Algoritmy a Dátové Štruktúry

Jana Katreniaková
katreniakova@dcs.fmph.uniba.sk

Utriedený zoznam pomocou B-stromov

B-strom

B strom rádu m je (2m+1)-árný strom ktorý spĺňa:

- Každý vrchol (stránka) má najviac medzi m a 2m kľúčov (okrem koreňa)
- Každý vrchol je
  - Listový a nemá žiadnych potomkov
  - Vnútorný s k kľúčmi a k+1 nasledovníkmi
- Všetky listy sú na jednej úrovni

B-strom ako zovšeobecnenie binárneho stromu

Vo vrchole X: kľúče k1, k2, . . . , km a ich podstromy T1..Tm.

- Všetky prvky menšie ako k1 sú v podstrome T0.
- Všetky prvky x také, že $ki < x < ki + 1$ pre $1 = i < m$ sú v podstrome Ti.
- Všetky prvky väčšie ako km sú v podstrome Tm.

Utriedený zoznam pomocou B-stromov

B-strom

B strom rádu m je (2m+1)-árný strom ktorý spĺňa:

- Každý vrchol (stránka) má najviac medzi m a 2m kľúčov (okrem koreňa)
- Každý vrchol je
  - Listový a nemá žiadnych potomkov
  - Vnútorný s k kľúčmi a k+1 nasledovníkmi
- Všetky listy sú na jednej úrovni

B-strom ako zovšeobecnenie binárneho stromu

Vo vrchole X: kľúče k1, k2, . . . , km a ich podstromy T1..Tm.

- Všetky prvky menšie ako k1 sú v podstrome T0.
- Všetky prvky x také, že $ki < x < ki + 1$ pre $1 = i < m$ sú v podstrome Ti.
- Všetky prvky väčšie ako km sú v podstrome Tm.

Insert do B-stromu

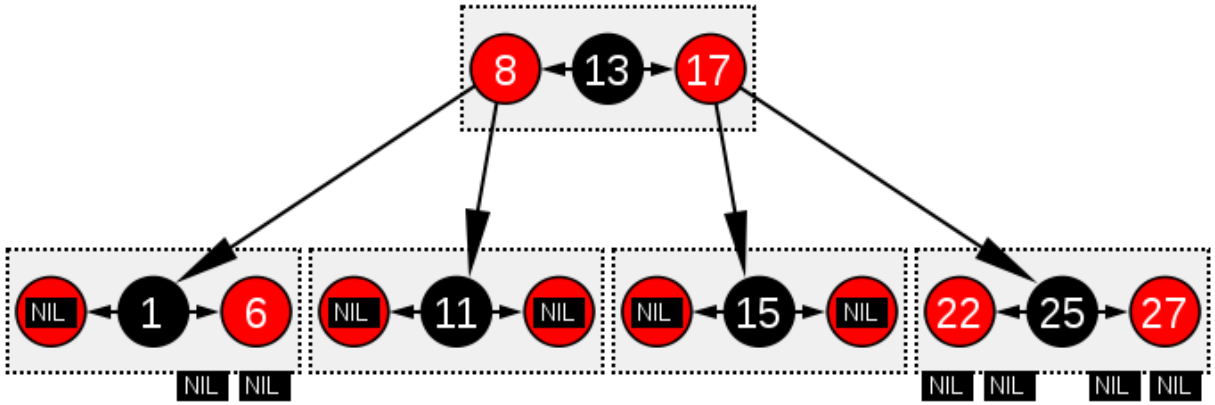
- V triviálnom prípade vieme kľúč priamo vložiť
- Ak vhodný vrchol X obsahuje 2m kľúčov potrebujeme strom upraviť
  - X sa rozdelí na dve časti Xa a Xb (tzv. štiepenie - split)
  - Všetkých 2m+1 kľúčov sa rozdelí rovnomerne mezi tieto vrcholy a ostane jeden nezaradený kľúč K
  -
    - Xa obsahuje kľúče ki <= K, 1 = i = m
      Xb obsahuje kľúče li >= K, 1 = i = m
  - Ešte potrebujeme začleniť kľúč K do vrcholu predchodcu

Delete z B-stromu

- Delete vrchola x – ako v bin. vyhľadávacom strome
  - Nahrádzame najbližším menším alebo najbližším väčším
- V prípade, že nemôžem vymazať kvôli počtu prvkov - požičiam si od suseda.
- A ak ani sused nemôže požičať, tak to znamená, že má m prvkov, čo s mojími m-1 a jedným od predchodcu je v poriadku
  - Jedine sme tým mohli spôsobiť problém u predchodcu.

Zhrnutie B-stromy

- Má menšiu hĺbku aj keď sa to asymptoticky nemení
- Všetky operácie asymptoticky ako v AVL strome
- Špeciálny prípad - tzv. červeno-čierny strom = (2-4)B-strom



Prioritná fronta

Abstraktná dátová štruktúra prioritná fronta

- Postupnosť objektov - u nás čísel (nezaujíma nás, ako prichádzali, ale ich hodnota)
- Operácie hľadanie prvku (s najväčšou prioritou), vloženie prvku, vymazanie prvku s najvyššou prioritou, ...

Možnosti implementácie

- Kompaktné množiny - polia, polia s dyn. veľkosťou
- Spájané zoznamy
- Stromy
- Halda

Prioritná fronta

Abstraktná dátová štruktúra prioritná fronta

- Postupnosť objektov - u nás čísel (nezaujíma nás, ako prichádzali, ale ich hodnota)
- Operácie hľadanie prvku (s najväčšou prioritou), vloženie prvku, vymazanie prvku s najvyššou prioritou, ...

Možnosti implementácie

- Kompaktné množiny - polia, polia s dyn. veľkosťou
- Spájané zoznamy
- Stromy
- Halda

Prioritná fronta pomocou kompaktných množín

Neutriedené zoznamy

- HighestPriority: O(N)
- Vkladanie: O(1) - na koniec
- Vyberanie: Nájdeme najvyššiu prioritu + zrušíme medzery

Utriedené zoznamy

- HighestPriority: O(1) - prvý prvok?
- Vkladanie: na konkretne miesto + pozor na diery O(N)
- Vymazanie: O(N) - resp. ak budem mat HighestPriority na konci tak O(1)

Prioritná fronta pomocou kompaktných množín

Neutriedené zoznamy

- HighestPriority: O(N)
- Vkladanie: O(1) - na koniec
- Vyberanie: Nájdeme najvyššiu prioritu + zrušíme medzery

Utriedené zoznamy

- HighestPriority: O(1) - prvý prvok?
- Vkladanie: na konkretne miesto + pozor na diery O(N)
- Vymazanie: O(N) - resp. ak budem mat HighestPriority na konci tak O(1)

Prioritná fronta pomocou spájaných zoznamov

Neutriedené spájané zoznamy

- HighestPriority: O(N)
- Vkladanie: na koniec O(1)
- Vyberanie: potrebujeme nájsť najvyššiu prioritu O(N)

Utriedené spájané zoznamy

- HighestPriority: O(1) - začiatok/koniec
- Vkladanie: O(N)
- Vyberanie: O(1) - prvý/posledný prvok

Prioritná fronta pomocou spájaných zoznamov

Neutriedené spájané zoznamy

- HighestPriority: O(N)
- Vkladanie: na koniec O(1)
- Vyberanie: potrebujeme nájsť najvyššiu prioritu O(N)

Utriedené spájané zoznamy

- HighestPriority: O(1) - začiatok/koniec
- Vkladanie: O(N)
- Vyberanie: O(1) - prvý/posledný prvok

Prioritná fronta pomocou stromov

Vyvážené stromy

- HighestPriority: O(log N) - vieme kde je minimum resp. maximum ale kym sa k nemu dostaneme...
- Vkladanie: O(log N)
- Vyberanie HighestPriority: O(log N) - ako teda každého iného prvku

Prioritná fronta pomocou haldy

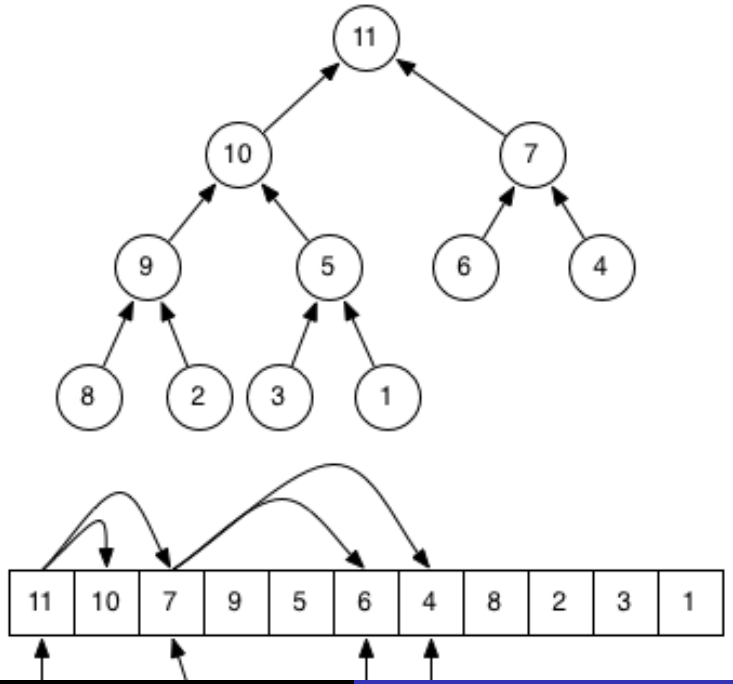
Halda

- Skoro úplný binárny strom - kaziť to môže iba najnižšia úroveň aj tam sú však prvky od začiatku úrovne
- Ideálne na reprezentáciu v poli
- Na rozdiel od stromu je otec najväčší (najmenší) z haldy
- Výška haldy je $1 + \lfloor log_2 N \rfloor$

Prioritná fronta pomocou haldy

Halda

- Skoro úplný binárny strom - kaziť to môže iba najnižšia úroveň aj tam sú však prvky od začiatku úrovne
- Ideálne na reprezentáciu v poli
- Na rozdiel od stromu je otec najväčší (najmenší) z haldy
- Výška haldy je $1 + \lfloor log_2 N \rfloor$



Malá odbočka - to sme už videli

Binárny strom nemusíme nutne reprezentovať pomocou pointrov. Dá sa ukladať (nie síce veľmi efektívne) aj v poli. Ale halda sa dá ukladať efektívne!

- Koreň stromu je A[1]
- Predchodca prvku A[i] je Parent(i) = $\lfloor i/2 \rfloor$
- Potomkovia prvku A[i] sú
  - Ľavý syn: Left(i) = 2*i
  - Pravý syn: Right(i) = 2*i + 1

Prioritná fronta pomocou haldy

INSERT(A,x)

```
1  k = ++length(A); A[k] = x;
2  while (k > 1 ∧ A[k/2] < A[k])
3  do Swap(k/2, k); k = k/2;
4  return k;
```

REMOVEHIGHEST(A)

```
1  max = A[1]; Swap(1, length(A) − −); k = 1;
2  while (2 ∗ k <= length(A))
3  do if A[2 ∗ k] > A[2 ∗ k + 1]
4        then if A[2 ∗ k] > A[k]
5                then Swap(k, 2 ∗ k); k = 2 ∗ k;
6        else if A[2 ∗ k + 1] > A[k]
7                then Swap(k, 2 ∗ k + 1); k = 2 ∗ k + 1;
8  return max;
```

Vytvorenie a úprava haldy

UpravHore

- Prvok buble hore až kým nenarazí na prvok nad ktorý už nemôže
- Vždy teda vymieňam iba $(k/2, k)$

UpravDole

- Prvok buble dole s tým, že sa vždy vymieňa so svojím väčším synom
- Vždy teda vymieňam buď $(k, 2 * k)$ alebo $(k, 2 * k + 1)$

Vytvorenie Haldy - čierna mágia

- `for (i=N/2; i>=1; i-) UpravDole(i);`
- Ale skúsime vysvetliť...

Vytvorenie a úprava haldy

UpravHore

- Prvok buble hore až kým nenarazí na prvok nad ktorý už nemôže
- Vždy teda vymieňam iba $(k/2, k)$

UpravDole

- Prvok buble dole s tým, že sa vždy vymieňa so svojím väčším synom
- Vždy teda vymieňam buď $(k, 2 * k)$ alebo $(k, 2 * k + 1)$

Vytvorenie Haldy - čierna mágia

- `for (i=N/2; i>=1; i-) UpravDole(i);`
- Ale skúsime vysvetliť...

Vytvorenie a úprava haldy

UpravHore

- Prvok buble hore až kým nenarazí na prvok nad ktorý už nemôže
- Vždy teda vymieňam iba $(k/2, k)$

UpravDole

- Prvok buble dole s tým, že sa vždy vymieňa so svojím väčším synom
- Vždy teda vymieňam buď $(k, 2 * k)$ alebo $(k, 2 * k + 1)$

Vytvorenie Haldy - čierna mágia

- `for (i=N/2; i>=1; i-) UpravDole(i);`
- Ale skúsime vysvetliť...

Zhrnutie Halda

- Reštrukturalizácia haldy – Hore, Dole O(log n)
- HighestPriority O(1)
- Vloženie prvku – Insert O(log n)
  - Samotné vloženie O(1)
  - Úprava smerom hore O(log n)
- Vymazanie najväčšieho – Delete O(log n)
  - Vymazanie a náhrada listom O(1)
  - Úprava smerom hore O(log n)
- Vytvorenie haldy O(n.log n)
- Vyhľadávanie prvku – Member O(n)