Föreläsning 14

Sökning och Sökträd

Innehåll

- Sökträd
- Strängsökning

Binära sökträd

- Om trädet är komplett så vet vi att både medel- och värstafallskomplexiteten är O(log n).
- Men… Det tar tid och kraft att se till att trädet är komplett. Ibland kan man tvingas bygga om hela trädet.
- Det räcker att se till att balansen är god…

Balanserade binära sökträd

- Finns flera olika metoder t.ex.:
  - AVL träd
  - Röd svarta träd (Red-black tree)
- B-träd
  - Annan trädstruktur med balanseringsfunktionalitet.

Flervägs sökträd

- Ett *m*-vägs sökträd (m-way search tree, m-ary search tree) är en generalisering av ett binärt sökträd.
- Trädet är ett ordnat träd där varje nod har högst m delträd.
- Etiketterna är sekvenser av upp till *m-1* värden i stigande sorteringsordning som fungerar som delningspunkter vid sökning.
- Oftast är etiketterna nycklar och värdet till en viss nyckel finns i lövet.

Flervägs sökträd

- Till en nod med $k+1$ delträd, $t_0, t_1, \ldots, t_k$ hör en sekvens med värden $v_1, v_2, \ldots, v_k$. Sorteringsvillkoret för trädet är att:
  - alla värden i t_0 går före v_1 (i sorteringsordningen)
  - alla värden i t_j ligger mellan v_j och v_{j+1} för $1<j<k$
  - alla värden i t_k går efter v_k
- Operationerna blir liknande de för binärt sökträd.
- Plattare träd. Höjden = $\log_m n$
- Mer jobb i noderna

B-träd

- Ett B-träd av ordning m är en typ av balanserat *m*-vägs sökträd som uppfyller följande:
  - Roten är antingen ett löv eller har minst två barn
  - Alla noder utom roten och löven har mellan *m*/2 och *m* barn
  - Alla löv är på samma djup

B-träd

- Insättning av nya element görs alltid på den djupaste nivån, i rätt löv för att bevara sorteringsordningen.
  - En insättning kan leda till att noden blir för stor (dvs > m). Då måste noden delas upp.
- Borttagning kan leda till att man måste justera värdena och slå ihop noder eller omfördela värden mellan dem.
- B-träd av ordning 3 kallas också 2-3 träd

B-träd analys:

- För ett B-träd av ordning m, med höjden h och n nycklar insatta gäller h = O(log n).
- För att välja rätt underträd vid sökning krävs att man stänger in sökt nyckel mellan två nycklar i noden. Om nycklarna är sorterade och lagrade i en vektor kan man använda binärsökning. Sökning i en nod O(log m)
- Nycklarna i vektorn måste skiftas runt vid splittring av en nod. Kostnad O(m)
- Eftersom m är en konstant blir det O(1) arbete i varje nod vid sökning och insättning. Antalet noder som berörs är uppåt begränsad av höjden.
- Värstafallskostnad för sökning och insättning O(log n)

Exempel på B-träd: 2-4 träd

- Varje nod har 1, 2 eller 3 nycklar och varje icke-löv har 2-4 barn.
- Regel för insättning:
  - Man letar sig fram till rätt löv på liknande sätt som i ett vanligt sökträd.
  - Den nya nyckeln sätts in där. Om det blir för många nycklar i det lövet splittras det.

Exempel på B-träd: 2-4 träd

- Regel för borttagning:
  - Man letar sig fram till rätt löv i noden, ta bort den som ska bort.
  - Halvsvåra fallet: Syskonen har ”extra på liknande sätt som i ett vanligt sökträd.
  - Enkla fallet: Det finns flera nycklar ” element som vi kan sno.
  - Svåra fallet: Vi får tomt och syskonet har bara ett element. Då måste vi göra en ”fuse”-operation.

Strängsökning

- Specialfall av sökning.
  - Man söker inte ett enstaka element utan en sekvens av element.
  - Elementet ofta tecken, men kan även vara andra typer av data.
- Formellt:
  - Vi har ett mönster *P* med längd *m* och vi söker i en sekvens *S* av längd *n* där $m<<n$.

Strängsökning…

- Ett antal algoritmer
  - Naiv
  - Knuth Morris Pratt
  - Booyer Moore
  - Rabin-Karp

Första försök till algoritm:

- Börja jämföra mönstret med sekvensen med start i position ett.
  - Jämför mönstret från vänster till höger tills man misslyckas.
  - Flytta då fram en position i sekvensen och försök igen.
- Värsta fallet: Varje element i S avläses m gånger, dvs O(n*m)
  - I praktiken bättre

Exempel

```
tetththeheehthtehtheththeheehtht
the
tetththeheehthtehtheththeheehtht
 the
tetththeheehthtehtheththeheehtht
  the
tetththeheehthtehtheththeheehtht
   the
tetththeheehthtehtheththeheehtht
    the
tetththeheehthtehtheththeheehtht
     the
```


Knuth Morris Pratt

- Utnyttjar en felfunktion f
  - som berättar hur mycket av mycket av den senaste jämförelsen man kan återanvända om man felar
  - är definierat som det längsta prefixet i P[0,...,j] som också är suffix av P[1,...,j] där P är vårt mönster.
  - visar hur mycket av början av strängen matchar upp till omedelbart före felet

Felfunktion exempel

- Om jämförelsen felar på position 4, så vet vi att a,b i position 2,3 är identiska med position 0,1

j	0	1	2	3	4	5
P[j]	A	B	A	B	A	C
f(j)	0	0	1	2	3	0

KMP-algoritmen

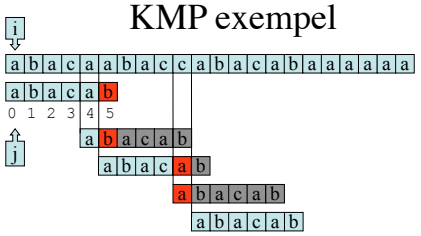
- Input: String *T* (text) with *n* characters and *P* (pattern) with *m* characters.
- Output: Starting index of the first substring of *T* matching P, or an indication that *P* is not a substring of T.

```
f <- KMPfailureFunction(P)
i <- 0
j <- 0
while i < n do
   if P[j] = S[i] then
       if j = m-1 then return i-m-1 // En matchning
       i <- i+1
       j <- j+1
   else if j > 0 then
       // ingen match, vi har gått j index direkt efter
       // matchande prefix i P
       j <- f(j-1)
   else
       i <- i+1
   return ingen matchning av delsträngen P i S
```

KMPfailureFunction(P)

```
i <- 1
j <- 0
while i < m-1 do
  if P[j] = P[i] then
     f(i)<- j+1
     i <- i+1
     j <- j+1
  else if j > 0 ingen match then
     j <- f(j-1) //j index efter pref som match. prefix
  else //ingen matchning
     f(i) <- 0
     i <- i+1
```


KMP exempel



j	0	1	2	3	4	5
P[j]	a	b	a	c	a	b
f(j)	0	0	1	0	1	2

KMP-Algoritmen

- Låt k = i - j
- För varje varv i while-loopen händer ett av följande:
  - S[i] = P[j], öka i och j med 1, k oförändrad.
  - S[i] < > P[j] och j > 0, i är oförändrad men k ökar med minst 1 eftersom den ändras från i - j till i - f(j-1)
  - S[i] < > P[j] och j = 0, öka i med 1 och minska k med 1 (ty j oförändrad).
- Alltså för varje varv i loppen ökar antingen i eller k med minst 1. Max antal varv blir 2n
- Detta antar att f redan är beräknad (som är O(m)).
- Total komplexitet: O(n + m)

Boyer-Moores algoritm

- Två idéer:
  - Gör matchningen baklänges, med start i mönstrets sista element.
  - Utnyttja kunskap om mönstrets uppbyggnad och informationen om värdet på den första felmatchande elementet i S för att flytta fram mönstret så långt som möjligt varje gång.
    - Om det finns upprepningar av element i mönstret så får man bara flytta fram till den högraste förekomsten.
    - Förskjutningstabell talar om hur långt man får flytta.

Rabin-Karp

- Beräkna ett hashvärde för mönstret och för varje delsträng av texten som man ska jämföra med
- Om hashvärdena är skilda, beräkna hashvärdet för det nästa M tecknen i texten
- Om hashvärdena är lika, utför en brute-force jämförelse mellan P och delsträngen
- Med andra ord:
  - Endast en jämförelse per deltext
  - Brute-force endast när hashvärdena matchar.

Rabin-Karp exempel

```
Hashvärdet för AAAAA = 37
Hashvärdet för AAAAH = 100
AAAAAAAAAAAAAAA
AAAAH
100 <> 37
AAAAAAAAAAAAAAA
 AAAAH
100 <> 37
AAAAAAAAAAAAAAAA
      AAAAH
100 = 100
```

Rabin-Karp hashfunktionen

- Vilken? Den får inte kosta för mycket....
- Betrakta de nästa M tecknen i söksträngen som ett M-siffrigt tal i basen b, där b är antalet bokstäver i alfabetet
- Textsekvensen t[i..i+M-1] avbildas på talet
  $x(i) = t[i]\mathbf{b}^{M-1} + t[i+1]\mathbf{b}^{M-2} + ... + t[i+M-1]$

Rabin-Karp hashfunktionen

- Billigt att beräkna x(i+1) från x(i)
  $x(i+1) = t[i+1]\mathbf{b}^{M-1} + t[i+2]\mathbf{b}^{M-2} + ... + t[i+M]$
- $x(i+1) = x(i)\mathbf{b}$ skifta ett vänster,
  $- t[i]\mathbf{b}^{M}$ ta bort den vänstraste termen
  $+ t[i+M]$ lägg till den nya högertermen
- Behöver inte räkna om hela talet utan gör bara en justering för det nya tecknet

Hash-värdet fortsättning

- Om M är stort blir blir (b^M) enormt därför så hashar man med mod ett stort primtal q
- $h(i) = ((t[i]\mathbf{b}^{M-1} \textbf{ mod } q) + (t[i+1]\mathbf{b}^{M-2} \textbf{ mod } q) + ... + (t[i+M-1] \textbf{ mod } q)) \textbf{ mod } q$
- $h(i+1) = (h(i)\mathbf{b} \textbf{ mod } q - (t[i]\mathbf{b}^{M} \textbf{mod } q) + (t[i+M] \textbf{ mod } q) \textbf{ mod } q$

Algoritm

```
hash_M <- Beräkna hashvärdet för M
hash_S <- Beräkna hashvärdet för
den första delsträngen
do
   if (hash_M = hash_S) then
      Bruteforce jämförelse av M och S
   hash_S + 1 tecken beräknas
while end of text or match
```


Komplexitet

- Om det är tillräckligt stort primtal q för hashfunktionen så kommer hashvärdet från två mönster vara distinkta
- I detta fall så tar sökningen O(N) där N är antalet tecken i strängen
- Men det finns alltid fall som ger i närheten av värsta fallet O(N*M) om primtalet är för litet