Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matevž Černe

Računalniški koncepti v nalogah s tekmovanja Bober

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Janez Demšar

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00066/2013
Datum: 02.04.2013

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

Kandidat: **MATEVŽ ČERNE**

Naslov: **RAČUNALNIŠKI KONCEPTI V NALOGAH S TEKMOVANJA BOBER**

COMPUTER SCIENCE CONCEPTS IN TASKS FROM THE BEBRAS COMPETITION

Vrsta naloge: Diplomsko delo univerzitetnega študija prve stopnje

Tematika naloge:

Tekmovanje Bober je primerna priložnost za poučevanje računalniških konceptov v osnovnih in srednjih šolah. Žal pa ostaja v večini šol neizkoriščena, kar je pogosto tudi posledica pomanjkljivega znanja učiteljev. Da to stanje presežemo, bi bilo potrebno pripraviti ustrezen material za poučevanje učiteljev.

V okviru diplomske naloge vzemite vzorec nalog s preteklih tekmovanj. Razdelite jih na kategorije glede na koncepte oz. področja računalništva, ki jih obravnavajo. Vsako področje najprej predstavite s teoretične perspektive, a na nivoju, primernem za učitelja brez posebne računalniške izobrazbe. Nato pokažite primere reševanja nekaj izbranih nalog s tega področja.

Mentor:

izr. prof. dr. Janez Demšar

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matevž Černe, z vpisno številko **63090012**, sem avtor diplomskega dela z naslovom:

Računalniški koncepti v nalogah s tekmovanja Bober

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Janeza Demšarja,
- 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 28. avgusta 2013 Podpis avtorja:

Zahvaljujem se vsem, ki so mi v času študija stali ob strani in me podpirali. Prav tako se zahvaljujem vsem, ki so pomagali pri nastajanju diplomskega dela.

Posebej se zahvaljujem mentorju izr. prof. dr. Janezu Demšarju za pomoč in nasvete pri izdelavi diplomskega dela.

Kazalo

Povzetek

Cilj diplomske naloge je bila priprava materialov za izobraževanje učiteljev, ki bi v prihodnosti s pomočjo nalog tekmovanja Bober poučevali računalniške koncepte na osnovnih in srednjih šolah. Naloge s tekmovanja Bober so bile razdeljene v osem kategorij, od katerih jih je šest obravnavalo področje računalništva. Za te je bil pripravljen pregled področja s teoretične perspektive na nivoju, primernem za učitelje brez posebne računalniške izobrazbe. Področja so bila razdeljena na poglavja v katerih so bili opisani posamezni algoritmi, podatkovne strukture ali tipi naloge. Vsakemu poglavju je dodan tudi primer naloge s tekmovanja Bober. Na koncu je predstavljena še vizija za prihodnost. V poučevanje računalništva v osnovnih in srednjih šolah bi bil vključen širši krog ljudi. Z njihovo pomočjo bi bilo učenje uporabe računalnika nadomeščeno z učenjem konceptov računalništva.

Ključne besede

tekmovanje Bober, koncepti računalništva, regularni izrazi, končni avtomati, kriptografija, kodiranje, dvojiški številski sistem, predstavitev podatkov, urejanje, polnjenje nahrbtnika, bisekcija, barvanje grafov, iskanje v globino, iskanje v širino, minimalno vpeto drevo, iskanje najkrajših poti, Dijkstrov algoritem, Huffmanov algoritem

Abstract

The main goal of our thesis was to prepare teaching materials to be used in primary and secondary school when teaching computing concepts. The tasks from Bebras Contest have been divided into eight categories, six of which belonged to the field of computer sciences. A review of the field on a level appropriate for teachers without a specific computer education was prepared. At the end of the thesis a vision for the future which would include a wider circle of people in the process of teaching computer sciences was presented. With their help teaching of computer use would be replaced with teaching of computer concepts to the children in primary in secondary school.

Keywords

Bebras Contest, computing concepts, regular expressions, finite-state machine, cryptography, coding, binary numbers, data presentation, sorting, knapsack problem, bisection, graph coloring, depth-first search, breadth first search, minimum spanning tree, shortest path, Dijkstra algorithm, Huffman algorithm

Poglavje 1

Uvod

V osnovnih in srednjih šolah je veliko tekmovanj in različnih področij. Poznamo tekmovanja iz matematike, logike, slovenskega jezika in še veliko drugih. V zadnjih letih pa se je pojavilo tudi tekmovanje v znanju računalništva, Bober. Ponavadi šole organizirajo krožke ali dodatne ure na katerih se pripravljajo na taka tekmovanja, čeprav te priprave niso zares nujne, saj se učenci in dijaki večino stvari naučijo med poukom. In tukaj pridemo do prvega problema: Koliko učiteljev dejansko zna učiti računalništvo?

Pojem učenje računalništva v šolah predstavlja bolj kot ne učenje uporabe računalnika in ne učenje računalništva. Vprašanje, ki se postavi, je naslednje: Ali se učenci med učenjem uporabe računalnika kaj naučijo? Odgovor je: Seveda ne. Računalništvo v veliko primerih učijo učitelji stari 40, 50 morda celo 60 let in učijo tisto, kar znajo: kako nekaj poiskati na internetu, kako spremeniti pisavo v Wordu in podobno. Dejansko stanje pa je tako, da zelo verjetno večina učencev ve o uporabi računalnika precej več kot učitelj.

Kako torej učiti računalništvo? Kot je v članku "Proč z računalniki v računalniških učilnicah"[5] napisal izr. prof. dr. Janez Demšar, je potrebno računalništvo učiti brez računalnikov. Učenje uporabe računalnika zamenjamo z učenjem konceptov računalništva.

Če se vrnemo nazaj na tekmovanje Bober: na tekmovanju ne tekmujejo v tem, kdo hitreje spremeni pisavo ali poišče neko stvar na internetu. Tekmu-

jejo v znanju računalništva, kako delujejo algoritmi, kako poiskati najhitrejšo pot med dvema krajema, kako skriti in potem prenesti besedilo sogovorniku, ne da bi ga drugi lahko prebrali.

V diplomski nalogi smo zajeli vzorec nalog s preteklih izvedb tekmovanja Bober. Naloge smo razdelili glede na koncepte in področja računalništva, ki so v posamezni nalogi obravnavana. Vsako od področij smo na nivoju primernem za učitelje brez posebne računalniške izobrazbe predstavili s teoretične perspektive. Nato smo na primerih prikazali, kako se naloge s posameznega področja rešujejo.

Naloge s tekmovanja Bober, pridobljene v še ne objavljenem delu[6], smo razdelili v osem kategorij (imena nalog ustrezajo imenom v omenjenem gradivu):

- Kodiranje in kriptografija - "Bevri", "Ure, ki jih ni", "Raznašalka pizz", "Koliko je ura?", "Kodiranje", "Bobrovsko štetje", "Zapisovanje slike", "Številke stanovanj", "Zapis znakov", "Preverjanje sodosti", "Slika iz pik", "Cocsozšla tigsa", "Golobi z dolgim nosom", "Rdečezeleno-modra", "Kratka imena datotek", "Opisovanje filmov", "Spletne oznake"

- Predstavitev podatkov - "Bim, bam", "Pleskar", "Prijatelji na omrežju", "Pavline ploščice", "Pogrinjki", "Drevo živali", "Drevo iz oklepajev", "Seznam stanovalcev", "Račun in drevo", "Družinsko drevo", "Morsejeva abeceda", "Klobuki", "Kolesa", "Skladi krožnikov", "Strategija"

- Postopkovne naloge - "Parkiranje", "Žabin sprehod", "Parkirna hiša", "Pomešane karte", "Stroj za prestavljanje krožnikov", "Pot do cveta", "Ugašanje", "Risanje cvetov", "Skladišče hlodov", "Kje je Franci", "Bobri predejo mrežo", "Po puščicah", "Smejkomat", "Dvoštevila", "Rože rastejo", "Ujemi barvo", "Labirinti", "Zmajeva krivulja", "Zaporedja števil", "Slika iz štampiljk", "Črkovna solata", "Bager", "Levo!", "Pirhi", "Prometni zamašek", "Premikanje vagonov", "Zmeda na postaji", "Zmeda na postaji (2)", "Preskakovanje"

- Algoritmi - "Postopek urejanja", "Preurejanje", "Urejevalni mostovi", "Urejanje kart", "Premetavanje žog", "Učinkovita čebela", "Najsladkejša pot", "Trajekt", "Nosač hlodov", "Veliki, srednji in mali čolni", "Ugibanje števila"

- Algoritmi na grafih - "Ben se vrača", "Telefonska mreža", "Čudni poštar", "Obiskovalni red", "Bobri in vidre", "Ne vrag, vrtičkar bo mejak", "Razvozlavanje", "E-Ceferin", "Cestne svetilke", "Lov na avtobuse", "Mostovje", "Ne pokažem!"

- Regularni izrazi in avtomati - "Opis imena datoteke", "Simbolične majice", "Zbiralec črk", "Uporabniška imena", "Preverjanje gesel", "Aibi"

- Matematika in logika - "Vodna logika (preprostejša)", "Vodna logika", "Ponarejeni kovanec", "Lažnivi jazbec", "Kdo-je-zavri", "Nagrada", "Ugani lik", "Sprememba", "Kvadratne čokolade", "Piskajoči bobri", "Prepogibanje", "Vodovodna napeljava", "Ugani barvo", "Ugani simbol", "Okraski", "Železnica", "Kdo laja?", "Virus"

- Ostalo - "P, E, P, E, L, K, A", "Povezava", "Velike veverice", "Sedem mest", "Nevarna gesla", "Nagrada po pošti", "Zaljubljeni Matjaž"

Osredotočili smo se na prvih šest kategorij, saj so neposredno povezane z računalništvom. V kategorijo *Ostalo* sodijo naloge, ki preverjajo znanje uporabe računalnika, *Matematične in logične* naloge smo izpustili, ker bolj sodijo na tekmovanja v matematiki ali logiki kot na tekmovanja v računalništvu. Vsako od kategorij smo potem razdelili na poglavja, ki opisujejo posamezen algoritem, podatkovno strukturo ali tip naloge. V vsakem poglavju je opisano, kako se rešuje problem, reševanje problema pa je prikazano tudi na nalogi s tekmovanja Bober.

Poglavje 2

Kodiranje in kriptografija

Kako računalnik shranjuje številke? Kako črke? In nazadnje, kako slike? V tem poglavju bomo opisali, kako pretvarjamo v dvojiški številski sistem in nazaj. Opisali bomo, kako v računalnik zapišemo črko in besedo ter kako učinkovito skrajšamo zapis besedila s pomočjo Huffmanovega kodiranja. Pokazali bomo tudi, kako računalnik vidi slike in kako jih stisne, da ne zavzamejo preveč prostora. S pomočjo parnostnih bitov pa bomo pokazali, kako shraniti ali poslati podatke tako, da bo naslovnik vedno vedel ali je dobil pravilne podatke ali so se ti med prenosom pokvarili. V drugem delu poglavja bomo pogledali, kako zakriptirati sporočilo, da ga med pošiljanjem preko interneta ne preberejo nepridipravi.

2.1 Dvojiški številski sistem

Za zapis števil v dvojiškem ali binarnem številskem sistemu potrebujemo dve številki: 0 in 1, oziroma dve stanji: pravilno/narobe, črno/belo ali kaj podobnega.

Pretvarjanje števil iz desetiškega sistema v dvojiškega in nazaj je enostavno.

V dvojiški sistem najlažje pretvarjamo tako, da med števili, ki so večkratniki števila 2^n (*1, 2, 4, 8, 16, 32,...*), poiščemo največje število, ki je še manjše

ali enako številu, ki ga pretvarjamo. Dobljeno število potem odštejemo od števila, ki ga želimo pretvoriti. Z ostankom pri odštevanju ponovimo postopek, kar ponavljamo, dokler ni ostanek 0. Pri vsakem od večkratnikov, ki smo ga uporabili, napišemo zraven števko 1, pri večkratnikih, ki jih nismo uporabili, pa napišemo števko 0.

Pri pretvarjanju iz dvojiškega sistema nazaj v desetiškega gre zgolj za seštevanje. Nad binarno število napišemo večkratnike števila 2^n. Števila, pod katerimi je številka ena, med seboj seštejemo in tako dobimo pretvorjeno število v desetiški sistem.

V računalništvu je binarni sistem zelo pomemben, saj z njegovo pomočjo zapisujemo podatke v računalnik in z njimi potem računamo. Uporablja se tudi za zapisovanje podatkov na CD, na katerega računalnik zapisuje tako, da v ploščico dela majhne luknjice na mestih kjer mora biti 1, mesta, na katerih mora biti 0, pa pusti cela. Binarni sistem se uporablja tudi pri pošiljanju podatkov iz enega računalnika drugemu, le da se tam pošlje višja ali pa nižja napetost, pri novejših, optičnih omrežjih, pa laser pošilja svetlobo.

2.1.1 Postopek

Primer pretvarjanja števil bomo prikazali s pretvarjanjem desetiškega števila 21 v dvojiški številski sistem ter potem nazaj v desetiškega.

Postopek pretvarjanja iz desetiškega v dvojiški sistem prikazuje slika 2.1.

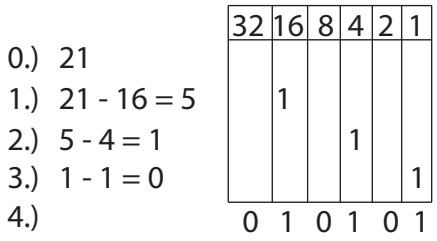
0. Na začetku imamo na levi število 21 v desetiškem sistemu na desni pa prazen prostor za vpisovanje enic in ničel, ki bodo predstavljale število v dvojiškem sistemu.

1. Na desni strani poiščemo največje število, ki je še manjše od 21; v tem primeru je to število 16. V stolpcu števila 16 to zabeležimo tako, da vanj vpišemo številko 1. Število 16 odštejemo od števila 21, ostane 5.

2. Zopet na desni poiščemo primerno število, v tem primeru je to 4, ki ga odštejemo od števila 5 in dobimo 1. V stolpec števila 4 vpišemo številko 1.

3. Od števila 1 je manjše ali enako le število 1, zato od ena odštejemo 1 in dobimo 0. V stolpec števila 1 zapišemo številko 1.

4. Na koncu v spodnjo vrstico, na mesta, kjer so v stolpcih enice, prepišemo enice, na mesta, ki so ostala prazna, pa napišemo ničle. Tako dobimo binarno število 010101, ki predstavlja desetiško število 21.



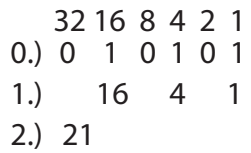
Slika 2.1: Primer pretvarjanja desetiškega števila 21 v dvojiško število.

Postopek pretvarjanja iz dvojiškega sistema nazaj v desetiški sistem prikazuje slika 2.2.

0. Na začetku imamo dvojiško število 010101, nad njim pa napisane večkratnike števila 2^n.

1. V prvo vrstico prepišemo vsa desetiška števila pod katerimi je številka 1.

2. Števila v prvi vrstici seštejemo in dobimo desetiško število 21, ki predstavlja binarno številko 010101.



Slika 2.2: Primer pretvarjanja dvojiškega števila 010101 v desetiško število.

2.1.2 Primer naloge s tekmovanja Bober

Slika 2.3: Naloga "Koliko je ura" s tekmovanja Bober.

Naloga je predstavljena le s sliko na kateri imamo ploščo s praznimi in polnimi polji. Vsak stolpec predstavlja eno število in tako lahko iz stolpca razberemo koliko je ura.

Prvi stolpec predstavlja dvojiško število 0001, kar je v desetiškem sistemu 1. Drugi stolpec predstavlja število 0010, kar je v desetiškem sistemu 2. Prva stolpca predstavljata uro, ki je dvanajst, minute pa so v naslednjih dveh stolpcih.

Tretji stolpec predstavlja število 0101, kar je v desetiškem sistemu 5, četrti stoplec pa število 0111, ki je v desetiškem 7.

Ura je tako 12:57.

Če bi želeli na uro gledati drugače, na primer po vrsticah, bi v zadnji vrstici naleteli na problem, saj je v tem primeru zadnja števka kar število enajst.

2.2 Kodiranje

2.2.1 Zapis in prenos besedil

V računalniku pa ne zapisujemo le številk, pač pa tudi črke. V računalništvu je nekaj standardov s pomočjo katerih zapisujemo črke in številke v pomnilnik. Primer takega standarda je ASCII v katerem so črke, številke ter drugi simboli, ki se uporabljajo pri zapisovanju besedil v računalnik. Po standardu ASCII črka A ni zaznamovana s številko 1, pač pa 65, mali a pa na primer 97.Standard ACSII je 7-biten, kar pomeni, da za predstavitev enega znaka potrebuje 7 bitov, številk binarnega sistema. Slika 2.4 prikazuje izvleček iz tabele ACSII, samo za velike tiskane črke.

Če bi želeli besedilo shraniti v računalniku ali poslati iz enega računalnika v drugega, moramo najprej črke pretvoriti v številke, ki jih potem pretvorimo v binarne številke in v taki obliki shranimo oz. pošljemo.

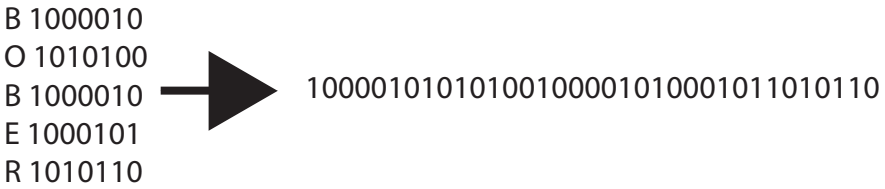
A	65	1000001	N	78	1001110
B	66	1000010	O	79	1001111
C	67	1000011	P	80	1010000
D	68	1000100	Q	81	1010001
E	69	1000101	R	82	1010010
F	70	1000110	S	83	1010011
G	71	1000111	T	84	1010100
H	72	1001000	U	85	1010101
I	73	1001001	V	86	1010110
J	74	1001010	W	87	1010111
K	75	1001011	X	88	1011000
L	76	1001100	Y	89	1011001
M	77	1001101	Z	90	1011010

Slika 2.4: Izvleček iz tabele ASCII za velike tiskane črke.

Postopek

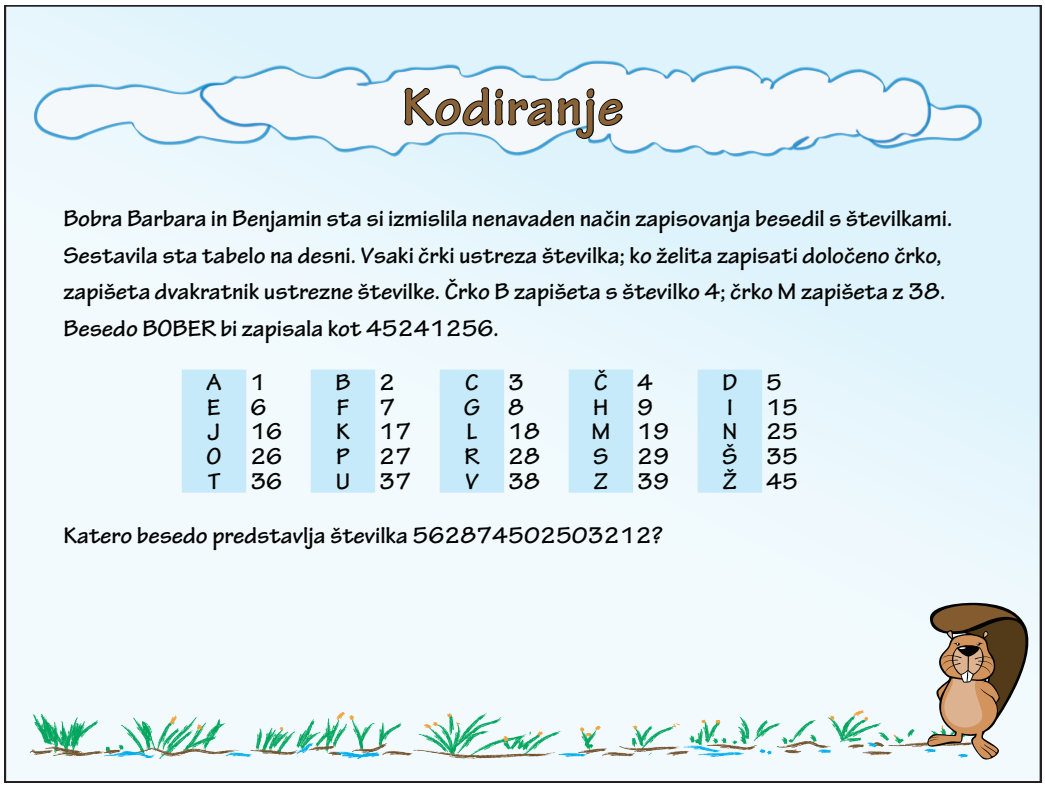
V primeru si bomo pogledali, kako lahko s pomočjo standarda ASCII zapišemo besedo.

Želeli bi napisati besedo BOBER. Vsako črko moramo posebej pretvoriti v dvojiški zapis in potem sestaviti besedo nazaj skupaj. Slika 2.5 prikazuje, kako besedo BOBER po črkah pretvorimo v binarni sistem in kako potem črke sestavimo nazaj v besedo.



Slika 2.5: Prikaz pretvarjanja besede iz črk v kodo ASCII.

Primer naloge s tekmovanja Bober



Slika 2.6: Naloga "Kodiranje" s tekmovanja Bober.

V nalogi imamo podoben primer kot prej, ko smo pretvarjali besedo v binarni sistem, le da tukaj pretvarjamo nazaj in imamo nekaj dodatnih pravil.

S pomočjo podane tabele moramo ugotoviti, katero besedo predstavlja zaporedje številk. Dodatno pravilo je, da je zaporedje številk sestavljeno tako, da je številka ob črki v tabeli pomnožena z dve. To pomeni, da bomo v kodirani besedi iskali sode številke, ki jih bomo potem delili z dve in dobili pravilno številko v tabeli.

Začnemo s številko 5, ki je liha zato vzamemo še naslednjo in tako dobimo številko 56. Ko številko 56 delimo z dve, dobimo 28, kar pomeni, da je prva črka R. Naslednja številka je 2, ki je soda in ob deljenju dobimo 1, kar pomeni

črko A. Naslednja številka je tudi soda, 8, in ob deljenju dobimo 4 oz. črko Č. Sledi številka 7, ki ni soda, zato vzamemo še naslednjo, ki je 4 in tako dobimo 74, kar ob deljenju z 2 vrne 37 ali črko U. Zopet dobimo številko 5, ki jo dopolnimo z naslednjo številko 0 in tako dobimo številko 50, ki jo delimo z 5 in dobimo 25 oz. črko N. Naslednja številka je 2, ki smo jo že videli in pomeni A, sledi pa ji 50, ki jo tudi že poznamo kot N. Za naslednjo številko, 3, tudi ugotvimo, da je liha, zato jo dopolnimo z 2, ki je takoj za njo. Ob deljenju z 2 dobimo 16 oz. črko J. Predzadnja številka je tudi liha, zato jo dopolnimo in dobimo 12. Številko 12 delimo z dve in dobimo 6 in s tem črko E.

Odkodirana beseda je RAČUNANJE.

2.2.2 Huffmanovo kodiranje

Kot smo opazili zgoraj, se besedilo, ko ga pretvorimo v binarno obliko, precej podaljša in tako zasede veliko prostora. Huffmanovo kodiranje se ukvarja s tem kako besedilo čim bolj skrajšati. David Huffman je leta 1952 opisal svojo metodo za skrajševanje besedil s katero lahko prihranimo od 20% do 90 % prostora. To mu je uspelo zato, ker je predpostavil, da se nekatere črke pojavljajo večkrat kot druge, zato je prav, da te črke zavzamejo manj prostora kot tiste, ki se pojavijo manjkrat. Vse skupaj je predstavljeno v obliki drevesa, kjer so bolj pogoste črke bližje korena, manj pogoste pa so bolj oddaljene.

Huffmanov algoritem gradnjo devesa začne na koncu in se premika proti vrhu. Vedno vzame dva najmanj pogosta znaka in jima določi vrednosti 0 in 1. Verjetnosti znakov potem združi v eno verjetnost in spet vzame dva znaka z najmanjšo verjetnostjo, ki jima spet dodeli 0 in 1. Seveda lahko ena od teh verjetnosti predstavlja tudi že združeno verjetnost iz prejšnjega koraka, tako je naprimer vrednost za osnovni znak sedaj lahko 10.

Glavni čar Huffmanovega algoritma je v tem, da nobeno zaporedje ne predstavlja začetka kakega drugega zaporedja. Tako kljub temu, da ne vemo kako dolga je koda posameznega znaka, vedno vemo kateremu znaku ustreza.

Postopek

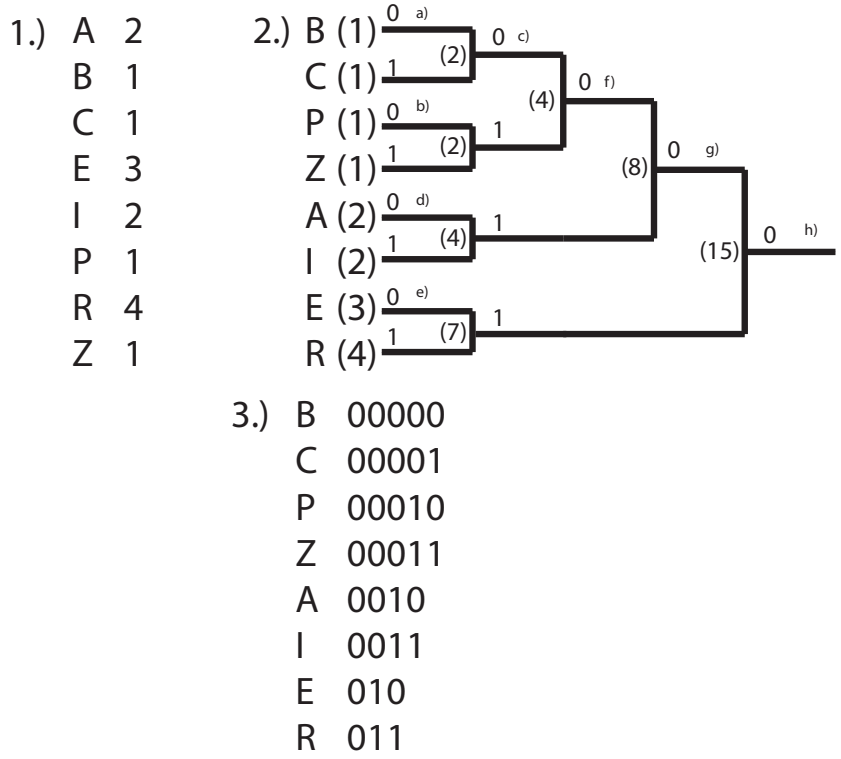
S pomočjo Huffmanovega algoritma bomo zakodirali besedilo *RIBA REŽE RACI REP*. Če bi besedilo kodirali s pomočjo standarda UTF-8, bi za vsako črko porabili 8 bitov, kar znese 120 bitov.

Postopek prikazuje slika 2.7

1. Najprej naredimo statistiko črk, ki se pojavijo v nizu.
2. Črke uredimo po številu pojavitve od najmanjše do največje, ter začnemo z združevanjem.
   (a) Izberemo dve števili z najmanjšim številom pojavitev, tako najprej združmo črki B in C. Črki B pripišemo število 0, črki C pa 1. Seštejemo število pojavitev in rezultat napišemo na črto, ki združuje obe črki.
   (b) Zopet izberemo črki z najmanjšim številom pojavitev, P in Z ter ju združimo. Črki P pripišemo število 0, črki Z pa 1. Sešetejemo število pojavitev in zapišemo.
   (c) Sedaj imamo štiri črke ali kombinacije črk, ki imajo enako število pojavitev. Najprej združimo črki BC s črkama PZ. Kombinacija BC dobi številko 0, PZ pa 1. Seštejemo pojavitve in zapišemo.
   (d) Združimo še črki A in I, A dobi številko 0, I pa 1. Seštejemo pojavitve in zapišemo.
   (e) V tem koraku združimo črki E in R, E dobi številko 0, R pa 1. Seštejemo pojavitve in dobimo 7.
   (f) Zopet izberemo kombinaciji z najmanjšima številoma pojavitev, združimo kombinacijo BCPZ s kombinacijo AI, BCPZ dobi številko 0, AI pa 1. Seštejemo pojavitve in zapišemo.
   (g) Zgornjo kombinacijo, BCPZAI, združimo še s kombinacijo ER, BCPZAI dobi 0, ER pa 1.

(h) Na koncu še napišemo poljubno številko (0 ali 1), v tem primeru smo izbrali 0.

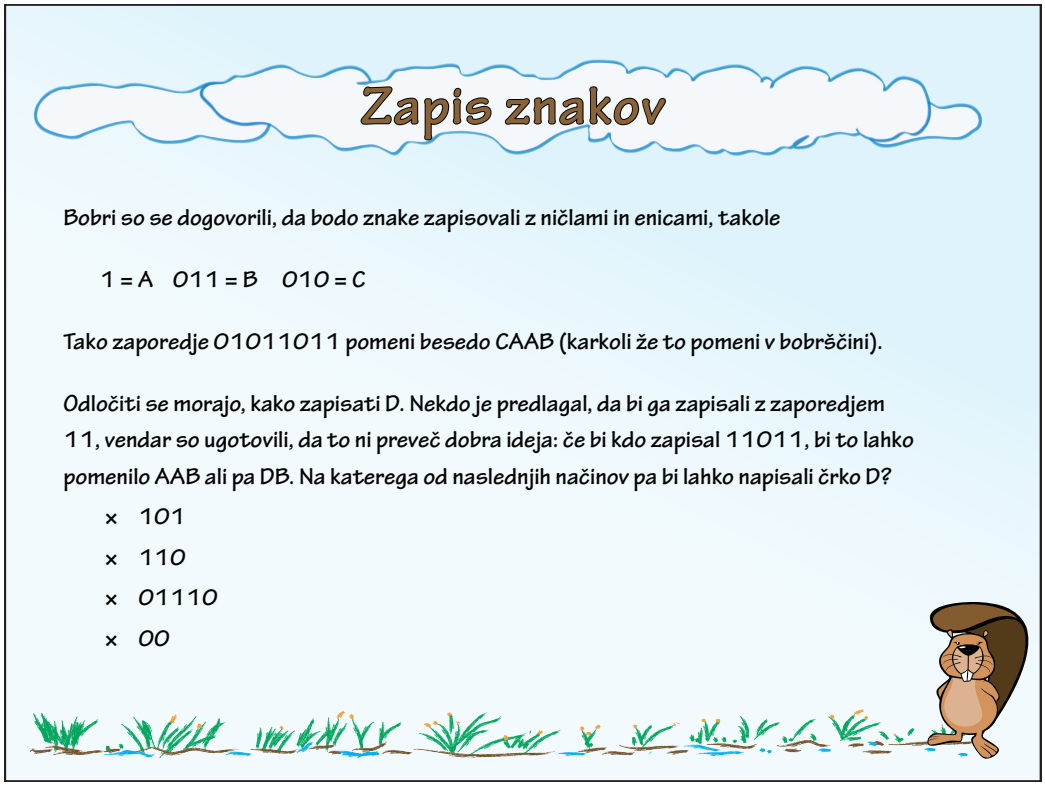
3. Kode za posamezne črke dobimo tako, da se iz korena po povezavah premikamo do njih in po vrsti zapisujemo številke, ki so na ̧vejah.



Slika 2.7: Prikaz kodiranja besedila s pomočjo Huffmanovega algoritma.

S pomočjo Huffmanovega algoritma smo dolžino kodiranega niza skrajšali na 57 bitov brez abecede. Slabost Huffmanovega kodiranja je, da je potrebno pred besedilom poslati tudi abecedo s katero smo kodirali besedilo, zato se Huffmanovo kodiranje obrestuje pri daljših besedilih.

Primer naloge s tekmovanja Bober



Slika 2.8: Naloga "Zapis znakov" s tekmovanja Bober.

V nalogi moramo ugotoviti katero izmed danih zaporedij se ne more pojaviti, če uporabljamo že podane znake.

Prvo zaporedje, 101, ne more biti uporabljeno za zapisovanje znaka D, saj se lahko pojavi pri zapisovanju zaporedja znakov AB (**101**1). Drugo zaporedje, 110, tudi ne more biti uporabljeno, saj se pojavi ob zapisovanju znakov AAB (**110**11). Tretje zaporedje, 01110, prav tako ne more biti uporabljeno. Pojavi se lahko ob zapisovanju znakov BAB (**01110**11).

Edino pravilno zaporedje je zadnje zaporedje, 00, saj se z njim ne začne nobeno drugo, že dogovorjeno zaporedje.

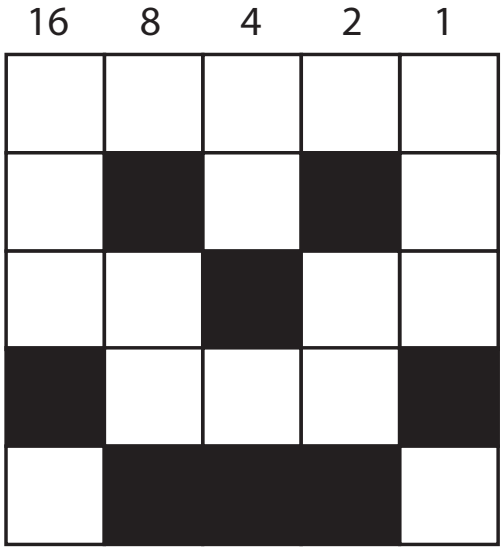
Naloga je prikaz delovanja Huffmanovega algoritma, natančneje tega, da se nobeno zaporedje ne začne z drugim zaporedjem.

2.2.3 Stiskanje slik

V prejšnjih poglavjih smo spoznali kako zapisujemo črke in številke, v tem pa bomo spoznali še kako zapisujemo slike. Slike so v računalniku shranjene kot niz kvadratkov - točk, v angliščini *pixel*. Slika, ki je shranjena v večji ločljivosti (z manjšimi kvadratki - točkami), je manj groba.

Več kot je točk, več prostora zasede slika, zato jih je potrebno stiskati. Pri črno-belih slikah je ta postopek veliko lažji kot pri barvnih, tako da bomo ostali pri tem. Postopek s katerim bomo stiskali slike se imenuje bitno kodiranje. Postopek deluje tako, da nad sliko zapišemo potence števila 2, kot to prikazuje slika 2.9. V vsaki vrstici seštejemo vrednosti števil polj, ki so črne barve in to zapišemo. Tako za vsako vrstico porabimo le eno številko.

Sliko lahko zapišemo tudi tako, da opišemo, v kakšnem zaporedju se pojavljajo črni in beli kvadratki. Ta postopek se imenuje kodiranje s stiskanjem čet, v angleščini *run length encoding (RLE)*. Vedno začnemo na začetku vrstice in preštejemo število belih kvadratkov do prvega črnega in to številko zapišemo. Potem preštejemo število črnih kvadratkov do naslednjega belega in zapišemo. Postopek ponavljamo dokler ne pridemo do konca vrstice. Pri vsaki vrstici tako dobimo zaporedje števil, ki povedo kako se barve v kvadratkih izmenjujejo ter koliko kvadratkov iste barve si sledi zapovrstjo.



Slika 2.9: Črnobela bitna slika.

Postopek

Za primer bomo vzeli kar sliko 2.9 in jo pretvorili v številke.

Najprej bomo primer rešili s pomočjo bitnega kodiranja. Začnemo s prvo vrstico, ki ima sama bela polja kar pomeni, da je njena vrednost 0. Druga vrstica ima dva črna kvadratka, eno pod številko 8, drugo pod številko 2, vrednosti seštejemo in dobimo vrednost vrstice 10. Naslednja vrstica ima en črn kvadratek pod številko 4, zato je njena vrednost 4. Četrta vrstica ima dva črna kvadratka, enega pod 16 ter drugega pod 1, ko seštejemo, dobimo 17, kar zapišemo. Zadnja vrstica je sestavljena iz treh črnih kvadratkov, ki so pod števili 8, 4 in 2. Seštevek teh števil je 14, kar tudi zapišemo. Sliko tako shranimo s petimi števili: **0, 10, 4, 17, 14.**

Sliko kodiramo še s pomočjo kodiranja s stiskanjem čet. V prvi vrstici najprej preštejemo koliko je belih kvadratkov. Belih kvadratkov je 5, črnih kvadratkov pa ni, tako da je vrednost te vrstice samo 5. Naslednja vrstica se začne z enim belim kvadratkom, ki mu sledi en črn, kateremu spet sledi

en bel, ki mu sledi še en črn, na koncu pa je še en bel kvadratek. To vrstico zato predstavimo kot 1, 1, 1, 1, 1. Tretja vrstica se začne z dvema belima kvadratkoma, ki jima sledi en črn, kateremu sledita še dva bela, zato vrstico predstavimo kot 2, 1, 2. Četrta vrstica se začne z nič belimi kvadratki, ki jim sledi en črn, kateremu sledijo trije beli, ter še en črn. Vrstico tako predstavimo kot 0, 1, 3, 1. Zadnja vrstica se začne z enim belim kvadratkom, ki mu sledijo trije črni kvadratki ter še en bel. Predstavimo jo kot 1, 3, 1. Celotno sliko lahko tako predstavimo kot:

5
1 1 1 1 1
2 1 2
0 1 3 1
1 3 1

Primer naloge s tekmovanja Bober

Slika 2.10: Naloga "Slika iz pik" s tekmovanja Bober.

Podobno, kot smo zgoraj kodirali sliko, jo moramo sedaj v tej nalogi dekodirati. Najlažje je, da si narišemo polje v velikosti 5×5, in barvamo polja, ki morajo biti črne barve.

V prvi vrstici tako pobarvamo prvo polje, potem tri polja pustimo prazna in pobarvamo še zadnje polje. Druga vrstica ima enako kodo kot prva, zato spet pobarvamo prvo in zadnje polje. Tretja vrstica je sestavljena iz 0 belih polj ter 5 črnih, kar pomeni, da pobarvamo vse kvadratke v vrstici. Četrta in peta vrstica sta enaki kot prvi dve, tako v obeh pobarvamo prvo in zadnje polje. Dobili smo črko H.

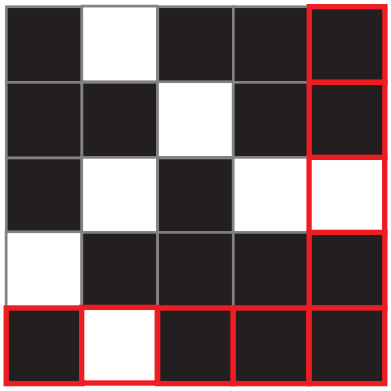
2.2.4 Popravljanje napak

Pri prenosu podatkov po omrežju ter tudi pri shranjevanju je dobro vedeti ali so podatki, ki smo jih dobili, res prispeli brez napake. V računalništvu je veliko postopkov s katerimi se preverja pravilnost prejetih podatkov. Nekateri od teh postopkov samo ugotovijo ali je prišlo do napake, drugi pa znajo napako tudi odpraviti.

Eden od postopkov s katerimi lahko odkrijemo napako je postopek preverjaja parnosti. Preverjanje deluje tako, da na koncu vsakega niza bitov, ki jih pošiljamo, dodamo tako imenovani parnostni bit. Želeli bi, da je število enic v nizu sodo, zato preštejemo vse enice v nizu. Če je število že sodo, potem na koncu niza dodamo ničlo, če je število enic liho, na koncu dodamo enico in tako dobimo sodo število enic. Ko bo prejemnik dobil naše sporočilo, bo tudi on preštel število enic; če število enic ne bo sodo, to pomeni, da se je na poti en od bitov pokvaril zato bo od nas zahteval, da mu sporočilo še enkrat pošljemo.

Če bi želeli z zgornjim postopkom tudi popravljati podatke, jih moramo poslati v mreži, kot jo prikazuje slika 2.11. Tudi v tem primeru je zadnji bit v vsaki vrstici parnostni bit. Posebnost pa je zadnja vrstica, ki vsebuje same parnostne bite in sicer glede na stolpce. Tako imamo v zadnjem stolpcu in zadnji vrstici parnostne bite, ki za vsako vrstico določajo sodost. Če se med prenosom en od bitov pokvari, lahko zelo hitro ugotovimo, kateri bit je to, saj samo pogledamo katera vrstica ter kateri stolpec nimata sodega števila enic. Koordinate, ki jih s tem dobimo, nam določajo kateri bit se je pokvaril in ga tako lahko popravimo.

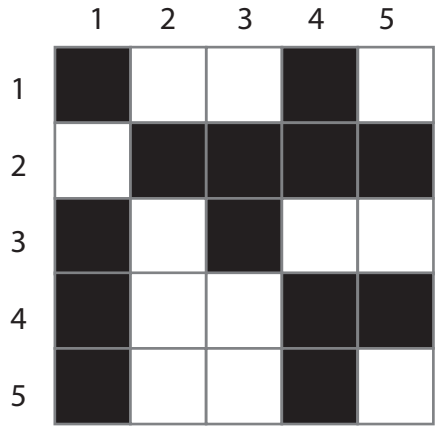
Če želimo, da je prenos podatkov odporen na večje število napak, uporabljamo druge postopke, ki pa jih tukaj na bomo predstavili.



Slika 2.11: Mreža, v kateri sta zadnji stolpec in spodnja vrstica namenjena preverjanju parnosti podatkov.

Postopek

V primeru bomo namesto enic in ničel uporabljali črne in bele kvadratke. Med pošiljanjem zakodiranega besedila prikazanega na sliki 2.12 se je en od bitov pokvaril.



Slika 2.12: Primer pokvarjene mreže.

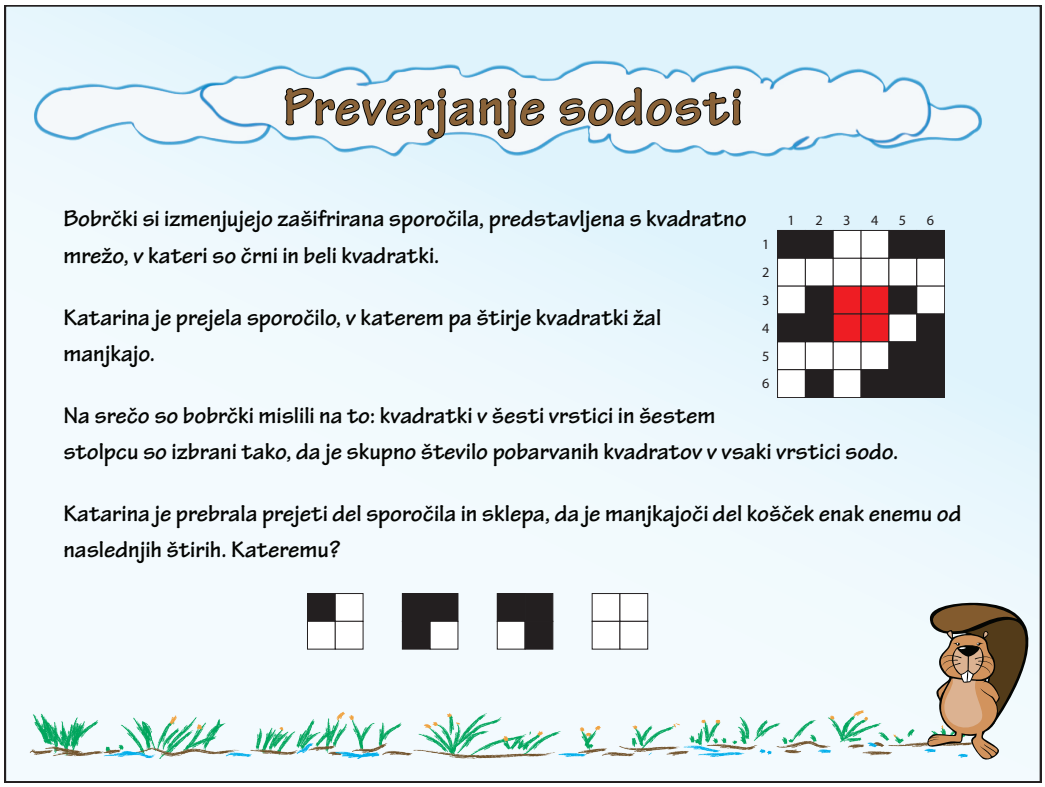
Najprej preverimo katera od vrstic nima sodega števila črnih kvadratkov.

Prva vrstica ima dva črna kvadratka, število dva je sodo, torej je ta vrstica pravilna. Druga vrstica ima štiri črne kvadratke, kar je tudi sodo število. V naslednji vrstici je tudi sodo število črnih kadratkov, dva. Četrta vrstica ima tri črne kvadratke; tri ni sodo število, kar pomeni, da je napaka v tej vrstici. Da bomo prepričani, preverimo še zadnjo vrstico, v kateri sta dva črna kvadratka.

Sledi preverjanje stolpcev. Prvi stolpec ima štiri črne kvadratke, kar je pravilno. V drugem stolpcu je le en črn kvadratek, kar nam pove, da je napaka v tem stolpcu. Preverimo še vse ostale stoplce. V tretjem stolpcu sta dva črna kvadratka, v četrtem so štirje in v petem sta tudi dva, kar pomeni, da so vsi trije stolpci pravilni.

Napaka je tako v četrti vrstici v drugem stolpcu. Napako popravimo in tako dobimo pravilno sporočilo.

Primer naloge s tekmovanja Bober



Slika 2.13: Naloga "Preverjanje sodosti" s tekmovanja Bober.

Podobno kot v prejšnjem primeru, se je tudi pri tej nalogi pokvarilo nekaj bitov. Možne rešitve so podane kot odgovori. Najlažje je, da za vsako od podanih rešitev preverimo ali je ustrezna ali ne. To preverimo tako, da v vrsticah in stolpcih, na katere vpliva rešitev, preverimo sodost.

Prva možna rešitev povzroči lihost črnih kvadratkov v tretji in četrti vrstici ter tretjem in četrtem stoplcu. Druga rešitev je pravilna, saj zagotovi sodost v celotni tabeli; v tretji in četrti vrstici so tako štirje črni kvadratki, v tretjem in četrtem stolpcu pa po dva črna kvadratka. Tretja rešitev sicer poskrbi za sodost v četrtem stoplcu ter tretji in četrti vrstici, vendar je v tretjem stolpcu le en črn kvadratek. Zadnja rešitev pusti mrežo tako, kot

smo jo dobili, brez sodosti v četrtem stolpcu ter v četrti vrstici.

2.3 Kriptografija

Kriptografija je veda, ki govori o varnem komuniciranju oz. kako zakriti sporočilo, da ga lahko prebere le naslovnik. Sporočilo, ki ga pošiljamo, ponavadi imenujemo čistopis, kriptirano besedilo pa tajnopis. Sporočilo z nekim postopkom zakriptiramo, prejemnik pa ga odkriptira.

Predstavili bomo dva postopka za kriptiranje sporočil:

- **Cezarjeva (pomična) šifra** deluje tako, da vsako črko besedila zamenjamo s črko, ki je v abecedi nekaj mest za njo.
- **Permutacijska šifra** pa deluje tako, da črke med seboj premešamo na nek sistematičen način.

Pri obeh šifrah mora naslovnik vedeti, na kakšen način je bilo sporočilo zakriptirano, da ga lahko potem odkriptira. To imenujemo ključ. Pri Cezarjevi šifri je ključ številka, ki pove, za koliko so črke zamaknjene, pri transpozicijski šifri pa je to zaporedje števil s katerim so bile številke premešane.

2.3.1 Postopek

S pomočjo Cezarjeve in permutacijske šifre bomo zakriptirali stavek: *DANES JE VROČE*.

Pri Cezarjevi šifri bomo za ključ izbrali številko 7, kar pomeni, da bomo vsako črko zamaknili za sedem mest v desno. Tako bo na primer črka A postala črka G. Postopek šifriranja je prikazan na sliki 2.14. Najprej pretvorimo črko D v črko K, črko A v črko G, črko N v črko U, črko E v črko L ter črko S v črko A. Tako iz besede *DANES* dobimo besedo *KGULA*. Druga beseda je *JE*; ko jo pretvorimo, dobimo *RL*. Iz zadnje besede *VROČE* dobimo *DŽVLJ*. Stavek se tako glasi: *KGULA RL DŽVJL*.

Slika 2.14: Primer kodiranja besedila s Cezarjevo šifro.

Permutacijska šifra od nas zahteva, da si izmislimo zaporedje s pomočjo katerega bomo premešali črke. Izbrali smo si zaporedje 4 2 1 3. Naš stavek ima 12 črk, zato bomo morali zaporedje uporabiti trikrat. Stavek zapišemo v eno vrstico, nad stavek pa napišemo zaporedje s katerim ga želimo zakriptirati, kot to prikazuje slika 2.15. Številka nad črto nam pove, na katero mesto jo moramo v šifriranem besedilu prestaviti. Tako črko N s tretjega mesta prestavimo na prvo, črko A pustimo na drugem mestu. Črko E s četrtega prestavimo na tretje mesto, ter črko D s prvega na četrto. To ponovimo še za ostale črke in dobimo stavek: *NAEDE JV SČEOR*.

Slika 2.15: Primer kodiranja besedila s transpozicijsko šifro.

2.3.2 Primer naloge s tekmovanja Bober

Slika 2.16: Naloga "Golobi z dolgim nosom" s tekmovanja Bober.

Na problem pri tej nalogi lahko gledamo kot na posebno vrsto permutacijske šifre. Šifra je tukaj dolga toliko, kot je dolgo besedilo in besedilo obrne okoli. Če besedilo beremo od zadaj naprej, dobimo besedilo, ki ga je bober Matjaž poslal Alenki. Prav tako lahko preberemo Alenkino sporočilo Matjažu, napisala mu je: Hvala za kartico Alenka.

Poglavje 3

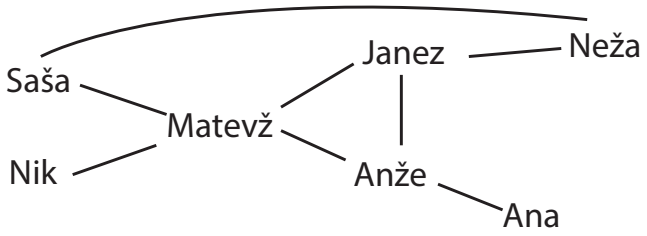
Predstavitev podatkov

V tem poglavju bomo pogledali kako se s pomočjo grafa lahko predstavi zemljevid, ki prikazuje ceste v neki državi ali kaj podobnega. Spoznali bomo posebne vrste grafov, drevesa, s pomočjo katerih lahko predstavljamo relacije med družinskimi člani, ter odločitvena drevesa, ki nam lahko pomagajo pri odločanju kako se zjutraj obleči. Za konec si bomo pogledali še eno najbolj običajnih stvari v življenju, le da ne vemo, da se jim tako reče, sklade, s pomočjo katerih zlagamo krožnike, skodelice ali obleke.

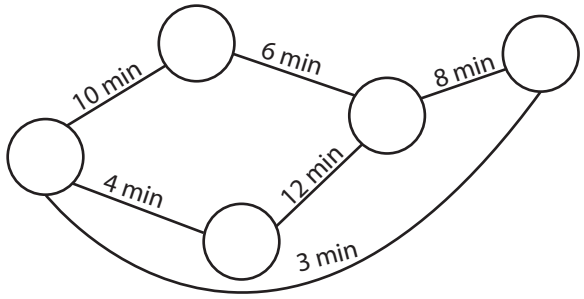
3.1 Predstavitev relacij z grafom

Grafi so matematične strukture, sestavljene iz točk in povezav med temi točkami. Z grafi lahko predstavimo najrazličnejše probleme, od zemljevidov, telefonskih povezav pa vse do povezav med osebami ali pa povezav mestnega prometa.

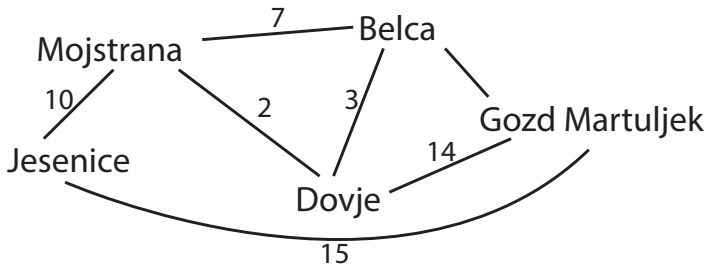
Točke ponavadi ustrezajo objektom, zato so poimenovane po njih, kot je to prikazano na sliki 3.1. Vrednost lahko dodelimo tudi povezavam med točkami. Povezave predstavljajo razadaljo med dvema točkama, ali pa čas potovanja od ene točke do druge, kar je prikazano na sliki 3.2. Seveda lahko tudi hkrati poimenujemo točke in povezave ter tako dobimo prave zemljevide z razdaljami med kraji, kot to prikazuje slika 3.3.



Slika 3.1: Prikaz povezav med osebami s pomočjo grafa.

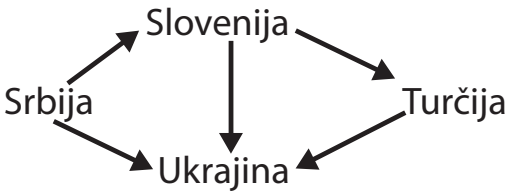


Slika 3.2: Prikaz povezav med točkami na grafu z razdaljami, napisanimi na povezavah.

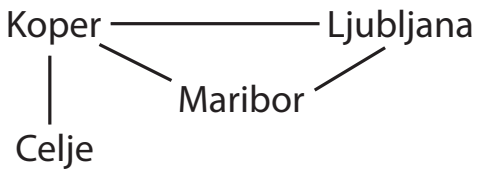


Slika 3.3: Prikaz povezav med točkami na grafu z razdaljami, napisanimi na povezavah.

Poznamo usmerjene in neusmerjene grafe. Usmerjeni grafi, na primer graf na sliki 3.4, so grafi, v katerih je smer povezave med točkama določena (v košarki je Srbija premagala Slovenijo, Slovenija pa Srbije ni premagala, je pa premagala Turčijo in Ukrajino). Neusmerjeni grafi 3.5 pa so grafi, v katerih povezave nimajo v naprej določene smeri in lahko potujemo v obe smeri (z avtobusom se lahko peljemo iz Kopra v Ljubljano, prav tako pa se lahko peljemo tudi iz Ljubljane v Koper).

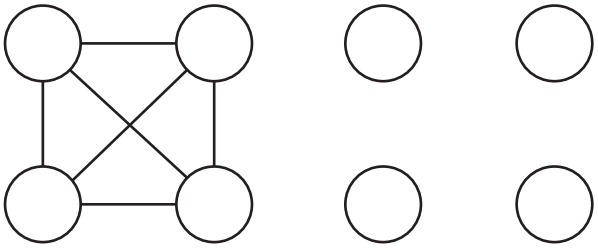


Slika 3.4: Primer usmerjenega grafa.



Slika 3.5: Primer neusmerjenega grafa.

Graf je poln, če so vse točke povezane z vsemi točkami, ter prazen, če ni povezav med točkami. Na sliki 3.6 je levo prikazan poln graf, desno pa prazen.



Slika 3.6: Primer polnega grafa, na levi, ter praznega grafa, na desni.

3.1.1 Primer naloge s tekmovanja Bober

Slika 3.7: Naloga "Prijatelji na omrežju" s tekmovanja Bober.

V nalogi so opisane relacije med bobrčki. Najlažje jo rešujemo tako, da sami narišemo graf, za katerega veljajo pravila, potem pa ta graf poiščemo med danimi grafi.

To najlažje naredimo tako, kot je to prikazano na sliki 3.8:

1. Od Miha narišemo povezavo do Lane, Janeza in Patrika.

2. Janeza povežemo z njegovima prijateljema, Mihom in Ano. Janez in Miha sta že povezana, saj je Miha tudi Janezov prijatelj.

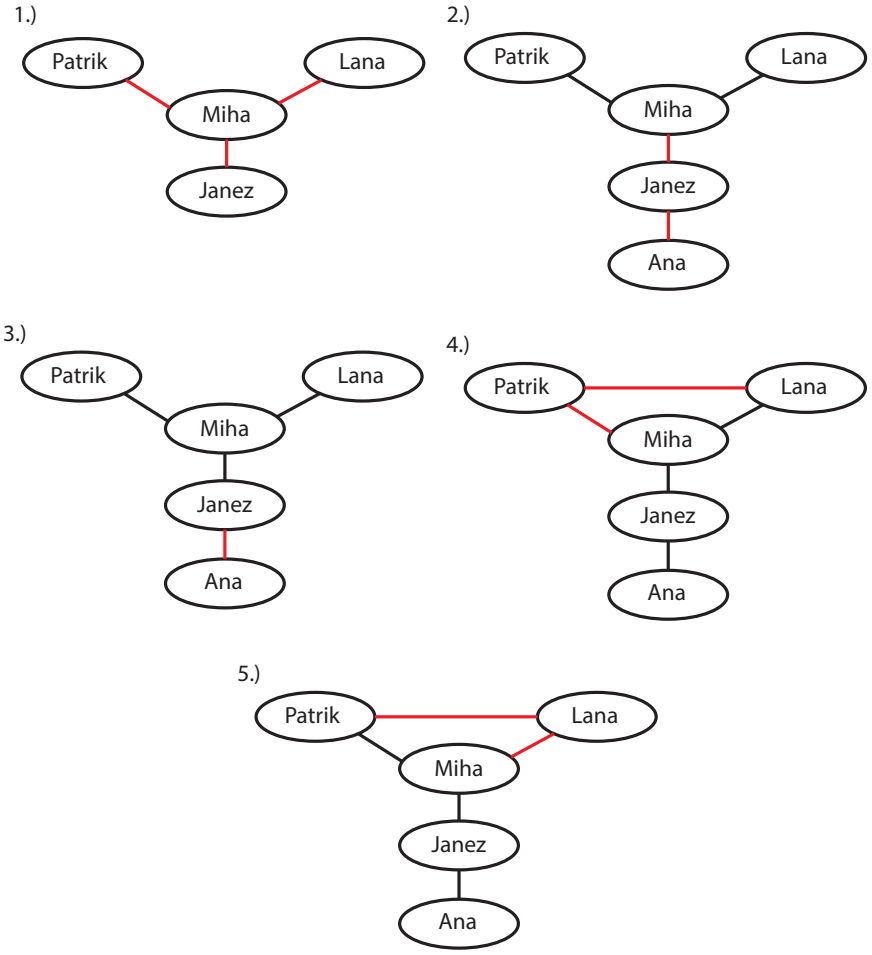
3. Janez je že povezan z Ano, tako da nove povezave tukaj ne potrebujemo.

4. Patrik je že povezan z Mihom, tako ga povežemo le še z Lano.

5. Lana je povezana še z Mihom in Patrikom, vendar sta tudi onadva že povezana z njo, tako da so povezave že narisane.

   Sedaj moramo le še poiskati graf, ki izgleda tako kot naš. Tak graf je na sliki z nalogo čisto spodaj.

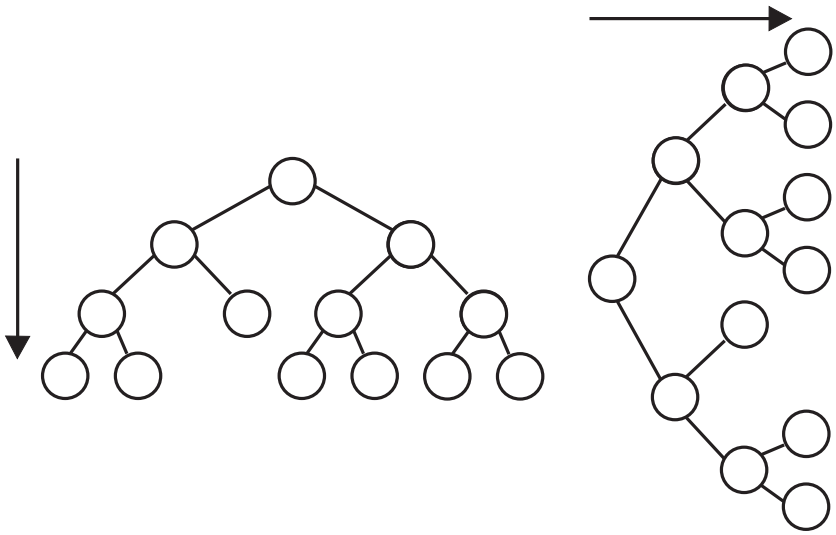
   Naloge se lahko lotimo tudi na drugačen način. Ugotovimo, da sta v desnem in zgornjem grafu dve točki s štirimi prijatelji, v podatkih pa noben od bobrčkov nima več kot treh prijateljev, kar pomeni da ta dva grafa nista pravilna. Ugotovimo tudi, da je Ana tista, ki ima le enega prijatelja, Janeza. To pomeni, da osamljena točka na grafu predstavlja Ano, ki je povezana z Janezom. V podatkih vidimo, da ima Janez le dva prijatelja, na levi sliki pa so pri točki, ki bi morala predstavljati Janeza, prikazane tri povezave. Tako nam ostane le še spodnja slika, ki je tudi rešitev te naloge.



Slika 3.8: Postopek rešvanja naloge "Družinsko drevo" s tekmovanja Bober. Z rdečo barvo so označene povezave, ki nastanejo v posameznem koraku.

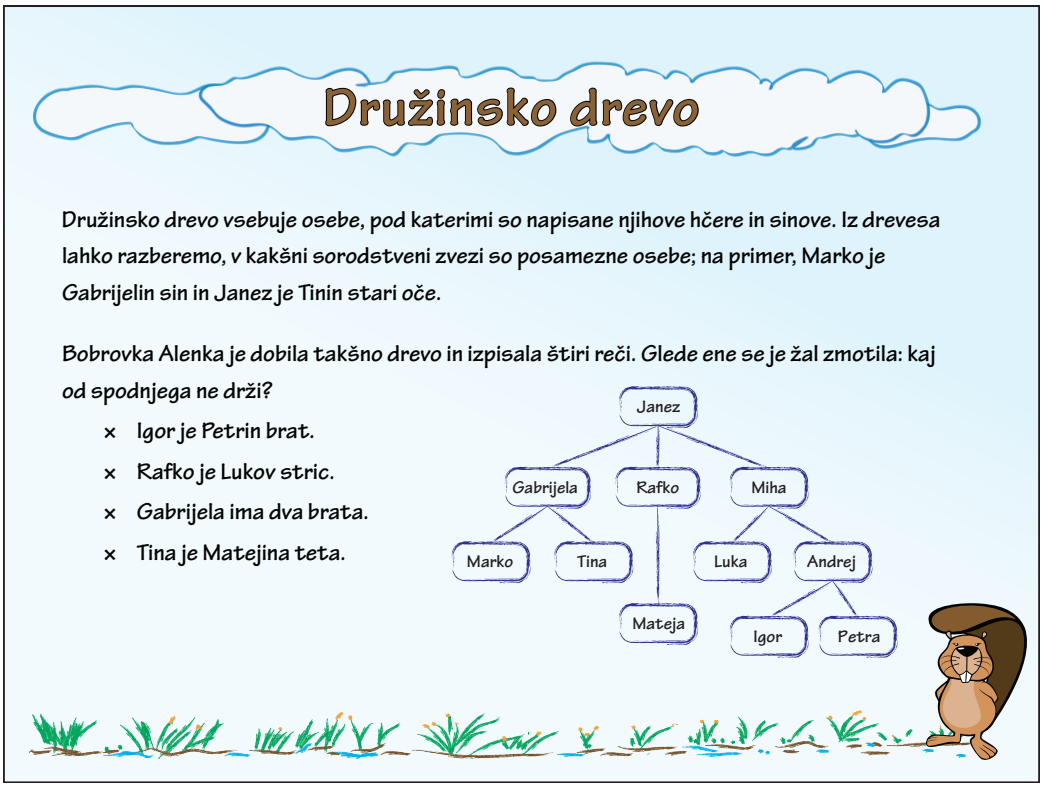
3.2 Predstavitev podatkov z drevesom

Drevesa so posebna vrsta grafov. Drevo je usmerjen graf, v katerem v vsako točko vodi le ena povezava, iz točke pa poljubno število povezav. Izjema je le koren grafa, v katerem je izhodišče drevesa. Koren grafa nima vhodnih povezav. Kljub temu, da so povezave pri drevesih usmerjene, jih ponavadi ne rišemo s puščicami. Drevo narišemo od zgoraj navzdol ali iz leve proti desni, kot to prikazuje slika 3.9, in s tem določimo, v katero smer potekajo povezave (dol oz. desno).



Slika 3.9: Prikaz drevesa. Levo drevo poteka od zgoraj navzdol, desno pa z leve proti desni.

3.2.1 Primer naloge s tekmovanja Bober



Slika 3.10: Naloga "Družinsko drevo" s tekmovanja Bober.

Drevesa so zelo uporabna tudi za predstavitve družinskih relacij, kjer vsak nivo nižje predstavlja drugo generacijo. Pri tej nalogi so predstavljene štiri generacije; tako je Janez Igorjev in Petrin pradedek.

Nalogo rešujemo tako, da preverimo podane relacije eno za drugo. Igor je Petrin brat, saj imata oba istega očeta, Andreja. Rafko je Lukov stric, ker je Miha Lukov oče, hkrati pa je Miha tudi Rafkov brat. Gabrijela ima dva brata, Rafka in Miha, njihov oče pa je Janez. Tina ni Matejina teta, pač pa je Matejina sestrična, saj je Tinina mama Gabrijela, ki je sestra Matejinega očeta, Rafka.

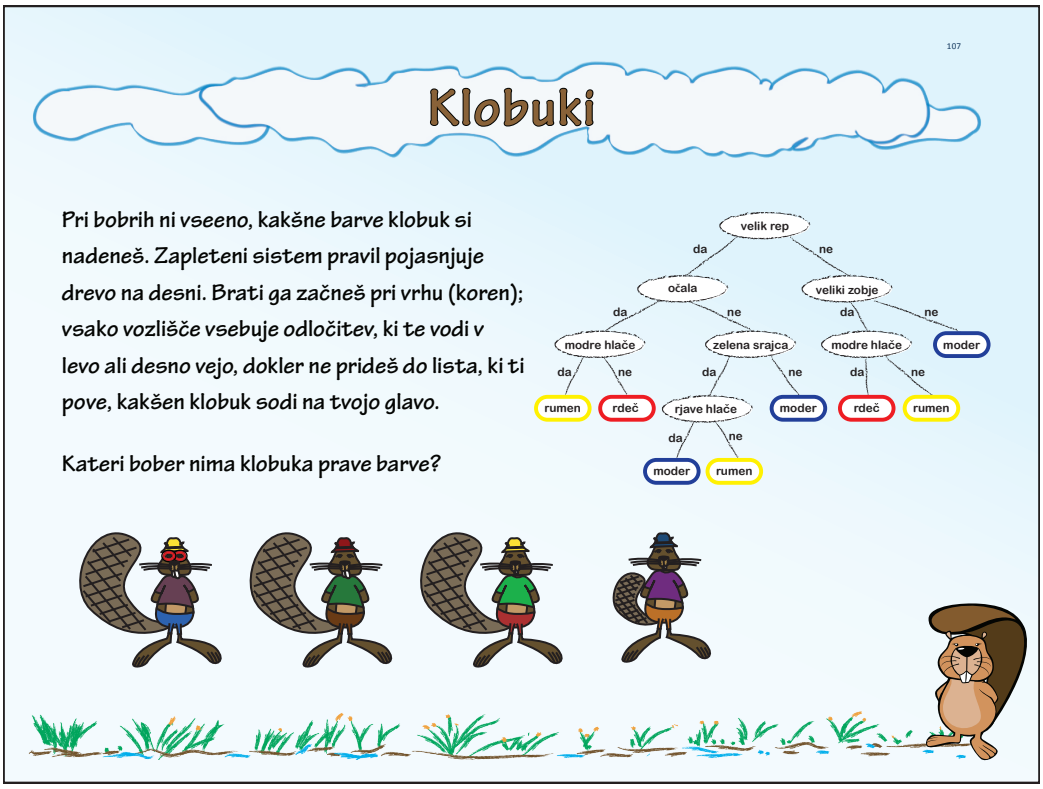
3.3 Odločitvena drevesa

Odločitvena drevesa, v angleščini *Decision Tree*, so posebne oblike grafov oz. dreves in nudijo podporo pri odločanju. Z odločitvenimi drevesi si računalnikarji pomagamo tudi, ko rišemo diagrame poteka za različne algoritme.

Z odločitvenimi drevesi si lahko pomagamo moški, ki nimamo smisla za oblačenje. Vse, kar potrebujemo, je nekdo s smislom za oblačenje, verjetno oseba ženskega spola, ki za vsa naša oblačila naredi odločitveno drevo, ki pove, kaj sodi skupaj in kaj ne.

Posebna vrsta odločitvenih dreves so binarna odločitvena drevesa. To so drevesa, kjer se na vsaki točki lahko odločimo le med dvema različnima možnostima, na primer levo ali desno.

3.3.1 Primer naloge s tekmovanja Bober



Slika 3.11: Naloga "Klobuki" s tekmovanja Bober.

Na sliki je predstavljeno binarno odločitveno drevo, ki bobrom pomaga, da pravilno izberejo barvo klobuka.

Nalogo rešujemo tako, da za vsakega bobra posebej preverimo, ali nosi pravilen klobuk. To naredimo tako, da se sprehodimo skozi drevo in na koncu vidimo, ali ima bober na glavi pravilno barvo klobuka.

Začeli bomo z bobrom na levi. Ta bober ima velik rep, tako gremo v levo polovico drevesa. Nosi očala, kar pomeni, da gremo spet v levo polovico. Ker ima modre hlače, to pomeni, da mora imeti rumen klobuk. Ta bober ima pravo barvo klobuka.

Drugi bober ima prav tako velik rep, nima pa očal. Naslednje vprašanje

je, ali ima zeleno srajco, ker je odgovor ne, bi moral imeti moder klobuk, vendar ima rjavega. Ta bober nima pravilno izbrane barve klobuka.

Naslednji bober ima prav tako kot prejšnji velik rep in ne nosi očal. Tako pridemo do vprašanja ali ima zeleno srajco. Ker ima zeleno srajco, preverimo še, ali ima rjave hlače. Ker nima rjavih hlač, mora biti njegov klobuk rumene barve. Barva klobuka pri tem bobru je pravilna.

Zadnji bober nima velikega repa, zato gremo v desni del drevesa. Ker tudi nima velikih zob, mora biti njegov klobuk modre barve. Zadnji bober je izbral pravilno barvo klobuka.

3.4 Skladi

Kadar pospravljamo pomivalni stroj in krožnike zlagamo nazaj na police omare, nevede uporabljamo sklad, v angleščini *stack*.

Za trenutek si kuhinjsko polico predstavljajmo kot sklad. Ko krožnike zlagamo na police, lahko krožnik položimo le na vrh kupa krožnikov. Prav tako tudi, ko pripravljamo mizo, lahko s police vzamemo le zgornji krožnik in ne poljubnega iz sredine.

Ravno tako deluje sklad pri računalnikih, le da ne zlagamo krožnikov, pač pa številke ali kaj drugega.

3.4.1 Primer naloge s tekmovanja Bober

Slika 3.12: Naloga "Skladi krožnikov" s tekmovanja Bober.

V navodilu je napisano katere krožnike dobijo veliki bobri (rdeče) in katere mali bobri (zelene). V nalogi je potrebno preveriti pravilnost postavitve krožnikov. Krožniki so naloženi na sklad, kar pomeni, da lahko vsak, ki pride, vzame krožnik le z vrha kupa.

Pri prvem možnem odogovoru bo najprej prišel velik bober in vzel rdeč krožnik, potem bosta prišla dva mala bobra in vzela zelena krožnika. Sledila bosta velik bober, ki bo vzel rdeč krožnik in mali bober, ki bo vzel zelenega. Na koncu pridejo še trije, dva velika, ki bosta vzela rdeča in en mali, ki bo vzel zelenega. Ta sklad krožnikov je pravilno naložen.

Pri drugem odgovoru bosta najprej prišla dva velika bobra, ki bosta vzela

rdeča krožnika, sledil bo mali bober, ki bo vzel zelenega. Nadaljevala bosta dva velika bobra, ki bosta vzela rdeča krožnika in dva mala bobra, ki bosta vzela manjša, zelena krožnika. Tudi ta sklad krožnikov je pravilno naložen.

Pri tretjem odgovoru bodo bobri prihajali izmenično. Najprej pride velik bober in vzame rdeč krožnik, potem pride mali bober in vzame zelen krožnik. To se ponovi trikrat in vedno se izide, kar pomeni, da je sklad krožnikov pravilno naložen.

Pri zadnjem odgovoru najprej pride mali bober in vzame zelen krožnik. Sledi velik bober, kateremu pripada rdeči krožnik vendar je na voljo le zelen, kar pomeni, da je ta sklad krožnikov sestavljen napačno.

Poglavje 4

Postopkovne naloge

V tem poglavju bomo pregledali nekaj nalog pri katerih morajo tekmovalci bodisi določiti algoritem, bodisi je ta določen z nalogo. Algoritmi niso splošni, temveč so vezani na konkreten problem.

Tipi nalog so razdeljeni po Bloomovi taksonomiji za kognitivno področje[22, 15]:

- **Poznavanje, sledenje opisu** - nekaj nalog zahteva bolj ali manj trivialno sledenje opis,u podanemu v nalogi. Glavna zahtevnost naloge je, da si moramo predstavljati vmesno stanje po vsakem koraku. Pri teh nalogah je pomembno, da natančno sledimo postopku, ki je dan z navodili. Primer je naloga *Parkiranje*, ki je razložena med primeri nalog s tekmovanja Bober v podpoglavju tega poglavja 4.1.1.

- **Razumevanje** - v ta sklop sodijo naloge, pri katerih ukazi niso več trivialni in zahtevajo razumevanje opisa, podanega v nalogi. Naloge so zahtevne tudi zaradi razumevanja uporabljene notacije. Primer s tekmovanja Bober je naloga *Smejkomat*.

- **Uporaba in razumevanje algoritma** - v opisu naloge ni več podano zaporednje ukazov, pač pa je opisan postopek (algoritem), ki ga je potrebno razumeti, da lahko uspešno rešimo nalogo. Na tekmovanju Bober je bil primer take naloge *Zaporedja števil*, kjer je potrebno

ugotoviti, kako deluje zaporedje, če želimo rešiti nalogo.

- **Analiza** - naloga *Prometni zamašek* s tekmovanja Bober zahteva od nas naprednejše sledenje podanemu algoritmu. Razumeti moramo kako algoritem deluje, kakšna so pravila in to znanje uporabiti pri iskanju najbolj optimalne rešitve problema.

Z ostalimi višjimi nivoji Bloomove taksonomije se nismo ukvarjali, saj v nalogah niso izraziteje zastopani.

4.1 Primeri nalog s tekmovanja Bober

4.1.1 Parkiranje

Slika 4.1: Naloga "Parkiranje" s tekmovanja Bober.

Nalogo najlažje rešujemo tako, da za vsak odgovor posebej preverimo, ali nas pripelje do cilja.

Začnemo s prvim, ki od nas zahteva, da gremo najprej *naprej* ter potem *levo*. Ker smo še vedno na cesti, lahko vozimo naprej, kot zahtevajo navodila, tako gremo *naprej* in *levo*. Z zavojem v levo smo zavili s ceste, kar pomeni da ta odgovor ni pravilen.

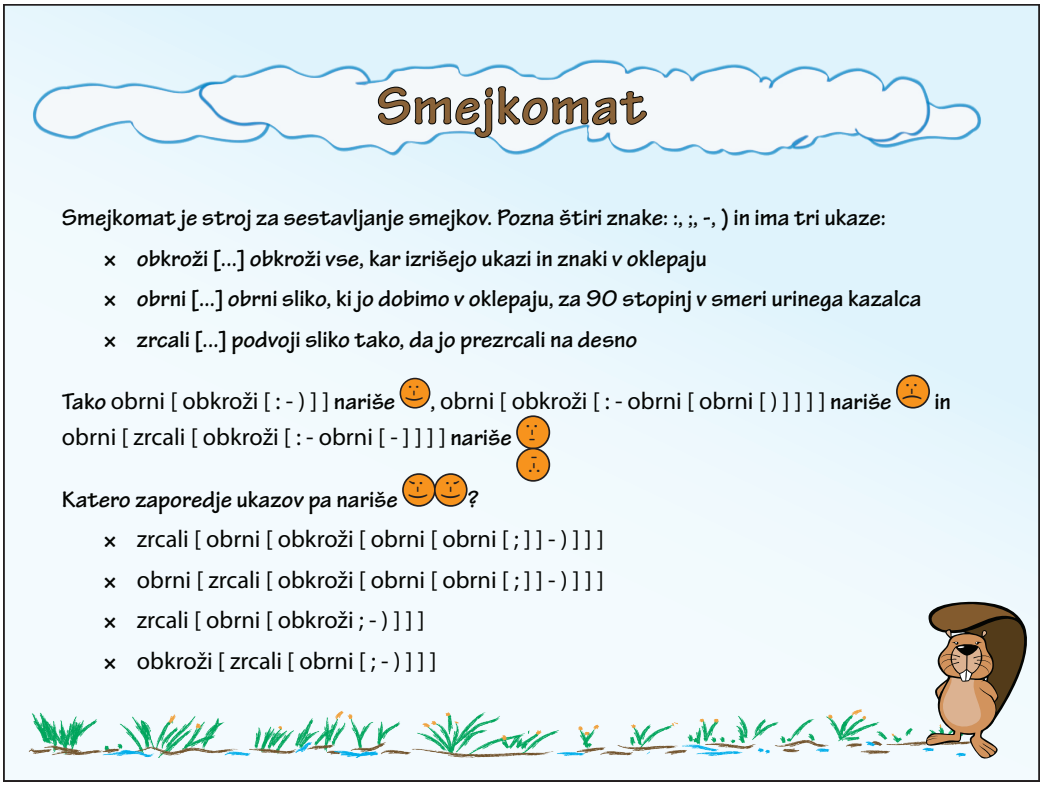
Drugo zaporedje navodil se prav tako začne z *naprej* in *levo*, za kar vemo, da je pravilno, nadaljuje pa se z *naprej* in *desno*. Še vedno smo na poti,

zato nadaljujemo in gremo *naprej* in *levo*, kar je tudi še pravilno. Naslednja ukaza sta *naprej* in *levo*. Zadnji zavoj v levo nas zapelje s ceste, kar pomeni, da navodila v tem odgovoru niso pravilna.

Tretje zaporedje se začne tako kot drugo, z *naprej, levo, naprej, desno, naprej, levo* in *naprej*, vendar naslednji zavoj ni v levo, pač pa v desno. Tako še vedno ostanemo na cesti, zato preberemo že zadnje navodilo, ki je *naprej*. Navodila pri tretjem odgovoru so pravilna.

Zadnje zaporedje se začne z navodilom *levo* in tako že na začetku zgrešimo cesto, kar pomeni, da zaporedje navodil ni pravilno.

4.1.2 Smejkomat



Slika 4.2: Naloga "Smejkomat" s tekmovanja Bober.

Naloga opisuje, kaj naredi posamezen ukaz, ki ga lahko uporabimo pri sestavljanju smejkov. Nalogo se da rešiti na več načinov. Eden od njih je, da sami napišemo zaporedje ukazov, po katerem dobimo želeno sliko, vendar je zaporedij ukazov, ki vodijo do rešitve, preveč in bi za to porabili preveč časa.

Tu bomo nalogo rešili s preverjanjem podanih zaporedij in izločanjem tistih, ki niso pravilna.

Zadnji ukaz, ki še nariše želeno sliko, je ali *zrcali* ali *obrni*. Tako četrto zaporedje, pri katerem je zadnji ukaz *obkroži*, ni pravilno. Ostala zaporedja preverimo enega za drugim.

Preverjamo lahko na dva načina, ali začnemo na koncu in gremo proti začetku ali obratno. Tu bomo začeli na koncu in preverjali pravilnost ukazov, kar pomeni, da bomo delali ravno nasprotno od tega, kar nam ukaže ukaz. Pri ukazu *zrcali* bomo sliko prepognili po navpični črti in preverili, ali se vse ujema, pri ukazu *obrni* bomo sliko obrnili za 90 stopinj v nasprotni smeri urinega kazalca, pri ukazu *obkroži* bomo izbrisali krog okoli smejka.

Pri prvem zaporedju je prvi ukaz *zrcali*, torej po navpični črti zložimo oba smejka skupaj in preverimo, če se ujemata. Naslednji ukaz je *obrni*, zato smejka obrnemo v nasprotni smeri urinega kazalca. Ukaz *obkroži* pomeni, da moramo izbrisati krog okoli smejka. Sledi ukaz *obrni*, zato vse skupaj obrnemo v nasprotni smeri urinega kazalca. Pri naslednjem ukazu, ki je spet *obrni*, ugotovimo, da je nekaj narobe. Vidimo, da se podpičje obrne enkrat več kot minus in zaklepaj, kar pomeni, da smejko ne more izgledati tako, kot prikazuje slika.

Drugo zaporedje se začne z ukazom *obrni*, kar smejka postavi pokonci. Naslednji ukaz je *zrcali*, kar pomeni, da moramo sliko prepogniti po navpični črti in tako dobimo dva polkroga enega na drugem. Zaporedje tako ni pravilno.

Prvi ukaz v tretjem zaporedju je *zrcali*, tako smejka prepognemo po navpični črti. Sledi ukaz *obrni*, ki ga mi obravanamo kot obračanje v nasprotni smeri urinega kazalca. Zadnji ukaz je *obkroži*, ki obkroži podpičje, minus in zaklepaj. Tretje zaporedje je pravilno.

4.1.3 Zaporedja števil

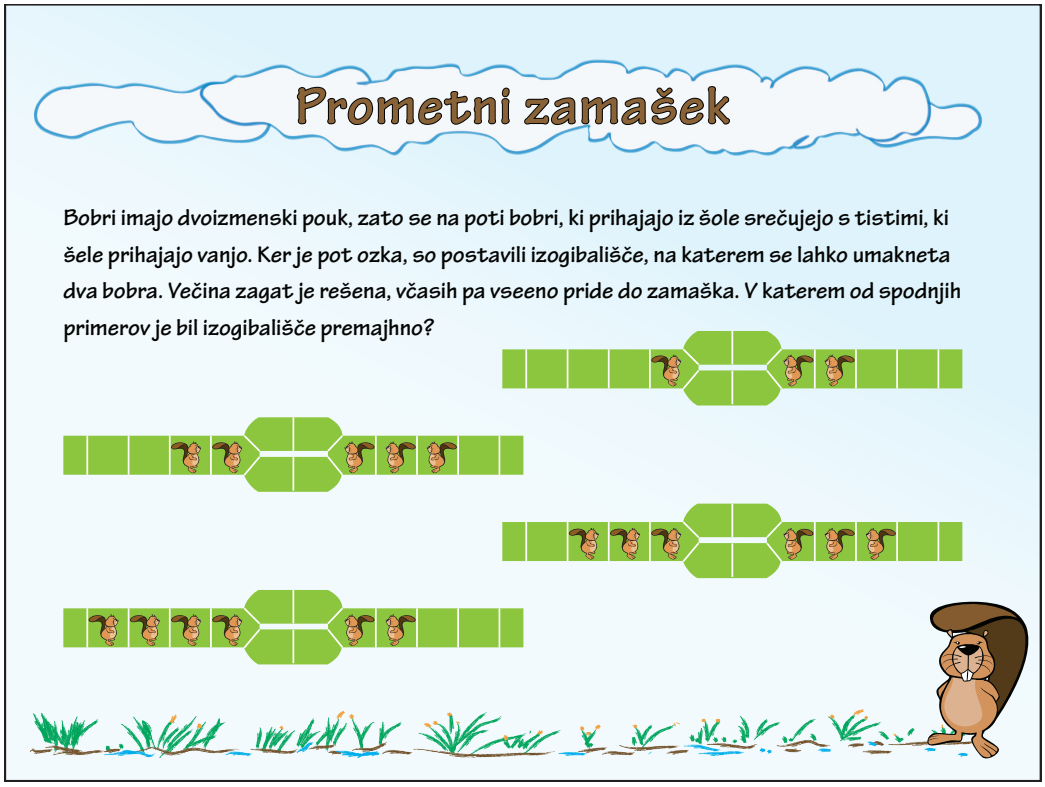
Slika 4.3: Naloga "Zaporedja števil" s tekmovanja Bober.

V nalogi je podan postopek, ki se ga moramo držati pri sestavljanju zaporedja števil.

Število 12 je sodo, zato ga delimo z 2 in dobimo 6, ki je tudi sodo. Po deljenju števila 6 z 2 dobimo število 3, ki je liho. Število 3 pomnožimo s 3 in prištejemo 1, tako dobimo 10. Do sedaj je naše zaporedje: 12, 6, 3, 10. Število 10 je sodo zato ga delimo z 2 in dobimo število 5. Število 5 je liho in ga zato pomnožimo s 3 in prištejemo 1. Dobimo število 16, ki je sodo zato ga delimo z 2. Naslednje število je 8, ki je prav tako sodo. Z deljenjem 8 dobimo število 4, ki je tudi sodo, zato ga spet delimo in dobimo 2. Število 2 delimo z 2 in dobimo 1. S tem zaključimo z gradnjo zaporedja, ki je sedaj

12, 6, 3, 10, 5, 16, 8, 4, 2, 1 in obsega 10 števil.

4.1.4 Prometni zamašek



Slika 4.4: Naloga "Prometni zamašek" s tekmovanja Bober.

V opisu naloge je napisano, da se na izogibališče lahko umakneta največ dva bobra, kar pomeni, da sta ali na levi ali na desni strani lahko največ dva. Seveda jih je lahko na eni strani več, pod pogojem, da sta na drugi strani največ dva.

Nalogo rešujemo tako, da za vsako možno rešitev pogledamo ali ustreza pravilu. Pri prvem primeru sta na levi strani dva bobra, na desni strani pa so trije. Bobra z leve se lahko umakneta na izogibališče, med tem pa se bobri z desne sprehodijo mimo. V drugem primeru imamo na levi strani

enega bobra na desni pa dva. Tudi to ne predstavlja težave, saj se lahko eni ali drugi umaknejo na izogibališče. Tretji primer prikazuje situacijo, ko so na levi strani štirje bobri, na desni pa sta dva. Bobra z desne se lahko umakneta na izogibališče in naredita prostor za bobre z leve. V zadnjem primeru so tako na levi kot desni strani trije bobri. Dva bobra se sicer lahko umakneta na izogibališče, vendar pa tretji povzroči zamašek, kar pomeni, da je izogibališče premajhno.

Poglavje 5

Algoritmi

Kako računalniki urejajo zaporedja števil po velikosti, besede po abecedi? V naslednjem poglavju bomo razložili pet različnih algoritmov za urejanje, med vsemi tudi najhitrejšega, imenovanega kar hitro urejanje. Pokazali bomo tudi, kako optimalno napolniti omejen prostor, ter kako s pomočjo bisekcije poiskati številko, besedo, ...

5.1 Urejanje

Velikokrat se zgodi, da moramo kaj urediti po abecedi, velikosti, teži, ... V tem poglavju bomo pogledali, kako delujejo algoritmi, ki jih uporabljajo računalniki, ko morajo nekaj urediti. Pri vseh algoritmih bomo urejali isto zaporedje števil in s tem prikazali razliko med algoritmi. Uporabljeno zaporedje števil je prikazano na sliki 5.1.

85	34	3	8	89	44	74	20

Slika 5.1: Zaporedje števil, ki jih bomo urejali.

Človek pri urejanju uporablja "metodo ostrega pogleda": vidi vse številke naenkrat in jih brez posebnega sistema spravi v prav vrstni red. Računalnik ne more delovati tako, temveč mu moramo podati točen opis algoritma. Poleg tega računalnik ne "vidi"vseh objektov hkrati, temveč zmore primerjati le dva naenkrat.

5.1.1 Urejanje z izbiranjem

Urejanje z izbiranjem, v angleščini *Selection sort*, nam je najbližje. Predstavljajmo si, da imamo naštete vse številke, kot to prikazuje slika 5.1, sedaj to zaporedje razdelimo na dva dela, del, v katerem so urejena števila, in del, v katerem so še neurejena števila. Vedno poiščemo najmanjšo številko v neurejenem delu in jo zamenjamo s prvo v neurejenem delu, tako se urejeni del poveča.

Slabost urejanja z izbiranjem je v tem, da za n števil potrebujemo n^2 primerjanj teh števil, kar nam pri veliko številih lahko vzame kar nekaj časa.

Postopek

V primeru bomo urejali zaporedje v katerem so števila neurejeno napisana eno za drugim, kot to prikazuje slika 5.1.

Na sliki 5.2 je po korakih prikazano kako deluje algoritem:

0. Na začetku imamo neurejen seznam. Urejenega dela (na levi) ni, imamo samo neurejeni del (na desni).

1. V neurejenem delu poiščemo najmanjšo številko, ki je v tem primeru 3, in jo zamenjamo s prvo številko v neurejenem delu, 85. Tako dobimo prvo urejeno število in meja med urejenim in neurejenim delom se prestavi za eno številko v desno.

2. Spet poiščemo najmanjšo številko, 8, in jo zamenjamo s 34 ter popravimo mejo med neurejenim delom.

3. Najnižja številka je tokrat 20, ki jo zamenjamo s 85.

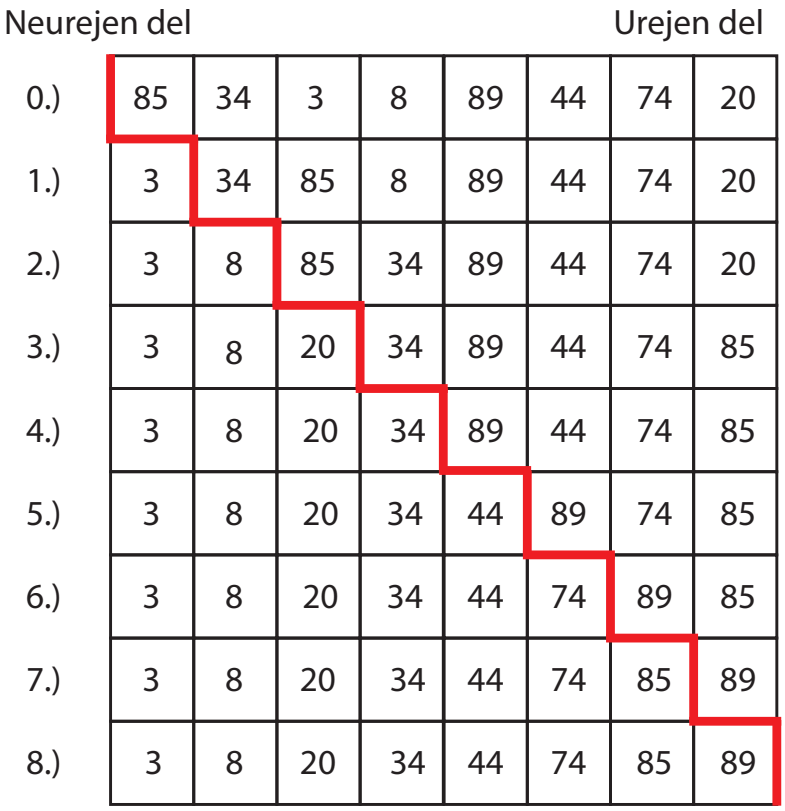
4. Naslednja številka je 34, ki je že urejena in tako ostane kar na svojem mestu.

5. Nadaljujemo s številko 44, ki zamenja številko 89.

6. Naslednjo uredimo številko 74, ki v zaporedju zamenja številko 89.

7. Uredimo še številko 85, ki zamenja številko 89.

8. Na koncu dobimo urejen seznam številk, od najmanjše do največje.



Slika 5.2: Prikaz urejanja z izbiranjem.

5.1.2 Urejanje z vstavljanjem

Urejanje z vstavljanjem, v angleščini *Insertion sort*, pogosto uporabljamo kadar ročno urejamo kakšne stvari. Z vstavljanjem urejamo tako, da v zaporedju po vrsti jemljemo neurejena števila in jih vstavljamo v urejeni del na mesto, ki jim v tistem trenutku pripada (med številko, ki je manjša in številko, ki je večja od te). Postopek ponavljamo, dokler ne dobimo urejenega seznama. V najboljšem primeru imamo že v začetku urejen seznam ali pa moramo urediti le nekaj števil in je ta postopek veliko hitrejši od urejanja z izbiranjem. V najslabšem primeru pa moramo urediti vsa števila in ta postopek ni nič hitrejši od urejanja z izbiranjem.

Postopek

Kot pri urejanju z izbiranjem bomo tudi tukaj za primer urejali zaporedje, v katerem so števila neurejeno napisana eno za drugim, kot to prikazuje slika 5.1.

Na sliki 5.3 je po korakih prikazano, kako deluje algoritem:

0. Na začetku imamo neurejen seznam. Urejenega dela (na levi) ni, imamo samo neurejeni del (na desni).

1. Prva številka, 85, je avtomatično urejena, zato začnemo z drugo številko 34, ki jo vzamemo in vstavimo pred 85, meja med neurejenim in urejenim delom se tako premakne v desno.

2. Številko 3 prav tako vzamemo in vstavimo na začetek urejenega dela zaporedja.

3. Naslednja številka je 8, ki jo vstavimo med števili 3 in 34. Mejo med urejenim in neurejenim delom prestavimo za eno mesto v desno.

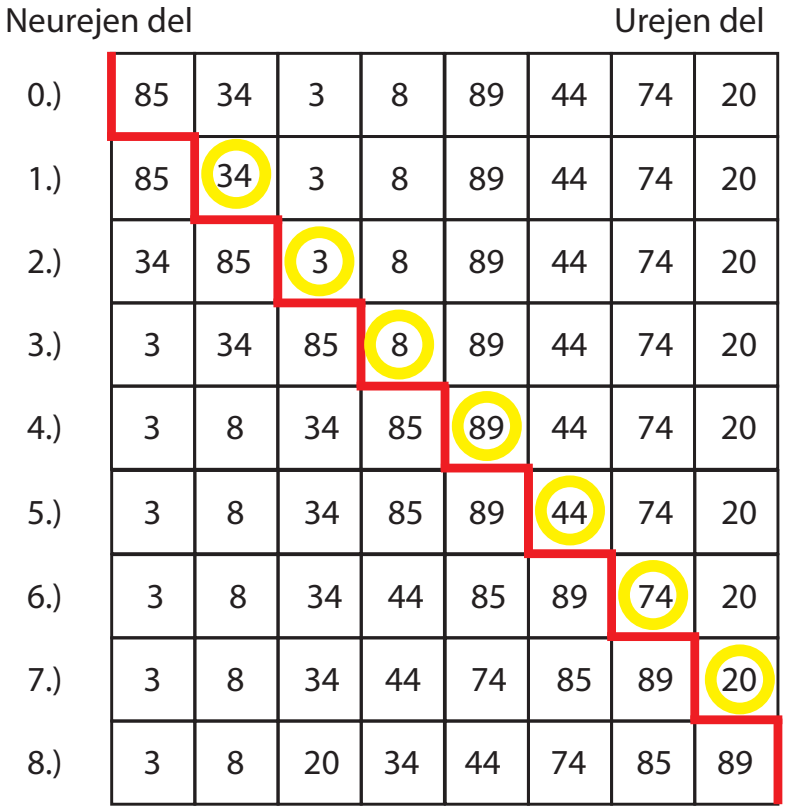
4. Sledi številka 89, ki je do sedaj največja, zato mora čisto na desno stran. Ker je številka že na svojem mestu, samo prestavimo mejo v desno.

5. Nadaljujemo s številko 44, ki jo vstavimo med števili 34 in 85.

6. Številko 74 vstavimo med številki 44 in 85.

7. Nazadnje vzamemo še številko 20 in jo vstavimo med številki 8 in 34.

8. Na koncu dobimo urejen seznam številk, od najmanjše do največje.



Slika 5.3: Prikaz urejanja z vstavljanjem.

5.1.3 Urejanje z mehurčki

Urejanje z mehurčki, v angleščini *Bubble sort*, je dobilo ime po tem, da večji elementi med urejanjem potujejo na konec zaporedja kot mehurčki proti gladini vode. Urejanje deluje tako, da primerja dva elementa zaporedja in večjega postavi na desno, manjšega pa na levo. Urejanje se vedno začne

na začetku zaporedja, potem pa števila potujejo proti levi oz. proti desni. V najslabšem primeru, ko je element z najnižjo vrednostjo čisto na desni strani zaporedja, za urejanje porabimo toliko primerjav kot pri urejanju z izbiranjem. V najboljšem, ko so vsa števila že urejena, pa le toliko primerjav, kolikor imamo elementov v zaporedju.

Postopek

Za prikaz delovanja algoritma za urejanje z mehurčki bomo uporabili neurejeno zaporedje števil prikazano na sliki 5.1. Z razliko od prejšnjih algoritmov, bo urejeni del tokrat na desni strani, neurejeni del pa na levi.

Na sliki 5.4 je po korakih prikazano kako deluje algoritem:

0. Na začetku imamo neurejen seznam. Urejenega dela (na desni) ni, imamo samo neurejeni del (na levi).

1. Najprej primerjamo prvi dve številki, 85 in 34. Številki zamenjamo, saj je 85 večja od 34. Potem primerjamo 85 s 3 in ju spet zamenjamo, prav tako naredimo s številko 8. Številk 85 in 89 ne zamenjamo, saj je 89 večja. Sedaj ostale številke primerjamo s 89, ker je 89 večja od vseh jo postavimo na konec seznama. Tako dobimo prvo urejeno številko.

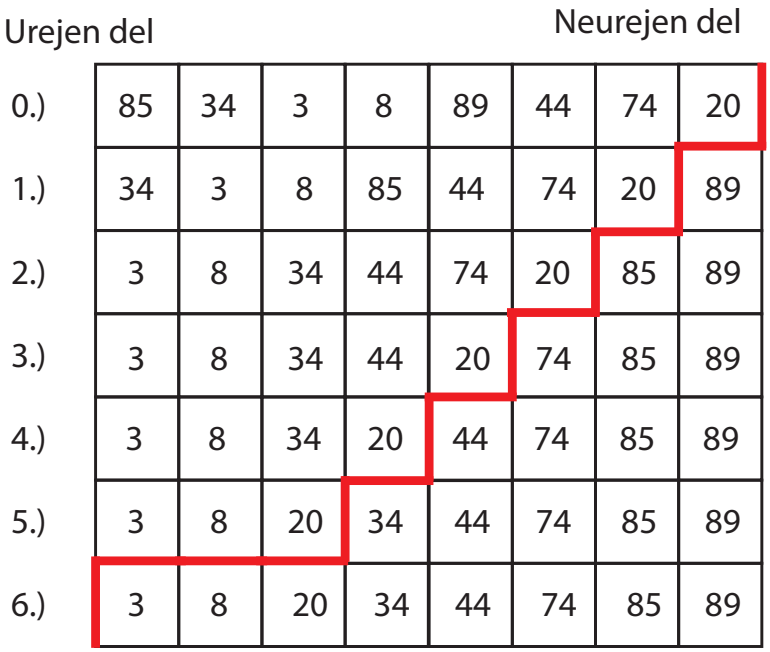
2. Na začetku seznama je sedaj številka 34, ki jo vzamemo in primerjamo s 3 in 8, od katerih je večja. Pridemo do številke 85, ki je večja. Naprej primerjamo vse preostale številke v neurejenem delu s 85. Vse številke so manjše, zato 85 postavimo na konec neurejenega dela zaporedja in tako razširimo urejeni del zaporedja.

3. Vsa števila, do števila 74, so manjša od svojih naslednikov. Številko 74 primerjamo z 20 in ju zamenjamo, saj je 20 manjša. Urejeni del se razširi še za eno mesto.

4. Številko 44 zamenjamo s številko 20, saj je 20 manjša in tako razširimo urejeni del zaporedja.

5. Ker je tudi številka 34 večja od številke 20, jo zamenjamo in dobimo nov urejeni del zaporedja.

6. Ko številk ni več potrebno menjati, dobimo urejen seznam števil od najmanjše do največje.



Slika 5.4: Prikaz urejanja z mehurčki.

5.1.4 Hitro urejanje

Do sedaj smo opisali tri algoritme za urejanje, ki pri svojem delovanju uporabijo veliko primerjav preden uredijo zaporedje. Algoritem, ki je v praksi najhitrejši, se imenuje kar hitro urejanje, v angleščini *Quick sort*.

Deluje tako, da med vsemi števili, ki jih imamo na voljo, naključno izberemo eno in potem vsa števila primerjamo s tem številom. Med primerjanjem števila ločimo na večja in manjša od danega števila. V naslednjih korakih po-

stopek ponovimo najprej na večjih številih, potem na manjših številih (lahko tudi obratno).

Pri hitrem urejanju potrebujemo veliko manj primerjanj kot pri zgornjih dveh algoritmih. Žal se števila primerjanj ne da natančno izračunati, saj je odvisno od tega, kako "dobro"naključno izberemo števila, s katerimi potem primerjamo ostala števila. V povprečju pri hitrem urejanju za n števil potrebujemo $n \log n$ primerjanj.

Postopek

Spet bomo za primer uporabili neurejeno zaporedje, ki ga prikazuje slika 5.1. V tem primeru, za razliko od prejšnjih algoritmov, zaporedja ne bomo delili na urejeni (levi) del in neurejeni (desni) del. Tukaj bomo med samim urejanjem dobivali krajša urejena zaporedja, ki se bodo skozi postopek sama združila v celoto. Vedno bomo urejali tako, da bomo urejali najprej števila, ki so večja od izbranega števila, nato števila, ki so manjša.

Na sliki 5.5 je po korakih prikazano kako deluje algoritem:

0. Na začetku imamo neurejen seznam. Naključno izberemo številko s katero bomo primerjali vsa ostala števila, v našem primeru je to 74.

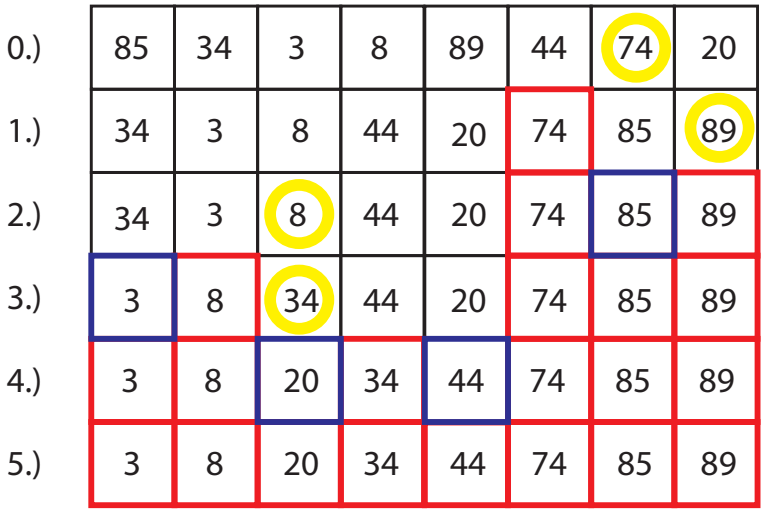
1. Število 74 označimo za urejeno, ostala števila pa razdelimo na levo - manjša od 74, na desno - večja od 74. Iz desnega dela ponovno naključno izberemo številko, tokrat 89.

2. Številko 89 označimo za urejeno in z njo primerjamo edino še ostalo številko, 85, ki jo postavimo na levo, saj je manjša od 89. Ker na desni strani ne moremo več deliti števil, je tudi 85 že urejena, zato jo v tem koraku označimo z modro, v naslednjem pa z rdečo. Na levi strani naključno izberemo eno od števil, v tem primeru 8.

3. Številko 8 označimo za urejeno, ostala pa zopet razdelimo na večja in manjša. Ker je od številke 8 manjša le številka 3, lahko tudi slednjo, tako kot 85 v prejšnjem koraku, označimo za urejeno. Med preostalimi števili na desni ponovno naključno izberemo eno, 34.

4. Številko 34 označimo za urejeno, ostali dve pa razdelimo na večjo in manjšo. Ker sta na vsaki strani po ena številka, ju lahko obe označimo za urejeni.

5. Na koncu dobimo urejen seznam števil od najmanjše do največje.



Slika 5.5: Prikaz hitrega urejanja.

5.1.5 Urejanje z zlivanjem

Urejanje z zlivanjem, v angleščini *Merge sort*, deluje na drugačen način kot zgoraj razložena urejanja. Pri urejanju z zlivanjem potrebujemo vsaj dve že urejeni zaporedji, ki ju potem zlijemo v eno zaporedje. Recimo, da imamo dva kupa knjig, ki sta urejena po abecedi in ju želimo po abecedi urediti v en kup. Knjige zlagamo tako, da iz vsakega kupa vzamemo po eno knjigo, naslova knjig primerjamo in tisto, ki je prej po abecedi odložimo na nov kup. Iz kupa, na katerem je bila odložena knjiga, vzamemo novo knjigo, ki jo zopet primerjamo in tisto, ki je prej po abecedi odložimo na kup. Postopek ponavljamo, dokler iz dveh kupov ne naredimo samo enega, novega.

Urejanje z zlivanjem se uporablja, ko imamo veliko zelo dolgih zaporedij, ki jih moramo združiti v eno, saj zlivamo po dve zaporedji hkrati in tako lahko delo razdelimo. Na primer, če imamo šestnajst zaporedij, jih najprej zlijemo v osem zaporedij, potem v štiri zaporedja, ki jih zlijemo v dve zaporedji in na koncu v eno zaporedje. Seveda je postopek izvedljiv tudi, če število zaporedij ni potenca števila 2, vendar v tem primeru nekatere vrste čakajo.

Postopek

Delovanje algoritma bomo prikazali s šestnajstimi števili v štirih urejenih zaporedjih A, B, C in D, ki so prikazana na sliki 5.6, kjer je prikazano tudi delovanje algoritma.

1. V novo zaporedje E zlijemo zaporednji A in B.
   (a) Vzamemo številki 3 in 1 in ju primerjamo; ker je 1 manjša, jo pošljemo naprej, številko 3 obdržimo
   (b) Iz zaporedja B vzamemo novo številko, 6. Številki primerjamo in pošljemo številko 3 naprej, številko 6 pa obdržimo.
   (c) Naslednja številka v zaporedju A je 5, ki jo primerjamo s 6 in pošljemo naprej, saj je manjša.
   (d) Iz zaporedja A dobimo novo številko, 10. Ker je 6 manjša, jo pošljemo naprej.
   (e) Nova številka iz zaporedja B je 7, ki je manjša od 10, zato jo pošljemo naprej.
   (f) Naslednja številka iz zaporedja B je 9, ki je tudi manjša od 10, zato gre naprej.
   (g) V zaporedju B ni več številk, zato se števili 10 in 14 iz zaporedja A zlijeta v zaporedje E.
2. V novo zaporedje F zlijemo zaporedji C in D.

(a) Iz zaporedij C in D vzamemo prvi številki, 2 in 4, in ju primerjamo. 2 je manjša zato jo pošljemo naprej, 4 pa zadržimo.

(b) Iz zaporedja C vzamemo novo številko, 7 in jo primerjamo z obstoječo številko, 4. Številka 4 je manjša, zato gre naprej, 7 obdržimo.

(c) Naslednja številka iz zaporedja D je 6, ki je manjša od 7, zato jo pošljemo naprej.

(d) Iz zaporedja D vzamemo novo številko, 11, ki je večja od 7. Številko 7 pošljemo naprej.

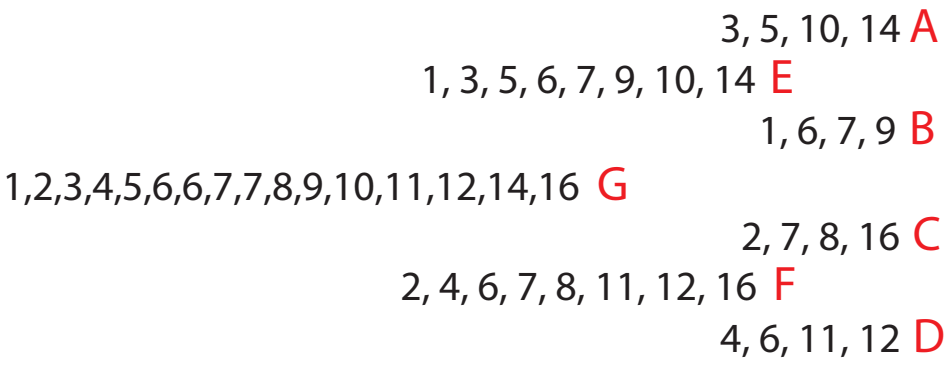
(e) Iz zaporedja C dobimo številko 8, ki jo pošljemo naprej, številko 11 pa obdržimo.

(f) Naslednja številka v zaporedju D je 16, ki je večja od 11. Številko 11 pošljemo naprej, številko 16 pa obdržimo.

(g) Iz zaporedja C dobimo številko 12, ki je manjša od 16, zato jo pošljemo naprej.

(h) Ker v zaporedju C ni več številk, v zaporedje F zlijemo še 16.

3. Po enakem postopku zlijemo v novo zaporedje G še zaporedji E in F.



Slika 5.6: Prikaz urejanja z zlivanjem.

5.1.6 Primer naloge s tekmovanja Bober

Slika 5.7: Naloga "Postopek urejanja" s tekmovanja Bober.

Najprej pogledamo, kateri postopek je bil do sedaj uporabljen za urejanje števil. V drugem koraku vidimo, da se je številka 7 uredila skrajno desno, zamenjali pa sta se tudi števili 5 in 4. V tretjem koraku se je uredila številka 6, pred tem pa se je prestavila tudi številka 5, vse do številke 6 (preden se je ta zamenjala s številko 1).

Na podlagi tega lahko ugotovimo, da gre za urejanje z mehurčki. Vse, kar moramo zdaj narediti, je, da stanje v 4. koraku uredimo z danim algoritmom. Tako se 0 in 2 zamenjata, številka 3 ostane na svojem mestu. Zamenjata se tudi številki 4 in 1, ostale številke so urejene.

Dobimo zaporedje: 0, 2, 3, 1, 4, 5, 6, 7, ki je enako odgovoru A.

5.2 Polnjenje nahrbtnika

Problem polnjenja nahrbtnika, v angleščini *Knapsack problem*, se ukvarja s problemom, ki ga v običajnem svetu zastavimo kot vprašanje, kako optimalno napolniti nahrbtnik z omejeno prostornino oz. notranjostjo. Postavimo se v kožo nekoga, ki koplje diamante. Po enem mesecu je izkopal kar nekaj velikih kosov, sedaj pa jih mora odnesti v zlatarno in jih tam prodati. Žal so diamanti preveliki, da bi lahko nesel vse naenkrat, zato se mora odločiti, katere bo odnesel ter katere bo zavrgel. Velikost diamanta ponavadi ne odraža vrednosti, pri ocenjevanju vrednosti je pomembnih tudi veliko drugih dejavnikov.

Diamante tako razporedi po njihovi vrednosti in ne njihovi velikosti. Nahrbtnik začne polniti z bolj vrednimi diamanti in nadaljuje proti manj vrednim. Ko pride do diamanta, ki ne gre več v nahrbtnik, ga izpusti in nadaljuje naprej, dokler ne napolni nahrbtnika.

Ta postopek ni vedno najboljši, saj se lahko zgodi, da z nekaj najbolj vrednimi diamanti napolnimo nahrbtnik, pri tem pa nam ostane več malo manj vrednih, s katerimi bi ob pravi kombinaciji dosegli višjo skupno vrednost. Vendar bi za iskanje najbolj optimalne rešitve porabili veliko več časa. Prednost postopka je, da je hiter in relativno učinkovit.

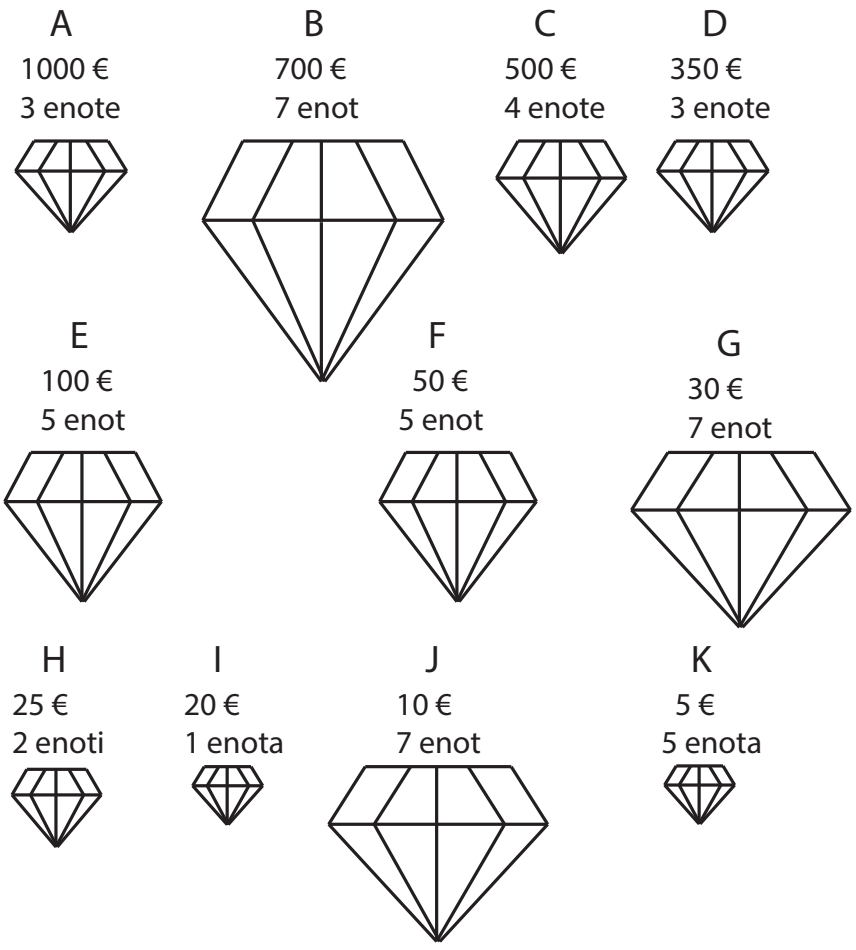
5.2.1 Postopek

Algoritem bomo prikazali kar na primeru diamantov. V nahrbtnik lahko zložimo za 30 enot diamantov, na voljo imamo pa dimante, ki skupaj zasedejo 51 enot. Vrednost diamantov in koliko enot zasede sta prikazana na sliki 5.8

Vedno začnemo z najbolj vrednimi diamanti in nadaljujemo z manj vrednimi. Najprej v nahrbtnik spravimo diamant A, ki zasede 3 enote, ostane nam jih še 27. Naslednji diamant, ki ga bomo odnesli, je diamant B, ki zasede 9 enot, tako nam jih ostane še 18. Nadaljujemo z diamantom C, na voljo imamo še 14 enot. V nahrbtnik po vrsti spravimo še diamante D, E in F, ki skupaj zasedejo za 13 enot prostora. Diamantov G, H in J ne moremo

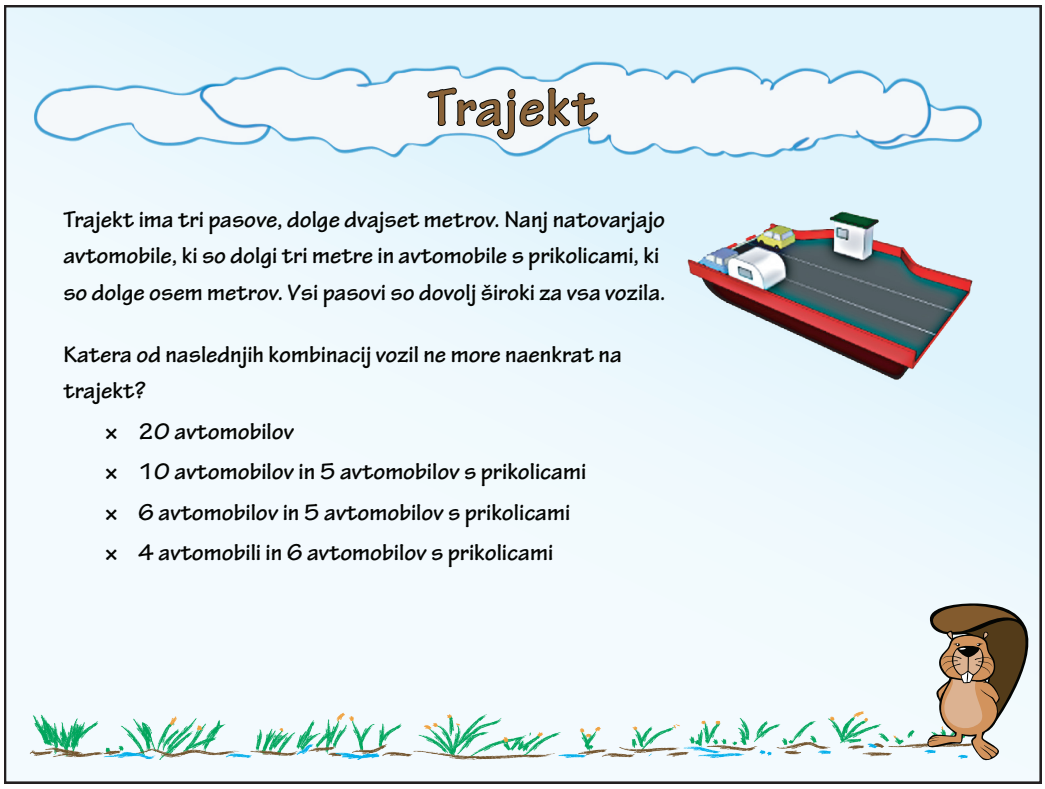
shraniti, saj zavzamejo preveč prostora. Dovolj velika sta le še diamanta I in K, vendar ima diamant I prednost, saj je vreden več kot diamant K.

Nahrbtnik je tako poln, v njem pa so diamanti v vrednosti 2.720 €. To je veliko več, kot če bi diamante polnili po velikosti (začenši z manjšimi), ko sploh ne bi mogli do konca napolniti nahrbtnika in bi nam to prineslo le 1.150 €.



Slika 5.8: Diamanti za primer prikaza postopka polnjenja nahrbtnika.

5.2.2 Primer naloge s tekmovanja Bober



Slika 5.9: Naloga "Trajekt" s tekmovanja Bober.

Preveriti je potrebno katere od danih kombinacij je možno vkrcati na trajekt in katerih se ne da. Na voljo imamo tri pasove dolge 20 metrov. Predpostavimo, da je dolžina avtomobila oz. avtomobila s prikolico kar vrednost le tega.

Najprej preverimo kombinacijo dvajsetih avtomobilov. Začnemo s polnjenjem prvega pasu, na katerega lahko vkrcamo šest avtomobilov, ki so skupaj dolgi 18 metrov. Prav toliko jih lahko vkrcamo na drugi in tretji pas, kar pomeni, da lahko na trajekt vkrcamo največ 18 avtomobilov.

Naslednja je kombinacija desetih avtomobilov in petih avtomobilov s prikolicami. Začnemo z bolj vrednimi, kar pomeni, da najprej vkrcamo avto-

mobile s prikolicami. Na prvi in drugi pas tako vkrcamo po dva avtomobila s prikolicami, na tretjega pa en avto s prikolico. Na prvem in drugem pasu nam ostane še 4 metre prostora, na tretjem pa 12. Na prvi in drugi pas lahko vkrcamo po en avto, na tretjega pa štiri avtomobile. To pomeni, da lahko na trajekt spravimo največ šest avtomobilov in pet avtomobilov s prikolico, kar od nas zahteva tretja kombinacija.

Zadnja kombinacija, ki jo moramo preveriti, je tista s štirimi avtomobili in šestimi avtomobili s prikolicami. Tudi tukaj začnemo z bolj vrednimi in tako na vse tri pasove vkrcamo po dva avtomobila s prikolico. Na vsakem od treh pasov nam tako ostane še štiri metre prostora, kar pomeni, da lahko na vsak pas vkrcamo le po en avtomobil.

Edina kombinacija, ki jo lahko peljemo s tem trajektom, je tretja, šest avtomobilov in pet avtomobilov s prikolico.

5.3 Bisekcija

Bisekcija, v angleščini *Bisection*, se uporablja, ko moramo preiskati ogromno števil, napak ali telefonskih številk, ki so primerno urejene, na primer po velikosti ali abecedi.

Bisekcija deluje tako, da vedno razdeli dani prostor na dva enako velika dela, na polovico. Pri iskanju to pomeni, da v vsakem koraku zmanjšamo prostor iskanja za polovico in tako zelo hitro najdemo, kar iščemo. To pomeni, da lahko iskano najdemo v $\log n$ korakih, kjer je n velikost našega prostora.

5.3.1 Postopek

Algoritem bomo prikazali na primeru iskanja telefonske številke v imeniku. Imenik je dolg in mi bi radi kar najhitreje poiskali Janeza ter njegovo telefonsko številko.

Postopek iskanja je prikazan na sliki 5.10.

0. Na začetku imamo celoten imenik, v katerem je petnajst številk.

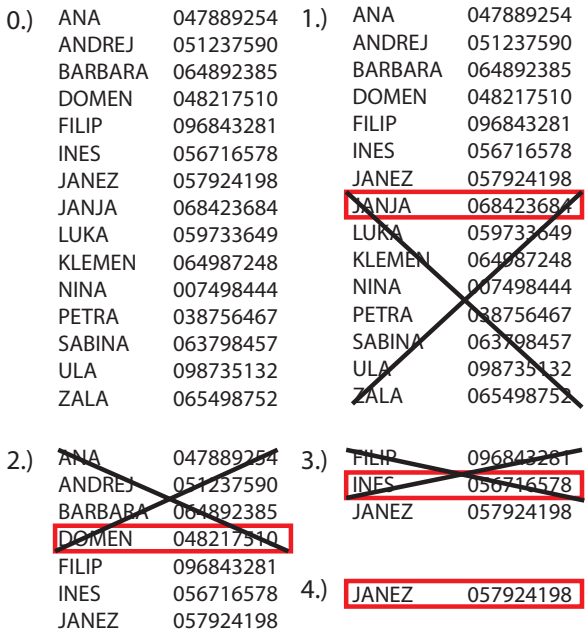
1. Telefonski imenik odpremo na sredini, na strani na kateri je Janja. Janez mora biti v imeniku pred Janjo, zato lahko odrežemo celoten spodnji del imenika, skupaj z Janjo. Ostane nam še 7 številk.

2. Preostali del imenika zopet razdelimo na pol, torej ga odpremo na strani, na kateri je Domen. Črka J je po abecedi za črko D, kar pomeni, da lahko odrežemo vse številke nad Domnom, vključno z njim. Ostanejo nam še 3 številke.

3. Ponovimo postopek in zopet razdelimo na pol. Tokrat odpremo na strani, na kateri je Ines. I je po abecedi pred J, zato spet odrežemo zgornji del imenika. Ostane nam le še ena številka.

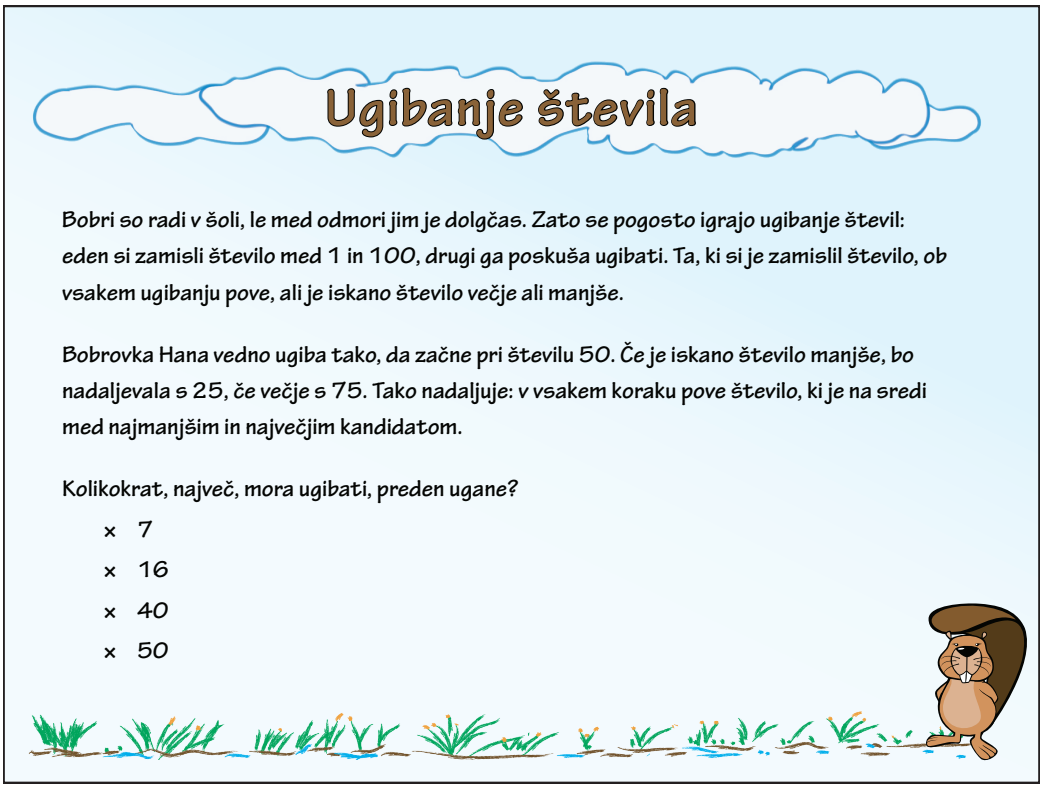
4. V tem koraku smo našli Janeza in s tem njegovo številko.

Za iskanje telefonske številke smo tako porabili štiri korake, kar je zaokrožen dvojiški logaritem števila petnajst.



Slika 5.10: Prikaz postopka iskanja telefonske številke v imenku v pomočjo bisekcije

5.3.2 Primer naloge s tekmovanja Bober



Slika 5.11: Naloga "Ugibanje števil" s tekmovanja Bober.

V navodilu naloge je razloženo, da gre za iskanje števil z bisekcijo. Hana tako najprej število 100 razdeli na dva dela, kjer je sredina 50, potem pa tudi 50 razdeli na dva dela in tako dobi 25 in 75 za nadaljevanje. Ni nam potrebno poznati končne številke, da bi ugotovili, koliko korakov potrebujemo, da jo najdemo, saj moramo samo deliti prostor toliko časa, dokler nam ne ostane le ena številka.

Predpostavimo, da je številka manjša od 50 in nadaljujemo s 25. V nadaljevanju 25 razdelimo na pol in dobimo 13 (vedno zaokrožimo navzgor). 13 spet razdelimo na dva dela in dobimo 7. Število 7 spet razdelimo na pol in dobimo 4. Ko razdelimo 4 na pol, dobimo 2. Od števila 2 je lahko manjše le

število 1, večje pa lahko le število 3, saj smo število 4 že preverili.

Otroci tako spoznajo logaritem, čeprav ne vedo, kako se imenuje. Dvojiški logaritem od števila 100 je 6.6 in če to zaokrožimo navzgor dobimo 7, ki je naša rešitev. Do rešitve bi lahko prišli tudi tako, da bi po vrsti šteli potence števila 2 in tista s katero bi prišli čez število 100 je naša rešitev. V tem primeru je to število 7, saj je $2^6 = 64$ in je tako manjše od 100, $2^7 = 128$, ki pa je že večje od 100.

Poglavje 6

Algoritmi na grafih

Problemov z grafi je v računalništvu ogromno, zato je tudi algoritmov, ki se ukvarjajo z reševanjem teh problemov, ogromno. V naslednjih poglavjih bomo predstavili problem barvanja grafov, iskanje v širino in globino, Primov algoritem za iskanje minimalnega vpetega drevesa ter Dijkstrov algoritem za iskanje najkrajših poti v grafu.

6.1 Barvanje grafov

Veliko problemov je takih, da imamo opravka s primeri, kjer moramo paziti, da se več stvari ne dogaja istočasno ali pa da določene stvari ne smejo stati zraven nekaterih drugih.

Značilen primer pri teh problemih je načrtovanje šolskega urnika, ko učitelj ne more biti na dveh mestih hkrati, prav tako dva razreda ne moreta biti v isti učilnici.

V računalništvu in matematiki je bolj znan problem barvanja zemljevidov. Želeli bi pobarvati zemljevid in pri tem uporabiti kar se da malo barv, pri čemer državi, ki imata skupno mejo (razen, kadar je to v eni točki), ne smeta biti iste barve. Problem je bil prvič opisan leta 1852, a šele leta 1976 so dokazali, da se da katerikoli zemljevid pobarvati z največ štirimi barvami.

Problem lahko predstavimo z grafom: točke predstavljajo države, dve

državi sta povezani, če imata skupno mejo. Naloga je vsaki točki določiti barvo.

Točnega algoritma za reševanje tega problema ne poznamo, čas za reševanje pa ponavadi narašča eksponentno z velikostjo grafov. To pomeni, da se čas barvanja z vsako novo točko podvoji.

6.1.1 Primer naloge iz tekmovanje Bober

Slika 6.1: Naloga "Ne vrag, vrtičkar bo mejak" s tekmovanja Bober.

Nalogo bomo reševali tako, da bomo za graf, prikazan na sliki, poskušali najti pravilno razporeditev vrtov na posamezni livadi.

Naš graf ima v vsaki točki vsaj dve povezavi. Rdeč vrt na prvi livadi ima samo enega soseda, rumen vrt posledično samo eno povezavo. To pomeni,

da na prvi livadi ni Majinega vrtička.

Ko v grafu preštejemo število povezav iz posamezne točke, ugotovimo, da gre iz ene točke pet povezav, kar pomeni, da en od vrtičkov meji na kar pet drugih vrtičkov. Na drugi in četrti livadi ni nobenega vrtička, ki bi mejil na pet drugih, kar pomeni, da na teh dveh livadah ni Majinega vrtička.

Ostane nam še tretja livada, na kateri moder vrt meji na pet drugih vrtov, prav tako pa so pravilne tudi ostale povezave med vrtovi. Majin vrtiček je na tretji livadi.

6.2 Iskanje v globino

V več zabaviščnih parkih imajo med vsemi atrakcijami tudi labirint. Večina se iskanja poti iz labirinta loti brez pravega sistema, z ugibanjem katera pot je prava. Iskanje v globino ponuja sistematičen način odkrivanja poti iz labirinta. V angleščini se algoritem imenuje *Depth-first search* in, kot že ime samo pove, najprej preiskuje v globino.

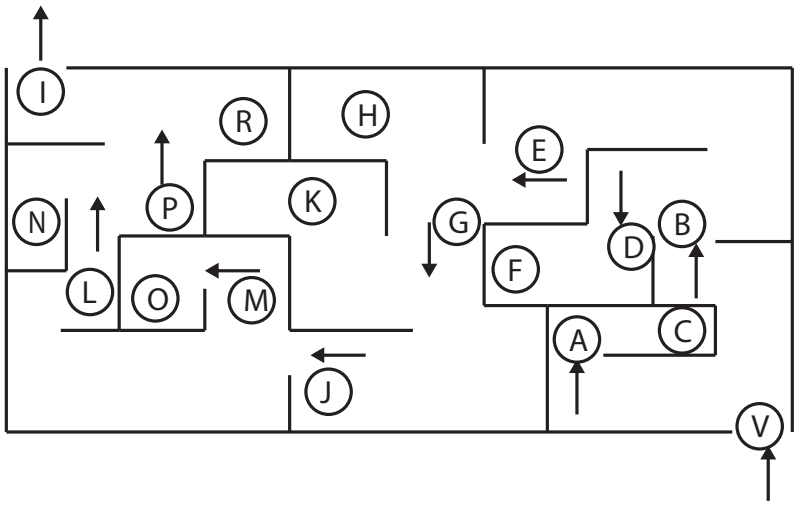
Za razliko od iskanja v širino ta algoritem najprej razišče eno vejo drevesa, potem nadaljuje na drugi, tretji, ..., dokler ne najde cilja. Algoritem vedno najprej razišče levo vejo. Algoritem lahko uporabimo tudi na grafih, kjer se po povezavah premikamo med točkami.

Slabost algoritma se pokaže pri grafih, ki vsebujejo cikel, saj v tem primeru algoritem v neskončnost skače med dvema globinama. Težavo rešimo tako, da označimo točke, v katerih smo že bili in jih ob ponovnem odkritju enostavno zavržemo.

6.2.1 Postopek

Za razlago bomo uporabili labirint na sliki 6.2, ki smo ga za potrebe te naloge preslikali v drevo, prikazano na sliki 6.3. Na zemljevidu labirinta in drevesu labirinta so s točkami, velike črke, označena križišča, kjer se bomo morali odločiti za smer levo ali desno. Začetna točka je označena z V - vhod,

končna pa z I - izhod. Puščice na zemljevidu labirinta pa nam povedo, v katero smer smo na križišču obrnjeni.



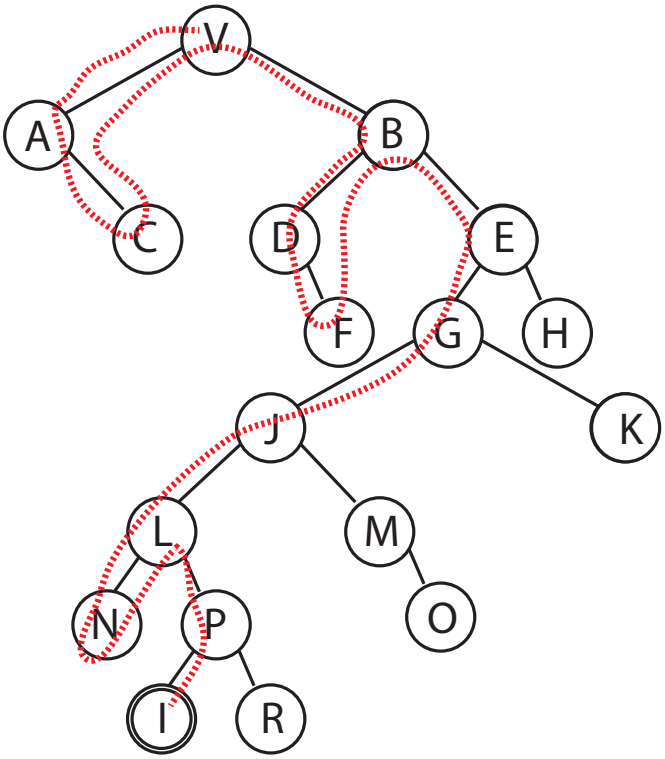
Slika 6.2: Labirint.

Začnemo torej v točki V, kjer je tudi prvo križišče. Algoritem določa, da moramo vedno najprej zaviti levo. Tako najprej odkrijemo točko A, kjer ne levo ne moremo zaviti, zato gremo v desno, kjer odkrijemo točko C, ter tudi slepo ulico.

Vrnemo se nazaj na zadnje še ne do konca raziskano križišče, v tem primeru je to V. Ker smo levo že zavili, sedaj zavijemo desno in tako pridemo do točke B, v kateri po pravilu spet zavijemo levo v točko D. Iz točke D lahko nadaljujemo le na desno, kjer odkrijemo točko F, iz katere ni več možnosti za nadaljevanje.

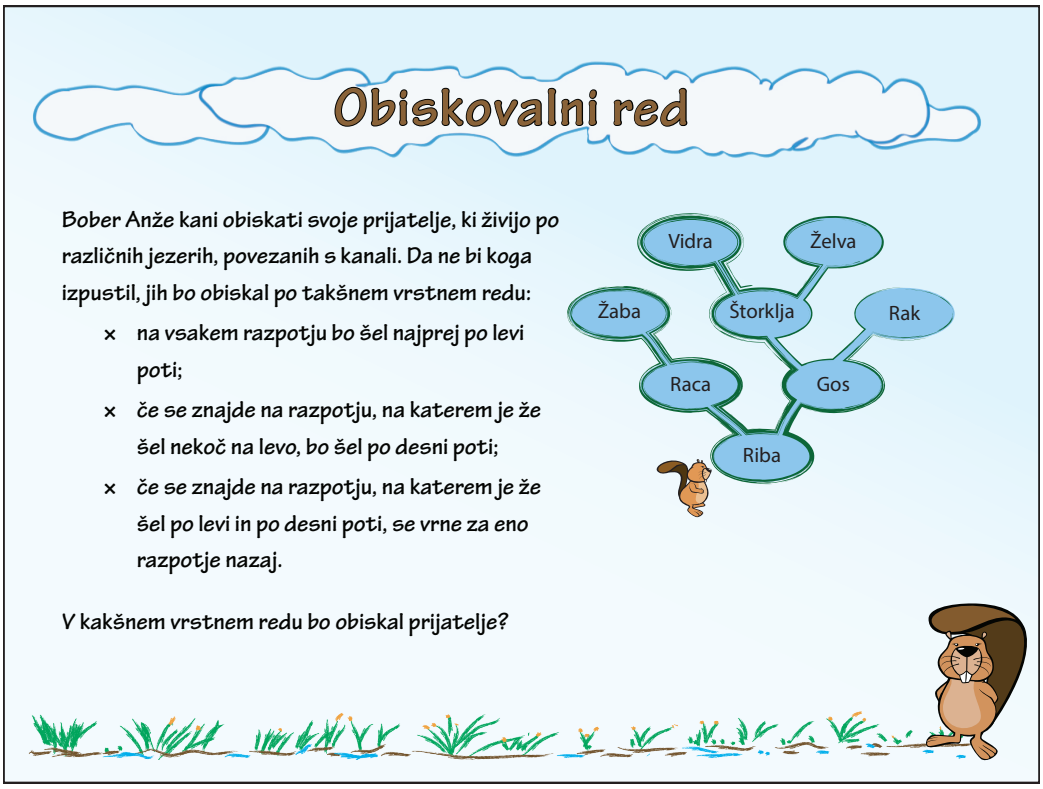
Spet se vrnemo nazaj, tokrat na točko B in zavijemo desno ter nadaljujemo v točki E. V točki E spet po pravilu zavijemo levo in tako pridemo do točke G. V točki G sledimo pravilu in zavijemo levo do točke J. V točki J nadaljujemo po levi in pridemo v točko L, kjer zavijemo levo v točko N, za katero ugotovimo, da iz nje ni več poti naprej.

Vrnemo se v točko L in iz nje nadaljujemo po desni v točko P. Iz točke P po levi pridemo do cilja, izhoda iz labirinta.



Slika 6.3: Prikaz labirinta s pomočjo drevesa.

6.2.2 Primer naloge s tekmovanja Bober



Slika 6.4: Naloga "Obiskovalni red" s tekmovanja Bober.

Že v samem besedilu naloge je razloženo preiskovanje v globino.

Predpostavimo, da Anže z obiski začne pri ribi. Po pravilih, ki so podana (*preiskovanje v globino*), je naslednji obisk pri raci, saj mora iti levo. Naslednji obisk bo pri žabi, ki je prav tako na levi strani. Po obisku žabe se mora vrniti nazaj do ribe, saj je žaba povezana le z raco, raca pa le z ribo in žabo. Prav tako na razpotju nima možnosti zaviti v desno; prvo razpotje, na katerem je možno zaviti desno, je kar izhodišče, pri ribi. Anže svojo pot nadaljuje pri goski, ki je na razpotju med štorkljo in rakom. Spet upošteva pravilo, da gre najprej v levo in se odpravi k štorklji, kjer ponovno zavije v levo do vidre. Pri vidri se mora obrniti in vrniti nazaj do štorklje, kjer zavije

desno in pride do želve. Od želve se mora vrniti nazaj mimo štorklje do gosi, kjer zavije v desno in obišče še raka.

Končna vrstni red obiskov je tako *riba, raca, žaba, gos, štorklja, vidra, želva, rak.*

6.3 Iskanje v širino

Ko se odpravljamo v šolo, službo, na obisk k sorodnikom ali na potovanje, si želimo, da bi šli po najkrajši poti in tako prihranili čas. Pri iskanju te poti si lahko pomagamo z iskanjem v širino, v angleščini *Breadth-first search (BFS)*. Iskanje v širino poišče najbližjo pot iz korena drevesa do želenega cilja. Algoritem najprej preišče vse točke, ki so oddaljene za eno povezavo, potem se premakne naprej na tiste, ki so oddaljene za dve povezavi in tako naprej, dokler ne najde cilja.

6.3.1 Postopek

Algoritem bomo razložili na primeru, ki ga prikazuje slika 6.5. Naprava za navigacijo je računalnik in v začetku vidi samo začetno točko - koren, ostale točke mora odkriti. Prav tako računalnik ne ve, v kateri smeri je cilj, zato mora vedno raziskati vse možnosti. V vročih poletnih mesecih je zelo zaželena destinacija morje, zato se bomo v tem primeru odpravili iz Ljubljane v Koper.

V Ljubljani se ustavimo v prvem križišču in ker ne vemo, v katero smer je Koper, se najprej odpeljemo v Kranj, kjer ugotovimo, da sta naslednji možni destinaciji Tržič in Bled, potem se odpravimo v Vrhniko in zabeležimo naslednji destinaciji, ki sta Ajdovščina in Postojna. Nazadnje se odpravimo še v Domžale, kjer je križišče proti Kamniku ali Celju.

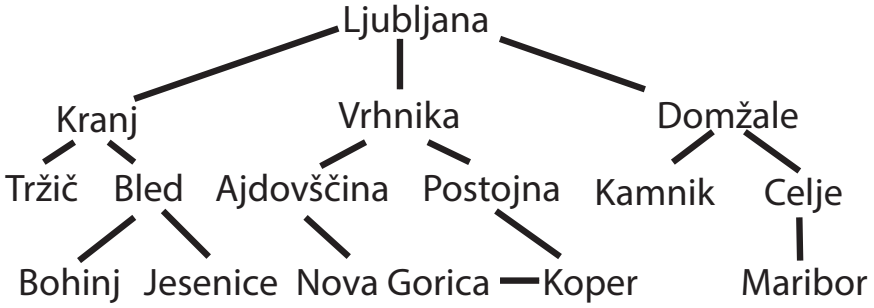
Ker v prvem koraku nismo našli našega cilja, raziskujemo naprej. Začnemo v Tržiču, za katerega ugotovimo, da je končna točka in iz njega ne moremo nadaljevati nikamor. Naslednja točka, ki jo obiščemo je Bled, iz katerega lahko nadaljujemo v Bohinj ali Jesenice, kar si spet le zabeležimo, ne da bi ju obiskali. Nadaljujemo v Ajdovščini od koder lahko nadaljujemo v Novo

Gorico. Iz Postojne lahko nadaljujemo v Koper, vendar to še ni konec, saj v Koper lahko nadaljujemo šele v naslednjem koraku, tako nadaljujemo odkrivanje v Kamniku, za katerega ugotovimo, da je končna točka. Na koncu nam ostane le še Celje, od koder lahko nadaljujemo v Maribor.

Tudi v drugem koraku nismo prišli do cilja, zato raziskujemo naprej. Tokrat začnemo v Bohinju, za katerega ugotovimo, da je končna točka in se iz nje ne da nadaljevati naprej. Enako ugotovimo tudi za Jesenice. V Novi Gorici ugotovimo, da lahko v naslednjem koraku nadaljujemo v Koper in si to zabeležimo. Naslednja točka je Koper, ki je hkrati tudi naš cilj. Zapomnimo si pot do Kopra, ki je Ljubljana - Vrhnika - Postojna - Koper. Nadaljujemo z raziskovanjem, tako za Maribor ugotovimo, da je končno vozlišče.

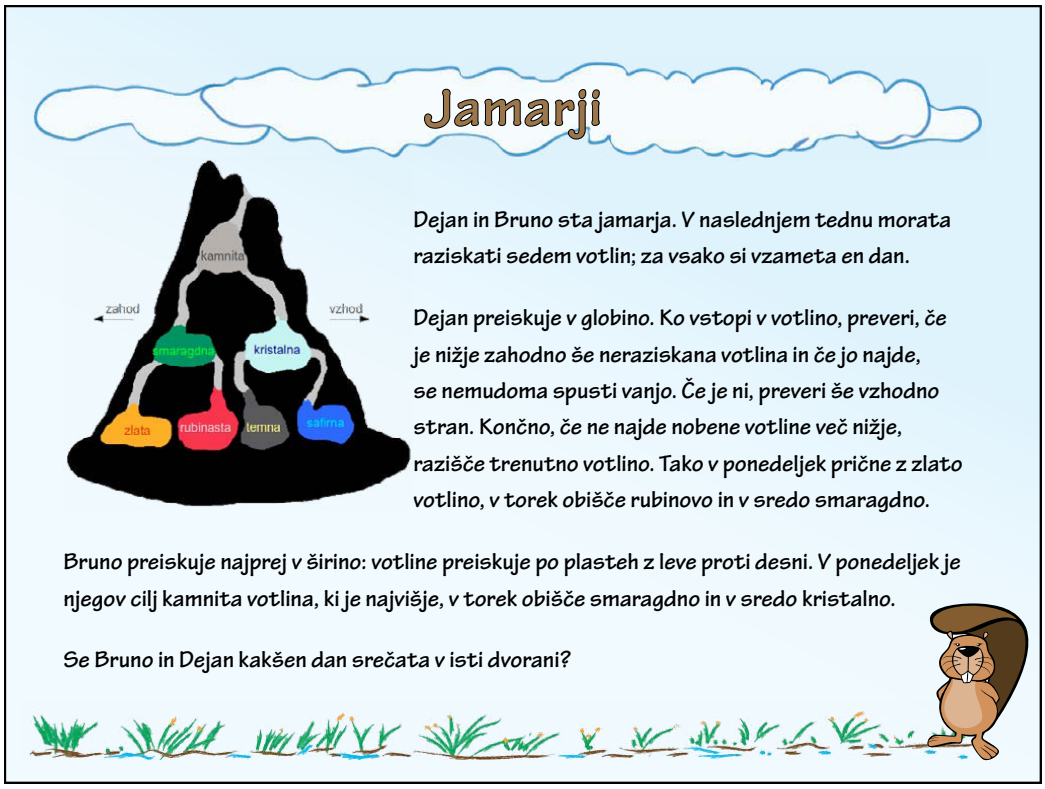
V tretjem koraku smo našli naš cilj, vendar nam je ostalo še eno vozlišče. Še drugič tako odkrijemo naš cilj, Koper, le da je tokrat po do njega drugačna, Ljubljana - Vrhnika - Ajdovščina - Nova Gorica - Koper.

Algoritem nam je tako vrnil dve različno dolgi poti. V realnosti je teh poti še veliko več, saj naprave za navigacijo poznajo tudi vse stranske ceste in jih upoštevajo pri iskanju najkrajše poti.



Slika 6.5: Skica zemljevida Slovenije.

6.3.2 Primer naloge s tekmovanja Bober



Slika 6.6: Naloga "Jamarji" s tekmovanja Bober.

Pri tem primeru naloge gre tako za iskanje v globino kot iskanje v širino. Bober Dejan išče v globino, bober Bruno pa v širino. Vsak bober se vsak dan premakne v drugo jamo.

V ponedeljek je Dejan v Zlati jami, Bruno pa v Kamniti jami. V torek Dejan nadaljuje v Rubinasti jami, Bruno pa je v Smaragdni jami. V sredo je Bruno v Kristalni jami, Dejan pa v Smaragdni jami. V četrtek je Dejan v Temni jami, Bruno pa v Zlati jami. V petek Dejan raziskuje v Safirni jami, Bruno pa v Rubinasti jami.

Postopka od tukaj naprej ni več potrebno nadaljevati, saj je Bruno že bil v obeh jamah, ki ju mora Dejan še preiskati. Torej se ne moreta srečati.

6.4 Minimalno vpeto drevo

V Sloveniji veliko denarja porabimo za gradnjo cest in avtocest. Pa so te ceste res zgrajene najbolj optimalno glede na to koliko mest je povezanih za denar, ki smo ga vložili? Za iskanje optimalne rešitve si lahko pomagamo z mimimalnim vpetim drevesom, v angleščini *Minimum spanning tree*. Pri gradnji minimalnega vpetega drevesa nas ne zanima več oddaljenost med kraji temveč, kako s cestami povezati vse kraje, pri tem pa porabiti čim manj asfalta.

Minimalno vpeto drevo iščemo na poljubnih grafih, ki imajo na povezavah med točkami napisano tudi ceno, ki prikazuje, koliko nas stane, da pridemo iz ene točke v drugo. Vpeto drevo je minimalno, če je seštevek vseh cen med točkami najmanjši možni za ta graf.

Pri določanju minimalnega vpetega drevesa si lahko pomagamo z dvema algoritmoma:

- **Primov algoritem** - iskanje minimalnega vpetega drevesa začne v poljubni točki. Za povezavo do naslednje točke izberemo tisto, ki nas najmanj stane in hkrati povezuje do točke, ki je še ni v drevesu. Ta postopek ponavljamo dokler ne povežemo vseh točk.
- **Kruskalov algoritem** - najprej izberemo povezavo z najnižjo ceno. Dodamo vse povezave z enako ceno in pri tem pazimo, na že povezane točke. Poiščemo naslednjo najnižjo ceno in ponovimo postopek. Postopek ponavljamo, dokler ne povežemo vseh točk.

6.4.1 Postopek

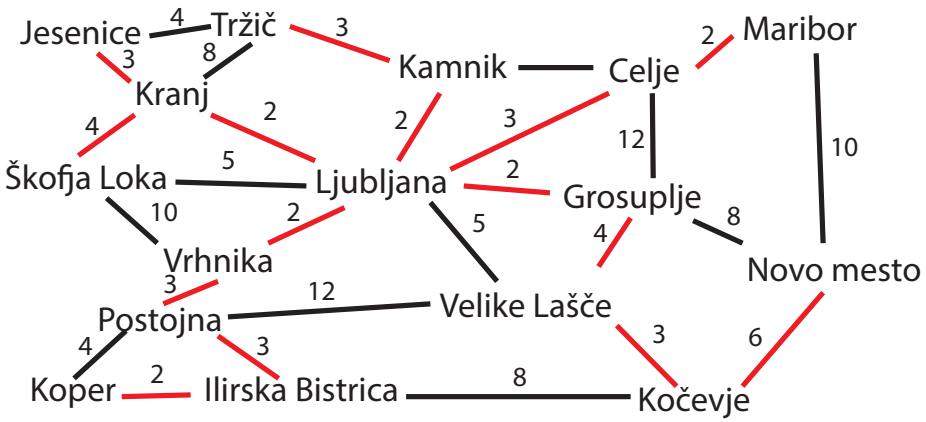
Odločili smo se, da bomo znanemu slovenskemu podjetju pomagali zgraditi ceste. Pridobili smo zemljevid Slovenije 6.7 s cenami (v miljonih €) za asfaltiranje cest med posameznimi kraji. Ker denar priskrbi država in morajo na razpisu izbrati najcenejšega ponudnika, bomo problem reševali z minimalnim vpetim drevesom. V tem primeru bomo problem reševali s Primovim algoritmom.

Začetno točko lahko poljubno izberemo; izbrali bomo Kranj. Iz Kranja vodijo poti v Tržič (8), Ljubljano (2), Škofjo Loko (4) ter na Jesenice (3). Cene poti so različne, Primov algoritem pa predvideva, da vedno izberemo najcenejšo pot. Najcenejša pot je v tem primeru do Ljubljane, saj nas stane le 2 milijona. Odprejo se nam nova mesta, ki jih moramo povezati. Nova mesta dodamo prejšnjim in popravimo cene, če sedaj obstajajo nižje cene do teh mest. Nov seznam je tako: Tržič (8), Jesenice (3), Škofja Loka (4), Vrhnika (2), Velike Lašče (5), Grosuplje (2), Celje (3) in Kamnik (2).

V graf najprej povežemo Vrhniko, potem Grosuplje in nazadnje še Kamnik. Tako imamo povezana mesta: Ljubljana, Grosuplje, Vrhnika, Kranj in Kamnik, dobimo pa veliko novih mest, ki jih dodamo v seznam in obstoječim popravimo cene. Nov seznam je tako: Tržič (3), Jesenice (3), Škofja Loka (4), Velike Lašče (4), Celje (3), Novo mesto (8) in Postojna (3). Najnižja cena v seznamu je sedaj 3, tako v drevo povežemo Tržič, Jesenice, Postojno ter Celje in posodobimo seznam, ki sedaj vsebuje: Škofja Loka (4), Velike Lašče (4) Novo mesto (8), Maribor (2), Ilirska Bisrtrica (3) in Koper (4).

Najnižjo ceno ima sedaj Maribor, ki ga povežemo v drevo. Cene do ostalih krajev se nikjer ne znižajo, zato dvignemo mejo na 3 milijone in v drevo povežemo še Ilirsko bistrico. V seznam dodamo Kočevje (8) in popravimo ceno do Kopra, ki je sedaj 2 milijona. Najnižja cena je tako 2 milijona, zato v drevo povežemo še Koper. Naslednja najnižja cena je 4 milijone, tako v drevo povežemo Škofjo Loko in Velike Lašče. Popravimo cene v seznamu, ki sedaj vsebuje: Novo mesto (8) in Kočevje (3). Najprej v drevo povežemo Kočevje, kar spremeni ceno Novemu mestu na 6 milijonov. Na koncu povežemo še Novo mesto in tako dobimo ceste, ki jih je potrebno asfaltirati, da se lahko pripeljemo v vsako slovensko mesto.

Rešitev so povezave, ki so na sliki 6.7 označene z rdečo barvo. Ko seštejemo vse cene, ugotovimo, da za asfaltiranje cest potrebujemo 39 milijonov €.



Slika 6.7: Skica zemljevida Slovenije s cenami (v milijonih €) za asfaltiranje cest med kraji.

6.4.2 Primer naloge s tekmovanja Bober

Slika 6.8: Naloga "Telefonska mreža" s tekmovanja Bober.

Predpostavimo, da začnemo v brlogu skrajno desno, saj je možna le ena povezava med obema brlogoma. V naslednjem koraku izberemo edini možni brlog, ki je oddaljen 6 metrov. Iz tega brloga imamo dve povezavi z utežjo 2 metra in eno povezavo z utežjo 1 meter, izberemo slednjo. Z dodanim brlogom se odprejo nove povezave z utežmi 2, 4 in 5 metrov, ki se dodajo povezavama z utežema 2 metra. Izberemo dve povezavi z utežema 2 metra (eno, ki povezuje notranji brlog, ter drugo, ki povezuje brlog levo od notranjega). V naslednjem koraku izberemo povezavo do brloga, ki je oddaljen 4 metre, ter dodamo novo povezavo do brloga, ki je oddaljen 3 metre. V zadnjem koraku dodamo še brlog (desno spodaj), ki je oddaljen 3 metre.

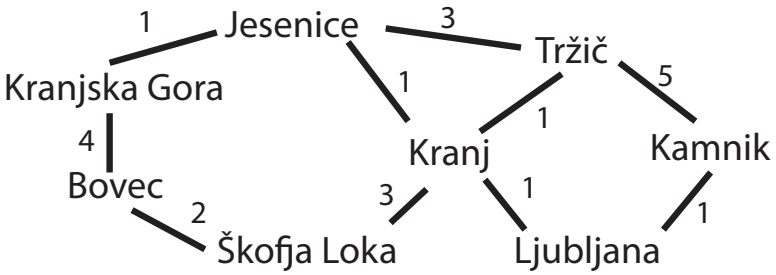
Bobri za svojo telefonsko mrežo potrebujejo 18 metrov žice.

6.5 Iskanje najkrajših poti v grafu

Nekaj o iskanju najkrajših poti smo povedali že v poglavju o iskanju v širino 6.3, vendar smo tam zemljevid obravnavali kot drevo. V tem poglavju bomo zemljevid obravnavali kot graf, ki ima na svojih povezavah ceno poti med dvema točkama. Ta cena lahko predstavlja čas, da nekaj opravimo, razdaljo med dvema krajema, ali kaj tretjega. Na sliki 6.9 je primer grafa, ki ima na povezavah napisano ceno povezave.

Za reševanje problema iskanja najkrajše poti v grafu se uporabljata dva algoritma:

- **Dijkstrov algoritem** - določimo izhodiščno vozlišče. V nadaljevanju računamo najkrajšo pot do vozlišč. Ponavljamo, dokler ne povežemo vseh vozlišč.
- **Bellman - Fordov algoritem** - deluje tako kot Dijkstrov algoritem, z razliko, da so vrednosti uteži pri povezavah lahko negativne.



Slika 6.9: Skica zemljevida Slovenije s cenami povezav med kraji.

6.5.1 Postopek

Razložili bomo Dijkstrov algoritem, pri tem pa si bomo pomagali z grafom na sliki 6.9. Poiskali bomo najkrajšo razdaljo med Kamnikom in Bovcem, hkrati pa izračunali tudi razdalje iz Kamnika do vseh ostalih mest na zemljevidu.

Na sliki 6.10 je prikazan celoten algoritem po korakih:

1. Začnemo v Kamniku, do katerega je razdalja 0. Do mejnih točk, Ljubljane in Tržiča, so cene 1 in 5.

2. Točko Ljubljana, ki je oddaljena najmanj, označimo za obiskano in tako dobimo novo točko Kranj, ki je od Ljubljane oddaljena za 1. Do Kranja je sedaj cena 2 (1 od Kamnika do Ljubljane + 1 od Ljubljane do Kranja).

3. Med obiskane točke dodamo Kranj, ter dodamo dve novi potencialni točki: Jesenice, s ceno 3, in Škofjo Loko, s ceno 5. Ob tem, ko smo povezali Kranj, moramo popraviti tudi ceno do Tržiča, ki je bila prej 5, sedaj pa je le 3 (Kranj s ceno 2 + cena povezave s Tržičem 1).

4. Sedaj imamo dve točki s ceno 3 (Jesenice in Tržič). Poljubno izberemo eno, saj izbira ne bo vplivala na dolžino poti, temveč le na pot, po kateri bomo prišli do cilja. Izberemo Tržič, ki ga označimo za obiskanega in mu dodelimo končno ceno 3.

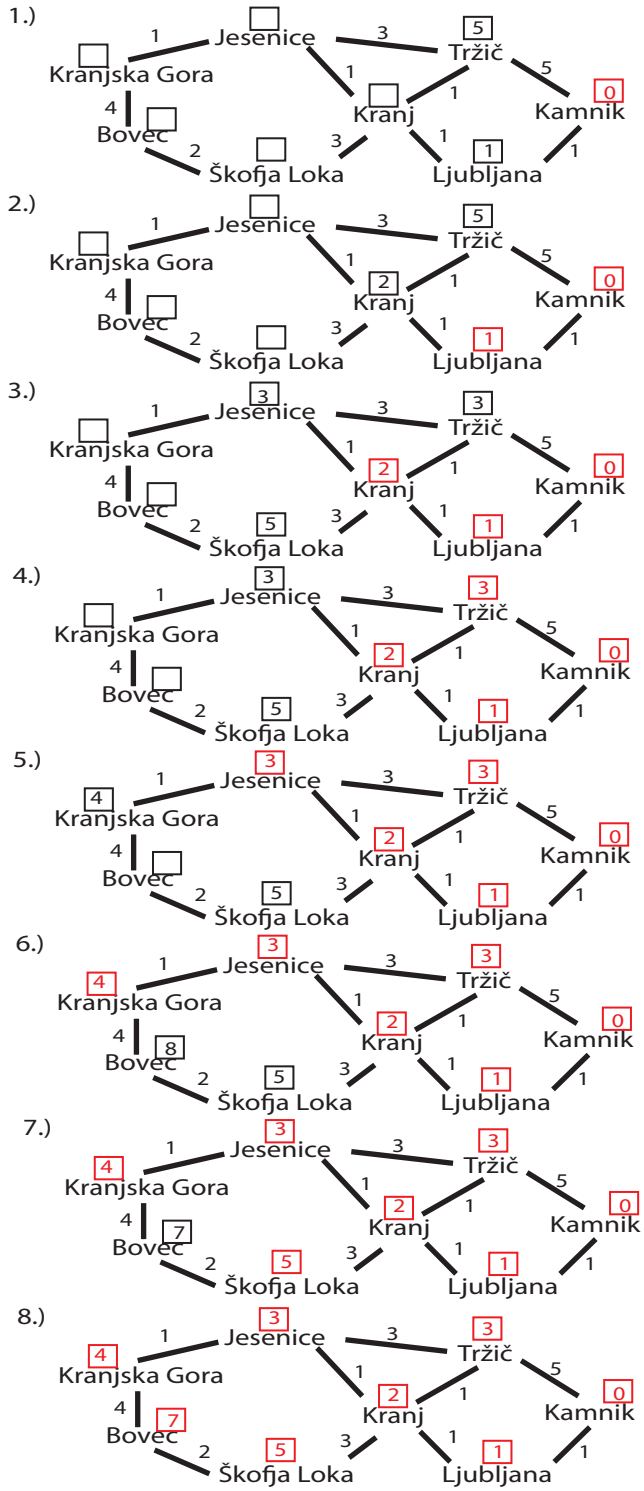
5. Med možnimi točkami (Jesenice - 3 in Škofja Loka - 5) izberemo najcenejšo, Jesenice, in jo prestavimo med obiskane. Dobimo novo mejno točko, Kranjsko Goro, s ceno 4.

6. Kranjska Gora ima med vsemi najnižjo ceno, zato jo prestavimo med obiskane in tako odkrijemo Bovec, ki ima ceno 8.

7. Kljub temu, da smo že našli naš cilj, pa ga še ne bomo izbrali, saj se moramo držati algoritma in vedno izbrati najcenejšo pot. Utegne se namreč zgoditi, da se bo do ciljnega kraja našla druga, krajša pot.

Tako izberemo pot do Škofje Loke, s ceno 5. Povezava med Škofjo Loko in Bovcem ima ceno 2, zato moramo Bovcu popraviti ceno in je ta sedaj 7.

8. Med obiskane dodamo še Bovec in dobimo najkrajšo pot iz Kamnika do Bovca. Pot gre preko Ljubljane in Škofje Loke.



Slika 6.10: Prikaz iskanja najkrajših poti v grafu s pomočjo Dijkstrovega algoritma.

6.5.2 Primer naloge s tekmovanja Bober

Slika 6.11: Naloga "Ben se vrača" s tekmovanja Bober.

Z Benom imamo na začetku dve možnosti, lahko izberemo levo pot s ceno 3 ali desno pot s ceno 6 minut. Ker je pot s ceno 3 minute cenejša, najprej razvijemo to. V naslednjem koraku imamo še vedno na voljo pot s ceno 6, ter novo pot s ceno 3+2 minut. Ker je 3+2 manjše od 6, izberemo prvo in razvijemo vozlišče, ki ima sedaj ceno 5 minut. Odpreta se nam dve novi možnosti, pot s ceno 3 ter pot s ceno 5 minut. Spet izberemo cenejšo in tako dobimo vozlišče s ceno 8 minut. Nadaljujemo podobno, na voljo imamo dve možnosti, pot po poti 5+5 ali po poti 8+1 minut, izberemo slednjo. V nadaljevanju imamo na izbiro možnosti 4 in 1 minute. Izberemo cenejšo in tako razvijemo novo vozlišče, ki je od začetnega oddaljeno za 10 minut. Za

nadaljevanje sta nam na voljo dve poti 10+2 ali 9+4, ker je 12 manjše od 13 izberemo prvo. Na koncu imamo samo eno izbiro, tako razvijemo še zadnje vozlišče. Ben bo za pot do doma porabil 14 minut.

Naloge, podobne tej, so na tekmovanju Bober kar pogoste. Navadno so sestavljene tako, da jih je možno rešiti tudi z opazovanjem, brez poznavanja algoritma. Ponavadi imajo zemljevidi ozka grla, točke preko katerih moramo iti, tako, da iščemo najkrajše poti po odsekih, kar je bistveno hitreje.

Poglavje 7

Regularni izrazi in avtomati

V tem poglavju je razloženo kako delujejo regularni izrazi, s pomočjo katerih računalnik išče vzorce v besedilu ali preverja pravilnost vašega gesla. Na primeru bančnega avtomata pa bomo razložili kako delujejo končni avtomati.

7.1 Regularni izrazi

Z regularnimi izrazi, v angleščini *Regular expressions*, opišemo splošnejši vzorec zaporedja simbolov, ki nam ustreza (ali pa tudi ne).

Regularni izrazi se v računalništvu uporabljajo za iskanje besed in vzorcev v besedilu ter manipulacijo z njimi. Uporabljajo se tudi za preverjanje pravilnosti vnosa podatkov v spletne obrazce.

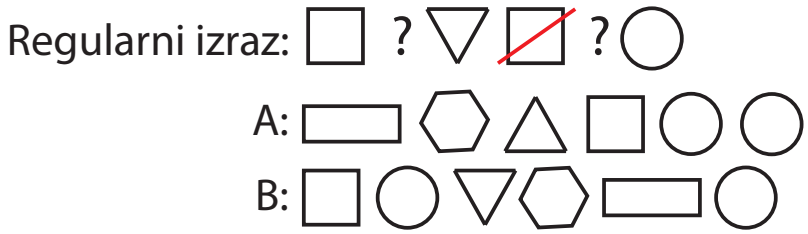
7.1.1 Postopek

Delovanje regularnih izrazov bomo prikazali s preprostim preverjanjem pravilnosti zaporedja geometrijskih likov. Regularni izraz in testna primera so prikazani na sliki 7.1.

Regularni izraz po vrsti prikazuje zaporedje likov, kateremu mora zaporedje ustrezati, da ga regularni izraz potrdi kot pravilnega. Pravilni vrstni red likov je: štirikotnik, poljuben lik, trikotnik, katerikoli lik razen štirikotnika, poljuben lik ter krožnica.

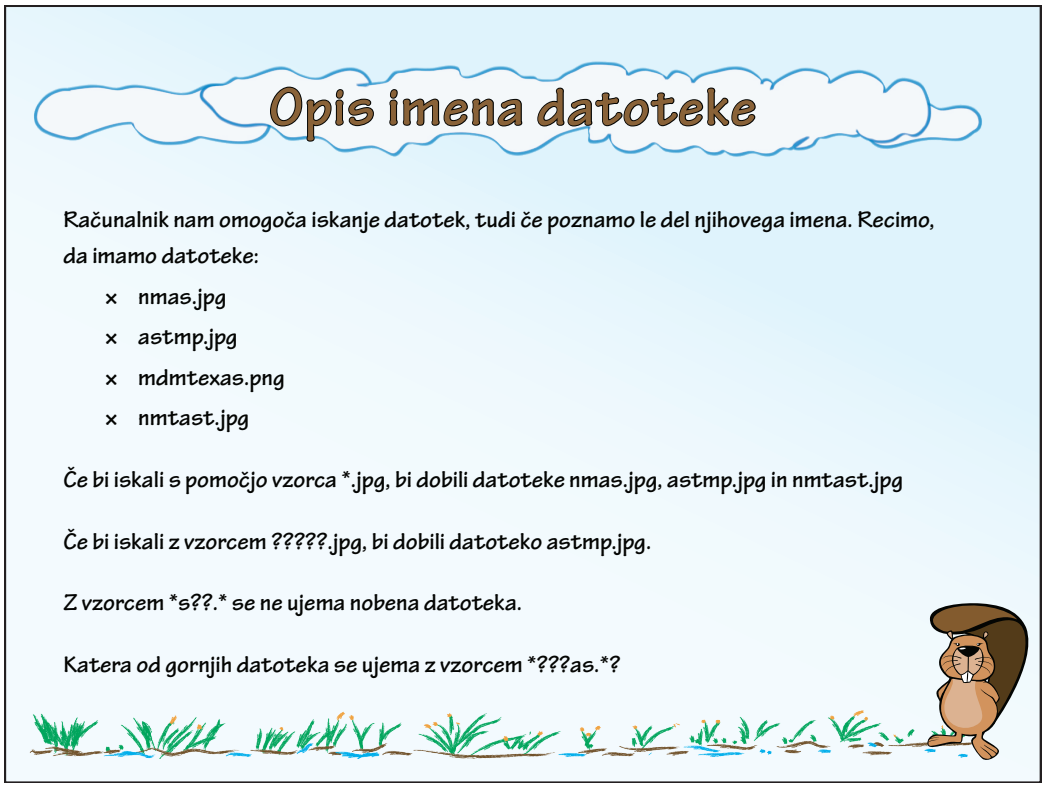
Prvi testni primer se začne s pravokotnikom, ki je štirikotnik, nadaljuje pa se s poljubnim likom, ki je v tem primeru šestkotnik. Naslednji lik je trikotnik, kateremu sledi kvadrat. Kvadrat je štirikotnik kar pomeni, da je izraz napačen, saj bi na tem mestu moral stati katerikoli lik razen štirikotnika. Če vseeno nadaljujemo naprej, sledita poljuben znak, ki je v tem primeru krog, ter še en krog, ki je tudi v regularnem izrazu krog.

Drugi testni primer se začne s kvadratom, ki je štirikotnik in se nadaljuje s krogom, ki je v tem primeru poljuben simbol. Naslednji simbol je trikotnik, tako kot tudi v regularnem izrazu, sledi pa mu šestkotnik, kar je pravilno, saj je pogoj, da ne sme biti štirikotnik. Naslednja lika sta pravokotnik, ki je v tem primeru poljuben lik, ter krog, ki ustreza regularnemu izrazu.



Slika 7.1: Primer regularnega izraza z dvema zaporedjema.

7.1.2 Primer naloge s tekmovanja Bober



Slika 7.2: Naloga "Opis imena datoteke" s tekmovanja Bober.

Naloga opisuje, kako lahko v računalniku iščemo datoteke s pomočjo regularnih izrazov. Znak za vprašaj predstavlja poljubno črko, znak za zvezdico pa poljubno število poljubnih črk. Vzorec tako zahteva poljubno število črk, kateremu sledijo tri poljubne črke, potem črki a in s ter piko, ki ji sledi še poljubno število črk. Reševali bomo tako, da bomo za vsako datoteko posebej preverili ali se ujema z vzorcem ali ne.

Ko s podanim regularnim izrazom primerjamo prvo datoteko, ugotovimo, da ne ustreza vzorcu, saj je prekratka. Dana datoteka ima pred črkama a in s le dve črki, kar je premalo.

Druga datoteka prav tako ne ustreza podanemu regularnemu izrazu, saj ne vsebuje zaporedja *as.*, ki ga zahteva regularni izraz.

Tretja datoteka se začne s poljubnim številom črk, v tem primeru tri črke, ki jim sledijo tri obvezne črke pred zaporedjem *as.*. Zaporedju sledi še poljubno število črk, ki so v tem primeru p, n in g. To ime datoteke ustreza danemu regularnemu izrazu.

Četrta datoteka, tako kot druga, ne vsebuje zaporedja *as.*, kar pomeni, da ime datoteke ne ustreza danemu regularnemu izrazu.

7.2 Končni avtomati

Končni avtomati, v angleščini *Finite-state machine*, se uporabljajo v primerih, ko mora računalnik bodisi prebrati zaporedje simbolov in ugotoviti ali zaporedje predstavlja dovoljeno kombinacijo ali ne, bodisi izvesti akcijo, ki je odvisna od zaporedja simbolov, pritiskov na gumbe ipd. Je stroj, ki za svoje delovanje uporablja neke vrste zemljevid, po katerem se premika med preverjanjem simbolov.

Zemljevid predstavimo z usmerjenimi grafi. Vozlišča grafov so stanja avtomata, povezave med vozlišči pa so pravila, po katerih lahko napredujemo v naslednje stanje. V avtomatu je eno ali več vozlišč označeno kot končno stanje avtomata; ponavadi je obkroženo dvakrat.

7.2.1 Postopek

Primer končnega avtomata je bančni avtomat. Avtomat je prikazan na sliki 7.3. Recimo, da želimo v tujini s svojega bančnega računa dvigniti 20 €.

Avtomat v tem primeru deluje na naslednji način:

1. Najprej v avtomat vstavimo kartico. Avtomat preveri ali je kartica domača ali tuja in temu primerno nadaljuje.

2. Avtomat ugotovi, da smo tujci in nas vpraša kateri jezik želimo.

3. Ko izberemo jezik, nadaljujemo na vnos številke PIN.

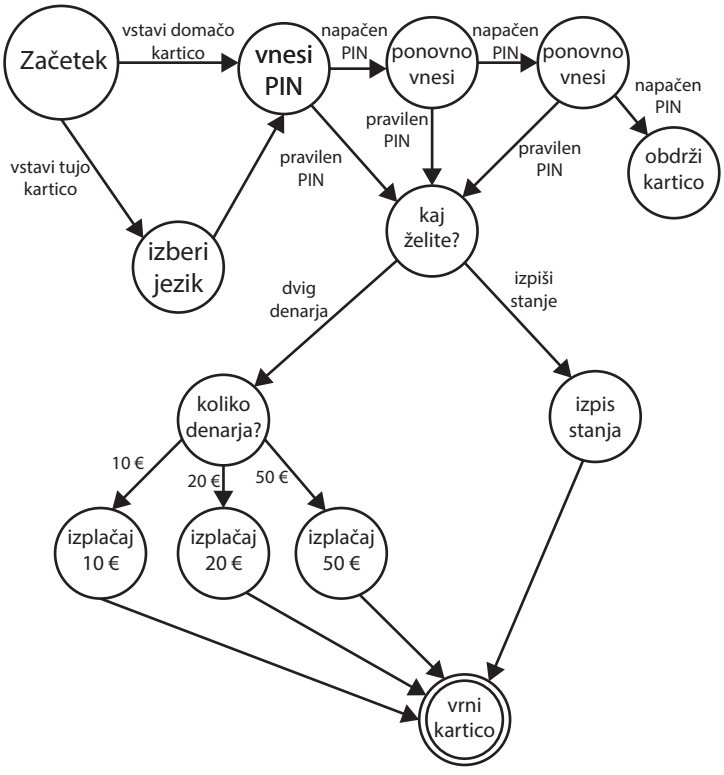
4. Za vnos pravilne številke imamo tri poizkuse, potem avtomat obdrži našo kartico.

5. Ko vnesemo pravilno številko, nas avtomat spusti naprej, kjer izberemo ali želimo dvigniti denar ali le preveriti stanje.

6. Izberemo dvig denarja. Avtomat nam na izbro ponudi zneske: 10, 20 in 50 €.

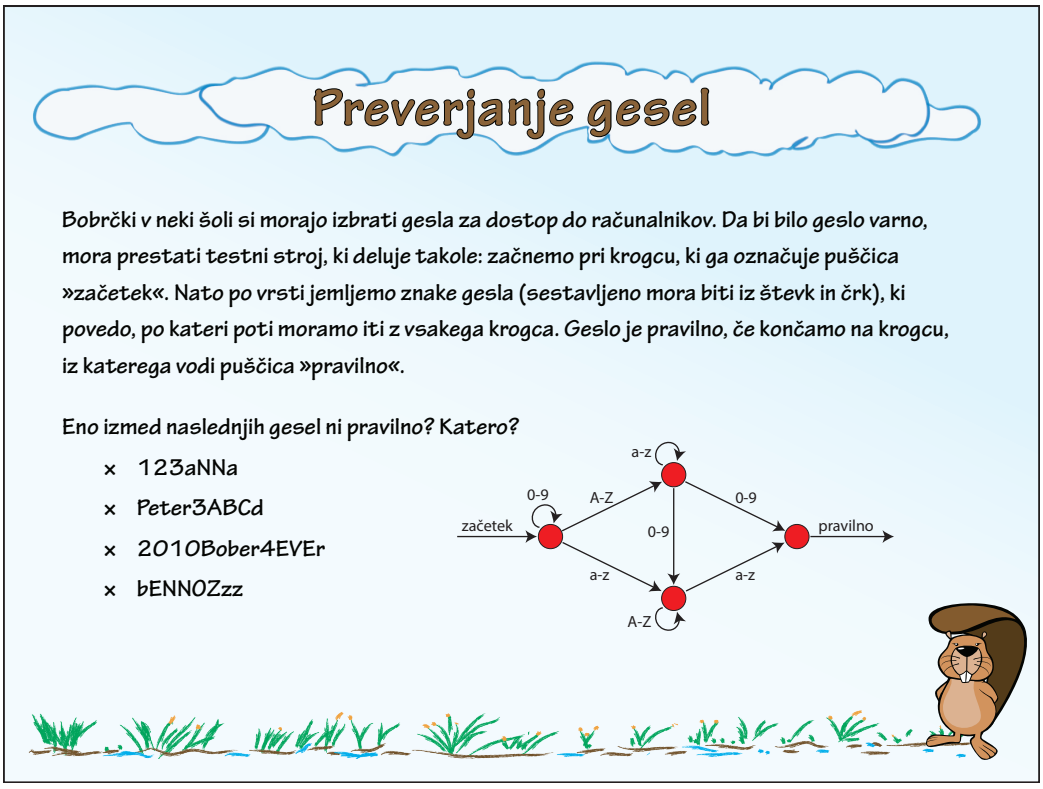
7. Izberemo dvig 20 €, avtomat nadaljuje na izplačilo želenega denarja.

8. Na koncu nam avtomat vrne kartico.



Slika 7.3: Primer končnega avtomata za bančni avtomat.

7.2.2 Primer naloge s tekmovanja Bober



Slika 7.4: Naloga "Preverjanje gesel" s tekmovanja Bober.

Nalogo s sestavljanjem gesel bomo rešili na dva načina. Pri prvem načinu bomo za vsako podano geslo pogledali, ali nas pripelje do konca avtomata, pri drugem pa bomo poskušali najti pravila, po katerih se preverjajo gesla in tako ugotovimo ali je geslo pravilno ali ne.

1. Začnemo s prvim geslom, ki se začne s številko. To je pravilno, saj lahko ostanemo v prvem stanju, če imamo številko. Drugi in tretji simbol sta prav tako številki, tako ostanemo v prvem stanju. Naslednji simbol je mala črka, kar pomeni, da napredujemo v spodnje stanje. Sledita dve veliki črki, kar je pravilno, saj ostajamo v istem stanju. Zadnji simbol

je mala črka, s katero napredujemo na konec grafa, kar pomeni, da smo dobili pravilno zaporedje.

Drugo geslo se začne z veliko črko, kar pomeni, da napredujemo v zgornje stanje. Sledijo štiri majhne črke, kar je pravilno, ker ostajamo v istem stanju. Naslednji simbol je številka in ker ni zadnji simbol, napredujemo na spodnje stanje. V tem stanju potrdimo še tri velike črke. Zadnji simbol je majhna črka, ki nas pripelje na konec grafa, zaporedje je pravilno.

Nadaljujemo s tretjim zaporedjem, ki se začne s štirimi številkami in nadaljuje z veliko črko, kar pomeni, da napredujemo v zgornje stanje. Štirim majhnim črkam sledi številka, ki nas pripelje v spodnje stanje. V spodnjem stanju dobimo tri velike črke, zaporedje pa zaključimo z majhno črko, ki nas pripelje v končno stanje.

Zadnje zaporedjee se začne z malo črko, kar nas pripelje v spodnje stanje. Sledi pet velikih črk, kar je pravilno, saj ostajamo v istem stanju. Naslednja je majhna črka, ki nas pripelje v končno stanje, vendar moramo potrditi še eno majhno črko. Zadnje črke ne moremo potrditi, saj iz končnega stanja ni izhodov. Zaporedje ni pravilno.

2. Pravila iščemo tako, da pogledamo možna zaporedja znakov. Tako vidimo, da se zaporedje lahko začne s poljubnim številom številk ali z eno veliko ali malo črko. Opazimo tudi, da se zaporedje lahko zaključi le z eno številko ali eno majhno črko. Vmes so lahko črke poljubne velikosti ter številke, če ustrezajo pravilom, ki so podana z grafom.

   S temi pravili sedaj pregledamo zaporedja. Vidimo, da se vsa zaporedja pravilno začnejo ter razen zadnjega tudi pravilno končajo. Pri zadnjem zaporedju sta na koncu dve zaporedni majhni črki, kar ob pregledu ostalih simbolov pokaže, da je zaporedje nepravilno.

Poglavje 8

Sklepne ugotovitve

V diplomskem delu je v šestih poglavjih predstavljenih nekaj pomembnih področij računalništva. Poleg razlag so zraven tudi rešeni primeri izbranih nalog, ki so se v zadnjih letih pojavile na tekmovanju Bober. Vsebina učiteljem nudi podporo pri pripravi učencev na tekmovanje Bober.

V prihodnosti bi lahko diplomsko delo še razširili in ga učiteljem ponudili v različnih oblikah. Učitelji bi lahko gradivo uporabljali kot enkratne priprave na tekmovanje Bober, ali pa bi priprave na tekmovanje zastavili kot celoletni krožek. Ena od oblik je tudi kot predmet, vendar pa bi v tem primeru potrebovali nekaj naklonjenosti tudi s strani pristojnega ministrstva.

Naloge bi pripravili v obliki prosojnic, kot so v uporabljenem gradivu[6]. Prosojnice bi učitelji lahko uporabljali, ko bi učence pripravljali na tekmovanje ali med razlaganjem snovi na primerih. Za učitelje bi pripravili dodatna gradiva, med katere sodijo učne priprave ter opisi ozadja, podobno kot je opisano v tem diplomskem delu. Pripravili bi tudi razlago vseh nalog, ki se pojavljajo na tekmovanjih Bober, s čimer bi pridobili tako učitelji kot učenci.

Zgoraj našteto bi ponudili v obliki spletne strani. Verjetno bi bilo smiselno spletno stran zastaviti tako, da bi vsebine lahko dodajal kdorkoli, nad upravljanjem strani pa bi bdela skupina strokovnjakov, ki bi vsebine po potrebi prilagajala. Spletno stran bi organizirali po temah, z opisom ter ozadjem teme. Poleg opisa bi dodali tudi seznam literature ter seznam nalog. Pri

vsaki od nalog bi poleg opisa napisali tudi v katero od tem sodi. Pod nalogo bi dodali možnost vpogleda v rešitev ter v razlago rešitve, podobno kot je to prikazano v tem diplomskem delu. Dodali bi povezave na podobne naloge ter na dodatni material in literaturo, v kolikor bi ta obstajala.

Želimo si, da bi s tem delom ter nadaljnjim prizadevanjem pripomogli k učenju računalništva in ne le učenja uporabe računalnika.

Literatura

[1] Bratko, Ivan. Prolog in umetna inteligenca. Ljubljana: Fakulteta za računalništvo in informatiko, 1997.

[2] Cormen, Thomas H. and Thomas H. Cormen. Introduction to algorithms. Cambridge, Mass.: MIT Press, 2001.

[3] Cormen, Thomas H. Introduction to algorithms. Cambridge, Mass.: MIT Press, 2009.

[4] Computer Science Unplugged, http://csunplugged.org/, 2011.

[5] Demšar, Janez. Proč z računalniki v računalniških učilnicah. Sobotna priloga, Delo, str. 2-3, 18.08.2012. 1

[6] Demšar, Janez. bober.pdf, neobjavljeno. 2012. 2, 97

[7] En.wikipedia.org. "Selection sort - Wikipedia, the free encyclopedia."1997. http://en.wikipedia.org/wiki/Selection_sort (dostopano 22. junij 2013).

[8] En.wikipedia.org. "Insertion sort - Wikipedia, the free encyclopedia."2013. http://en.wikipedia.org/wiki/Insertion_sort (dostopano 23. junij 2013).

[9] En.wikipedia.org. "Quicksort - Wikipedia, the free encyclopedia."1978. http://en.wikipedia.org/wiki/Quicksort (dostopano 23. julij 2013).

[10] En.wikipedia.org. "Bubble sort - Wikipedia, the free encyclopedia."2003. http://en.wikipedia.org/wiki/Bubble_sort (dostopano 23. julij 2013).

[11] En.wikipedia.org. "Merge sort - Wikipedia, the free encyclopedia."1945. http://en.wikipedia.org/wiki/Merge_sort (dostopano 24. julij 2013).

[12] En.wikipedia.org. "Knapsack problem - Wikipedia, the free encyclopedia."1897. https://en.wikipedia.org/wiki/Knapsack_problem (dostopano 11. avgust 2013).

[13] En.wikipedia.org. "Finite-state machine - Wikipedia, the free encyclopedia."n.d.. https://en.wikipedia.org/wiki/Finite-state_machine (dostopano 1. avgust 2013).

[14] En.wikipedia.org. "Regular expression - Wikipedia, the free encyclopedia."2006. http://en.wikipedia.org/wiki/Regular_expression (dostopano 2. avgust 2013).

[15] En.wikipedia.org. "Bloom's Taxonomy - Wikipedia, the free encyclopedia."1956. http://en.wikipedia.org/wiki/Bloom's_Taxonomy (dostopano 12. avgust 2013). 41

[16] Hr.wikipedia.org. "Regularni izraz - Wikipedija."2013. http://hr.wikipedia.org/wiki/Regularni_izraz (dostopano 2. avgust 2013).

[17] Maribor, SERŠ. "Kriptografija."n.d.. http://www.egradiva.net/moduli/upravljanje_ik/14_kriptiranje/01_datoteka.html (dostopano 27. julij 2013).

[18] Sl.wikipedia.org. "Minimalno vpeto drevo - Wikipedija, prosta enciklopedija."2013. http://sl.wikipedia.org/wiki/Minimalno_vpeto_drevo (dostopano 19. julij 2013).

[19] Sl.wikipedia.org. "Dijkstrov algoritem - Wikipedija, prosta enciklopedija."2013. http://sl.wikipedia.org/wiki/Dijkstrov_algoritem (dostopano 19. junij 2013).

[20] Sl.wikipedia.org. "Urejanje z navadnim izbiranjem - Wikipedija, prosta enciklopedija."2013. http://sl.wikipedia.org/wiki/Urejanje_z_navadnim_izbiranjem (dostopano 22. julij 2013).

[21] Sl.wikipedia.org. "Preprosti problem nahrbtnika - Wikipedija, prosta enciklopedija."2013. http://sl.wikipedia.org/wiki/Preprosti_problem_nahrbtnika (dostopano 11. avgust 2013).

[22] Štefanc, Damjan (2004). Učni cilji. Dostopno na: http://www.stefanc.net/docs/ucni_cilji.pdf 41

[23] Sl.wikipedia.org. "Dvojiški številski sistem - Wikipedija, prosta enciklopedija."2013. https://sl.wikipedia.org/wiki/Dvoji%C5%A1ki_%C5%A1tevilski_sistem (dostopano 27. julij 2013).

[24] Vidra.si. "Učne ure — Vidra."2012. http://vidra.si/ (dostopano 3. avgust 2013).

[25] Wiki.fmf.uni-lj.si. "Končni avtomat - MaFiRaWiki."2013. http://wiki.fmf.uni-lj.si/wiki/Kon%C4%8Dni_avtomat (dostopano 1. avgust 2013).

[26] Wiki.fmf.uni-lj.si. "Huffmanovo kodiranje - MaFiRaWiki."2006. http://wiki.fmf.uni-lj.si/wiki/Huffmanovo_kodiranje (dostopano 27. julij 2013).