Föreläsning 4
Datastrukturer (DAT037)

Fredrik Lindblad[1]

2016-11-10

[1] Slides skapade av Nils Anders Danielsson har använts som utgångspunkt. Se http://www.cse.chalmers.se/edu/year/2015/course/DAT037

Förra gången:

- Länkade listor, pekarjonglering.
- Komplexitet för implementeringar av listor.
- Invarianter, assertions, testning.
- Träd.

Cirkulära arrayer

Cirkulära arrayer

Implementationsteknik för köer.

Cirkulära arrayer

```
public static class CircularArrayQueue<A> {
    private A[] queue;
    private int size;
    private int front;  // nästa element
    private int rear;  // nästa lediga

    public CircularArrayQueue(int capacity) {
        if (capacity <= 0) {
            throw new IllegalArgumentException(
             "non-positive capacity");
        }
        queue = (A[]) new Object[capacity];
        size = front = rear = 0;
    }
    ...
```

Cirkulära arrayer

```
public void enqueue(A a) {
    if (size == queue.length) {
        doubleCapacity();
    }

    size++;
    queue[rear] = a;
    rear = (rear + 1) % queue.length;
}
```

Cirkulära arrayer

```
public A dequeue() {
    if (size == 0) {
        throw new NoSuchElementException(
         "queue empty");
    }

    size--;
    A a = queue[front];
    queue[front] = null;  // undvik minnesläckor
    front = (front + 1) % queue.length;

    return a;
}
```

Cirkulära arrayer

```
    private void doubleCapacity() {
        A[] newQueue =
            (A[]) new Object[2 * queue.length];

        for (int i = 0, j = front; i < size;
                i++, j = (j + 1) % queue.length) {
            newQueue[i] = queue[j];
        }
        queue = newQueue;
        front = 0;
        rear  = size;
    }
}
```

Prioritetsköer

Prioritetsköer

Köer där varje element har viss prioritet.

Gränssnitt (exempel):

- Konstruerare för tom kö.
- `insert`: Lägger till element.
- `find-min`: Ger tillbaka minsta elementet.
- `delete-min`:
  Tar bort och ger tillbaka minsta elementet.
- `increase-key`/`decrease-key`/`modify-key`:
  Ändrar ett elements prioritet.
- `merge`: Slår ihop två köer.

Prioritetsköer

Några tillämpningar:

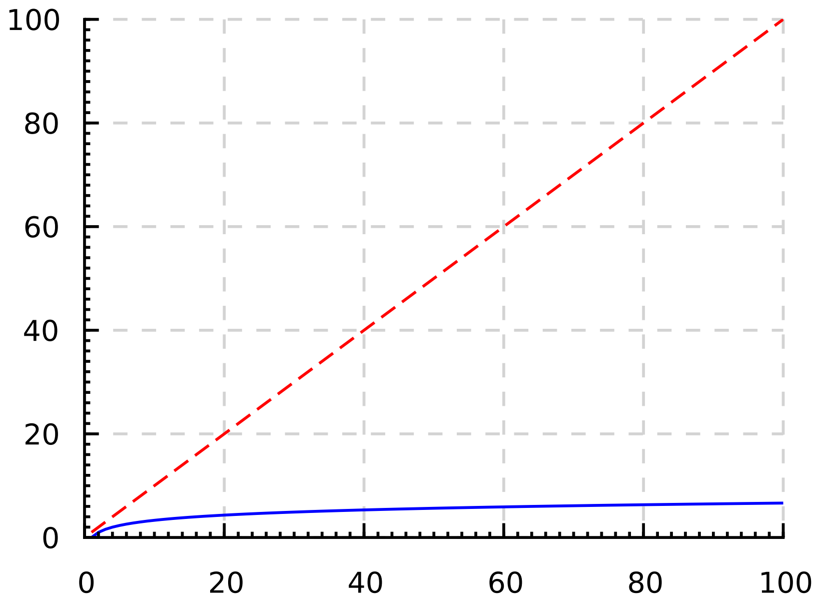
- Schemaläggning av processer.
- Sortering.
- Dijkstras algoritm (labb 3).

Labb 2: Implementera prioritetskö.

Om man implementerar prioritetskö-ADTn med listor, vad blir tidskomplexiteten (ev amorterad) för `insert` och `delete-min`?

- A – `insert`: $\Theta(1)$, `delete-min`: $\Theta(1)$.
- B – `insert`: $\Theta(1)$, `delete-min`: $\Theta(n)$.
- C – `insert`: $\Theta(n)$, `delete-min`: $\Theta(1)$.
- D – `insert`: $\Theta(n)$, `delete-min`: $\Theta(n)$.

Anta att prioriteter kan jämföras på konstant tid.



100
80
60
40
20
0
0
20
40
60
80
100
log₂ n
n

Binära heapar

Binära heapar

Kompletta binära träd med heapordningsegenskapen.

Heapordningsegenskapen

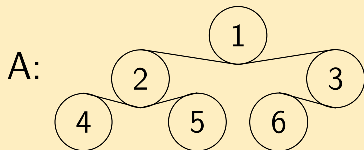
Varje nod är mindre än eller lika med alla sina barn.

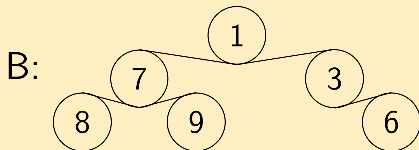
Komplett binärt träd

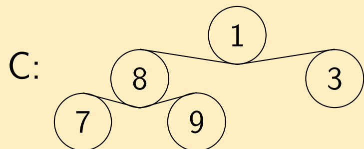
Så lågt som möjligt, alla nivåer helt fyllda utom möjligtvis den sista, som är fylld från vänster.

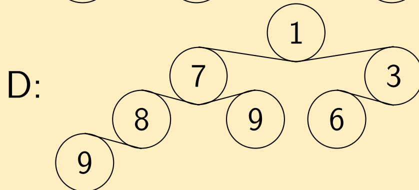
En binär heap med n noder har höjden $\Theta(\log n)$.

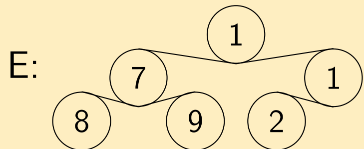
Identifiera alla binära heapar.

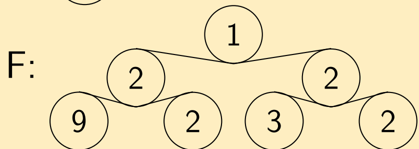












Binär heap implementerar prio. kö

- Tom kö: Tomt träd.
- `find-min`: Ge tillbaka roten.
- `insert`: Stoppa in sist. Bubbla upp.
- `delete-min`: Ta bort roten.
  Stoppa in sista elementet överst. Bubbla ned.
  Ge tillbaka gamla roten.
- `modify-key`: Ändra prioritet,
  bubbla åt rätt håll.

Bubbla upp/ned tills heapordningsegenskapen återställts.

Bubbla upp/ned

- Om värdet är mindre (prio är högre) än i noden ovan, byt plats och kolla rekursivt upp tills ordningen är rätt.
- Om värdet är större (prio är lägre) än i någon av noderna nedan, byt plats med barnet som är minst och fortsätt rekursivt i denna gren.

Tidskomplexitet

Om man kan hitta noderna snabbt:

- Tom kö: $\Theta(1)$.
- `find-min`: $\Theta(1)$.
- `insert`: $O(\log n)$ (kanske amorterat).
- `delete-min`: $O(\log n)$ (kanske amorterat).
- `modify-key`: $O(\log n)$.

(Givet att jämförelser tar konstant tid.)

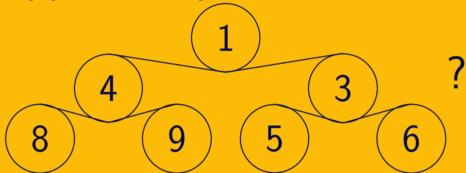
De flesta noderna är långt ned i trädet.
Medelkomplexitet för `insert`: $O(1)$.

Implementation av binära heapar

Kan representera trädet med array (labb 2).

- Roten på position 1 (eller 0).
- Sista elementet på position n $(n-1)$.
- Första tomma cellen på position $n+1$ (n).
- Nod is vänstra barn: $2i$ $(2i+1)$.
- Nod is högra barn: $2i+1$ $(2i+2)$.
- Nod is förälder $(i>1)$: $\lfloor i/2 \rfloor$ $((i>0)$: $\lfloor (i-1)/2 \rfloor)$.

Vad blir resultatet om `delete-min` appliceras på



?

A:

3	4	5	6	8	9

B:

3	5	6	4	8	9

C:

3	4	8	9	5	6

D:

3	4	5	8	9	6

`modify-key`

När man implementerar `modify-key`,
hur hittar man noden snabbt?

Kan använda extra datastruktur.

Exempel: Hashtabell.

`build-heap`

`build-heap`: Konverterar lista/array/... till heap.

- Stoppa in alla elementen i godtycklig ordning.
- Bubbla *ned* ett element i taget,
  med början på det nedersta, högraste.
  (Kan hoppa över alla löv.)

Heapordningsegenskapen uppfylls
(kan bevisas med induktion).

`build-heap`: tidskomplexitet

- Tid för viss nod: $O(\text{nodens höjd})$.
  (Givet $O(1)$-jämförelser.)
- Total tid: $O(\text{summan av alla höjder})$.
- Nästan alla noder är långt ned.
- Total tid: $\Theta(n)$.
- Bevis: Se boken.

Leftistheapar

`merge`

- Det verkar inte gå att slå ihop två binära heapar, implementerade med arrayer, på ett effektivt sätt.
- Med *leftistheapar*: $O(\log n)$

Leftistheapar

- Heapordnade (ofta pekarbaserade) binära träd, som inte är kompletta utan uppfyller annan balanseringsinvariant, leftist träd-egenskapen.
- Grundläggande operation: `merge`.
- Lätt att implementera `insert`, `delete-min` med hjälp av `merge`.

Null path length

För en nod, X, i ett träd är $\mathrm{npl}(X)$ (null path length of X) kortaste vägen till ett löv. (Jämför höjden som är längsta vägen till ett löv.) För tomt träd är null path length -1 (liksom för höjden).

Leftist träd-egenskapen

För varje nod i trädet gäller för dess vänster- och högerbarn, l och r, att $\mathrm{npl}(l) \geq \mathrm{npl}(r)$.
För ett leftist träd gäller att den högraste vägen har max $\lfloor \log(N+1) \rfloor$ noder, där N är totalt antal noder.

Sammanfattning

- Cirkulära arrayer
- Prioritetskö
- Heap