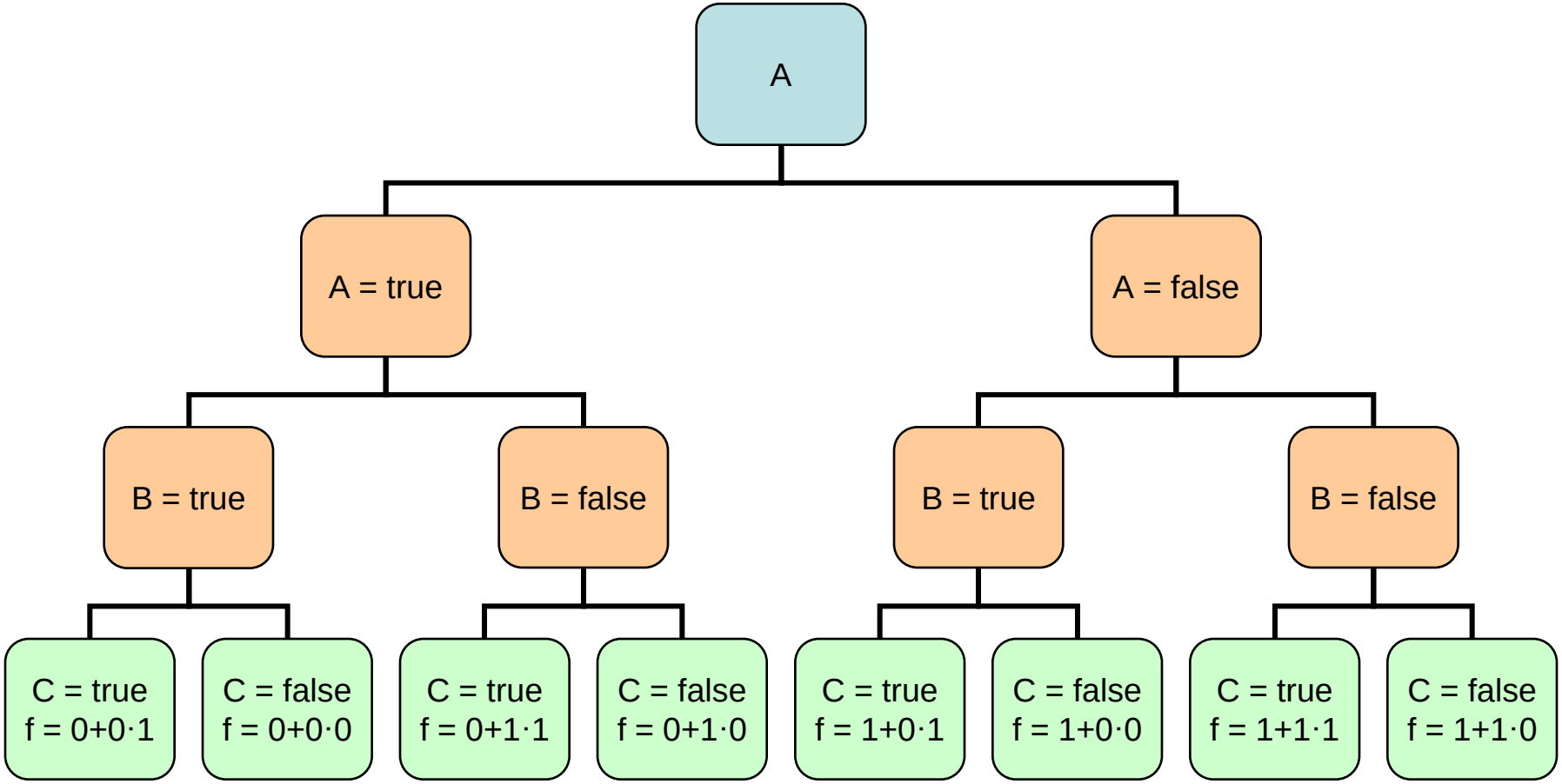
Stablo odlučivanja (Decision Tree)

Decision Tree - stablo u kojemu djeca (veze) predstavljaju odluke o uvjetima u nezavršnim (unutarnjim) čvorovima (roditeljima), a listovi su sva moguća rješenja polaznog problema

- olakšavaju rješavanje složenih problema rastavljajući kompliciran postupak na niz jednostavnih odluka
- kada za svaki uvjet postoje samo dva moguća odgovora (npr. DA-NE), stablo je binarno i može prikazati rješavanje bilo koje logičke funkcije (varijable mogu biti samo TRUE ili FALSE)

Primjer: $\overline{A} + \overline{B} \cdot C$



U listovima su sva moguća rješenja zadane funkcije.

Koja je najveća moguća brzina sortiranja?

Kao teorijsku mjeru brzine uzima se potrebni broj usporedbi pa je pitanje zapravo koliko je najmanje usporedbi potrebno za sortiranje n elemenata.

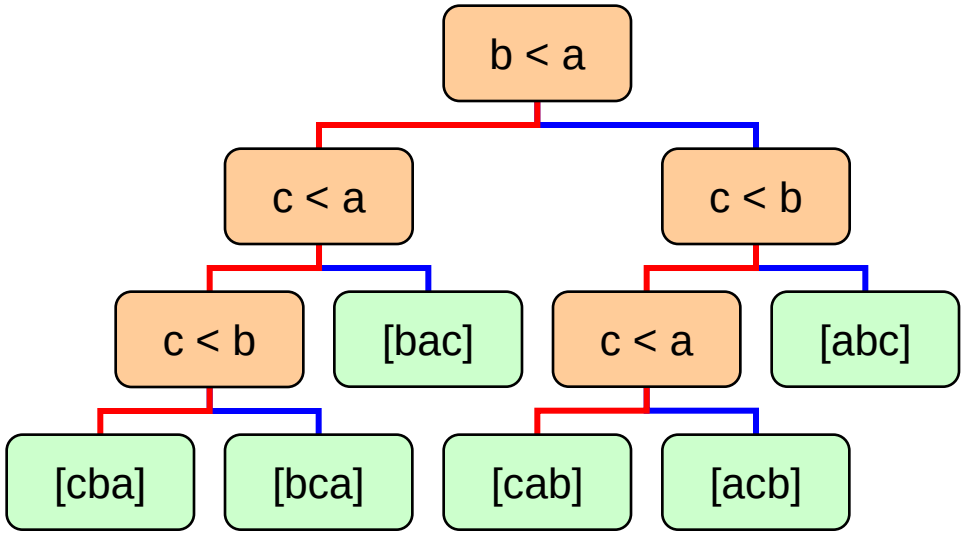
Decision Tree bilo kakvog sortiranja je binarno stablo u kojemu djeca (veze) predstavljaju DA-NE odluke o uvjetima (> ili ≤) u unutarnjim čvorovima, a listovi su svi mogući poretci elemenata.

Poredaka ima n! pa stablo odlučivanja mora imati najmanje toliko listova (možebitni višak jesu nemogući slučajevi - vidi primjer).

Struktura mu ovisi o algoritmu i poretku ulaznih podataka.

Primjer:
sotriranje polja [abc]

yes, no

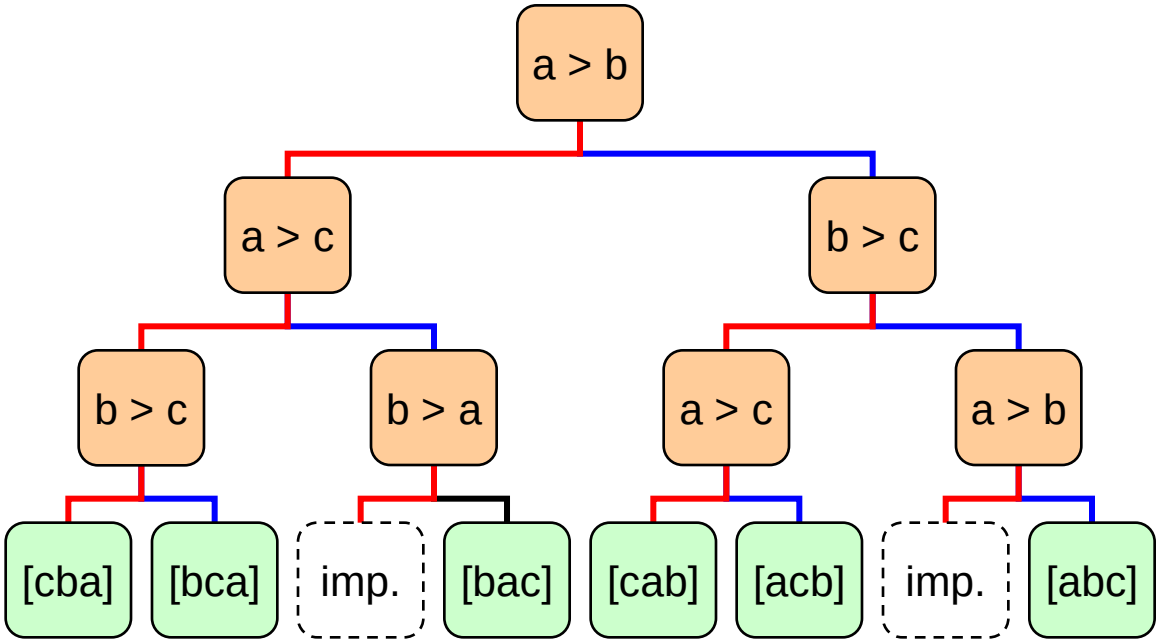


Insertion sort

- broj usporedbi ovisi o poretku ulaznih podataka

Bubble sort

- broj usporedbi ne ovisi o poretku ulaznih podataka



Potpuno binarno stablo u h razina ima $2^h - 1$ čvorova, od kojih 2^{h-1} listova i $2^{h-1} - 1$ unutarnjih čvorova.

Označimo: m = broj listova,

k = broj unutarnjih čvorova.

Nepotpuno stablo može imati manje ili jednako čvorova kao potpuno stablo pa vrijedi

$$m + k \leq 2^h - 1 \quad ; \quad m \leq 2^{h-1},\ k \leq 2^{h-1} - 1.$$

Broj mogućih poredaka n! je manji ili jednak broju listova, iz čega slijedi $n! \leq m \leq 2^{h-1}$, odnosno

$$2^{h-1} \geq n! \quad \Rightarrow \quad h - 1 \geq \log_2 n! \quad .$$

U najgorem slučaju moramo doći do najnižeg lista, a gornja relacija kazuje da ćemo pritom morati obaviti najviše $\log_2 n!$ usporedbi.

Dakle, najgori slučaj najbržeg mogućeg sortiranja zahtijeva $O(\log_2 n!)$ usporedbi.

Aproksimativna gornja granica te funkcije je $n \cdot \log_2 n$ jer je

$$\log_2 n! \leq \log_2(n \cdot n \cdot \ldots \cdot n) = \log_2 n^n = n \cdot \log_2 n .$$

Može se pokazati da je i proječan broj usporedbi također $\log_2 n!$ pa je prosječna složenost najbržeg sortiranja opet $O(n \cdot \log_2 n)$.

Brisanje čvora binarnog stabla za pretraživanje

Ponoviti gradivo o stablima iz ASP!

Binarno stablo za pretraživanje (Binary Search Tree) - stablo u kojemu su svi elementi lijevog podstabla nekog čvora manji, a desnog podstabla veći od promatranog čvora

Brisanje čvora mora riješiti tri moguća slučaja:

1) čvor je list (nema djece)
2) čvor ima samo jedno dijete
3) čvor ima dva djeteta

vodeći računa o posebnosti korijena.

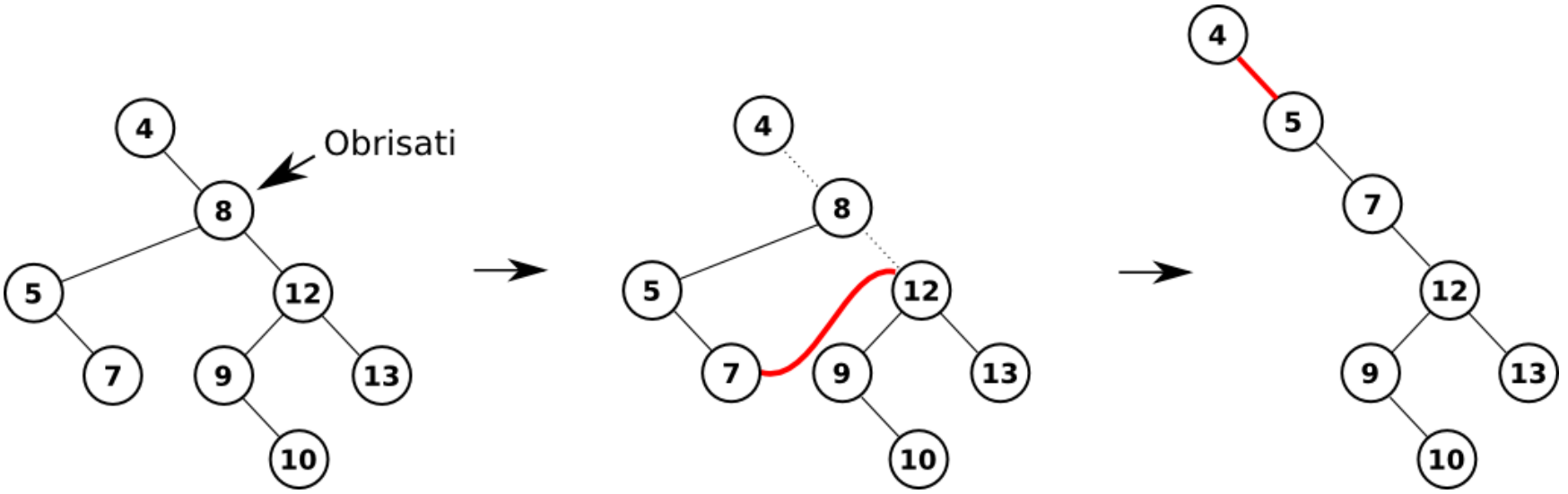
Prva dva slučaja su jednostavna, a treći se najčešće rješava na dva načina:

a) Brisanje sjedinjenjem (Deletion by Merging)
b) Brisanje kopiranjem (Deletion by Copying)

Brisanje sjedinjenjem (Deletion by Merging) - osjetno mijenja strukturu stabla (neupotrebljivo za uravnotežena stabla)

1. naći čvor koji se briše i njegovog roditelja
2. naći najveći manji od njega (ili najmanji veći)
   - to je najdesniji u lijevom podstablu (najljeviji u desnom) i taj sigurno nema desno (lijevo) dijete
3. desni (lijevi) pokazivač najvećeg manjeg (najmanjeg većeg) usmjeriti na desno (lijevo) dijete čvora koji se briše
4. pokazivač u roditelju usmjeriti na lijevo (desno) dijete čvora koji se briše
5. ukloniti čvor koji se briše (osloboditi memoriju)

Primjer: brisanje sjedinjenjem



BrisanjeCvoraStabla

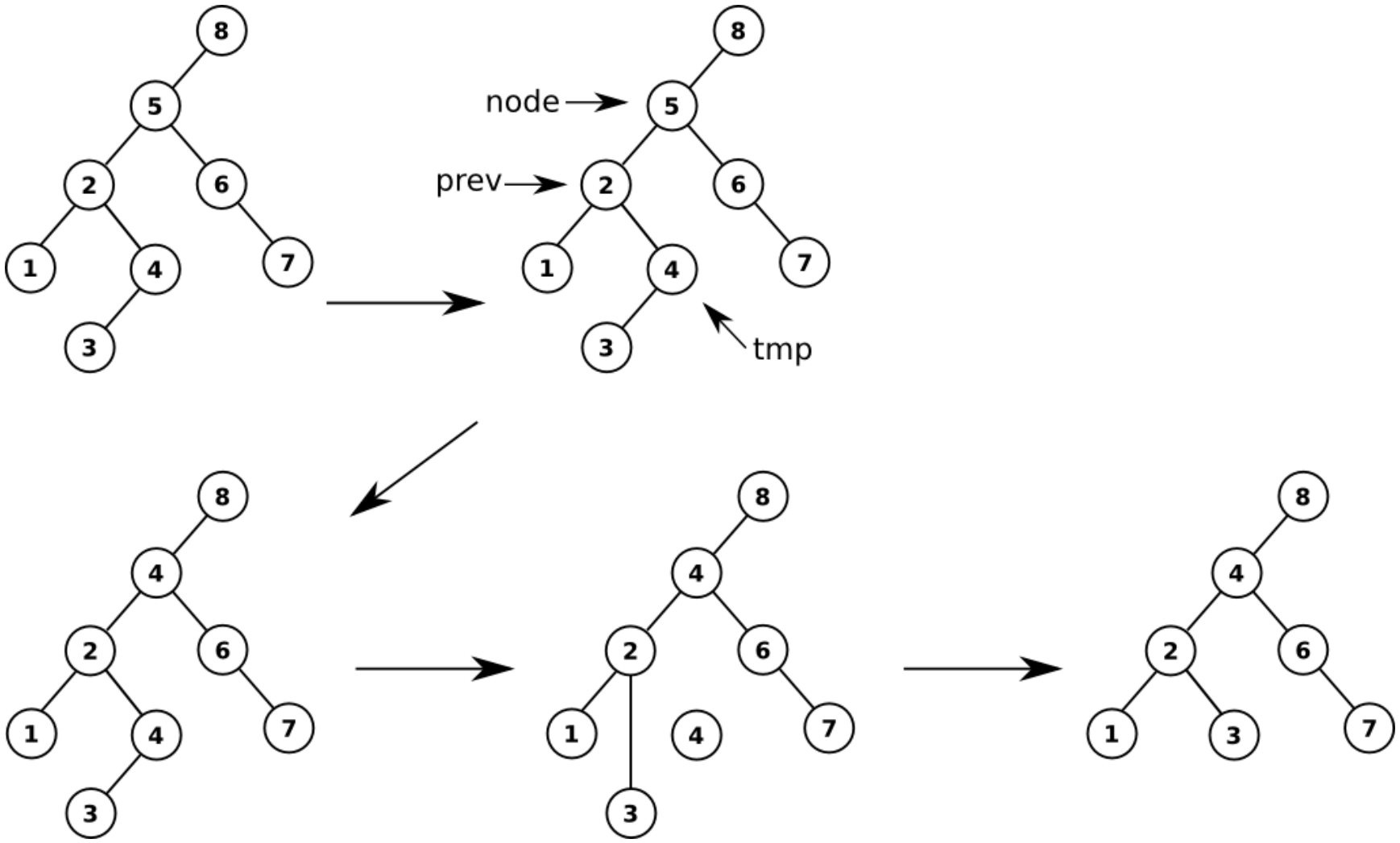
Brisanje kopiranjem (Deletion by Copying)

- ovim se algoritmom problem brisanja čvora s dva djeteta reducira na brisanje čvora s jednim djetetom ili bez djece (list)
- minimalno mijenja strukturu stabla

1. naći najbližeg prethodnika ili sljedbenika čvora koji se briše (najveći manji ili najmanji veći); zapamtiti njega i njegovog roditelja
   - najbliži je najdesniji u lijevom podstablu (najljeviji u desnom) i sigurno nema desno (lijevo) dijete
2. podatke iz najbližeg prethodnika (sljedbenika) prepisati u čvor koji se briše
3. desni (lijevi) pokazivač roditelja prethodnika usmjeriti na dijete prethodnika (“preskočiti” prethodnika)
4. ukloniti prethodnika (osloboditi memoriju)

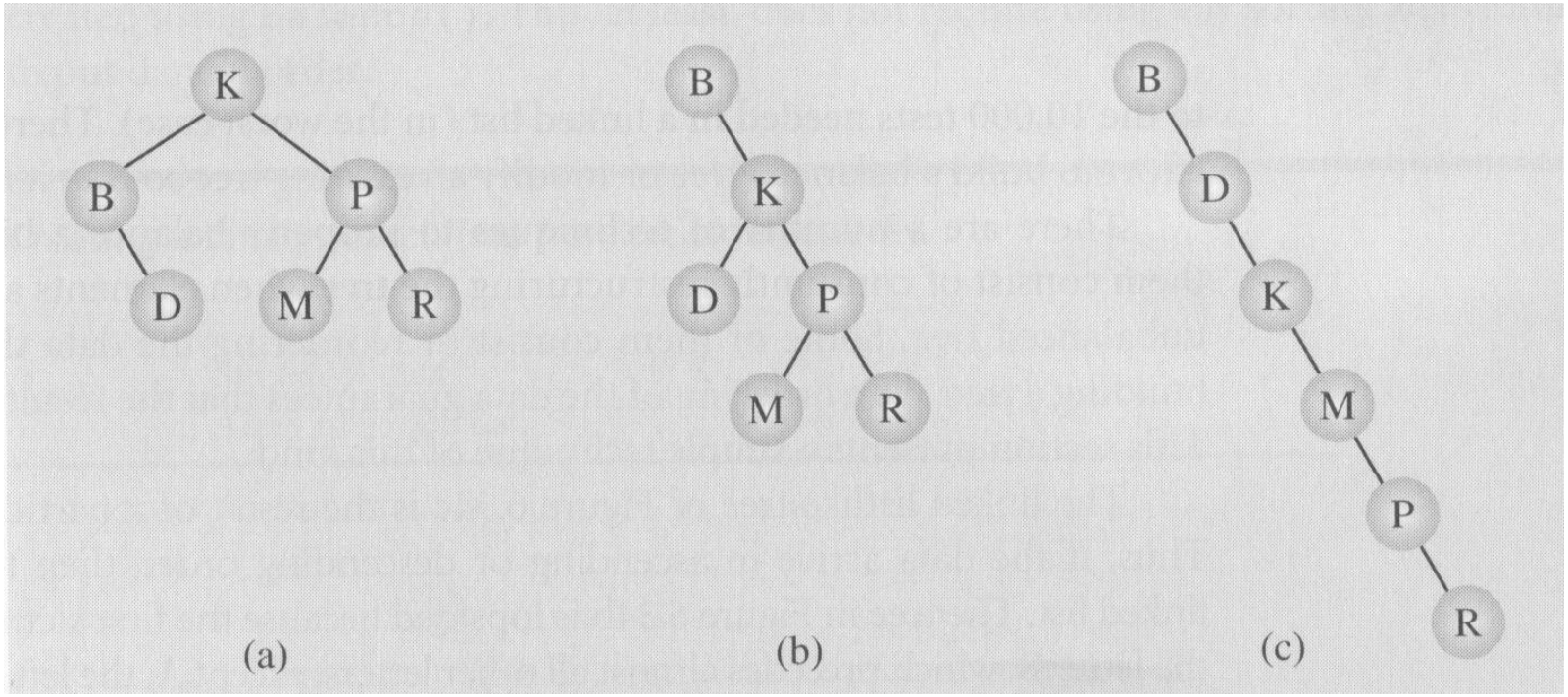
BrisanjeCvoraStabla

Primjer: brisanje kopiranjem



Uravnotežavanje stabla (Balancing a Tree)

Osnovna prednost stabla pred listom, brzina pretraživanja, se gubi ako je stablo neprikladne strukture, tj. neuravnoteženo
(krajnost: koso stablo = lista; primjer - traženje R).



Uravnoteženo stablo (Balanced Tree) – stablo u kojemu je razlika visina podstabala svakog čvora najviše jedan

Savršeno uravnoteženo stablo (Perfectly Balanced Tree) – uravnoteženo stablo u kojemu su svi listovi u najviše dvije razine

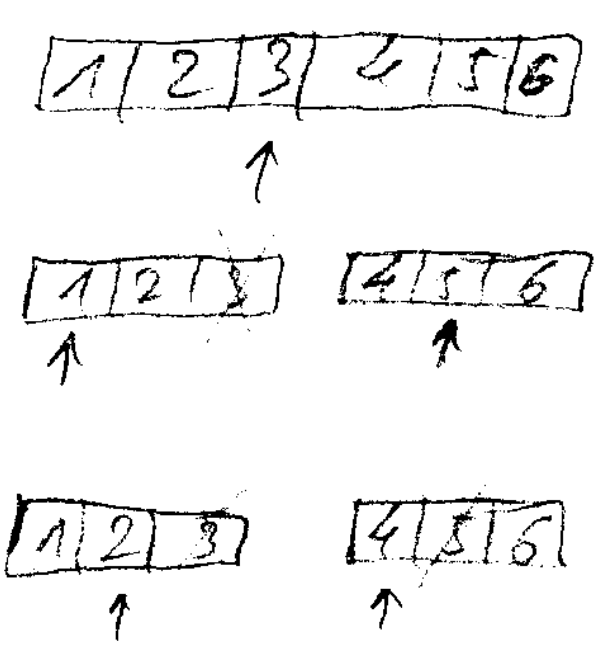
Uravnotežavanje:
1) promišljenim redoslijedom upisa podataka
2) restrukturiranjem stabla nakon izmjena

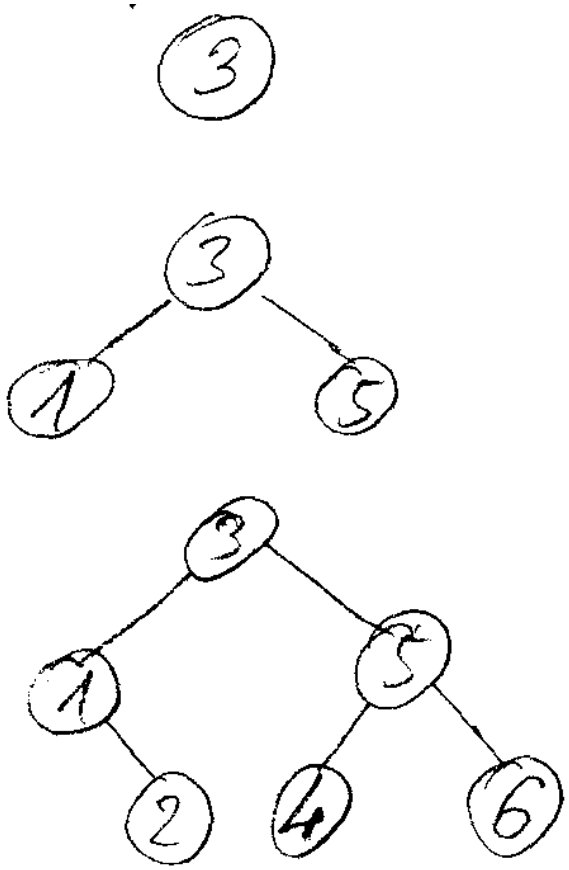
Promišljeni redoslijed upisa podataka

- algoritam sličan binarnom pretraživanju

1. prikupiti i sortirati sve ulazne podatke
2. korijen stabla = središnji element tog skupa; preostali podatci sada su u dva podskupa
3. središnji element lijevog podskupa postaje lijevo, a središnji element desnog podskupa postaje desno dijete korijena
4. s novim podskupima nastalim izdvajanjem središnjih elemenata postupiti na isti način, ponavljajući korake 2 i 3, pri čemu izdvojeni elementi imaju ulogu korijena svojih podstabala

Primjer:





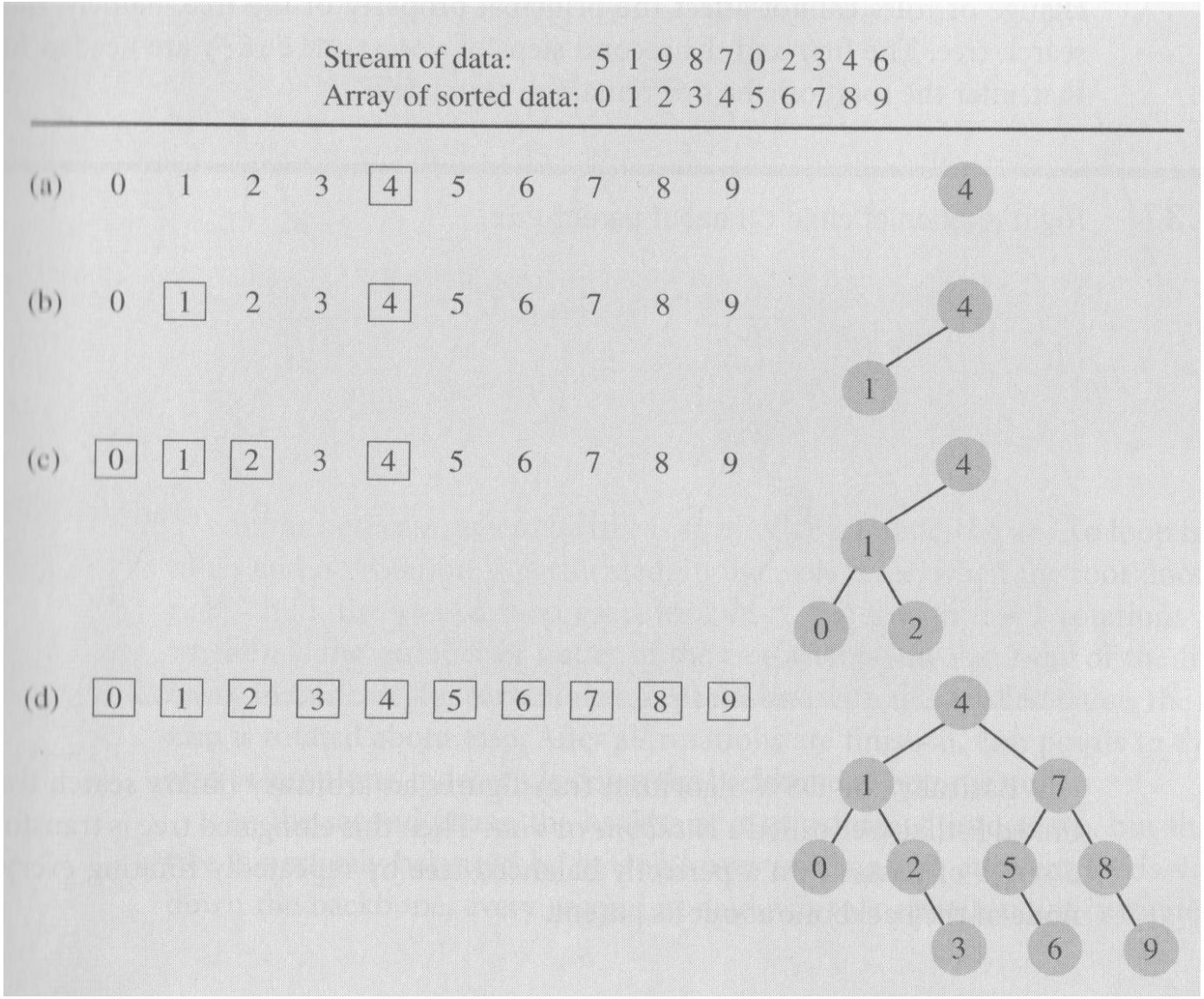
Sortiranje se može izbjeći upisivanjem podataka u neuravnoteženo stablo te potom *inorder* čitanjem i prepisivanjem u neku drugu strukturu, tj. polje.

Opisani algoritam prvo upisuje korijen, zatim njegovo lijevo i desno dijete pa istim redoslijedom djecu djece itd..

Za programiranje je osjetno jednostavnije upisati korijen, potom njegovo lijevo dijete pa lijevo dijete lijevog djeteta itd. i tek nakon toga desnu djecu, od najnižeg čvora prema korijenu.

```
MakeBalTree (data[], left, right)
if left <= right
    middle = (left + right)/2
    insert data[middle] into the tree
    MakeBalTree (data, left, middle-1)
    MakeBalTree (data, middle+1, right)
```

Primjer:


Stream of data: 5 1 9 8 7 0 2 3 4 6
Array of sorted data: 0 1 2 3 4 5 6 7 8 9
(a) 0 1 2 3 4 5 6 7 8 9
(b) 0 1 2 3 4 5 6 7 8 9
(c) 0 1 2 3 4 5 6 7 8 9
(d) 0 1 2 3 4 5 6 7 8 9
4
4
1
4
1
0
2
4
1
7
0
2
5
8
3
6
9

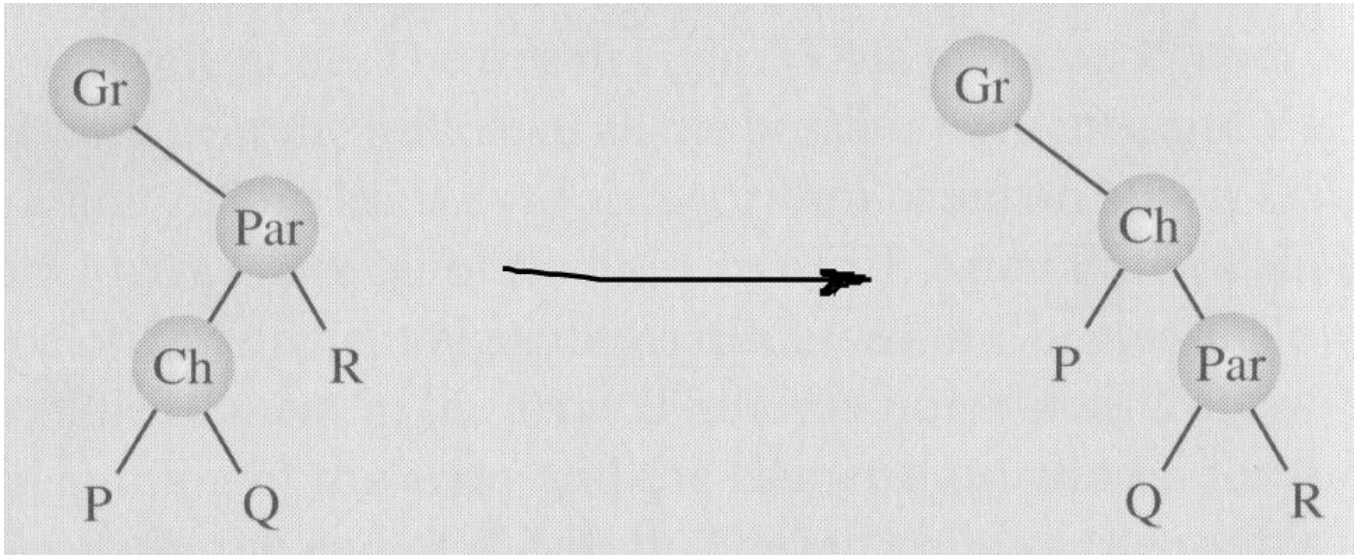
Nedostatci pripreme ulaznih podataka:

- potreba za dodatnom memorijom
- sortiranje
- rezultat ne mora biti popunjeno stablo

DSW algoritam – savršeno uravnotežavanje
- Colin Day, Quentin F. Stout i Bette L. Warren

Osnova tog algoritma su rotacije u stablu.
Rotacija - postupak kojim dijete postaje roditelj, a roditelj dijete, pri čemu se poštuju definicijska pravila stabla (hijerarhijska struktura stabla s obzirom na zadani kriterij)
Lijeva i desna rotacija su potpuno simetrične (isti algoritam sa zamijenjenim značenjem lijevo-desno).

Primjer: desna rotacija Ch oko Par:



`RightRotation (gr, par, ch)`

`if par` *is not the root* //not NULL

redirect right pointer of `gr` *to* `ch`

redirect left pointer of `par` *to right subtree of* `ch`

redirect right pointer of `ch` *to* `par`