Algoritmi i strukture podataka

- predavanja -

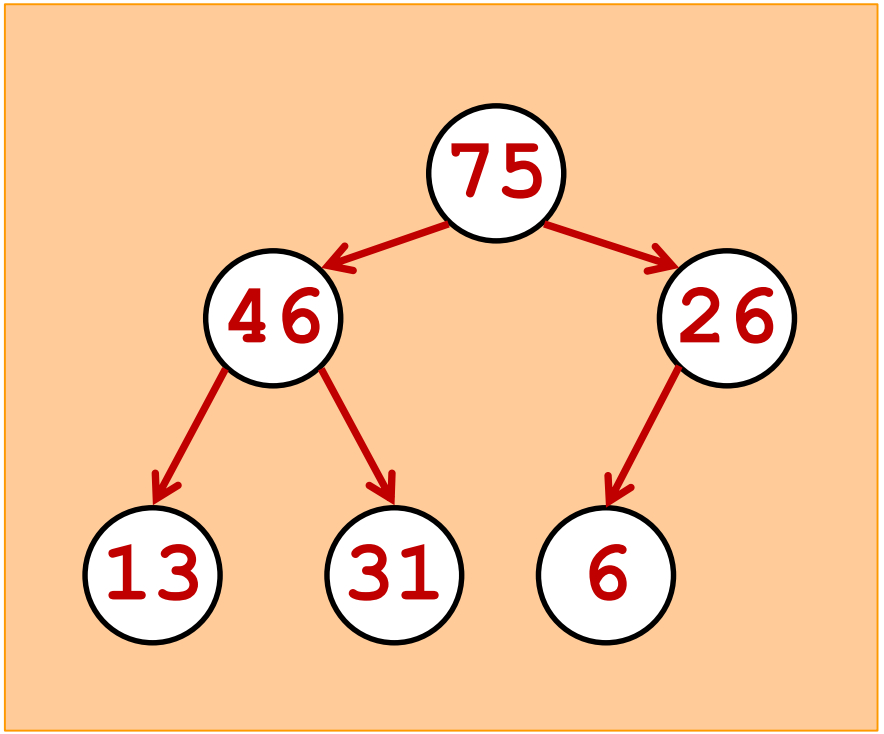
10. Gomila i sortiranje gomilom

Osnovni pojmovi

- prioritetni red je struktura podataka koja donekle podsjeća na obični red
  - sličnost s običnim redom je u tome što se podaci mogu dodavati (ubacivati) u prioritetni red te skidati (izbacivati) iz njega.
  - razlika je u tome što se ne skida onaj podatak koji je prvi bio dodan, već onaj koji ima najveću vrijednost (najveći prioritet)
- prioritetni red može se prikazati na razne načine:
  - sortiranom vezanom listom
  - sortiranim binarnim stablom
- najprirodniji i najučinkovitiji način prikaza prioritetnog reda je pomoću gomile (hrpe, *heap*)

Gomila

- gomila je potpuno binarno stablo gdje se čvorovi mogu uspoređivati nekom uređajnom relacijom (npr. <=) i gdje je bilo koji čvor u smislu te relacije veći ili jednak od svoje djece (ako postoje)



Oblikovanje strukture gomila

- najčešće se n elemenata složi u gomilu, pa je najjednostavnije koristiti potpuno binarno stablo za prikazivanje gomile, makar bi i druga binarna stabla mogla zadovoljavati
  - to se ostvaruje ubacivanjem jednog po jednog elementa u gomilu, čuvajući svojstvo gomile
  - počinje se od prazne gomile
  - na "dno" (list) gomile dodaje se član koji se onda uspoređuje i zamjenjuje sa svojim roditeljem, praroditeljem, prapraroditeljem itd. dok ne postane manji ili jednak nekoj od tih vrijednosti

Heap.h
HeapNLogN.cpp

Analiza najgoreg slučaja

- za analizu najgoreg slučaja algoritma uzmimo n elemenata
  - na i-toj razini potpunog binarnog stabla ima najviše 2^{i-1} čvorova
  - na svim nižim razinama do tada ima ukupno $2^{i-1} - 1$ čvorova, za $i > 1$.
  - stablo s k razina ima najviše 2^k-1 čvorova
  - stablo s $k-1$ razinom ima najviše $2^{k-1}-1$ čvorova.
- ako je stablo potpuno, započeta je posljednja razina, pa vrijedi $2^{k-1} - 1 < n \leq 2^k - 1$
  - iz ovoga slijedi:

$$2^{k-1} < n + 1 \Rightarrow (k-1)\log 2 < \log(n+1) \Rightarrow k < \log_2(n+1) + 1$$

$$n + 1 \leq 2^k \Rightarrow \log(n+1) \leq k \log 2 \Rightarrow \log_2(n+1) \leq k$$

$$\log_2(n+1) \leq k < \log_2(n+1) + 1 \text{ odnosno } k = \lceil \log_2(n+1) \rceil$$

Primjer

- za n = 14 treba $\lceil \log_2 15 \rceil = \lceil \ln 15/\ln 2 \rceil = \lceil 2.70805/0.693147 \rceil = \lceil 3.9 \rceil = 4$ razine
- za n = 15 treba $\lceil \log_2 16 \rceil = \lceil 4 \rceil = 4$ razine
- za n = 16 treba $\lceil \log_2 17 \rceil = \lceil 4.087 \rceil = 5$ razina

Ubrzanje algoritma - I

- u najgorem slučaju, petlja **`while`** izvršava se proporcionalno broju razina u gomili
  - skup podataka koji predstavlja najgori slučaj za ovaj algoritam je polje s rastućim podacima
- tada svaki novi element, onaj koji se ubacuje u gomilu pozivom funkcije **`ubaci`**, postaje korijen pa se kroz k razina obavlja zamjena
- vrijeme izvođenja je tada $O(n \log_2 n)$
- za prosječne podatke vrijeme za stvaranje gomile iz skupa podataka je $O(n)$, što je za red veličine bolje

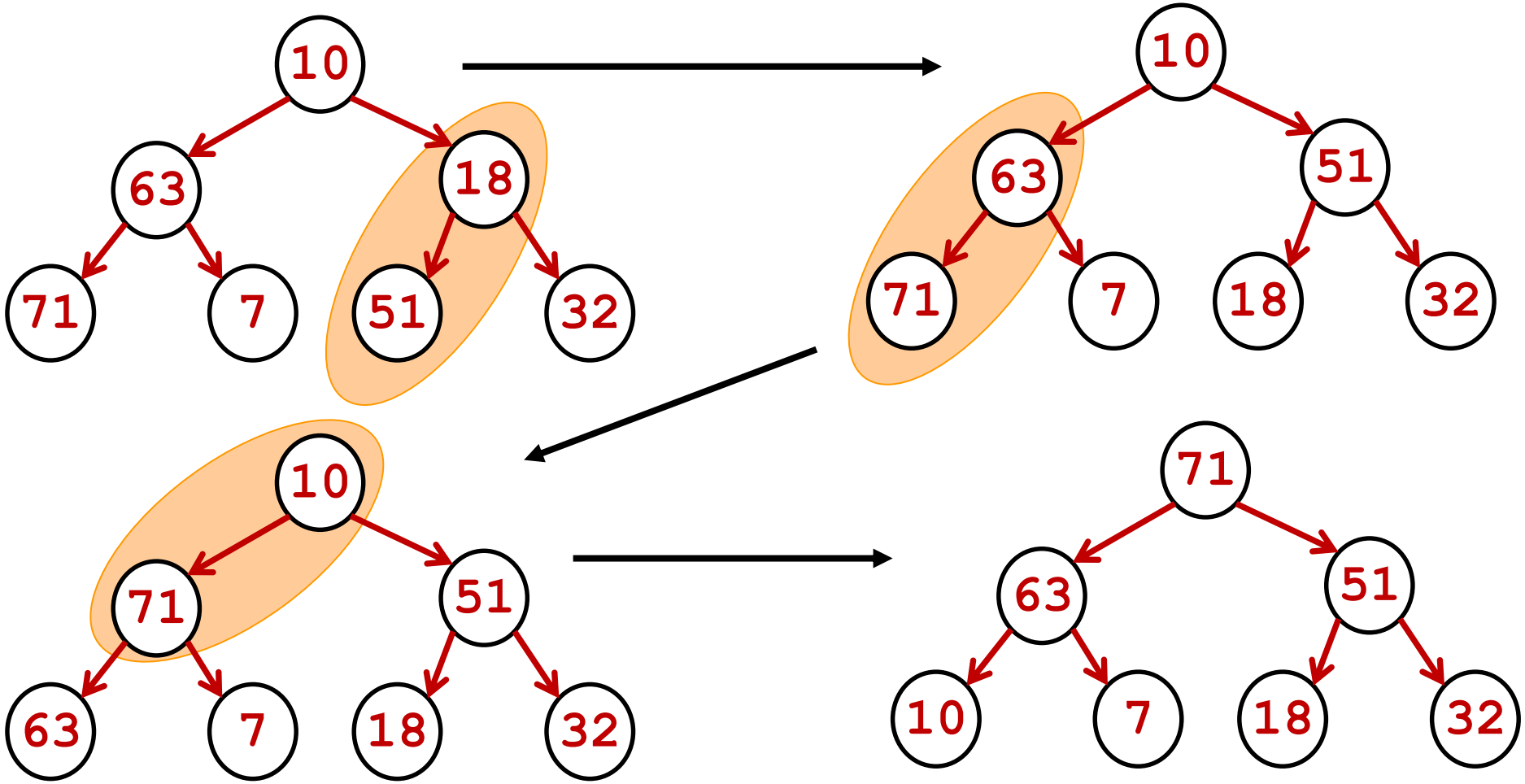
Ubrzanje algoritma - II

- za poboljšanje brzine obavljanja zadanih operacija stvoren je algoritam koji kreće od krajnjih čvorova prema korijenu, razinu po razinu
- samo podatak u korijenu može narušavati svojstvo gomile, dok podstabla zadržavaju to svojstvo
- tada je samo potrebno tu nepravilnost ispraviti i opet dobivamo željenu gomilu.
  - to čini funkcija **`podesi`** u primjeru
  - za krajnje čvorove svojstvo gomile je zadovoljeno, pa treba u **`stvori_gomilu`** funkciji provesti popravljanje svojstva gomile samo za korijen stabla

Heap.h
HeapNLib.cpp
HeapN.cpp

Primjer stvaranja gomile

- Stvaranje gomile za ulazni niz podataka: **10,63,18,71,7,51,32**



Ubrzanje algoritma

- za n podataka, $2^{k-1} \leq n < 2^k$, broj razina je $k = \lceil log_2(n+1) \rceil$
- za najgori slučaj broj iteracija u **`podesi`** iznosi $k-i$ za čvor na razini i gdje ima najviše 2^{i-1} čvorova
  - vrijeme izvođenja za **`stvori_gomilu`** je:

$$\sum_{i=1}^{k} 2^{i-1}(k-i)$$

eksponent se mijenja od *0* do *k -1*, a faktor od *k -1* do *0*

  - slijedi ekvivalentni izraz kad se izbaci faktor 0 i obrne redoslijed sumacije:

$$\sum_{i=1}^{k-1} i \cdot 2^{k-i-1} = \sum_{i=1}^{k-1} 2^{k-1} \cdot i \cdot 2^{-i} \leq n \sum_{i=1}^{k-1} \frac{i}{2^i} \leq 2n = O(n)$$

$2^{k-1} \leq n$

suma reda teži prema 2

Ubrzanje algoritma

- vrijeme izvođenja za najgori slučaj algoritma **`stvori_gomilu`** je $O(n)$, što je za red veličine bolje od $O(n \log_2 n)$ za uzastopno korištenje funkcije **`ubaci`**
- funkcija **`stvori_gomilu`** traži da su svi elementi za stvaranje gomile već prisutni, dok **`ubaci`** može ubaciti novi element u gomilu bilo kada
  - funkcije koje gomila treba brzo obaviti i radi kojih je napravljena ta struktura podataka su ubacivanje novih i brisanje najvećeg elementa iz skupa podataka
  - brisanje najvećeg podatka obavlja se izbacivanjem korijena i pozivanjem funkcije **`podesi`**, a ubacivanje novih radi se funkcijom **`ubaci`**
    - tako se postiže da se obje željene funkcije obavljaju u vremenu $O(\log_2 n)$

Ubrzanje algoritma

- gomila može biti napravljena za razne relacije njenih elemenata
  - gomilu s relacijom **veći od** zovemo *max heap*
  - gomilu s relacijom **manji od** zovemo *min heap*

Sortiranje gomilom

- heap sort:
  - element s vrha gomile zamjenjuje se s posljednjim elementom polja
  - gomila se skraćuje za 1 element i podešava
- složenost podešavanja je $O(log_2 n)$
- to se obavlja n puta pa je složenost sorta $O(n log_2 n)$
- razlika u vremenu izvođenja za različite redove veličine složenosti postane značajna za veliki n

HeapSort.cpp

Zadaci za vježbu

- Zadan je niz ulaznih podataka:

12, 15, 5, 3, 7, 2 18, 11, 4, 10

a) treba nacrtati potpuno binarno stablo koje je nastalo slijednim upisom ulaznih podataka

b) treba nacrtati podatkovnu strukturu gomila u koju su pohranjeni ulazni podaci

c) koliko iznosi apriorno vrijeme izvođenja za pretvorbu potpunog binarnog stabla u strukturu gomila?

Zadaci za vježbu

- Zadan je niz ulaznih podataka tipa `int`:

12, 5, 4, 10, 7, 8 11

a) treba nacrtati stablo koje predstavlja strukturu *gomila*, takvu da omogućuje rješenje zadatka pod b)

b) treba ilustrirati kako radi silazno sortiranje korištenjem strukture gomila (*heap sort*)

c) koliko je apriorno vrijeme potrebno za sortiranje n podataka?

Zadaci za vježbu

- Napisati program koji će u cjelobrojnom polju od **`n`** članova pronaći **`k`**-ti najveći član polja.
  a) Učitano polje sortirati po padajućim vrijednostima i ispisati član s indeksom **`k-1`**.
  b) Učitati **`k`** članova polja, sortirati ih po padajućim vrijednostima. Učitavati preostale članove polja. Ako je pojedini član manji od onoga s indeksom **`k-1`**, ignorirati ga, ako je veći umetnuti ga na pravo mjesto, a izbaciti član polja koji bi sad imao indeks **`k`**.
  c) Varirati postupke sortiranja te odrediti pripadna apriorna vremena i izmjeriti aposteriorna vremena izvođenja.
- Odrediti apriorna vremena trajanja, a izmjeriti aposteriorna vremena. Varirati postupak sortiranja.