

Föreläsning 12

Datastrukturer (DAT037)

Fredrik Lindblad¹

2016-12-12

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

Förra gången

- ▶ Sorting – stabel, in-place
- ▶ Urvalssortering, insättningsortering – $O(n^2)$
- ▶ Mergesort, Quicksort, Heapsort – $O(n \log n)$

In-place sortering

- ▶ Notera att sorteringsalgoritmen inte är in-place om den implementeras med rekursion (i avsaknad av "tail-call optimisation" e d): rekursion använder minne (stack).
- ▶ Quicksort-implementeringen från förra föreläsningen är alltså inte strikt in-place.

Radixsortering

Hinksortering, bucket sort

- ▶ Sortering av lista med max N distinkta element.
- ▶ Allokera N hinkar.
- ▶ Stoppa in varje element i rätt hink.
- ▶ Gå igenom hinkarna och bygg ny lista.
- ▶ Stabil om implementerad på rätt sätt.

Hinksortering, bucket sort

Nedan är bucket en funktion som tar element till heltal $\in \{ 0, \dots, N - 1 \}$.

```
bucket-sort(N, bucket, xs):  
    buckets = array of size N containing empty lists  
  
    for x in xs do  
        buckets[bucket(x)].add-last(x)  
  
    ys = new empty list  
  
    for b in 0, ..., N - 1 do  
        for y in buckets[b] do  
            ys.add-last(y)  
  
    return ys
```

Vad är tidskomplexiteten för hinksortering?

Anta att bucket tar konstant tid.

- ▶ $\Theta(Nn)$.
- ▶ $\Theta(n \log n)$.
- ▶ $\Theta((N + n) \log(N + n))$.
- ▶ $\Theta(N + n)$.
- ▶ $\Theta(Nn \log Nn)$.

Radixsortering

- ▶ Hinksortering är inte praktiskt om n är "mycket" mindre än N .
- ▶ Alternativ: Radixsortering, lexikografisk sortering av nycklar med d delnycklar.
- ▶ Exempel: Tal bestående av d siffror.

Radixsortering

Least significant digit (LSD) radix sort
för tal med d siffor:

- ▶ Sortera talen stabilt med avseende på den minst signifikanta siffran.
- ▶ Sortera talen stabilt med avseende på den näst minst signifikanta siffran.
- ▶ Och så vidare...

Radixsortering

- ▶ Stabil.
- ▶ Om hinksortering med $O(1)$ bucket används:
 $\Theta(d(N + n))$,
där N är talbasen (decimalt: 10, binärt: 2).
- ▶ Notera att man kan välja N själv.
Exempel: 32-bitars tal kan delas upp i
4 delar med 8 bitar ($d = 4, N = 256$),
eller 2 delar med 16 bitar ($d = 2, N = 65536$).

Sortering:
Undre gräns

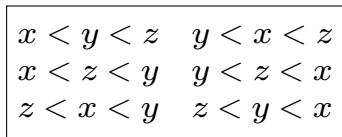
Sortering: Undre gräns

Sortering baserad enbart på binära jämförelser kräver, i värsta fallet, $\Omega(n \log n)$ jämförelser.

Beslutsträd

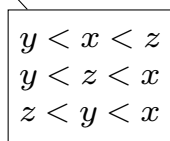
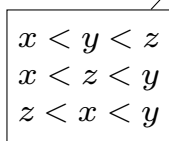
- ▶ Träd som dokumenterar vilka jämförelser en sorteringsalgoritm utför då den sorterar listor av en viss längd n .
- ▶ Låt oss anta att alla element är olika.
- ▶ Interna noder: Jämförelser. ("Är $a[i] < a[j]$?")
- ▶ Ett barn per resultat ($a[i] < a[j]$, $a[i] > a[j]$).
- ▶ Löv: Sorterade listor ($a[3] < a[1] < a[5] < \dots$).
- ▶ Jämförelsebaserade sorteringsalgoritmer:
Lövs få endast förekomma när jämförelserna ovanför i trädet ger tillräcklig information för att kunna sortera listan.

Beslutsträd



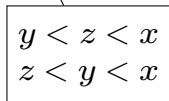
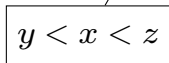
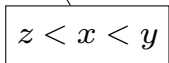
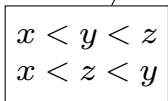
$x < y$

$y < x$



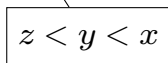
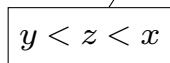
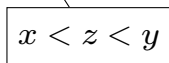
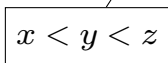
$x < z$ $z < x$

$x < z$ $z < x$



$y < z$ $z < y$

$y < z$ $z < y$



Ge en skarp undre gräns för antalet löv i ett beslutsträd.

- ▶ $n \lceil \log_2 n \rceil$.
- ▶ $n(n + 1)/2$.
- ▶ $n!$.
- ▶ 2^n .

Undre gräns

- ▶ Djupet av ett visst löv = antalet jämförelser för att sortera viss lista.
- ▶ Trädets höjd = värsta fallet.
- ▶ Ska visa att höjden är $\Omega(n \log n)$.

Undre gräns

- ▶ Ett binärt träd med höjd h har $\leq 2^h$ löv.
- ▶ Ett icke tomt binärt träd med ℓ löv har höjd $\geq \log_2 \ell$.
- ▶ Ett beslutsträd för sortering av listor med distinkta element har $\geq n!$ löv (minst ett per möjlig permutation av listan).
- ▶ Beslutsträdet har alltså höjd $\geq \log_2(n!) = \Omega(n \log n)$.

Sortering i praktiken

- ▶ Adaptiv sortering – anpassar sig till indata, snabbare vid nästan sorterad lista (ex. insättningsortering)
- ▶ Hybridsortering – välj bland olika sorteringsalgoritmer beroende på indatas storlek, t.ex. sortera små dellistor (<15-20 element) med enkel kvadratisk algoritm.
- ▶ Olika algoritmer beroende på elementtyp.
- ▶ Java 8: Sortering av primitiva typer – variant av Quicksort med två pivotelement. (Jämförelse inte så dyrt men kopiering kan vara det, stabil sortering ej intressant (jmf. Mergesort))
- ▶ Java 8: Sortering av objekt – variant av Mergesort, stabil, adaptiv, få jämförelser, mer omflyttning