Föreläsning 13
Datastrukturer (DAT037)

Fredrik Lindblad[1]

11 december 2017

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

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).
- Tänk på att dessa representerar mängder av funktioner, även om man skriver $f(n) = O(g(n))$.
- Regler.

Ordonotation

Exempel på regler:

- Polynom i n av grad $k = O(n^k)$
- $\log(n^k) = O(\log n)$
- $f(n) + g(n) = O(\max(f(n), g(n)))$
- $\log(n!) = O(n \log n)$
- $O(1) < O(\log n) < O(n) < O(n \log n) < \ldots$

Komplexitetsanalys

Tidskomplexitet

- Utgår från att elementera operationer tar konstant tid, $O(1)$.
- Lägg ihop de asymptotiska tiderna för alla operationer som exekveras.
- if-satser: Max-tiden av de två alternativen om inte en mer sofistikerad analys kan göras.
- loopar: Hur många gånger upprepas loopen. För nästlade loopar, håll koll på hur många gånger inre loopar upprepas beroende på indexens värden yttre loopar.

Komplexitetsanalys

Tidskomplexitet

- Ibland kan man få en snävare uppskattning genom att lyfta ut inre loop i analysen, exempelvis för graf-algoritmer och antalet bågar.
- Rekursiva metoder: Hur många gånger anropas metoden för olika storlek på data. Lyft ut rekursiva anropen vid analys.

Även något om utrymmeskomplexitet för datastrukturer och extra minnesanvändning hos algoritmer.

Komplexitetsanalys

För funktioner där tiden beror på annat än indatas storlek

- Värsta- och bästafallskomplexitet.
- Medel- och amorterad komplexitet.

Medelkomplexiteten är ofta komplicerad att räkna ut, eftersom den beror på den statistiska fördelningen av indata.

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.

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. I java: gränssnitt.
- Datastruktur: Implementation. I java: klass.

Generics

- Gör implementeringar så universella som möjligt.
- Generics i java: typvariabler, extends.
- Ibland behöver typen ha vissa egenskaper (hashCode, equals, Comparable, ...)

Invarianter, assertions, testning

Invarianter för datastrukturer, pre och postconditions för metoder bidrar till dokumentation och underlättar förståelse och testning.
Assertions i java.

Några (klasser av) abstrakta datatyper

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

Tips: Återanvänd kod

- Ofta behöver man inte implementera datastrukturer själv. Utgå från standarddatatyper.
- 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).

Stackar, FIFO-köer

- Enkla implementationer med (olika sorters) listor.
- Cirkulära arrayer.

Träd

Begrepp, ex. höjd, djup. Genomlöpningar.

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:

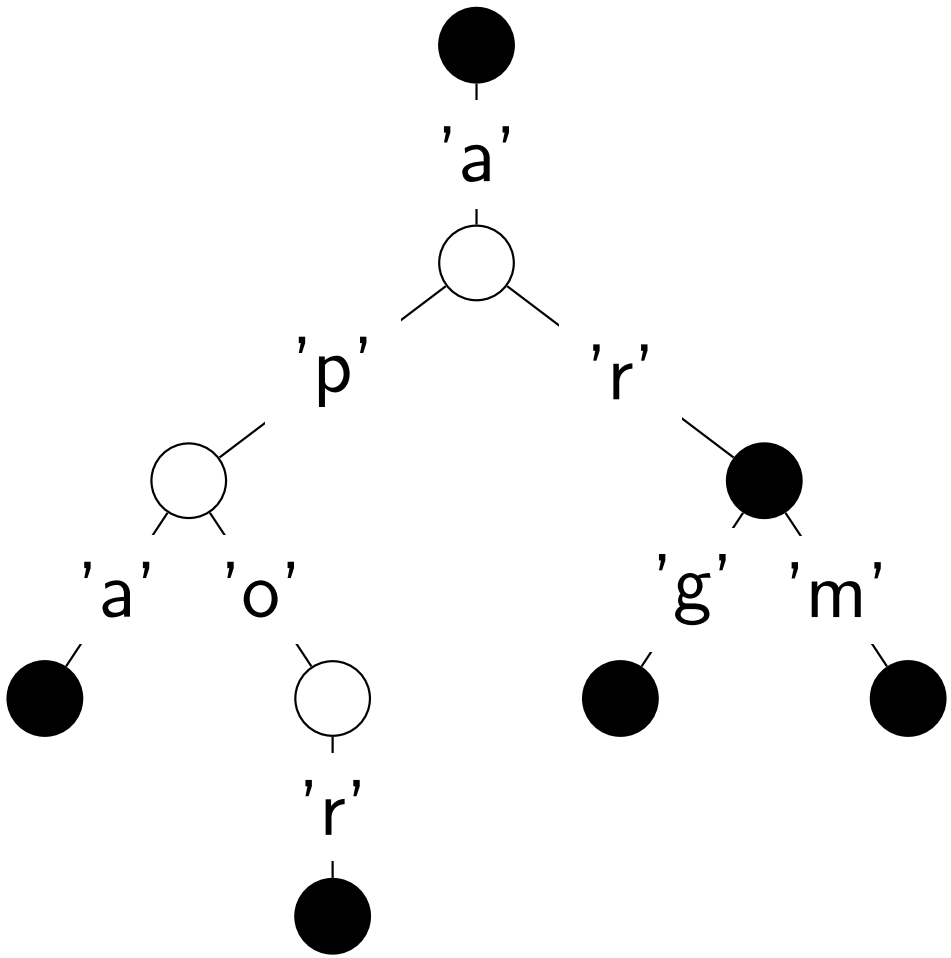
- Obalanserade binära sökträd: Långsamma vid insättning av sorterad sekvens.
- Trädrotationer
- Höjdbalanserade.
  - AVL-träd (invariant, hur insättning går till)
  - Rödsvarta träd (invariant)
  - B-träd (principen för icke-binära sökträd)
- Självbalanserade: Splay-träd (splaying, hur standardoperationerna går till)

För de olika träden enligt : tidskomplexitet ($O(\log n)$ för balanserade, $O(\log n)$ amorterat för splay-träd)

Mängder, avbildningar

Prefixträd:

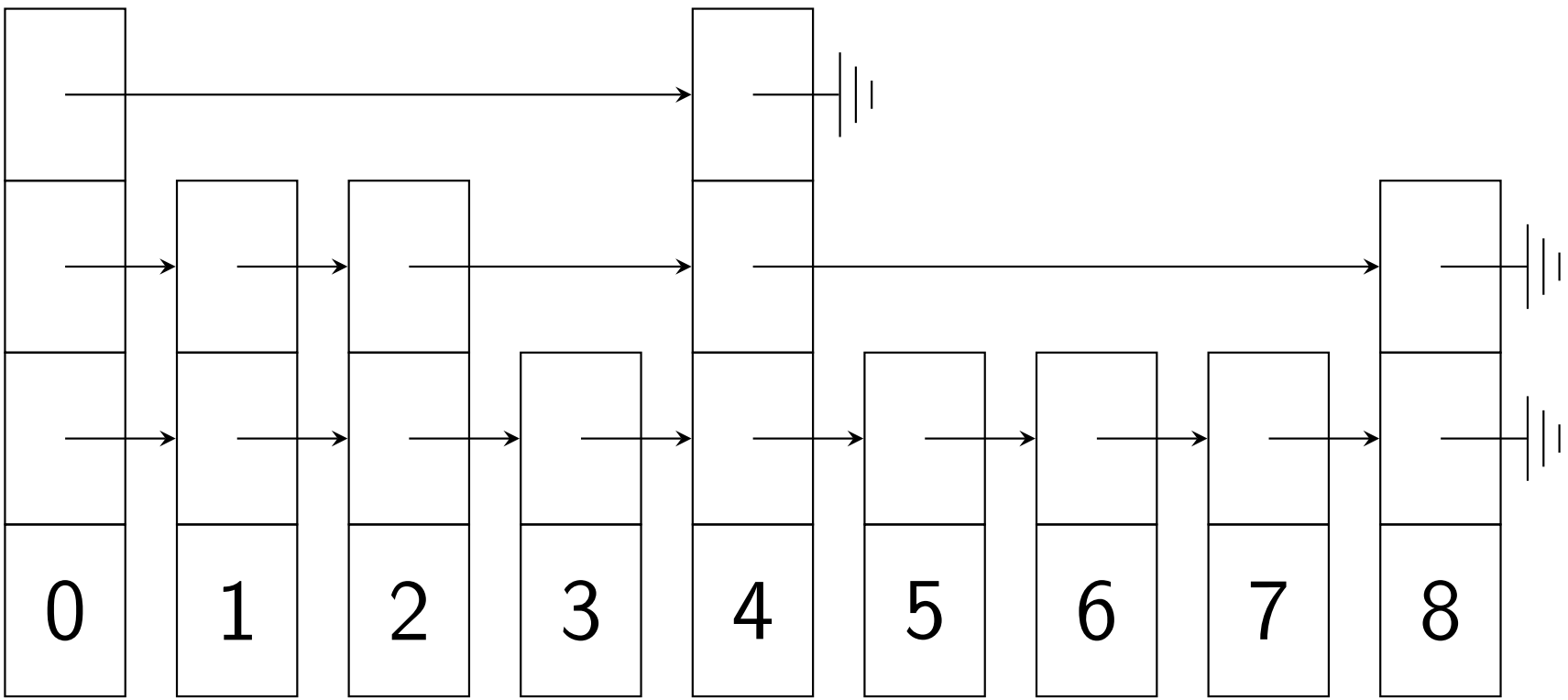
- Nycklar: Strängar av tecken.



Mängder, avbildningar

Skipplistor:

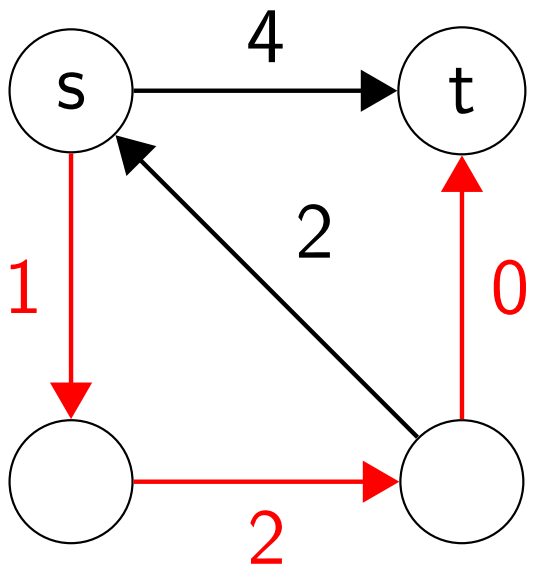
- Insättning: Randomiserad algoritm.
- Förväntad tidskomplexitet.



Grafer

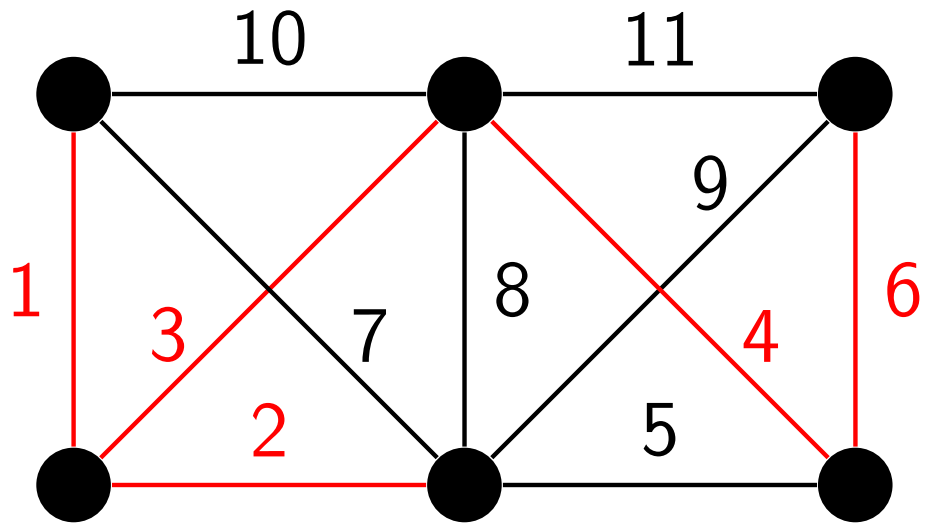
- ADT: $G = (V, E)$.
- Begrepp: riktad, viktad, cyklisk, ...
- Datastrukturer: Grannlistor, grannmatriser.

Kortaste vägen



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

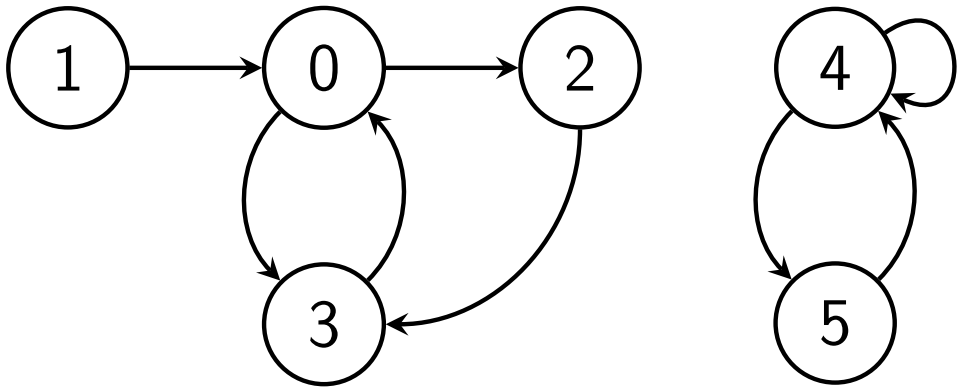
Minsta uppspännande träd



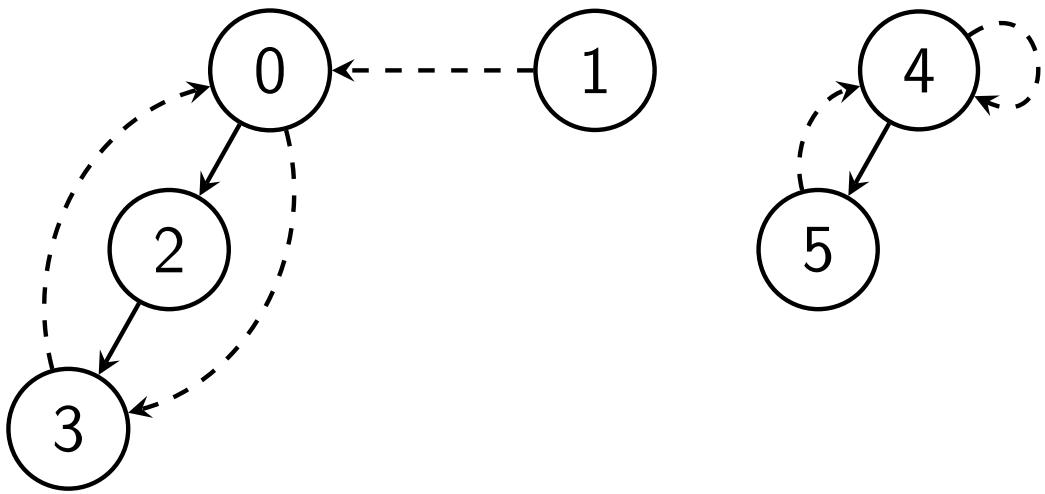
- Prims algoritm: Liknar Dijkstras.
- Kruskals algoritm: Hanterar osammanhängande grafer.

Djupet först-sökning

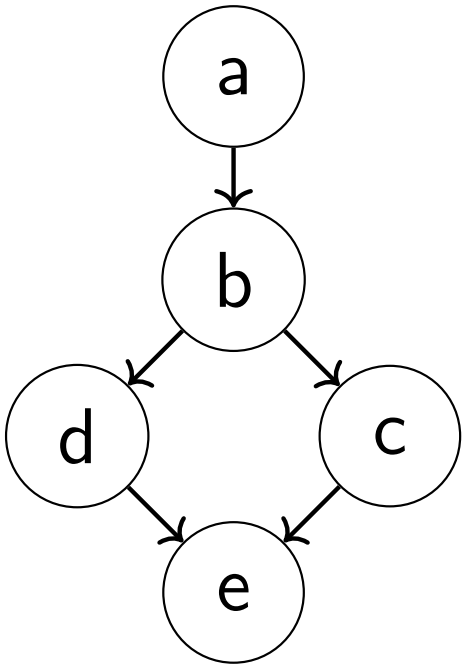
Graf:



En möjlig uppspännande skog:



Topologisk sortering

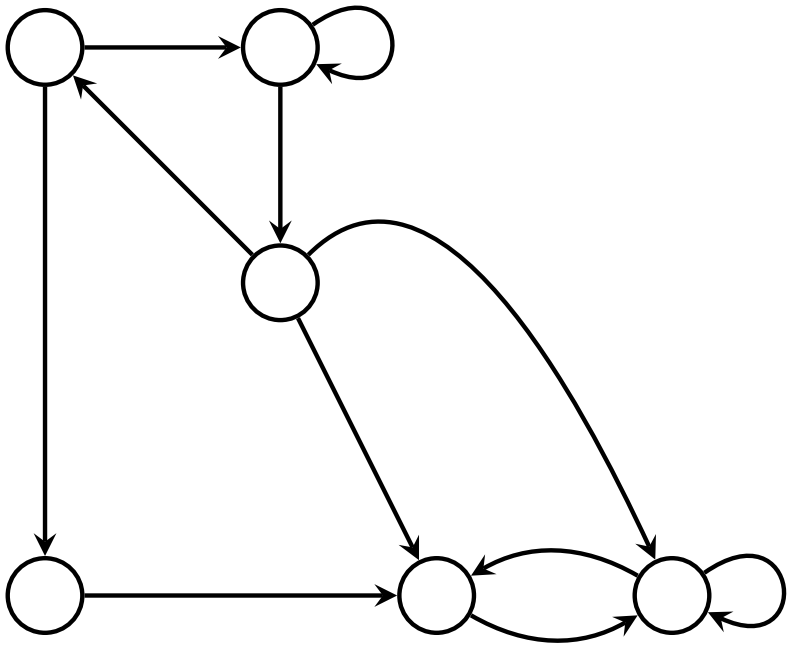


$\mapsto$ a, b, d, c, e eller a, b, c, d, e.

Några algoritmer:

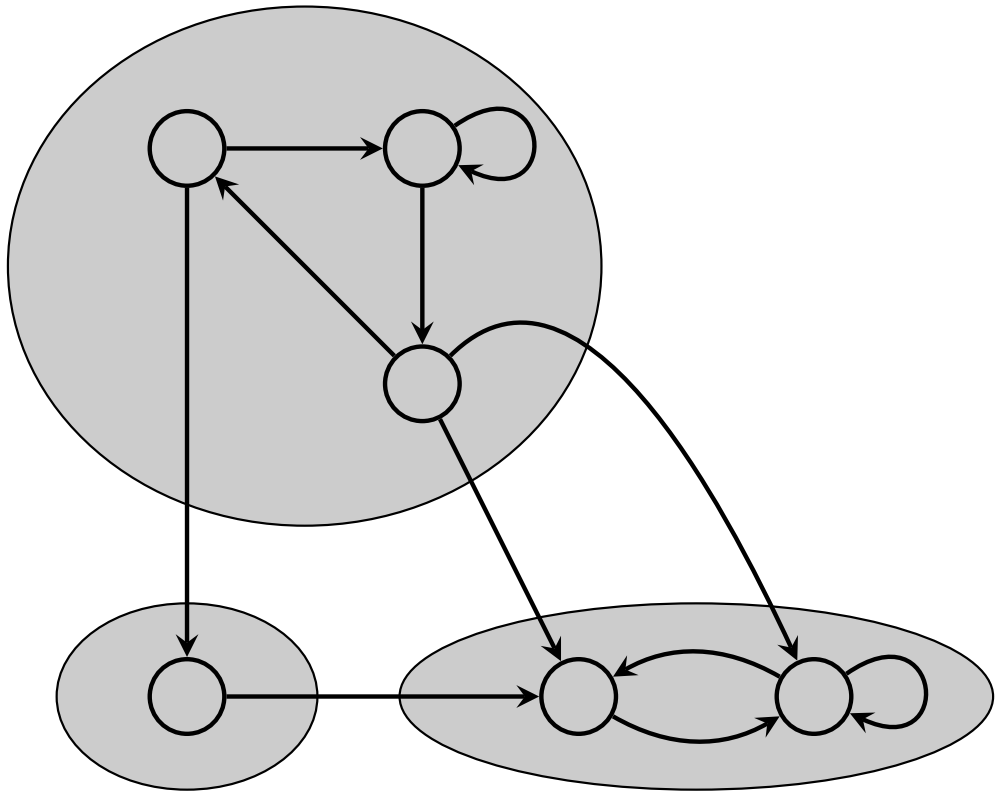
- Djupet först-sökning, gå igenom skogen i preordning, från höger till vänster.
- Kö med noder vars virtuella ingrad är 0.

Starkt sammanhängande komponenter



En algoritm: Två djupet först-sökningar.

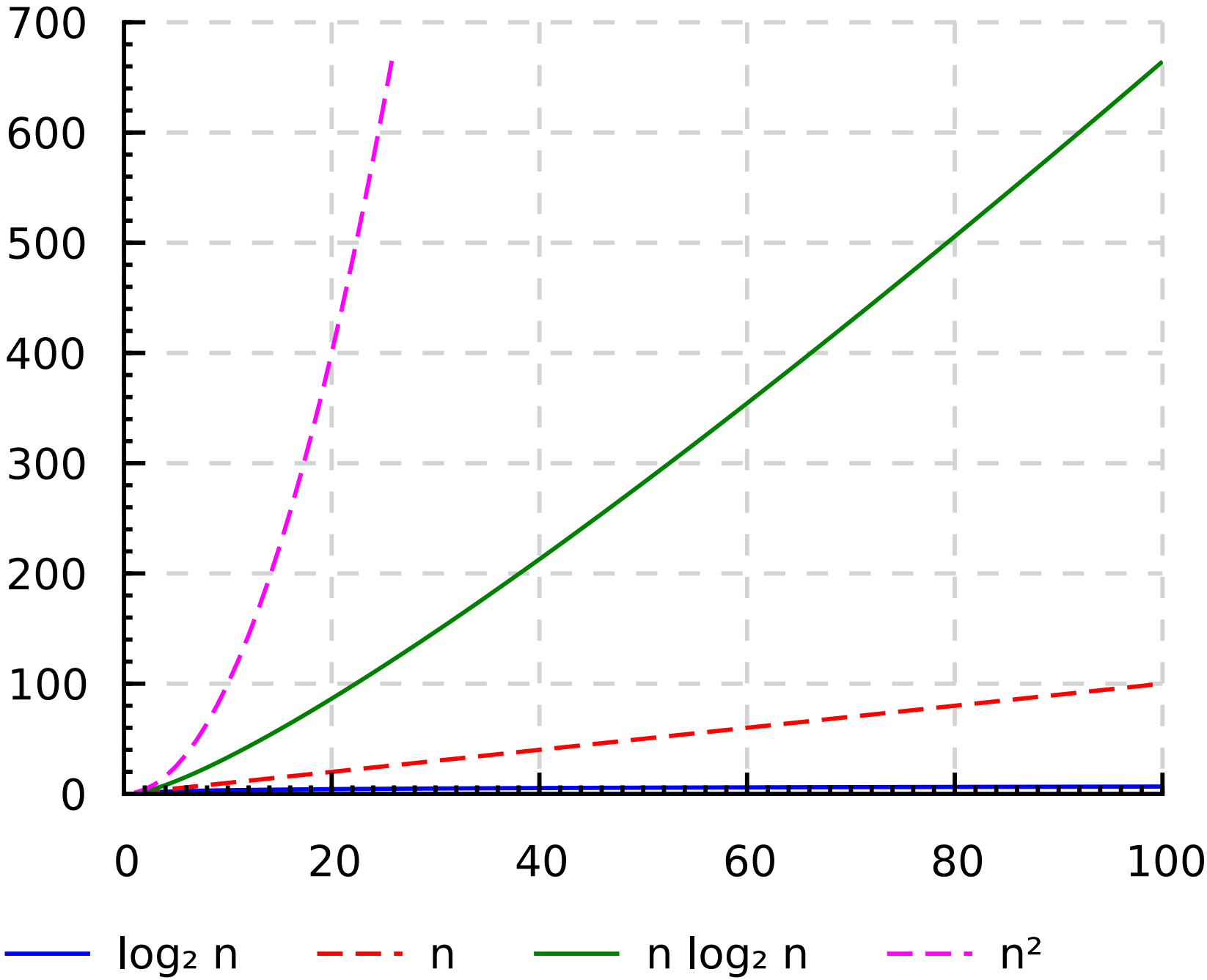
Starkt sammanhängande komponenter



En algoritm: Två djupet först-sökningar.

Sortering

- Undre gräns: $\Omega(n \log n)$.
- Urvalssortering: $O(n^2)$.
- Insättningssortering: $O(n^2)$.
- Mergesort: $O(n \log n)$.
- Quicksort: $O(n \log n)$ (medel).
- Heapsort: $O(n \log n)$.
- 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

Diverse

- Iteratorer
- Java Collections Framework
- Rekursion
- Giriga algoritmer
- Divide and conquer

Skillnader från gamla tentor

Vissa saker har utgått ur kursinnehållet, men återfinns bland uppgifterna i gamla tentor.

- Datastrukturer i funktionella språk, persistens
- Potentialmetoden