Programiranje I – RIN
Računalništvo I – MA

DREVO

7. predavanje

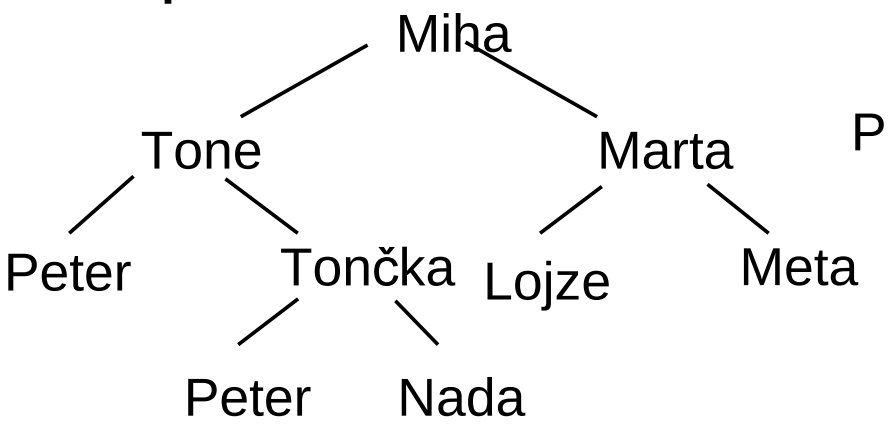
Splošno

Drevo je nelinearna podatkovna struktura;

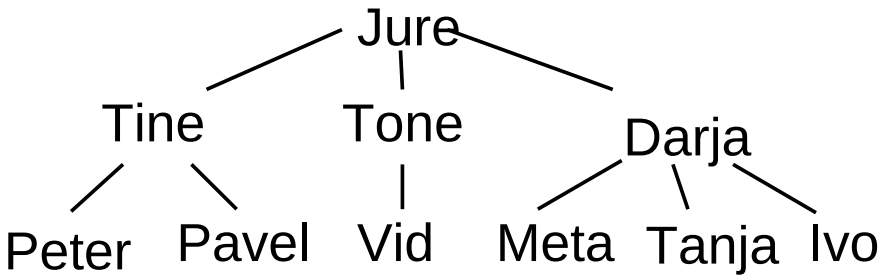
Vozlišče lahko ima več naslednikov, vendar kvečjemu enega prednika;

Primer: dve obliki rodovnika:

Drevo prednikov:

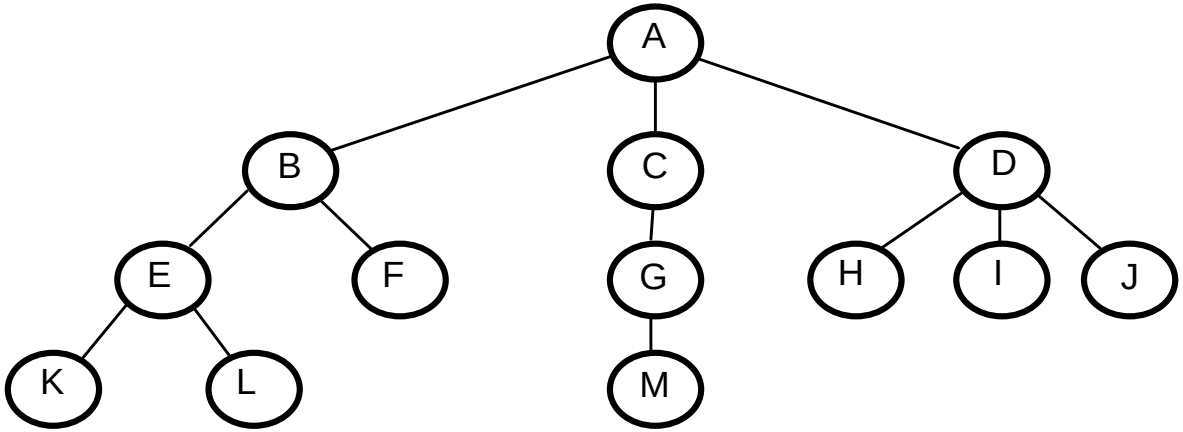


Drevo potomcev:



Predstavitev dreves

- Običajni grafični zapis in predstava:



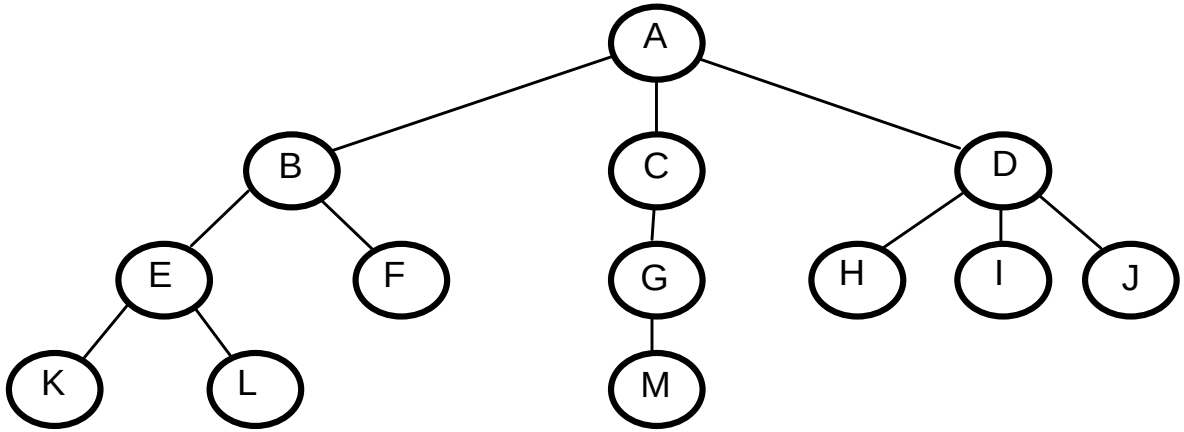
- Drevo stoji “na glavi” oz. rišemo njegove “korenine”

Terminologija

- **vozlišče**: element drevesa; ima podatek (ključ) in informacijo o njega izhajajočih poddrevesih;
- **listi** (končna vozlišča): vozlišča brez potomcev;
- **Koren:** vrhnje vozlišče, vozlišče brez predhodnika
- **notranja vozlišča**: vsa razen listov;
- **sin** (ali naslednik): koren poddrevesa nekega vozlišča; v::=sin(v);
- **oče** (ali predhodnik): obratno; v::=oče(v);
- **bratje**: koreni poddreves istega vozlišča; vozlišča z istim očetom
- **predniki**: vozlišča na poti od korena do nekega vozlišča;

Vozlišče

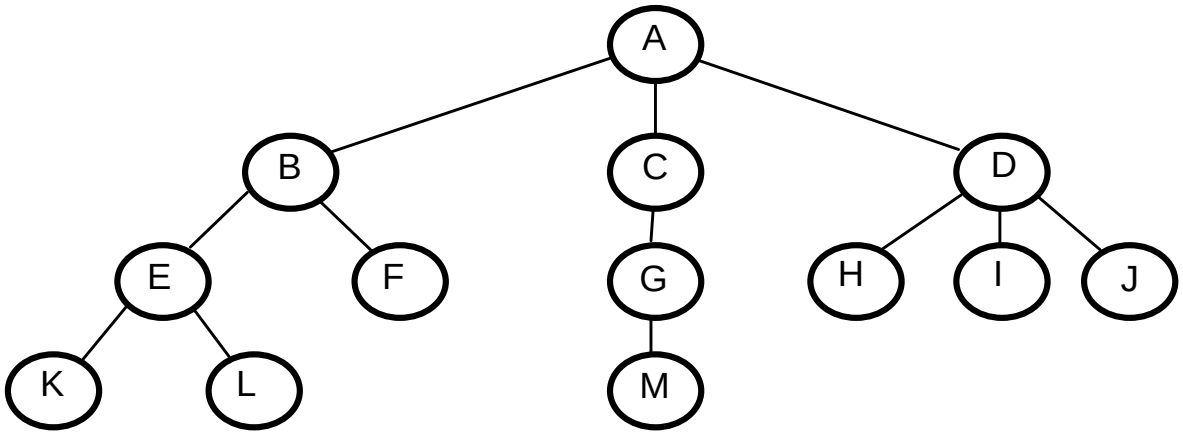
- Vsi “krogci”. V vozliščih hranimo podatke, ter vzdržujemo strukturo



- Vozlišča: A, B, C, D, ...

Listi / koren

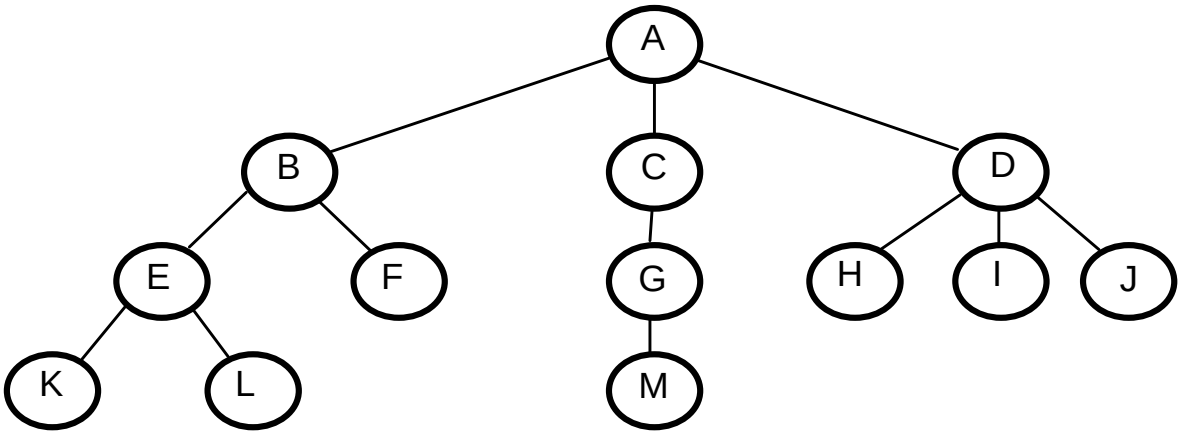
- Listi: vozlišča brez potomcev
- Koren: vozlišče brez predhodnika



- Listi: K, L, F, M, H, I in J
- Koren: A

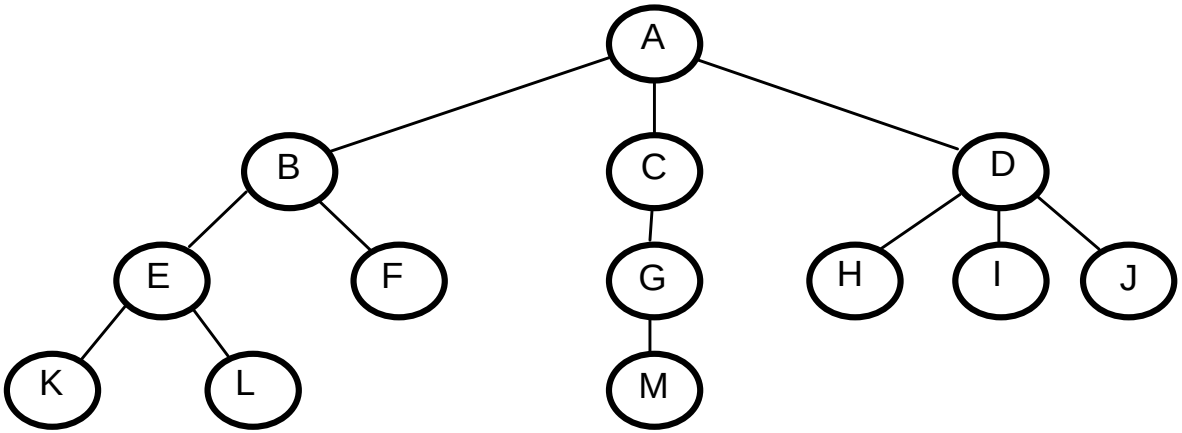
sin / oče

- Vozlišče B je sin vozlišča A in oče vozlišč E in F
- Vozlišče A nima očeta (je koren) in ima sinove B, C in D.



Bratje, predniki

- Vozlišča B, C in D so bratje (njihov oče je A)
- Predniki vozlišča L so E, B in A.

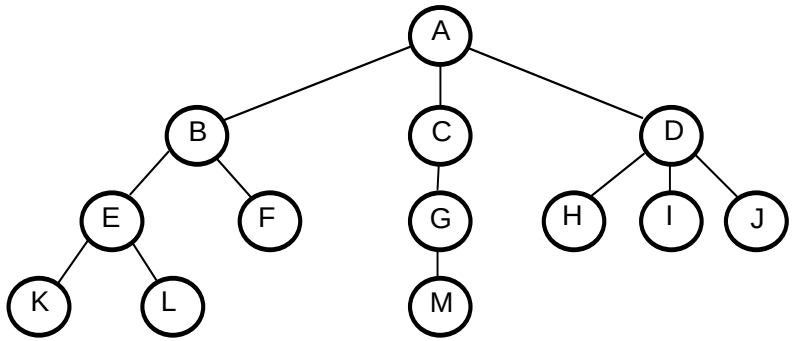


Pojmi

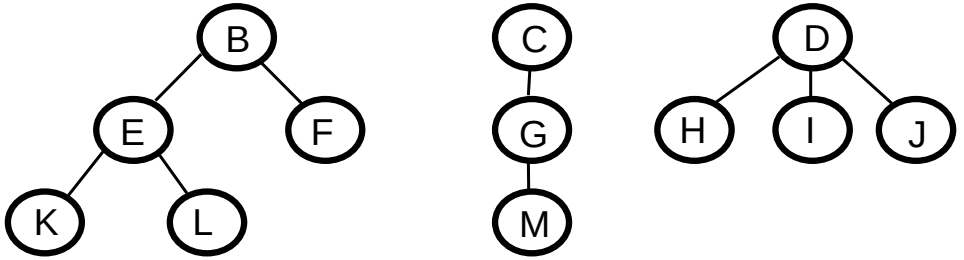
- **poddrevo**: celotno drevo, ki izhaja iz vozlišča
- **stopnja vozlišča**: podana s številom iz njega izhajajočih poddreves (število neposrednih naslednikov);
- **stopnja drevesa**: najvišja stopnja njegovih vozlišč;
- **nivo vozlišča**: koren ima nivo 1. Če ima oče nivo n, ima sin nivo n+1;
- **višina drevesa**: definirana z vozliščem z najvišjim nivojem;
- **gozd**: množica disjunktnih dreves.

- **urejeno drevo**: vrstni red poddreves v vsakem vozlišču je podan in pomemben.

Poddrevo



Poddrevesa vozlišča A

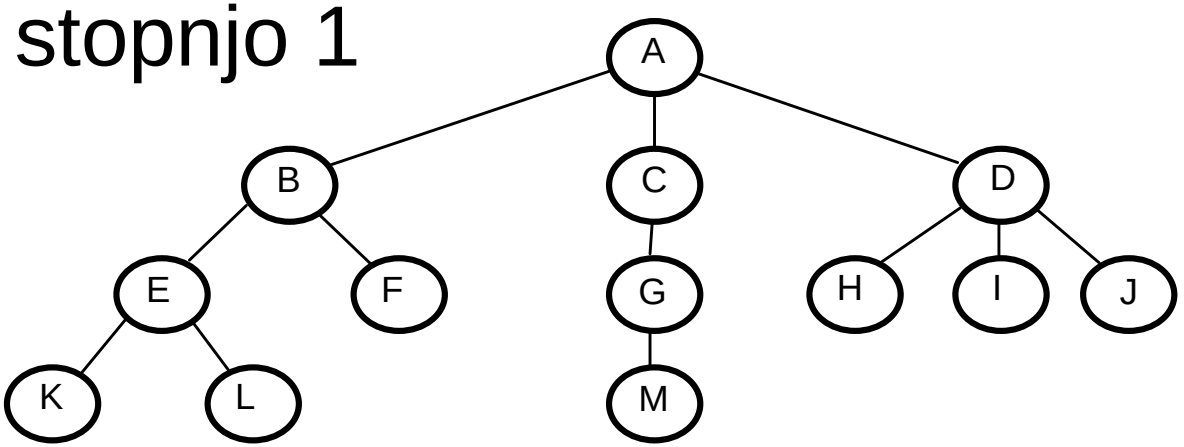


Poddrevesa vozlišča D

Poddrevesi vozlišča B

Stopnja vozlišča / stopnja drevesa

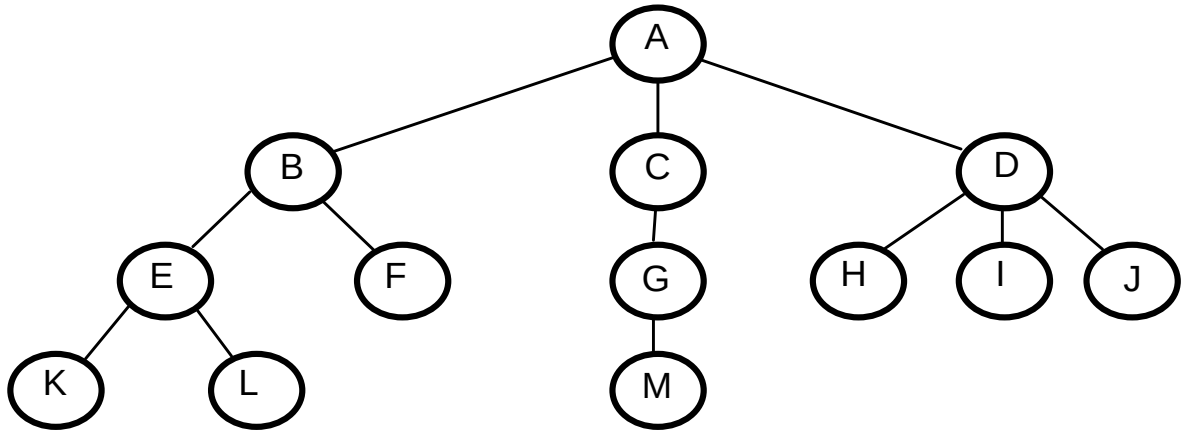
- Vozlišče A ima stopnjo 3
- Vozlišče K ima stopnjo 0 (tako kot vsi listi)
- Vozlišče C ima stopnjo 1



- Drevo ima stopnjo 3 (maksimalna stopnja vozlišč je 3) – trojiško drevo

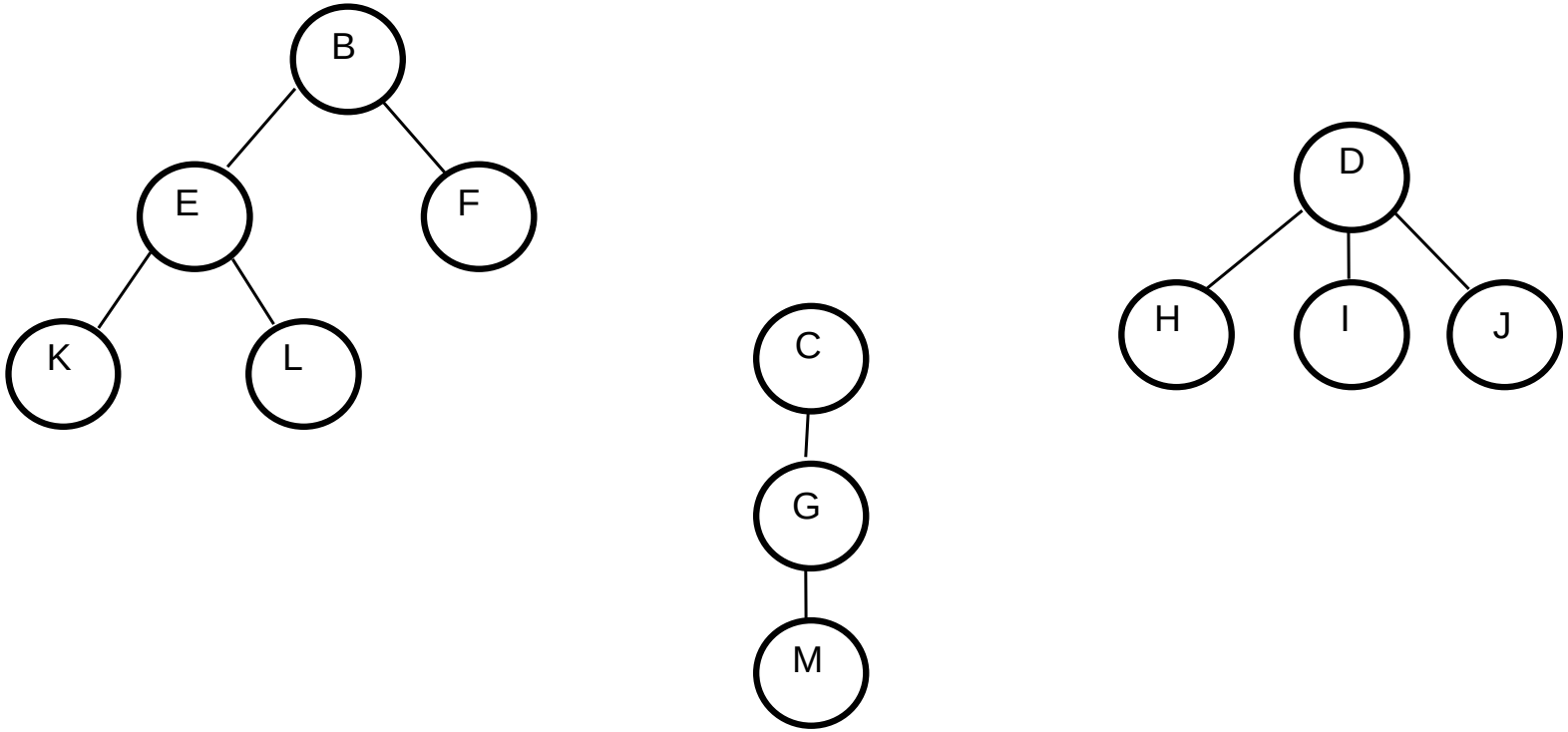
nivo vozlišča / višina drevesa

- Vozlišče A ima nivo 1 (edino koren ima nivo 1)
- Vozlišče K ima nivo 4
- Vozlišče C ima nivo 2



- Drevo ima višino 4

Gozd



O drevesih

- Slovar pojmov:
  - http://www.cs.usask.ca/resources/tutorials/csconcepts/1998_6/bintree/glossary.html

Vrste dreves

- dvojiška drevesa
  - drevo prednikov
  - aritmetičnega izraza
- 2-3 drevesa
- AVL drevesa
- B-drevesa
- najbolj leva drevesa
- ...
- http://www.cs.usask.ca/resources/tutorials/csconcepts/1998_6/bintree/index.html

Dvojiško drevo

- urejeno drevo
- stopnja vozlišč največ dva
- definicija
  - dvojiško drevo je bodisi prazno ali pa ga sestavlja posebej odlikovano vozlišče koren, ki ima levo in desno poddrevo
  - Levo in desno poddrevo sta spet dvojiški drevesi

Dvojiško drevo

- **Definicija:**
  - dvojiško drevo je bodisi prazno ali pa ga sestavlja posebej odlikovano vozlišče koren, ki ima levo in desno poddrevo
  - levo in desno poddrevo sta spet dvojiški drevesi

APS dvojiško drevo

structure DVOJISKO DREVO

declare

pripravi: 0 → dvojisko drevo;

sestavi: (dvojisko drevo, podatek, dvojisko drevo) → dvojisko drevo;

vrni: dvojisko drevo → podatek;

levo poddrevo: dvojisko drevo → dvojisko drevo;

desno poddrevo: dvojisko drevo → dvojisko drevo;

prazno: dvojisko drevo → {true, false};

APS dvojiško drevo

where

levo poddrevo(*pripravi*) ::= NAPAKA;

levo poddrevo(*sestavi*(l,k,d)) ::= l;

desno poddrevo(*pripravi*) ::= NAPAKA;

desno poddrevo(*sestavi*(l,k,d)) ::= d;

prazno(*pripravi*) ::= true;

prazno(*sestavi*(l,k,d)) ::= false;

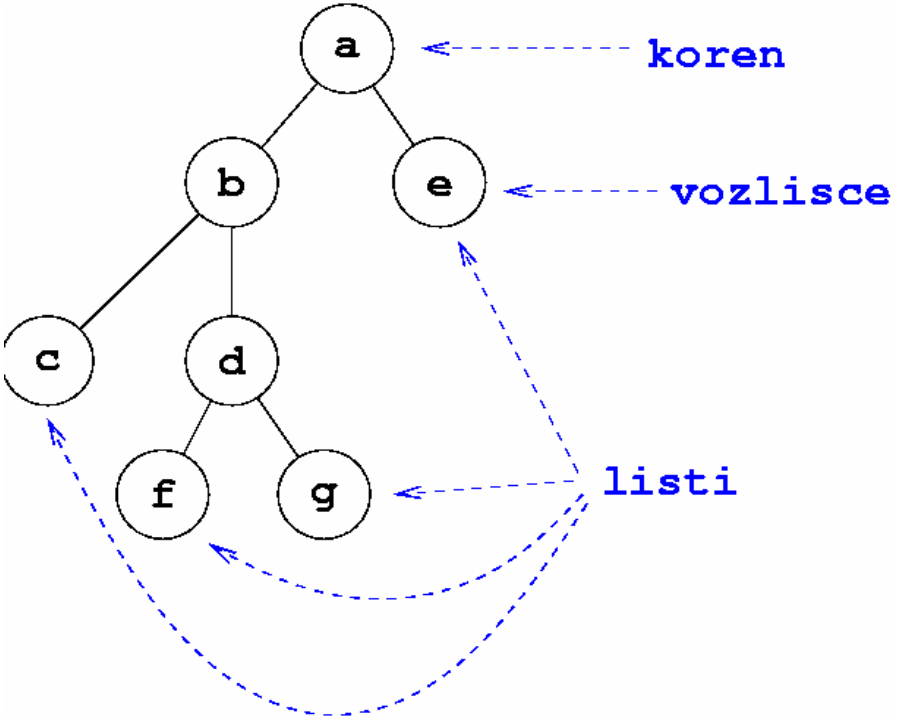
vrni(*pripravi*) ::= NAPAKA;

vrni(*sestavi*(l,k,d)) ::= k;

end;

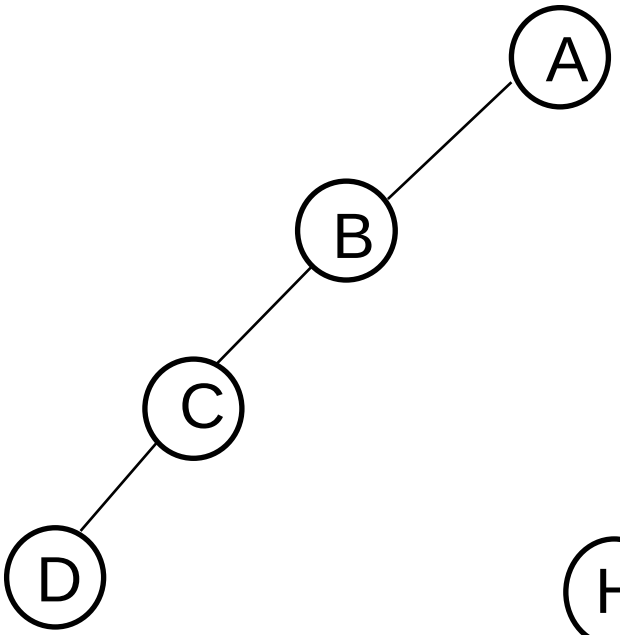
Naloge

- Zrcalna kopija
- Preštej liste
- Drevo povprečij

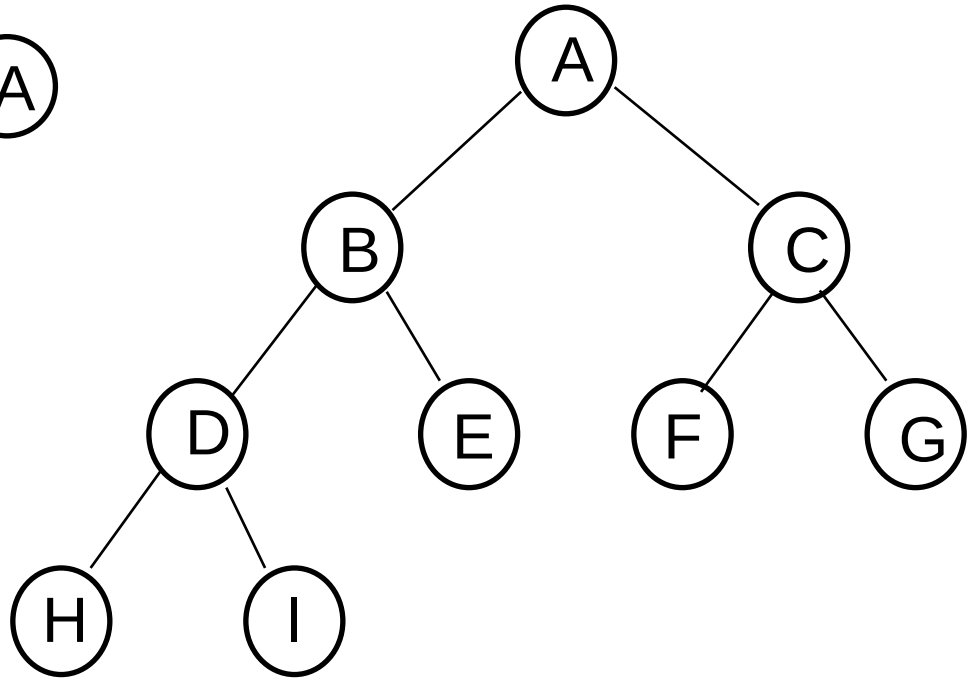


Posebne oblike dvojiških dreves

Izrojeno dvojiško drevo



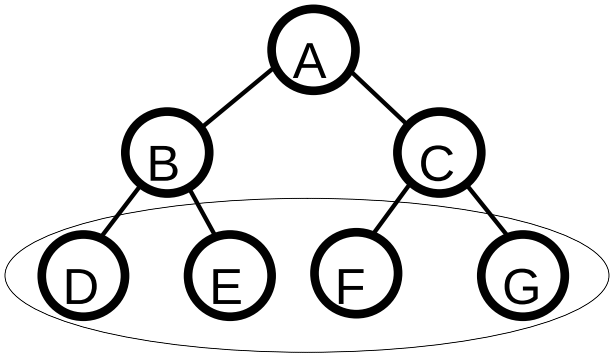
Levo poravnano dvojiško drevo



Lastnosti dvojiških dreves

Splošni pojmi kot pri splošnih drevesih;

Največje število vozlišč na nivoju i:



$$2^{i-1}$$

npr.:

nivo=3

število vozlišč = 2^{3-1}

Lastnosti dvojiških dreves

Največje število vozlišč v drevesu višine k:

$$n_{\max} = \sum_{i=1}^{k} 2^{i-1} = 2^k - 1$$

oz.: najmanjša višina drevesa z n vozlišči:

$$k = \log_2(n+1)$$

Predstavitev dvojiških dreves

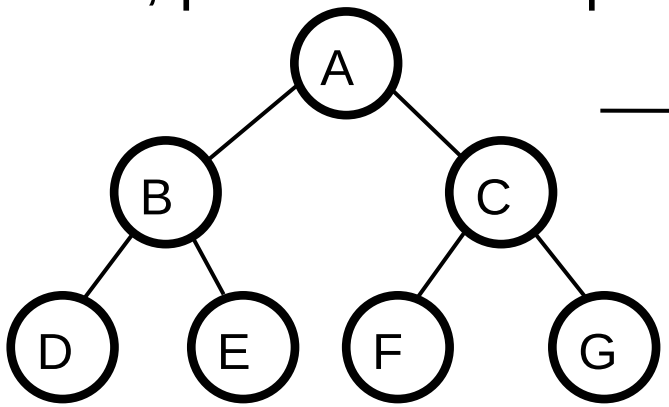
- tabela
  - Na indeks 0 kar pozabimo!
  - razvrstimo elemente po nivojih
  - na istem nivoju od leve v desno
  - zgled
  - levi sin(i) = 2i, če i ima levega sina, nedef. sicer
  - desni sin(i) = 2i+1, če i ima desnega sina, nedef. sicer
  - oce(i) = i / 2

- kazalčna predstavitev
  - (podatek, kazalec na očeta)
  - (podatek, kazalec na levega in desnega sina)
  - (podatek, kazalec na očeta, kazalec na levega in desnega sina)
  - odvisno od uporabe

Statična predstavitev drevesa

z linearno tabelo: ugodno, če je drevo polno ali levo poravnano; KAKO OHRANITI STRUKTURO!

vozlišča začenši s korenom po vrsti (nivojih) vpisujemo v tabelo; prazna mesta posebej označimo;



Vozlišča, nanizana zaporedoma v vektorju:

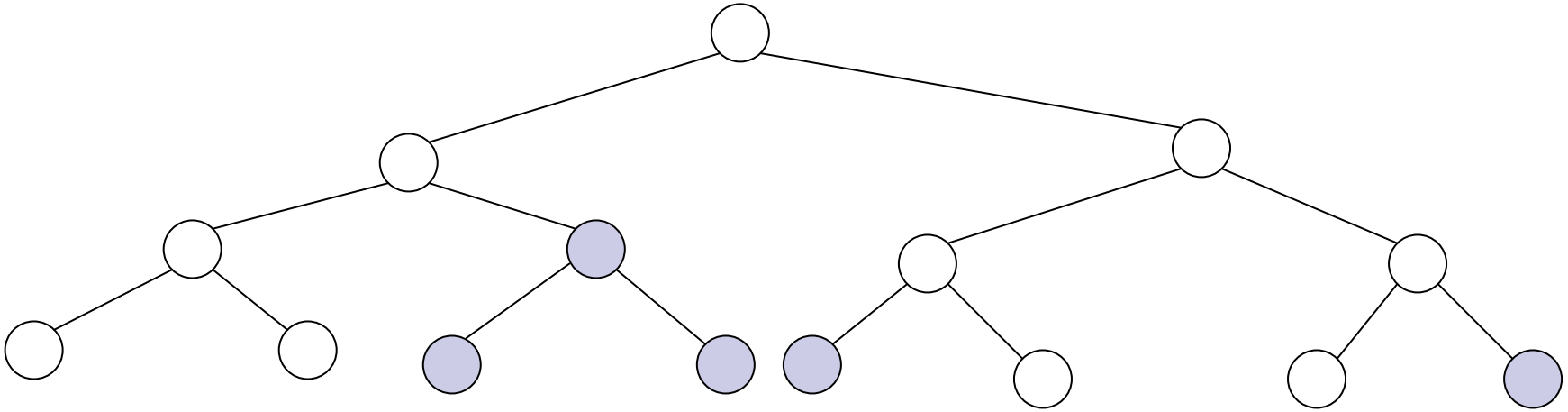
A	B	C	D	E	F	G

Pravila:
Oče(i):=i % 2 (ali ga ni)
Levi_sin:=oče*2 (ali ga ni)
Desni_sin:=oče*2+1(ali ga ni)

Če drevo ni polno, moramo za vsako nezasedeno vozlišče izpustiti prazno mesto v vektorju, da ohranimo indeksiranje.

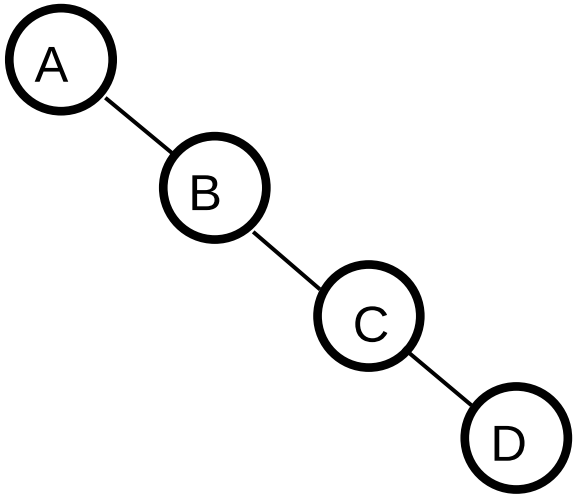
Predstavitev dvojiških dreves

```
int[] drevo = new int[100];
```



```
d[1] := 10; d[2] := 12; d[3] := 31; d[4] := 4;
d[5] := 0; d[6] := 51; d[7] := 16; d[8] := 71;
d[9] := 8; d[10] := 0; d[11] := 0; d[12] := 0;
d[13] := 9; d[14] := 10; d[15] := 0
```

Neugoden primer: izrojeno drevo



Izgled vektorja pri statični izvedbi izrojenega drevesa:

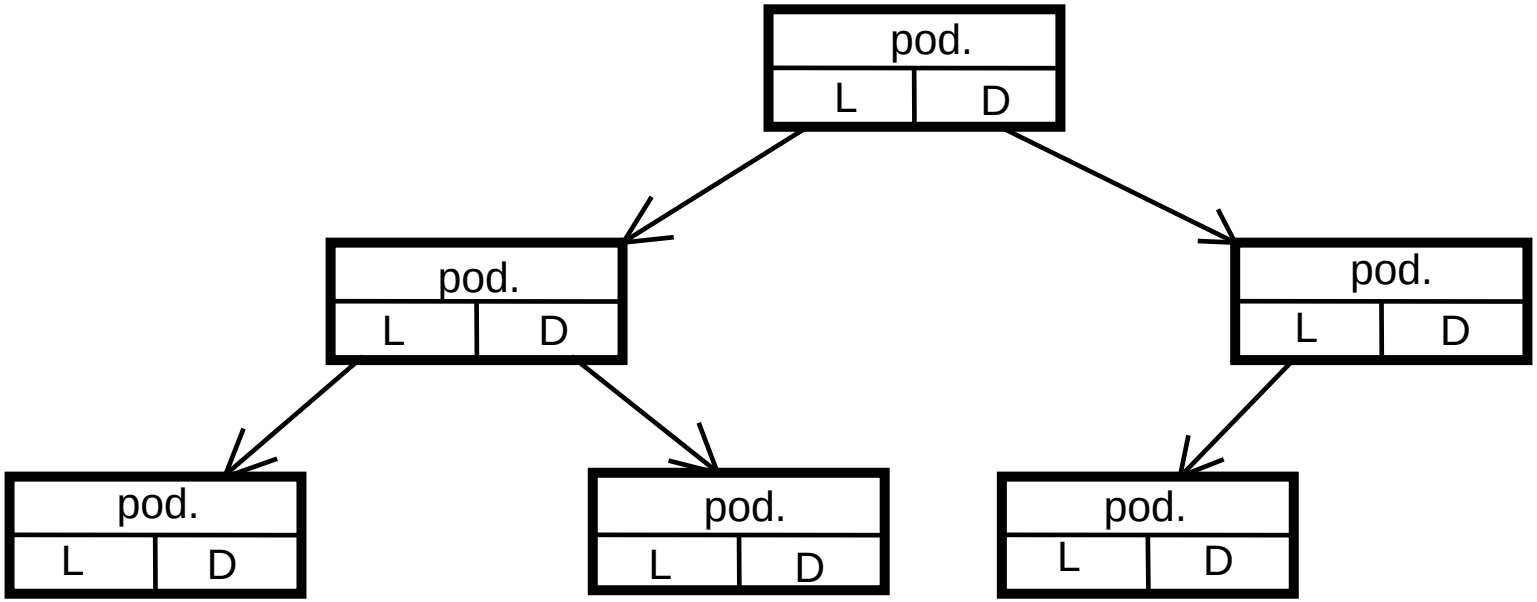
A	-	B	-	-	-	C	-	-	-	-	-	-	D

- Pri slabo zasedenem ali celo izrojenem drevesu moramo računati z veliko izgubo prostora. “Luknje”, ki so visoko v drevesu, povzročijo večjo izgubo, ker manjkajo tudi vsi njihovi nasledniki na nižjih nivojih (dve mesti na nivoju pod luknjo, štiri na naslednjem, itd.)

Predstavitev z naslovi (kazalci, referencami)

- Najbolj običajno
  - kazalec na levega sina
  - vsebina
  - kazalec na desnega sina

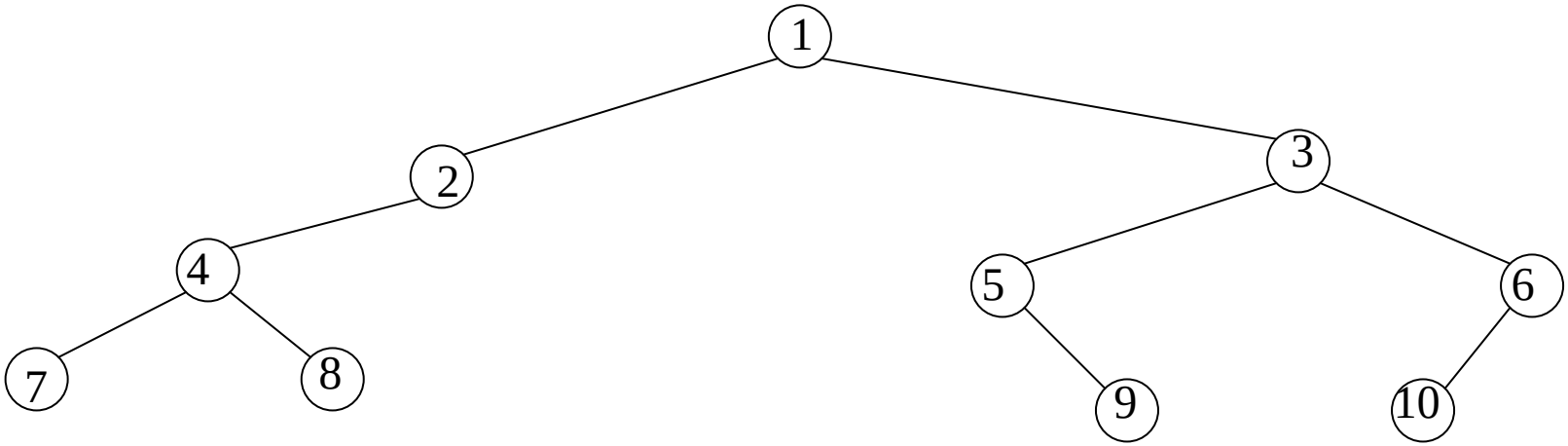
Dinamična predstavitev dreves



Pregled dvojiškega drevesa

- Obisk vozlišč v določenem vrstnem redu
  - v globino
  - po nivojih
  - po listih, ...
- Najpomembnejši:
  - **premi vrstni red**: obisk očeta, premi pregled(levo poddrevo), premi pregled(desno poddrevo)
  - **vmesni vrstni red**: vmesni pregled(levo poddrevo), obisk očeta, vmesni pregled(desno poddrevo)
  - **obratni vrstni red**: obratni pregled(levo poddrevo), obratni pregled(desno poddrevo), obisk očeta

Pregled dvojiškega drevesa



PREMI: 1, 2, 4, 7, 8, 3, 5, 9, 6, 10

VMESNI: 7, 4, 8, 2, 1, 5, 9, 3, 10, 6

OBRATNI: 7, 8, 4, 2, 9, 5, 10, 6, 3, 1

Pregled dvojiškega drevesa - demo

- http://www.cs.usask.ca/resources/tutorials/csconcepts/1998_6/bintree/2-2.html
- http://www.educa.fmf.uni-lj.si/www375/2000/Algoritmi/Java-examples/Tree.html

Premi vrstni red

- enaka ideja kot pri izpisu
- rekurzija

Rekonstrukcija drevesa

- iz dveh pregledov lahko naredimo drevo
- PREMI: 1, 2, 4, 7, 8, 3, 5, 9, 6, 10
- VMESNI: 7, 4, 8, 2, 1, 5, 9, 3, 10, 6
- KOREN: 1
- LEVO DREVO V: 7, 4, 8, 2
- LEVO P: 2, 4, 7, 8
- DESNO DREVO V: 5, 9, 3, 10, 6
- DESNO P: 3, 5, 9, 6, 10

Iskalno dvojiško drevo

- ni podvojenih elementov
- def: *dvojiško drevo, za katerega velja, da so vsi elementi v levem poddrevesu (ki je tudi iskalno drevo) manjši od korena in v desnem poddrevesu večji od korena. Tudi desno poddrevo je iskalno dv. drevo*
- Iskanje podatkov je lahko dokaj hitro: odvisno od učinkovitosti predstavitve

Iskanje v IDD

- http://www.cs.queensu.ca/home/jstewart/applets/bst/bst-searching.html

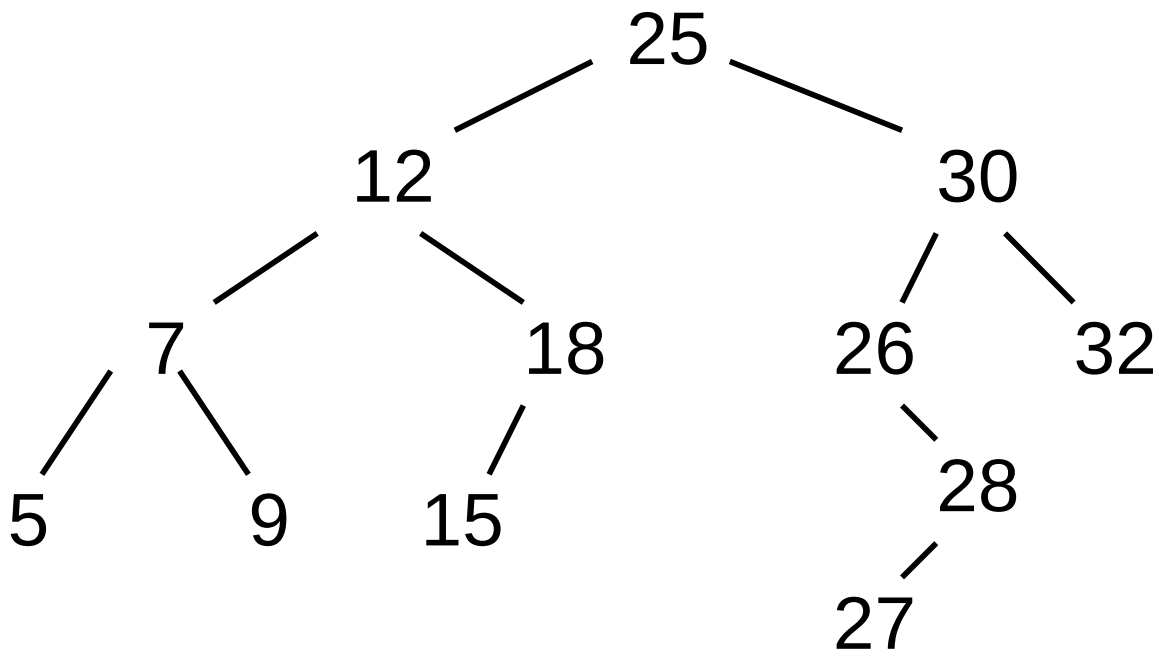
ISKALNO DREVO

- vsebuje podatke, urejene po enem ključu;
- za vsako vozlišče velja, da množica podatkov v levem poddrevesu ni večja, množica podatkov v desnem poddrevesu pa ni manjša od podatka v korenu
-
  ⇨ iskalno drevo ali binarni slovar
-

ISKALNO DREVO

- Binarni slovar:
  v slovarju iščemo neko geslo. Odpremo na poljubni strani; če je geslo pred to stranjo (korenom) iščemo v prvi polovici (levo poddrevo), sicer v drugi (desno poddrevo). Postopek rekurzivno ponavljamo, dokler ne najdemo gesla v korenu ali ugotovimo, da ga ni v slovarju (in ga po potrebi dodamo).

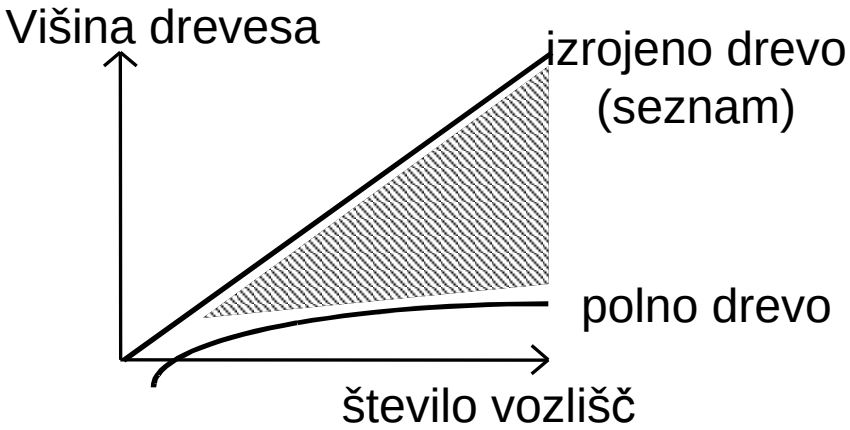
Primer:



Vzdrževanje urejenih podatkov v seznamu in v iskalnem drevesu

- število primerjav in premikov, potrebnih, da pregledamo iskalno drevo, je enako višini drevesa; rast višine drevesa s številom podatkov je počasna, posebej za velike n;
- višina polnega drevesa z n podatki je proporcionalna log(n);
- če v iskalno drevo vstavljamo že urejeno zaporedje podatkov, dobimo izrojeno drevo - seznam;
- višina izrojenega drevesa z n vozlišči je n.

Primerjava



- pričakovana višina iskalnega drevesa je v črtkanem območju. Želimo, da bi bila čim bližje spodnji krivulji.
- prednost vzdrževanja podatkov v drevesu je tem večja, čim več je podatkov. Dvakrat več podatkov (+1) kot v celotnem (polnem) drevesu poveča njegovo višino (potrebno število primerjav in premikov) le za 1.

Primer:

- za iskanje podatka v seznamu z 106 podatki potrebujemo v najslabšem primeru 106 primerjav in premikov.
- za iskanje podatka v (polnem!) iskalnem drevesu z 106 podatki potrebujemo kvečjemu

$$\log_2(10^6) = \frac{\log_{10}(10^6)}{\log_{10}(2)} = \frac{6}{0.303} \cong 20$$

primerjav.

- za iskanje podatka v drevesu z 2*106 podatki potrebujemo le eno primerjavo več.

Vstavi v iskalno drevo

- enako kot pri izpisu, pregledu
- rekurzija
- če je drevo prazno, vstavimo
- če je x > koren, vstavimo desno
- če je x < koren, vstavimo levo
- če je x enak korenu, ne naredimo nič

Vstavljanje podatkov

- začnemo pri korenu in se pomikamo proti listom;
- podatek, ki ga vstavljamo, primerjamo s podatkom v korenu; če je manjši, se premaknemo v levo, sicer v desno poddrevo;
- postopek ponavljamo, dokler:
  - ne najdemo podatka v drevesu; ali
  - ne pridemo do lista; podatka še ni v drevesu, zato ga vstavimo - tvorimo nov list.
- Problem: izrojena oblika – počasno iskanje
- Rešitev: uravnotežena drevesa (AVL, rdeče-črna, ...)