

# Podatkovne strukture in algoritmi

Vrste s prednostjo

binomske kopice

## Vrsta s prednostjo

Osnovne operacije nad vrsto s prednostjo:

**dodajanje:**  $\text{Insert}(S, x)$  – v  $S$  dodamo nov element  $x$ .

**najmanjši:**  $\text{Min}(S) \rightarrow y$  – v  $S$  poiščemo najmanjši element  $y$ .

**odreži:**  $\text{DelMin}(S)$  – iz  $S$  izločimo najmanjši element.

Operacije nad posplošeno vrsto s prednostjo:

**izločanje:**  $\text{Delete}(S, x) \rightarrow y$  – iz  $S$  izločimo element  $x$ . Rezultat  $y$  je lahko Boolova vrednost `true` ali `false`, ki sporoči ali je bil element uspešno izločen ali ne, ali pa operacija ničesar ne vrne.

**spreminjanje:**  $\text{Decrease}(S, x, d)$  – v  $S$  elementu  $x$  zmanjšamo (povečamo) vrednost za  $d$ .

**zlij:**  $\text{Merge}(S_1, S_2) \rightarrow S$  – zlije vrsti s prednostjo v novo vrsto s prednostjo.

**levi sosed:**  $\text{Left}(S, x) \rightarrow y$  – v  $S$  poiščemo element  $y$ , ki je največji element, kateri je še manjši od  $x$ . Če takšnega elementa ni, vrne `null`.

**desni sosed:**  $\text{Right}(S, x) \rightarrow y$  – v  $S$  poiščemo element  $y$ , ki je najmanjši element, kateri je še večji od  $x$ . Če takšnega elementa ni, vrne `null`.

## Dosedanje izvedbe

operacija	seznam	drevo	kopica
dodajanje	$O(1)$	$O(\log n)$	$O(\log n)$
najmanjši	$O(n)$	$O(\log n)$	$O(1)$
odreži	$O(n)$	$O(\log n)$	$O(\log n)$
izločanje	$O(n)$	$O(\log n)$	$O(\log n)$
spreminjanje	$O(n)$	$O(\log n)$	$O(\log n)$
zlij	$O(\mu)$	$O(\mu \log n)$	$O(\mu \log n)$
levi sosed	$O(n)$	$O(\log n)$	$O(n)$
desni sosed	$O(n)$	$O(\log n)$	$O(n)$

Komentarji:

**izločanje:** poznamo referenco elementa

**spreminjanje:** čas brisanja + čas vstavljanja

**zlij:**  $\mu = \min(|S_1|, |S_2|)$

# Uvod v binomske kopice

Tudi binomska drevesa (*binomial trees*).

Struktura ni več implicitna, torej se vračamo k referencam.

Težave so se pojavile pri zlivanju in iskanju sosedov. Naša želja je, da naredimo vsaj zlivanje dovolj hitro.

Morda bi se dalo še kaj postoriti pri samem vstavljanju? Vstavljanje je v resnici zlivanje stare kopice z novo, ki ima samo en element.

# Dvojiška kopica

- *invarianca 1*: vsi elementi v  $L$  in  $D$  so večji ali enaki  $k$ 
  - zagotavlja urejenost podatkovne strukture in
  - posledično hitro iskanje (najmanjšega elementa) –  $O(1)$
- *invarianca 2*: podkopici  $L$  in  $D$  sta približno enako veliki
  - zagotavlja uravnoteženost strukture in zato učinkovita popravljanja –  $O(\log n)$

# Zlivanje kopic

- imamo kopici A in B ter tvorimo kopico C
- koren kopice C naj bo  $\min(A.\text{koren}, B.\text{koren})$  – recimo, da je to (WLOG) A.koren; in
- koren druge kopice postane naslednik novega korena – koren kopice A dobi še *enega* naslednika
- nova kopica *ni* več dvojiška
- še vedno omogoča hitro iskanje (invarianca 1)
- ni pa nujno (Murphy), da je uravnotežena (invarianca 2), kar pomeni, da popraviljanja niso več učinkovita

# Ideje?

- iskanje:  $O(1)$  – **super**
- zlivanje:  $O(1)$  (zato tudi vstavljanje) – **super**
- popravljanje:  $O(\text{višina kopice})$  – **katastrofa**
- Kaj sedaj?

# Rešitev

TEHNIKA: Če nekje malce popustimo, bomo morda drugje lahko veliko pridobili – *princip ravnoteženja*.

(KAJ NAM PA OMEJUJE MEJO, DO KATERE LAHKO POPUSTIMO? KOLIKO LAHKO V TEM PRIMERU POPUSTIMO?)

- naj bo zlivanje (in iskanje) malce počasnejše –  $O(\text{višina kopice})$
- bo pa zato struktura bolj uravnotežena, kar bo razorožilo Murphyja (nasprotnika, *adversary*) in bo zato tudi višina kopice manjša (upamo, da  $O(\log n)$ )
- *ideja*:
  - vrsta s prednostjo naj sestoji iz  $O(\log n)$  podstruktur (kopic);
  - kopice naj bodo različno velike;
  - a največja naj ne bo višja od  $O(\log n)$



# Ideja zlivanja

- pogledjmo (dvojiški) zapis poljubnega števila

1010110<sub>2</sub>

- novo števko dodamo na začetek, ko *velikost števila preraste* največje število, ki ga lahko predstavimo s prejšnjim številom števč; in
- dodana števka *omogoča zapis še enkrat več vrednosti*;
- število števč je  $\lceil \lg n \rceil$

## Ideja zlivanja – nadalj.

- zlivanje dveh števil (seštevanje) – vedno istoležeči števk, ki pa predstavljata tudi enakoo veliki vrednosti:

1010110

10111

1101101

- čas potreben, da zlijemo dve števili, je sorazmeren:

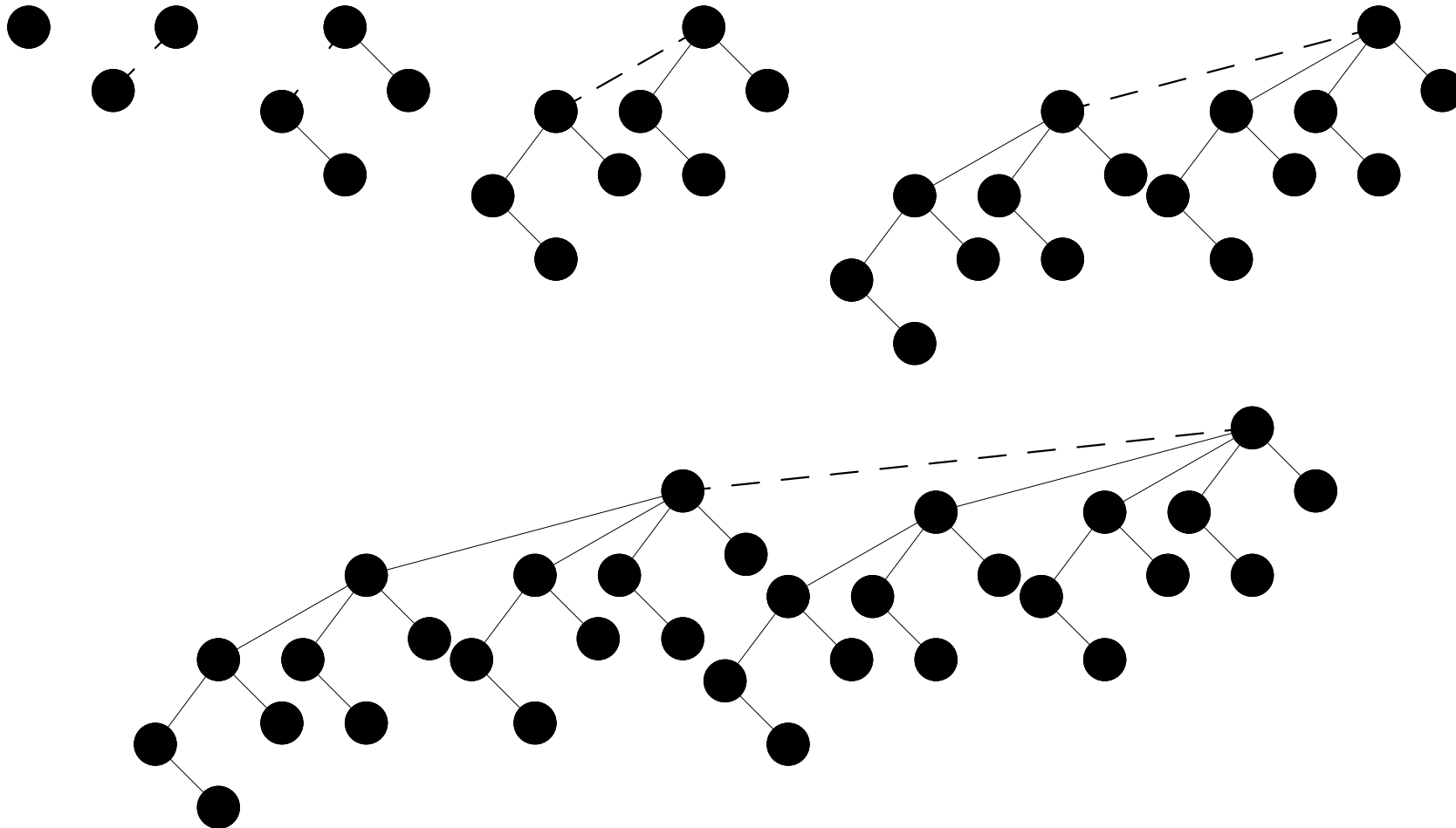
$$(\text{število števč večjega števila}) * (\text{čas seštevanja dveh števč}) = \\ \lceil \lg n \rceil * O(1) = O(\log n)$$

# Binomska drevesa

DEFINICIJA: binomsko drevo je rekurzivna podatkovna struktura, kjer velja:

- koren je najmanjši element v drevesu (urejenost);
- posamezen element je binomsko drevo  $B_0$
- binomsko drevo  $B_k$  sestoji iz dveh poddreves  $B_{k-1}$ , pri čemer je:
  - manjši od korenov poddreves tudi koren celotnega drevesa in
  - poddrevo z večjim korenom prvi naslednik celotnega drevesa  $B_k$

## Primeri binomskih dreves



## Lastnosti binomskega drevesa $B_k$

1. koren je najmanjši element v drevesu [iz definicije]
2. koren ima  $k$  naslednikov, pri čemer je prvo poddrevo  $B_{k-1}$ , drugo  $B_{k-2}$  in tako naprej do  $k$ -tega, ki je  $B_0$  [z indukcijo po  $k$ ]
3. v drevesu je  $n = 2^k$  elementov [z indukcijo po  $k$ ]
4. višina drevesa je  $k = \lg n$  [z indukcijo po  $k$ ]
5. število vozlišč na globini  $i$  je  $\binom{k}{i}$  [z indukcijo po  $k$ ] – zato so to *binomska drevesa*
6. ker »zlivamo« samo enako velika drevesa, je struktura uravnotežena
7. zlivanje dveh dreves lahko naredimo v času  $O(1)$

(DOKAŽITE ZGORNJE LASTNOSTI.)

# Binomska vrsta

Definicija:

- Binomska vrsta (kopica) sestoji iz *seznama* binomskih dreves  $B_0, B_1, \dots, B_{\lg n}$
- velikost zasedenega prostora je očitno  $O(n)$
  
- POZOR: odnosi med koreni kosameznih binomskih dreves v kopici niso določeni!

# Iskanje najmanjšega elementa

Ker gre za seznam, je sorazmerno dolžini seznama, ki pa je  $O(\log n)$ .

```
public int min(int x) {  
    int min;  
    binTree head;  
    head= binHeap.head();  
    tail= binHeap.tail();  
    min= head.min();  
    for (; tail != nil; head= tail.head(), tail= tail.tail())  
        min= Min(min, head.min());  
    return min;  
}
```

(KAKO SMO ŽE TO IZBOLJŠEVALI?)

# Zlivanje

- zlijemo enako veliki drevesi
- to počnemo na podoben način kot pri polnem seštevalniku (*full adder*)
  - ena števka iz prvega števila, ena iz drugega ter prenos
  - rezultat je nova števka ter prenos
  - rezultat ima enako število števk kot večje od števil ali eno več
- Čas zlivanja je sorazmeren dolžini daljšega sezname –  $O(\log n)$

(DEFINIRAJTE RAZRED `binTree`.)

(DEFINIRAJTE RAZRED `binHeap` (ENA OD METOD BO ZGORNJI `min`), KI JE IMPLEMENTACIJA VMESNIKA ZA VRSTE S PREDNOSTJO.)

(ZAPIŠITE ŠE KODO ZA METODO `merge` TER OSTALE OPERACIJE.)



## Osnovne operacije

**dodajanje:** isto kot zlivanje, le da zlivamo celo kopico z  $B_0$

**najmanjši:** glej zgoraj

**odreži:** ko odrežemo koren binomskega drevesa, nasledniki tvorijo v resnici seznam binomskih dreves, kar je ponovno kopica. Zato lahko naredimo naslednje:

1. iz izvirne kopice  $H$  izločimo drevo, z najmanjšim korenem  $B_k - O(1)$
2. drevesu  $B_k$  odrežemo koren in iz naslednikov naredimo kopico  $H_k - O(1)$
3. kopici  $H$  in  $H_k$  zlijemo v rezultat –  $O(\log n)$

## Posplošena vrsta s prednostjo

**spreminjanje:** za zmanjševanje podobno kot pri dvojiški kopici element splava navzgor dokler je potrebno –  $O(\log n)$ . Za zvečevanje razmislite za domačo nalogo!

**izločanje:** naredimo v dveh korakih:

1. najprej zmanjšamo prednost na  $-\infty - O(\log n)$
2. nato odrežemo najmanjši element –  $O(\log n)$

**zlij:** glej zgoraj

**sosed:** (levi in desni) – zahteva iskanje po celi strukturi –  $O(n)$

(ZAPIŠITE ŠE KODO ZA OSTALE OPERACIJE.)

## Rezultat

operacija	seznam	drevo	kopica	binomska
dodajanje	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
najmanjši	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$
odreži	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
izločanje	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
spreminjanje	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
zlij	$O(\mu)$	$O(\mu \log n)$	$O(\mu \log n)$	$O(\log n)$
levi sosed	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$
desni sosed	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$

Komentarji:

**izločanje:** poznamo referenco elementa

**spreminjanje:** čas brisanja + čas vstavljanja

**zlij:**  $\mu = \min(|S_1|, |S_2|)$

(V RAZPREDELNICI SO ASIMPTOTIČNE VREDNOSTI, KAJ PA V PRAKSI? IN VELIKOST STRUKTUR?)

# Odvečno delo

Kaj se zgodi, če samo velikokrat:

- dodajamo element,
- izločamo element,
- zlivamo kopici

Pri tem opravljamo nepotrebno delo, ki skrbi za strukturo kopice.

# Ideja

- pustimo strukturo kopice vnemar, ampak samo ohranjanjamo (dvojno povezan) seznam (obroč / krog) dreves; in
- šele, ko je potrebno, popravimo strukturo (operacije nad najmanjšim elementom).

Vse operacije iz prve skupine sedaj opravimo v konstantnem času, medtem ko so druge operacije veliko dražje.

A izkaže se, da je cena drugih operacij sorazmerna (odvisna) od števila opravljenih prvih operacij v nizu → če pogledamo celoten niz operacij nismo nič izgubili.

To je dobro znana tehnika amortizacije.

(KAJ PRIDOBIMO?)

# Fibbonacijeva kopica

Zelo preprosto povedano je to binomska kopica z amortiziranimi operacijami.

Pri analizi nastopajo Fibbonacijeva stevila in zato Fibbonacijeva kopica. (KOLIKO JE  $k$ .TO FIBBONACIJEVO STEVILO?)

Doslej smo se pri analizi časovne zahtevnosti vedno spraševali, kakšen je čas *ene operacije v najslabšem primeru*. Sedaj se sprašujemo, kako slabo je lahko izvajanje *niza operacij v najslabšem primeru*.

To, da druga vrednost ni slabša kot vsota prvih vrednosti je očitno. Pa je lahko boljša? (RAZMISLITE!)

# Zahtevnost

operacija	seznam	drevo	kopica	binomska	Fibonaccijska
dodajanje	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$
najmanjši	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(1)$
odreži	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
izločanje	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)^\dagger$
spreminjanje	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)^\dagger$
zlij	$O(\mu)$	$O(\mu \log n)$	$O(\mu \log n)$	$O(\log n)$	$O(1)$
levi sosed	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$
desni sosed	$O(n)$	$O(\log n)$	$O(n)$	$O(n)$	$O(n)$

Komentarji:

**izločanje:** poznamo referenco elementa

**spreminjanje:** čas brisanja + čas vstavljanja

**zlij:**  $\mu = \min(|S_1|, |S_2|)$

$\dagger$ : če ne popravljamo najmanjšega elementa je  $O(1)$