonterm_packed.end.data;
            if(diff==0)
                printf("INPUT ACCEPTED\n");
            else
                printf("EOF ENDING REASON arg=%d\n",diff);
            break;
        }
        set_data(&LRp.stack,0,LRp.stack.size++,value-RULES.COUNT); //state
        set_data(&LRp.stack,0,LRp.stack.size++,0); //symbol value
    }
    LRp.curr_input++;
}
else{
    //if(value<RULES.COUNT){

```

```

//int vval=packed.data[curr_state]-1;
int vval=packed.data[next_index]-1;
//printf("TERMINAL GOT DEFAULT REDUCTION value=%d vval=%d curr_state=%d\n",value
,vval,curr_state);
if(vval<RULES.COUNT){
    int reduced_symbol=-reduce_symb_table[vval];
    int reduce_size=reduce_table[vval]*2;
    LRp.stack_size-=reduce_size;
    //printf("TERMINAL REDUCE vval=%d reduce_size=%d curr_state=%d size=%d
    reduced_symbol=%d rule=%s\n",vval,reduce_size,curr_state,reduce_table[vval]
    ],reduced_symbol,rules_strings[vval]);
    printf("TERMINAL REDUCE rule=%s reduced_symbol=%d\n",rules_strings[vval],
    reduced_symbol);
    //set_data(&LRp.stack,0,LRp.stack_size-2,curr_state);
    if(rt_data(&LRp.stack,0,LRp.stack_size-2)==0)
        set_data(&LRp.stack,0,LRp.stack_size-2,curr_state);
    set_data(&LRp.stack,0,LRp.stack_size-1,reduced_symbol);
    LRp.curr_table++;//switch table
}else{
    printf("ERROR value=%d vval=%d\n",value,vval);
    error=1;
}
};
};
};
};

free_dyn_table(&table);
free_dyn_table(&table.data);
free_dyn_table(&row.list);
free_dyn_table(&translate_terminal);
free_dyn_table(&terminal_defaults);
//printf("full matrix len %d,%d=%d data_size=%d\n",WW,WH,WH*WW,WH*WW*sizeof(int));
//printf("reduced size data len=%d size=%d check is same total len=%d size=%d\n",packed.
end_data,packed.end_data*sizeof(int),packed.end_data*2,packed.end_data*sizeof(int)
*2);

return 0;
};

```