Föreläsning
Datastrukturer (DAT037)

Nils Anders Danielsson

2015-12-14

Idag

- Frågor? Är något oklart inför tentan?
- Sammanfattning.

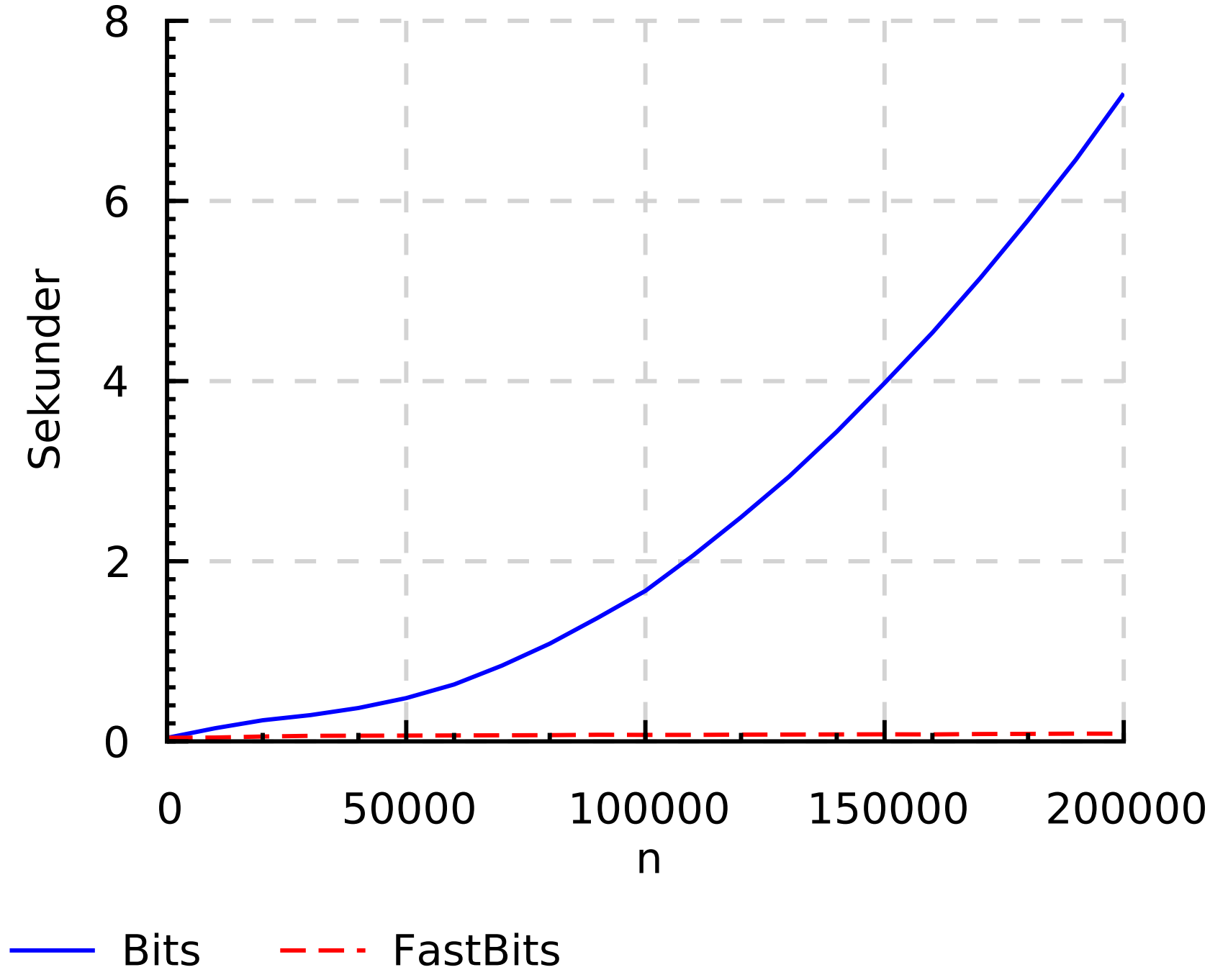
Exempel från föreläsning 1

Dåligt val av datastruktur

```
public class Bits {

    public static void main(String[] args) {

        int n = Integer.parseInt(args[0]);

        String bits = new String();

        for (int i = 0; i < n; i++) {
            bits += i & 1;
        }

        System.out.print(bits);

    }
}
```

Bättre val av datastruktur

```
public class FastBits {

    public static void main(String[] args) {

        int n = Integer.parseInt(args[0]);

        StringBuffer bits = new StringBuffer();

        for (int i = 0; i < n; i++) {
            bits.append(i & 1);
        }

        System.out.print(bits);

    }
}
```


8
6
4
2
0
Sekunder
0
50000
100000
150000
200000
n
Bits
FastBits

Hur har det gått?

Jag hoppas att ni nu (snart) har verktyg för att:

- Förutse (ungefär) hur snabbt (vissa av) era program kommer att gå.
- Göra (vissa) program mer effektiva.

Ta något lite mer komplicerat program som ni skrev innan ni gick den här kursen. Om ni hade skrivit programmet nu, hade ni gjort det på ett annat sätt? Vad hade ni gjort annorlunda?

- Svara kortfattat i Pingo.

Sammanfattning

Modeller

Vi har använt förenklade modeller av verkligheten för att analysera våra datastrukturer och algoritmer, i första hand den uniforma kostnadsmodellen.

Modellerna har begränsningar.

Ordonotation

Ordonotation gör det möjligt att dölja irrelevanta – men även relevanta – detaljer.

- Definition (O, Ω, Θ).
- Regler.

Vilka påståenden är sanna?

- $5n^2 + 3n + 7 = O(n^2)$.
- $5n^2 \cdot 3n \cdot 7 = O(n^2)$.
- $(\log_2 n)^2 = \Theta(\log n)$.
- $\log_2(n^2) = \Theta(\log n)$.
- $5n + 2\log_2(7n + 3) = O(n)$.
- $5n + 2\log_2(7n + 3) = O(\log n)$.

Ordonotation

Exempel på regler:

- Polynom i n av grad $k = O(n^k)$.
- $\log(n^k) = O(\log n)$.
- $f(n) + g(n) = O(\max(f(n), g(n)))$.

Tidskomplexitetsanalys

Vi har analyserat många algoritmers tidskomplexitet.

En enkel analys kan se ut så här:

- Vi känner till tidskomplexiteten hos vissa standardkomponenter.
- Vi räknar sedan hur många gånger standardkomponenterna körs.
  - ”En gång per varv i loopen.”
  - ”En gång per nod.”
  - ”En gång per kant.”

Pseudokod

- Blandning av programmeringsspråk, matematisk notation och naturligt språk.
- Mål: Fokusera på det viktiga, undvik onödiga detaljer.
- Ibland kan det vara bra med fler detaljer, ibland färre.

Abstrakt datatyp/datastruktur

- Abstrakt datatyp: Matematisk abstraktion.
- Datastruktur: Implementation.

Några (klasser av) abstrakta datatyper

- Listor.
- Stackar.
- FIFO-köer.
- Prioritetsköer.
- Mängder.
- Avbildningar (”maps”).
- Grafer.

Tips: Återanvänd kod

- Ofta behöver man inte implementera datastrukturer själv.
- Om man någon gång behöver det kan man ändå dra nytta av existerande standarddatastrukturer.

2012/08:2

Uppgiften är att konstruera en datastruktur som representerar ”dubbelriktade maps” (bijektioner mellan ändliga mängder):
`insert(`s`,`t`)`, `target(`s`)`, `source(`t`)`.

```
public class Bijection<S, T> {

  private Map<S, T> to; private Map<T, S> from;

  public Bijection() { to   = new HashMap<S, T>();
                       from = new HashMap<T, S>(); }

  public void insert(S s, T t) {
    if (to.containsKey(s) || from.containsKey(t)) {
      throw new IllegalArgumentException();
    }
    to.put(s, t); from.put(t, s);
  }

  public T target(S s) { return to.get(s); }

  public S source(T t) { return from.get(t); }
}
```

Listor

ADT med olika implementationer.

Länkade listor:

- Långsam indexering.
- Snabb insättning (exkl sökning) i mitten.
- Vaktposter? Enkellänkade, dubbellänkade?
- Pekarjonglering. Testa! Invarianter.

Dynamiska arrayer:

- Snabb indexering.
- Snabb insättning sist.
- Amorterad tidskomplexitet (bokföringsmetoden).

Amorterad tidskomplexitet

- Abstraktion som gör det lättare (?) att analysera vissa datastrukturer.
- Man kan låta tidiga, billiga operationer ”betala” för dyra, sena.

Amorterad tidskomplexitet

Bokföringsmetoden:

- Spara "mynt" i datastrukturen.
- Amorterad tidskomplexitet =
  faktisk tidskomplexitet
  + värdet av nya mynt
  − värdet av sparade mynt som används.
- Om vi börjar med noll mynt:
  Total amorterad tidskomplexitet övre gräns för total faktisk tidskomplexitet.
- Analysen fungerar ej för persistenta datastrukturer.

Persistenta datastrukturer

- Efter förändring finns gamla kopian kvar.
- `sort :: [Integer] -> [Integer]`, inte `sort :: [Integer] -> void`.
- Nackdel:
  Vissa operationer kan vara mindre effektiva.
- Fördel: Kanske lättare att förstå/testa.

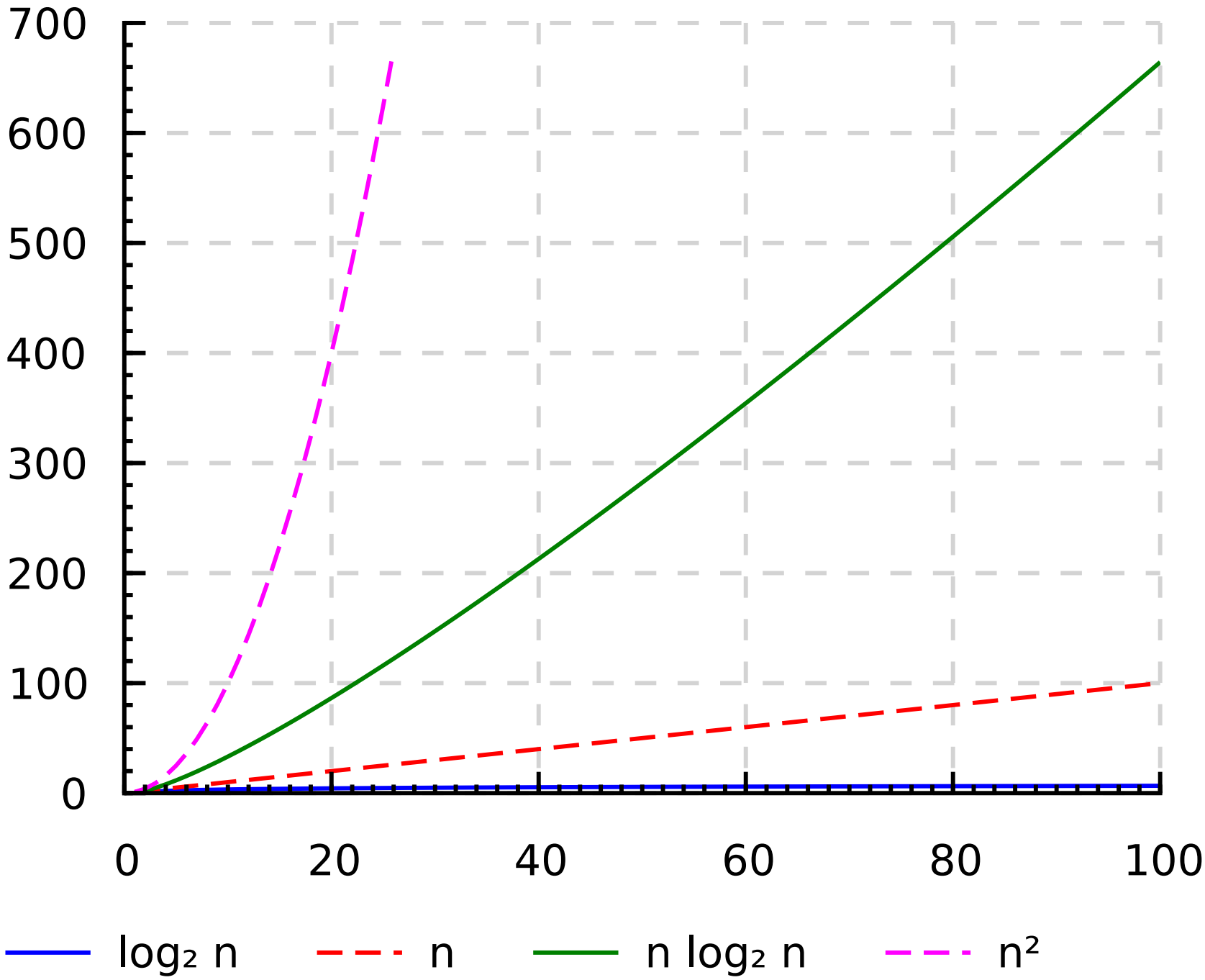
Vad är tidskomplexiteten för följande program, där `i++` körs n gånger?
Använd det logaritmiska kostnadskriteriet.
(Se `int` som en dynamisk array med bitar.)

```
int i = 0;
i++;
i++;
...
i++;
```

- $\Theta(1)$.
- $\Theta(\log n)$.
- $\Theta(n)$.
- $\Theta(n \log n)$.
- $\Theta(n^2)$.

Sortering

- Undre gräns: $\Omega(n \log n)$.
- Insättningssortering: $O(n^2)$.
- Heapsort: $O(n \log n)$.
- Mergesort: $O(n \log n)$.
- Quicksort: $O(n \log n)$ (medel).
- Hinksortering: $O(N + n)$ (N: antalet hinkar).
- Radixsortering: $O(d(N + n))$
  (N: "talbasen", d: största antalet "siffror").
- Stabil? Adaptiv? In-place?


700
600
500
400
300
200
100
0
0
20
40
60
80
100
log₂ n
n
n log₂ n
n²

Stackar, FIFO-köer

- Enkla implementationer med (olika sorters) listor.
- Cirkulära arrayer.
- Två listor. Persistent. $O(1)$ (amorterat) vid enkeltrådad användning.

Prioritetsköer

- Binära heapar:
  - Heapordnade kompletta binära träd.
  - Trädet representeras ofta av en array.
  - Bubbla upp/ned.
- Leftistheapar:
  - Heapordnade träd med leftistinvarianten: vänstra barnets högerryggrad $\geq$ högra barnets.
  - Effektiv `merge`.

Mängder, avbildningar

Hashtabeller:

- Hashfunktioner.
- Kollisioner.
- Kedjning, öppen adressering.
- Lastfaktor.

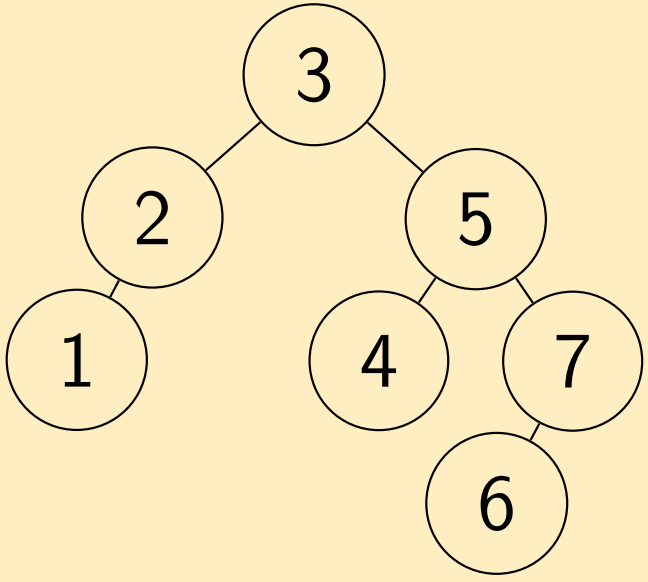
Mängder, avbildningar

Olika sorters sökträd:

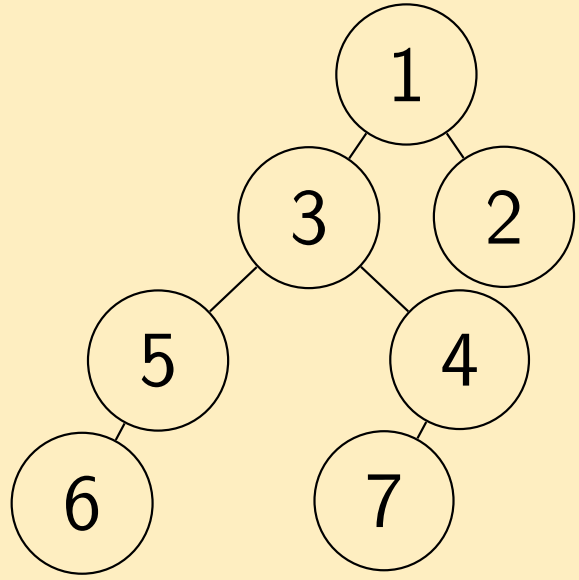
- Obalanserade binära sökträd:
  - Långsamma vid insättning av sorterad sekvens.
- AVL-träd:
  - Höjdbalanserade.
  - Rotationer.
- Rödsvarta träd, B-träd.

Klassificera träden:
Binär minheap, leftistminheap, AVL-träd.

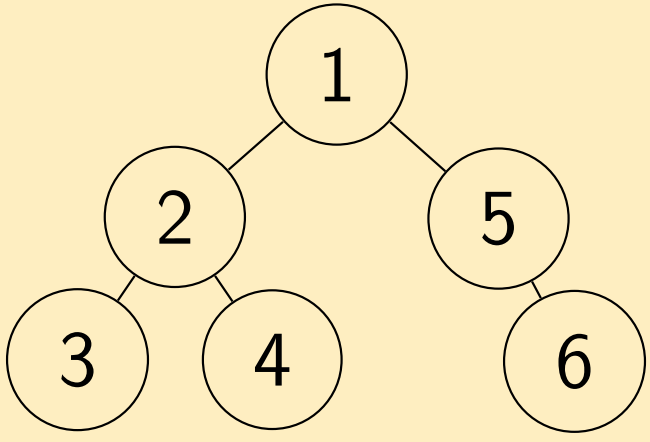
A:



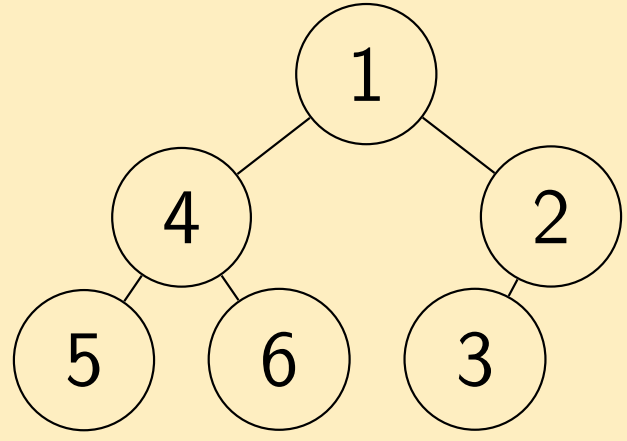
B:



C:



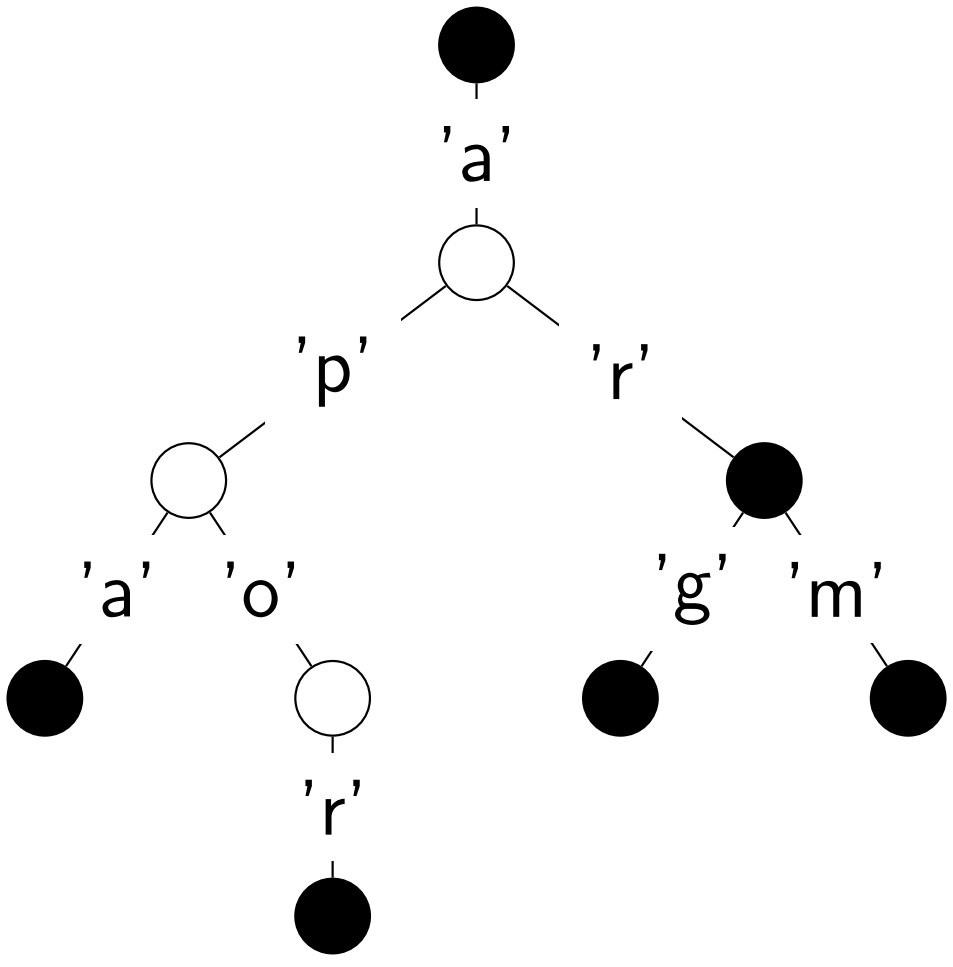
D:



Mängder, avbildningar

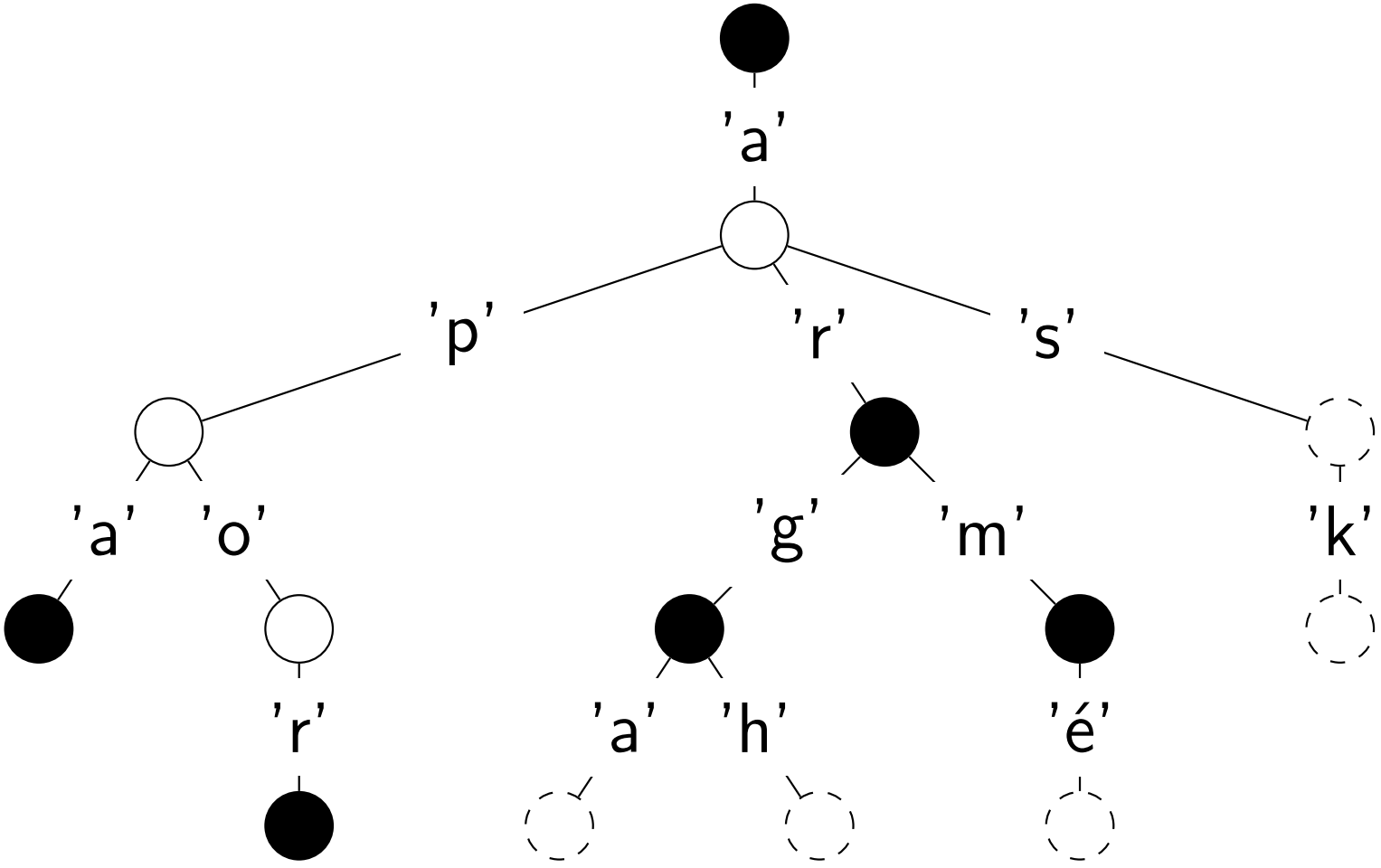
Prefixträd:

- Nycklar: Strängar av tecken.



Mängder, avbildningar

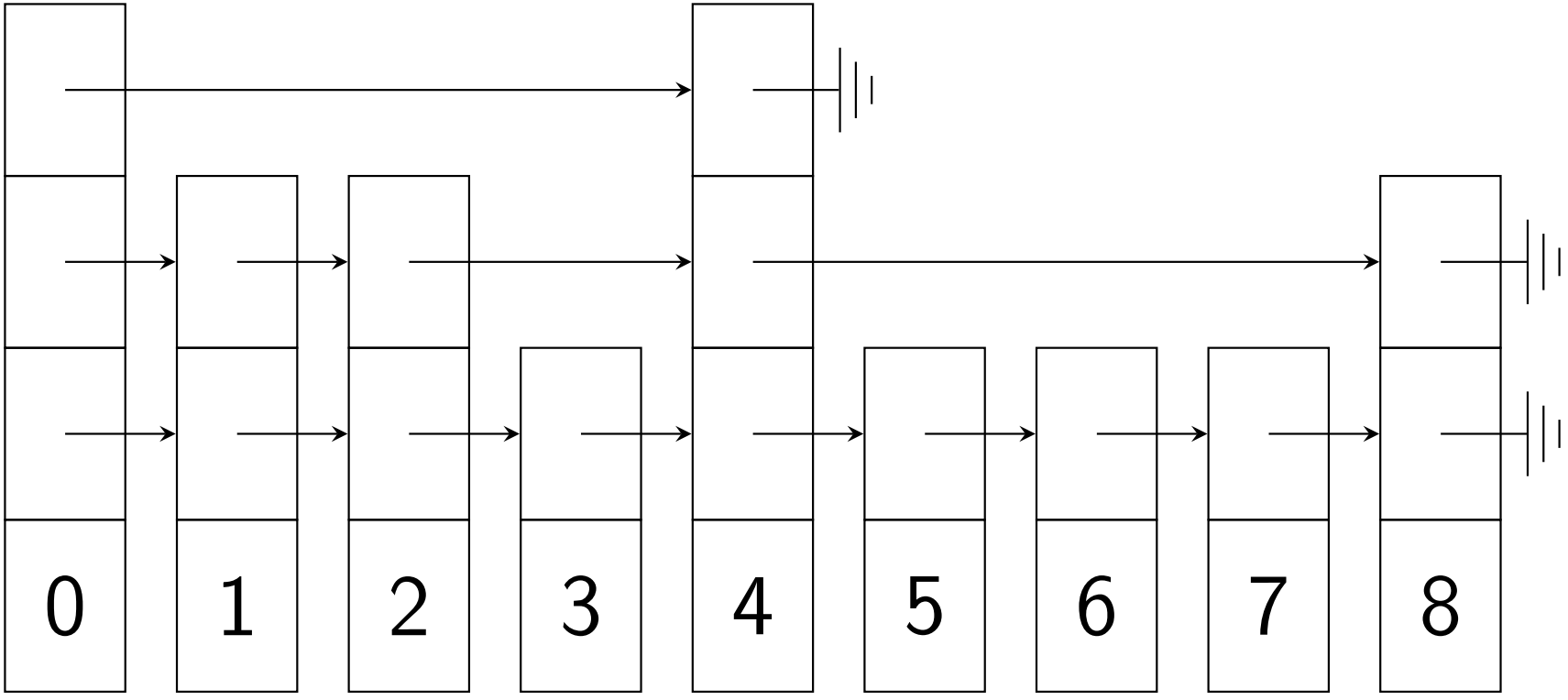
Tillägg: För att undvika långsam `toList`, se till att varje löv svarar mot en sträng som finns i trädet:



Mängder, avbildningar

Skipplistor:

- Insättning: Randomiserad algoritm.
- Förväntad tidskomplexitet.

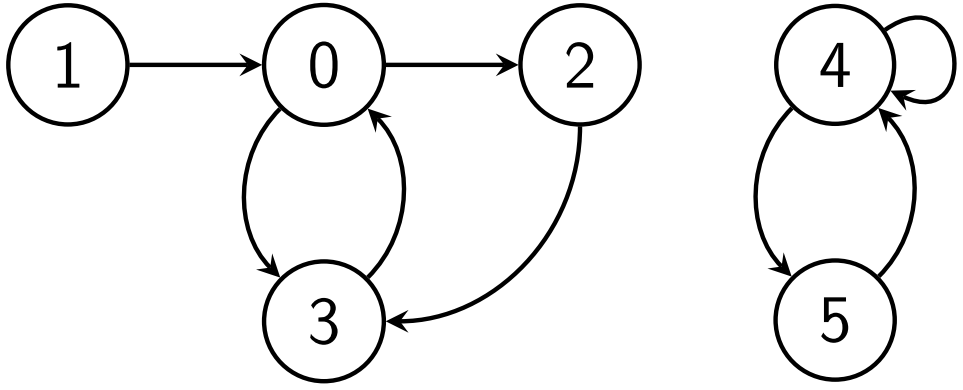


Grafer

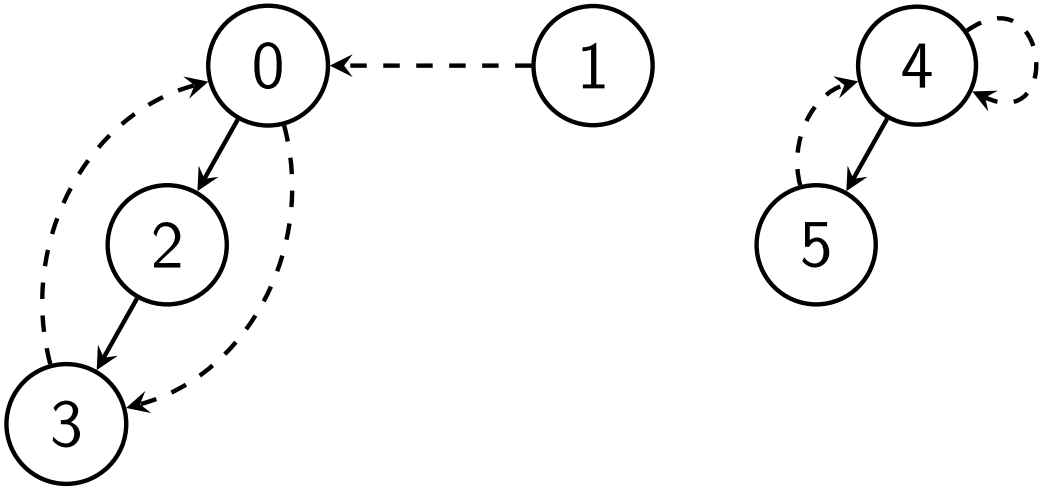
- ADT: $G = (V, E)$.
- Datastrukturer: Grannlistor, grannmatriser.

Djupet först-sökning

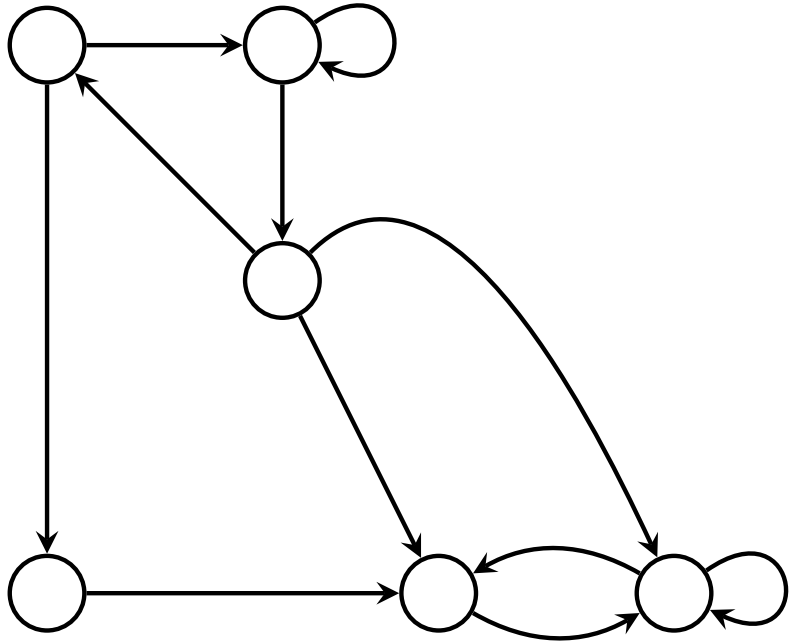
Graf:



En möjlig uppspännande skog:

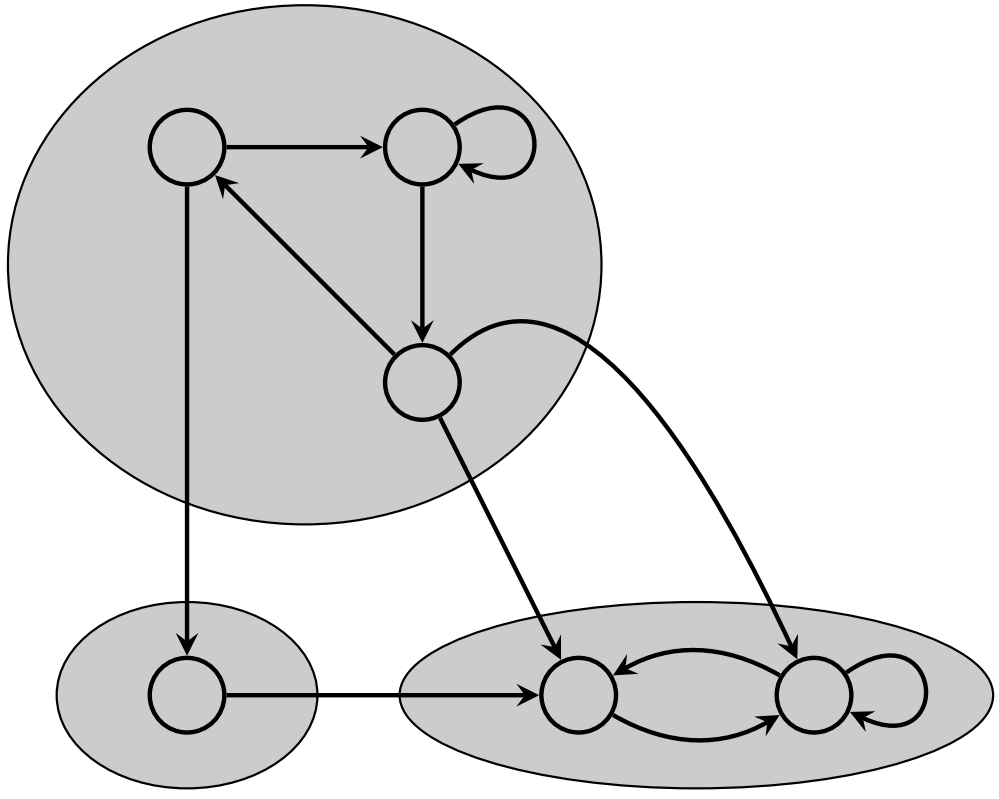


Starkt sammanhängande komponenter



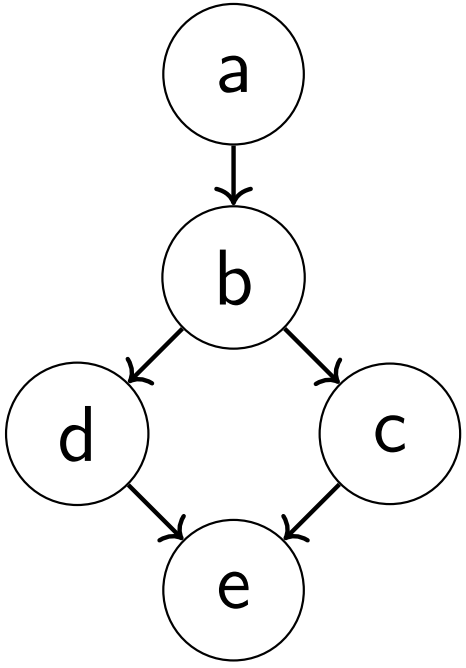
En algoritm: Två djupet först-sökningar.

Starkt sammanhängande komponenter



En algoritm: Två djupet först-sökningar.

Topologisk sortering

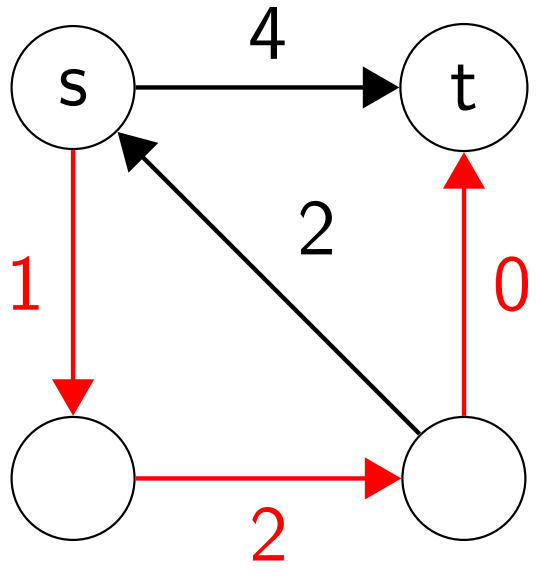


$\mapsto$ a, b, d, c, e eller a, b, c, d, e.

Några algoritmer:

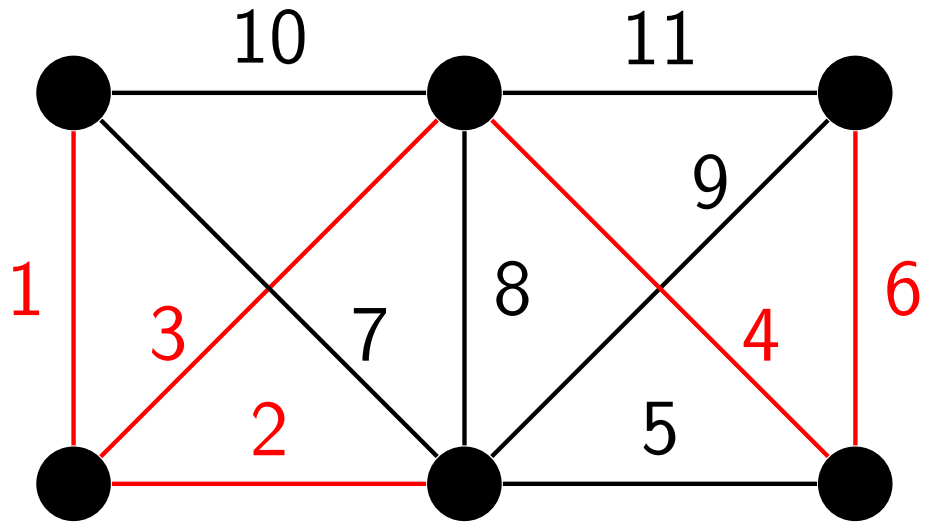
- Djupet först-sökning, gå igenom skogen i preordning, från höger till vänster.
- Kö med noder vars virtuella ingrad är 0.

Kortaste vägen



- För grafer utan vikter: Bredden först-sökning.
- Med ickenegativa vikter: Dijkstras algoritm.

Minsta uppspännande träd



Prims algoritm: Liknar Dijkstras.

Till sist

Svara gärna på kursenkäten.

- Ny examinator nästa år.

Lycka till!