Vaja 6 – B-drevesa

1. Splošna predstavitev problema

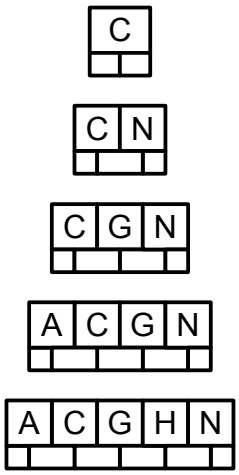
S širjenjem uporabe računalnikov se je hitro povečala količina podatkov, ki jih je bilo potrebno obdelati. Zaradi tega so se pojavile potrebe po podatkovni strukturi, ki bi omogočala hitro iskanje med podatki in njihovo urejanje, pri čemer so se iskanja osredotočila na razna drevesa, saj so ta imela najugodnejšo časovno zahtevnost pri iskanju. Težava pri klasičnih drevesih je v hitrem povečevanju njihove višine, ki neposredno vpliva na hitrost iskanja. Rešitev tega problema sta leta 1972 podala znanstvenika Rudolf Bayer in Edward M. McCreight, ki sta predstavila podatkovno strukturo B-drevo, to je popolnoma balansirano drevo, za katerega je značilna rast v širino namesto v globino. S to podatkovno strukturo je zlahka možno obvladovanje več milijonov ali celo milijard podatkov. Zaradi tega so danes B-drevesa nepogrešljiva komponenta tako pri operacijskih sistemih, kjer z njihovo pomočjo rešujemo problem datotečnih sistemov kot pri raznih podatkovnih bazah, kjer je zahtevan hiter dostop do podatkov.

Da B-drevo izpolni omenjene zahteve, se lahko zahvali posebni strukturi vozlišč. Za razliko od ostalih dreves, hrani vozlišče B-drevesa več ključev, posledično pa ima vozlišče tudi ustrezno več sinov. To hkrati pomeni, da je potrebno v vozlišču hraniti kazalce na te sinove in sicer velja, v kolikor je v vozlišču shranjenih n ključev, je v vozliščih tudi n+1 kazalcev na sinove. Vsi ključi v drevesu so urejeni v naraščajočem vrstnem redu, vozlišča pa so urejena podobno kot klasična binarna iskalna drevesa. Pri tem se vsak ključ v vozlišču obnaša ekvivalentno vozlišču binarnega iskalnega drevesa, saj sta nanj vezana levo in desno poddrevo, pri čemer velja, da so v levem poddrevesu ključi, ki so manjši od očetovega ključa, v desnem poddrevesu pa so ključi, ki so od tega večji. Značilno za B-drevo je tudi to, da je vedno popolnoma zbalansirano, ne glede na to ali v njo podatke shranjujemo ali jih iz nje brišemo. Vsi listi drevesa so poravnani na isti višini. Da je to možno zagotoviti, je potrebno določiti minimalno in maksimalno število ključev, ki jih lahko hranimo v vozliščih drevesa. Ob tem je treba povedati, da se ravno na tej točki pojavijo razhajanja med številnimi avtorji, kako ti vrednosti določiti preko nekega parametra. V večini primerov avtorji ta parameter imenujejo red drevesa, vendar pa ima ta izraz od avtorja do avtorja različen pomen, kar pa je še huje, način določitve minimuma in maksimuma ključev neposredno vpliva na implementacijo drevesa. Sami se bomo omejili na definicijo, ki jo najdemo v *Introduction to Algortihms*, avtorjev Cormen et. al.. Cormen uvaja tako imenovano *minimalno stopnjo B-drevesa*, ki jo označi s črko t. Tako velja, da je v vozlišču lahko minimalno t-1 in maksimalno 2t-1 ključev, s tem pa je določeno tudi minimalno in maksimalno število sinov vozlišča, saj je sinov vedno za enega več od ključev. Pri tem pa velja, da omejitev minimalnega števila ključev ne velja za koren drevesa. Zaradi jasnih razlogov je tako v korenu drevesa lahko tudi manj kot t-1 ključev.

Vstavljanje v drevo je seveda specifično, saj mora drevo enakomerno rasti in ostati zbalansirano. V vozlišče drevesa tako vstavljamo ključe, dokler ne dosežemo dovoljenega maksimuma. Ko je ta dosežen, se vozlišče razdeli v levega in desnega sina, sredinski ključ pa se premakne v očeta, kar zagotavlja urejenost vozlišč. V kolikor se v očetu prekorači dovoljen maksimum ključev, se razdeli tudi ta na enak način. To balansiranje se prenaša po drevesu navzgor, dokler za vsa vozlišča ne velja, da izpolnjujejo zahtevano število ključev. Glede na

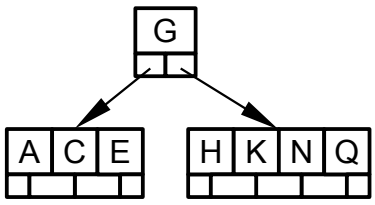
različno definicijo reda drevesa se pri nekaterih avtorjih najprej ključ v vozlišče shrani in se nato preverja, ali je bil maksimum ključev prekoračen in se potem ustrezno ukrepa, medtem ko drugi, tako kot tudi Cormen to preverjanje in delitev, če je potrebna, opravijo pred vpisom ključa. Poleg tega pa različne definicije vodijo do različnih B-dreves, ki pa vse izpolnjujejo predpisane lastnosti.

Za lažje razumevanje prej opisanega poglejmo rast drevesa z minimalno stopnjo 3, v katerega želimo shranili ključe C, N, G, A, H, E, K, Q, M, F, W, I, T, Z, D, P, R, X in Y. Glede na opisano, je minimalno število ključev, ki jih lahko hranimo v takem B-drevesu enako 2, maksimalno pa lahko v vozlišče shranimo 5 ključev. Zaradi tega lahko v drevo, preden se koren drevesa prvič razdeli, vstavimo prvih pet črk. Rast korena pred delitvijo je prikazana na sliki 1.



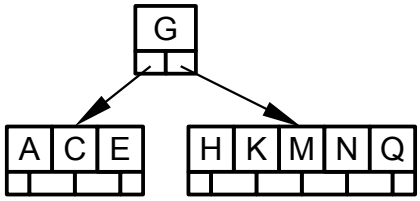
Slika 1: Rast B-drevesa ob vnosu črk C, N, G, A in H

Ker pri prvih petih črkah nismo prekoračili dovoljenega števila ključev v vozlišču, zato tudi ni bilo veliko dela ob vnosu. Bolj zanimivo postane ob vnosu črke E. Za njo v korenu ne bi bilo prostora, zato koren razdelimo ob sredinski črki G na dva sina. V levem sinu ostaneta črki A in C, v desnega sina pa vpišemo črki H in N, črka G pa se pomakne nivo više in tvori novi koren. Sedaj je v tako nastalih vozliščih dovolj prostora, da lahko vstavimo črko E. Ker je črka E v abecedi pred G, se črka urejeno zapiše v levega sina. Pri vpisu črke K ni težav, saj je prostora dovolj, ker je črka v abecedi za črko G, jo uvrstimo v desnega sina, prav tako pa tudi naslednja črka Q. Nastalo drevo je prikazano na sliki 2.



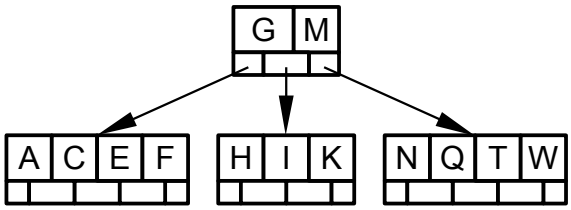
Slika 2: Drevo po vstavitvi črk E, K in Q

Drevo se bistveno ne spremeni niti ob vpisu črke M. Ker je tudi ta v abecedi za črko G, jo poskusimo vpisati v desnega sina in ker je tam še prostor za en ključ, M vpišemo na ustrezno mesto v desnem sinu. Še enkrat je potrebno poudariti, da so črke v vozliščih urejene, kar lahko preverimo tudi na sliki 3, kjer je prikazano drevo po vpisu črke M.



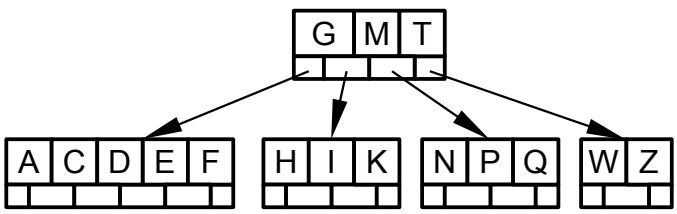
Slika 3: Drevo po vstavitvi črke M

Drevo se bistveno ne spremeni niti po vpisu črke F, saj je ta v abecedi pred črko G, zato jo vpišemo v levega sina. Ker je tam še dovolj prostora vpis te črke strukture drevesa ne spremeni. Temu pa ni tako ob vpisu črke W, ki se mora vpisati v desnega sina, vendar tam ni več prostora. Zato se desni sin razdeli v dva dela in sicer v novo nastalem levem vozlišču ostaneta tako črki H in K, črki N in Q pa se vpišeta v novo nastalega desnega sina. Sredinska črka M se premakne en nivo više, to je v koren drevesa, kjer se vpiše za črko G. B-drevo sedaj dobi tri sinove in sicer skrajno levega s ključi A, C, E in F, sredinskega s ključi H in K, ter skrajno desnega s ključi N in Q in prav v tega sina se vpiše tudi črka W, ki je v abecedi za črko M. Ker se naslednja črka, ki jo vpisujemo v drevo, to je črka I, v abecedi nahaja med črkama G in M, jo vpišemo v sredinskega sina med črki H in K. Črko T pa shranimo v skrajno desnega sina med črki Q in W. Nastalo B-drevo lahko vidimo na sliki 4.



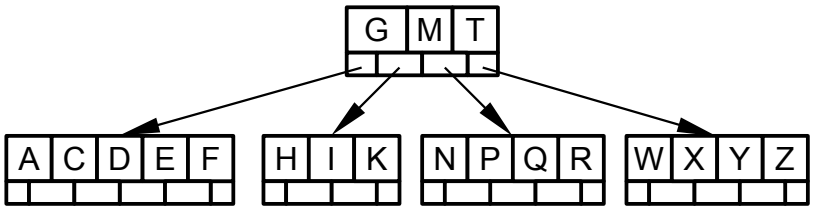
Slika 4: Drevo po vstavitvi črk F, W, I in T

V naslednji seriji bomo vnesli v drevo črke Z, D in P. Ker je črka Z v abecedi za M, se vpiše v skrajno desnega sina, v katerem je še dovolj prostora, zaradi česar se drevo bistveno ne spremeni. Enako na strukturo drevesa ne vpliva vpis črke D, saj ta leži pred črko G in se tako vpiše v skrajno levega sina, kjer je še dovolj prostora. Nekaj več dela pa nas čaka ob vpisu črke P, ki v abecedi leži za M in bi jo tako morali vpisati v skrajno desnega sina, kjer pa ni več prostora. Zato skrajno desno vozlišče razbijemo na dva dela. V levem sinu ostaneta črki N in Q, v desnega pa se vpišeta črki W in Z. Sredinska črka T se premakne en nivo više, to je v koren, medtem ko novo nastala levi in desni sin postaneta desna sinova B-drevesa. Ta ima sedaj štiri sinove in sicer od leve proti desni ima prvi sin ključe A, C, D, E in F, v drugem sinu imamo ključe H, I in K. V tretjem sinu imamo ključa N in Q, ter v zadnjem četrtem sinu še ključa W in Z. Po tej delitvi lahko brez težav v drevo vstavimo črko P in sicer v tretjega sina, saj leži med črko M in T. Rezultat opisanih operacij lahko vidimo na sliki 5.



Slika 5: Delitev desnega lista po vstavitvi črke Z, D, P

Kot zadnjo vstavimo še trojico črk R, X in Y. Črka R leži med črkama M in T, zato R vstavimo v tretjega sina za črko Q, kjer je dovolj prostora in to ne vpliva na samo strukturo drevesa. Za obe črki X in Y velja, da ležita za črko T in se zato vstavita v zadnjega, četrtega sina, kjer je tudi v obeh primerih dovolj prostora, tako da se zgolj skrajno desni list ustrezno razširi. Končno drevo, ki smo ga dobili po vstavitvi vseh črk je prikazano na sliki 6. Kljub temu, da smo v drevo vstavili kar 19 črk, so vsi listi ostali na globini 1, kar omogoča izjemno hitro iskaje, ne glede na to, kateri ključ iščemo.



Slika 6: Končno drevo po vstavitvi zadnje trojice črk R, X in Y

Še nekoliko bolj zahtevna operacija od vstavljanja ključev v drevo, je njihovo odstranjevanje iz drevesa, saj moramo ves čas ohranjati balansirano strukturo B-drevesa. Brisanje poteka tako, da najprej poiščemo ključ, ki ga želimo brisati. Nato pa imamo več možnosti, glede na to ali leži ključ v listu, v notranjem vozlišču, ali v korenu drevesa.

V kolikor je ključ v listu drevesa, je način brisanja naslednji:

1. izbrišemo želeni ključ in če v listu ostane dovolj elementov, da zadostijo zahtevanemu minimumu ključev, smo s postopkom brisanja končali.
2. V kolikor ostane v listu premalo ključev, je potrebno dodaten ključ poiskati pri neposrednem levem ali desnem bratu. Če dodaten ključ pri bratu obstaja, vzamemo ustrezen ključ od očeta (ključ med obema bratoma) ter ga premaknemo v list, izpraznjeno mesto v očetu pa nato nadomesti ključ iz brata, ki ga premaknemo navzgor.
3. V kolikor ostane v listu premalo ključev in ne neposredni desni in ne levi brat nimata kakšnega dodatnega ključa, je potrebno brata združiti. To naredimo tako, da iz očeta premaknemo ključ med obema sinoma v list z manjkajočim ključem, nato pa k temu priključimo brata. Pri tem se nam lahko zgodi troje:
    - V očetu ostane dovolj ključev in omejitvam je zadoščeno. V tem primeru s postopkom končamo.
    - V očetu ne ostane dovolj ključev, a ima neposredni levi ali desni brat še kakšen dodaten ključ. V tem primeru izvedemo točko 2 nad očetom.
    - V očetu ne obstaja dovolj ključev pa tudi v njegovem neposrednem levem in desnem bratu ne. V tem primeru združimo očeta z njegovim bratom, tako da na nivoju očeta izvedemo točko 3.

Druga možnost je ta, da želimo izbrisati ključ iz notranjega vozlišča. Za razliko od ključa v listu, ključa v vmesnem vozlišču ne moremo kar tako izbrisati, saj se vsak ključ v vozlišču obnaša kot nekakšno vozlišče binarnega iskalnega drevesa, zaradi tega tudi brisanje poteka na analogen način in sicer imamo naslednje možnosti:

1. Denimo, da je v neposrednem levem sinu dovolj ključev, da tudi po odstranitvi enega izmed njih vozlišče še vedno izpolnjuje kriterija minimuma in maksimuma. V tem

primeru v tem sinu poiščemo predhodnika ključa, ki ga želimo odstraniti ter tega nadomestimo z njegovim predhodnikom. Posledično se neposredni levi sin skrči.
2. V kolikor v neposrednem levem sinu ni dovolj prostora, preverimo ali je dovolj ključev v neposrednem desnem sinu ključa, ki ga želimo izbrisati. V kolikor je temu tako, poiščemo v tem sinu naslednika ključa, ki ga želimo izbrisati in ta ključ enostavno nadomestimo z njegovim naslednikom. Posledično se skrči neposredni desni sin, struktura drevesa pa ostane nespremenjena.
3. Če ne neposredni levi, ne desni sin nimata dovolj ključev, da lahko izvedemo 1. ali 2. točko, oba sinova združimo, iz vozlišča očeta pa izbrišemo tako želeni ključ kot enega od kazalcev na sina, saj je po združitvi sinov ta zdaj odveč. To nam strukture drevesa ne poruši, saj je v vsakem od sinov pred združitvijo natanko t-1 ključev, če ne bi bilo tako, bi lahko izvedli eno izmed obeh predhodnih točk. Po združitvi je tako v preostalem sinu 2t-2 ključev, kar je dovoljeno stanje. V kolikor imamo v očetu pred brisanjem samo t-1 ključev, je treba po opravljenem postopku brisanja to vozlišče združiti s svojim očetom.

Ostane nam še zadnja možnost in to je brisanje ključa iz korena. Pri brisanju ključev iz korena, je postopek podoben brisanju iz notranjih vozlišč, s tem da se zamenjave rekurzivno prenašajo po drevesu navzdol. Težava se pojavi, v kolikor leži koren med sinovoma z minimalnim številom ključev. V tem primeru združimo koren in oba sina, nato pa izbrišemo še želeni ključ po enakem postopku kot pri vmesnih vozliščih.

B-drevesa so namenjena iskanju informacij pri veliki količini podatkov. Pri tem je podatkov tako veliko, da jih ne moremo vseh zapisati v pomnilnik, ampak je večina zapisana na disku, ki je cenejši, a veliko počasnejši medij, zato je pomembno, da je potrebno čim manj krat posegati na disk. To dosežemo z dovolj veliko minimalno stopnjo drevesa. Samo iskanje je zelo podobno iskanju po binarnem iskalnem drevesu, le da imamo sedaj več ključev v vozlišču, ki jih je potrebno preveriti. V odvisnosti od velikosti iskanega ključa in ključev v vozlišču, ki ga preiskujemo, z iskanjem nadaljujemo v enem od njegovih sinov. Ob dovolj veliki minimalni stopnji je teh skokov v sinove relativno malo, kar zagotavlja maksimalno hitrost iskanja.

V nadaljevanju bomo opisane postopke zapisali še v obliki psevdokoda, pri čemer se bomo osredotočili samo na vstavljanje v drevo in iskanje po drevesu, saj gre za dovolj zahtevne postopke.

2. Pomoč pri implementaciji

Preden se lotimo opisa algoritmov, moramo definirati strukturo vozlišča B-drevesa. V vozlišču bomo imeli polje za hranjenje ključev, ki ga bomo imenovali *ključ*. Kazalce na sinove bomo hranili v polju *sin*. Trenutno število ključev bomo hranili v spremenljivki *n*. Podatek ali gre pri vozlišču za list, ali notranje vozlišče, bomo hranili v boolovi spremenljivki *list*. Prvi algoritem, ki ga bomo pogledali, je tisti, ki se tudi najpogosteje uporablja nad B-drevesi, to je iskanje ključa v drevesu. Poimenovali ga bomo `B_TREE_SEARCH`, kot vhodni parameter pa bomo podali kazalec na trenutno vozlišče, ki ga preiskujemo in ključ, ki ga želimo poiskati. Psevdokod za iskanje v B-drevesu je prikazan v izpisu 1.

```
function B_TREE_SEARCH(x, k)
 begin
   i := 1;
   while i ≤ x.n and k > x.ključ[i] do
     i := i+1;

   if i ≤ x.n and k = x.ključ[i] then
     return (x,i);

   if x.list then
     return NIL;
   else
     return B_TREE_SEARCH(x.sin[i], k);
end
```

Izpis 1: Psevdokod procedure za iskanje v drevesu `B_TREE_SEARCH`

Iz izpisa 1 je razvidno, da v kolikor ključa v drevesu ni, se iskanje konča v listu drevesa, kar kaže, da je hitrost iskanja ključa neposredno odvisna od višine drevesa. V kolikor bi bilo drevo tako veliko, da bi v pomnilniku lahko imeli zgolj eno samo vozlišče, bi morali pred rekurzivnim spustom v globino, ustreznega sina najprej prebrati z diska v pomnilnik. Ker v našem primeru drevo ne bo tako veliko, ta korak ne bo potreben.
Preden pa lahko v drevesu karkoli najdemo, pa je potrebno drevo najprej generirati. Drevo gradimo s pomočjo dveh funkcij `B_TREE_CREATE` in `B_TREE_INSERT`. Funkcija `B_TREE_CREATE` se kliče samo prvič, ko je potrebno inicializirati koren drevesa, njen psevdokod je prikazan v izpisu 2.

```
function B_TREE_CREATE(T)
 begin
   x := ALLOCATE_NODE();
   x.list := true;
   x.n := 0;
   T := x;
 end
```

Izpis 2: Psevdokod procedure za iskanje v drevesu `B_TREE_CREATE`

Funkcija `B_TREE_CREATE` dobi kot vhodni parameter koren drevesa T, ki ga inicializira. Ko je koren enkrat inicializiran, se ključi v drevo dodajajo s klicem funkcije `B_TREE_INSERT`. V tej funkciji nastopa tudi konstanta *t*, to je minimalna stopnja B-drevesa, ki torej določa minimalno in maksimalno dovoljeno število ključev v vozlišču. Psevdokod funkcije `B_TREE_INSERT` je prikazan v izpisu 3.

```
function B_TREE_INSERT(T,k)
 begin
   r := T;

   if r.n = 2t-1 then
     begin
       s := ALLOCATE_NODE();
       T := s;
       s.list := false;
       s.n := 0;
       s.sin[1] := r;
       B_TREE_SPLIT_CHILD(s,1,r);
       B_TREE_INSERT_NONFULL(s,k);
     end
   else
       B_TREE_INSERT_NONFULL(r,k);
end
```

Izpis 3: Psevdokod funkcije `B_TREE_INSERT`

Iz izpisa 3 lahko vidimo, da dokler ni koren drevesa poln, vstavljamo v koren s pomočjo klica metode `B_TREE_INSERT_NONFULL`, saj je potrebno poskrbeti za to, da bodo ključi v vozlišču pravilno urejeni. V kolikor pa v korenu ni več prostora, pa je potrebno vozlišče razdeliti, kar dosežemo s klicem funkcije `B_TREE_SPLIT_CHILD`. Ko je prostor v drevesu zagotovljen, kličemo spet funkcijo `B_TREE_INSERT_NONFULL`. Psevdokod te funkcije najdemo v izpisu 4.

```
function B_TREE_INSERT_NONFULL(x,k)
 begin
   i := x.n;

   if x.list then
     begin
       while i ≥ 1 and k < x.ključ[i] do
         begin
           x.ključ[i+1] := x.ključ[i];
           i := i-1;
         end;
         x.ključ[i+1] := k;
         x.n := x.n+1;
     end
   else
     while i ≥ 1 and k < x.ključ[i] do
       i := i-1;
     i := i+1;
     if x.sin[i].n = 2t-1 then
       B_TREE_SPLIT_CHILD(x,i, x.sin[i]);
       if k > x.ključ[i] then
         i := i + 1;
     B_TREE_INSERT_NONFULL(x.sin[i],k);
end
```

Izpis 4: Psevdokod funkcije `B_TREE_INSERT_NONFULL`

V izpisu 4 vidimo, da se lahko tudi v funkciji `B_TREE_INSERT_NONFULL` zgodi, da je treba kakšno vozlišče tudi razdeliti, da naredimo prostor za nov ključ tudi v tem primeru kličemo funkcijo `B_TREE_SPLIT_CHILD`. Psevdokod te funkcije je prikazan v izpisu 5.

```
function B_TREE_SPLIT_CHILD(x,i,y)
 begin
   z := ALLOCATE_NODE();
   z.list := y.list;
   z.n := t-1;

   for j := 1 to t-1 do
     z.ključ[j] := y.ključ[j+t];

   if not y.list then
     for j:=1 to t do
       z.sin[j] := y.sin[j+t];

   y.n := t-1;

   for j := x.n+1 downto i+1 do
     x.sin[j+1] := x.sin[j];

   x.sin[i+1]:=z;

   for j:=x.n downto i do
     x.ključ[j+1] := x.ključ[j];

   x.ključ[i] := y.ključ[t];
   x.n := x.n+1;
end
```

Izpis 5: Psevdokod funkcije `B_TREE_SPLIT_CHILD`

Iz izpisa 5 je razvidno, da funkcija `B_TREE_SPLIT_CHILD` ob delitvi vozlišča podatkov, ki se prepišejo v novo nastalo vozlišče, iz originalnega vozlišču ne izbriše, ampak samo ustrezno spremeni njegov števec ključev. To lahko pri visokih maksimalnih stopnjah drevesa prihrani precej časa.

3. Zahteve naloge

Implementirati je potrebno algoritme za generiranje B-dreves in iskanje po njem. B-drevo naj bo pri tem grafično prikazano v oknu aplikacije, po vzoru slik 1-6. Uporabnik naj ima možnost določiti minimalno stopnjo, ob tem pa se vsi ključi, ki so bili vpisani v trenutnem B-drevesu zapišejo v B-drevo z novo stopnjo, slika novega drevesa pa naj zamenja staro.
V B-drevesu bomo namesto črk hranili številke, ki jih lahko uporabnik dodaja posamično eno za drugo, lahko pa jih doda iz datoteke *vhod.txt*. Kot že samo ime pove, gre v tem primeru za tekstovno datoteko, katere format je prikazan v izpisu 6.

```
5 8 24 10 7 15 22 3 6 19 17 9 37 25
```

Izpis 6: Primer datoteke vhod.txt

Razen datoteke *vhod.txt* lahko program odpre tudi druge datoteke s testnimi podatki, zapisane v enakem formatu, ki jih uporabnik izbere preko dialoga za nalaganje datotek.