Övning 7 - Tillämpad datalogi 2013

KTH, CSC/CB, Mikael Lindahl

Sammanfattning

Idag räknade vi igenom en tentamen från 12 januari 2010. OBS i många uppgifter kan man svara på olika sätt. Det viktiga är att visa att man förstått.

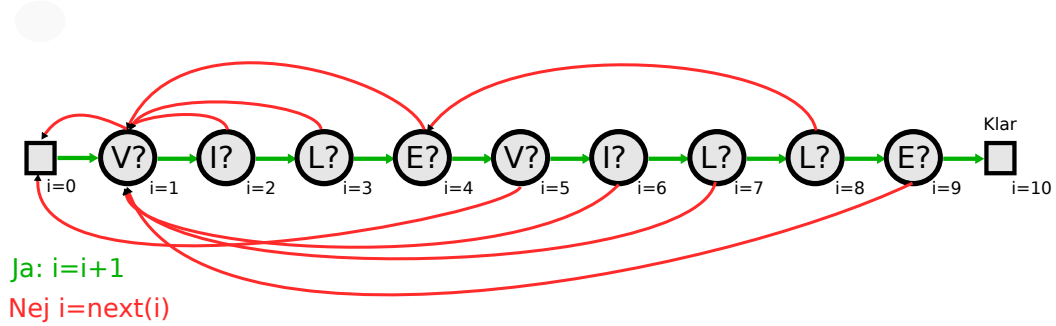
DD1320, TENTAMEN I TILLÄMPAD DATALOGI Tisdagen den 12 januari 2010 kl 14–18

Maxpoäng = 100. 50 poäng ger E, men den som fått 47-49 poäng kan få komplettera. Gränserna för högre betyg är 60, 70, 80, och 90 poäng. Skriv upp antal bonuspoäng från labbar respektive hemtal på tentaomslaget. Tentorna beräknas vara rättade om två veckor och kan sedan hämtas på studentexpeditionen.

Hjälpmedel: En algoritmbok och ditt handskrivna formelblad. Lämna in formelbladet tillsammans med tentan.

1. Spelautomat

(10p) Konstruera och rita upp en KMP-automat som söker efter Farmvilles läskigare efterträdare VILEVILLE. Ange även next-vektorn!



i	1	2	3	4	5	6	7	8	9
$next(i)$	0	1	1	1	0	1	1	4	1

2. Spelsökning

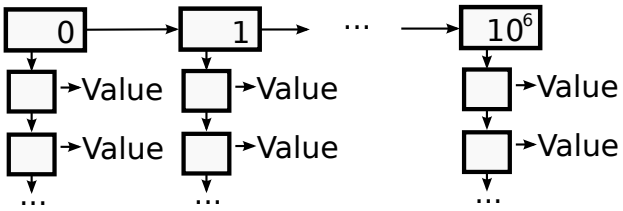
Speljätten Cawr Games har 1 073 741 824 olika spel i sin databas. Hittills har man använt binärträd för att få snabb sökning, men man vill byta till hashning, med en hashtabell som rymmer en miljon objekt.

(10p) a. Vilken krockhanteringsmetod skulle du använda? Motivera ditt val, och rita en bild som visar hur det går till.

(10p) b. Hur snabb skulle hashsökningen bli jämfört med binärsökningen? Anta att antalet jämförelser bestämmer tiden

1) Krockhanteringsmetod

Vi har runt en miljard element som vi vill lagra i en hashtabell med en miljon platser. För linjär eller kvadratisk probning som krockhantering behöver vi ha en hashtabell som är minst lika stor som antalet element som ska in. Det har vi inte och och därför bör vi istället välja krockhantering med listor. Vi kommer då dynamiskt bygga ut tabellen med listor för varje position när krockar uppstår. Så här kan vi illustrera en hashtabell med krocklistor:

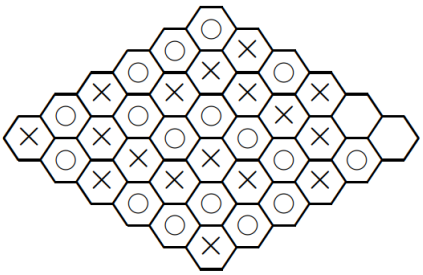


2) Prestanda

Om vi har en bra hashfuntkion som sprider värdena jämt i tabellen skulle vi i snitt få omkring 1000 krockar per position i hashvektorn (en miljard delat på en miljon). Tiden för en hash sökning skulle i genomsnitt ta $1000/2 = 500$ jämförelser. Med ett binärträd skulle vi istället få $log_2(1073741824) = 30$ jämförelser. Oturligt nog skulle hashtabellen bli 500/30=16 gånger långsammare än binärträdet.

3. Rekursivt spel

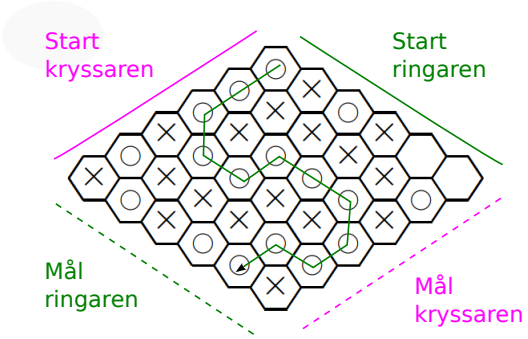
(10p) I spelet Hex ska kryssaren bygga en väg mellan övre vänsterkant och nedre högerkant och ringaren bygga en väg mellan övre högerkant och nedre vänsterkant. **Båda kan inte lyckas (inser du varför?) men i exemplet har ringaren just vunnit (ser du hur?)** Formulera en rekursiv tanke som kontrollerar om det finns en vinnande väg.



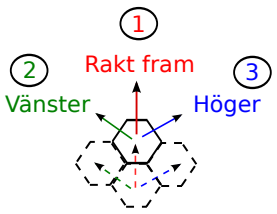
(4p) Föreslå en datastruktur för att representera spelplanen, kryssen och ringarna.

1) Rekursiv tanke

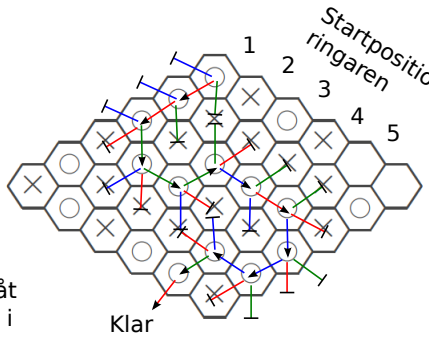
Bilden nedan illustrerar var respektive spelare startar och slutar samt en vinnande väg för ringaren.

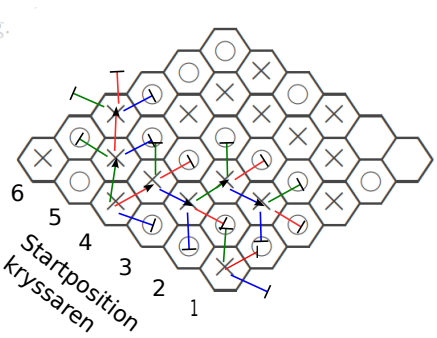


För en position i spelplanen (se figure nedan) kan vi se att det finns tre rikningar som vi bör kolla vidare. Antingen till vänster, höger eller rakt fram. Positionerna snett bakåtvvänster och snett bakåt höger har redan besökts i steget innan och vi behöver därför inte gå dit. I bilden nedan kollar vi om den den rekursiva tanken "kolla rakt fram, vänster, höger" håller för kryssaren och ringaren. Vi ser att för ringaren hittar vi en lösning medan för kryssaren gör vi det inte.



Behöver inte kolla snett bakåt vänster/höger för de gjordes i steget innan





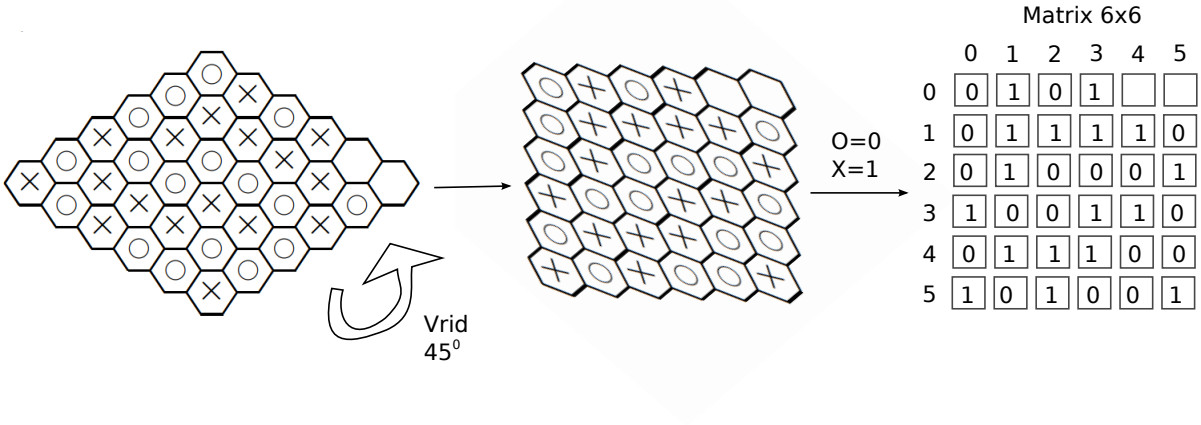
Vår rekursiva tanke blir:

1. Kolla nod rakt fram
2. Kolla nod vänster
3. Kolla nod höger

Basfall: Om nod inte är spelarens tecken stanna, returera False, Om nod är målnod, returnera True
Kolla för alla startnoder.

2) Datastruktur

Om vi vrider spelplanen 45 grader ser vi att vi kan representera den som en matris. Ring får ta värdet 0 och kryss 1.



4. Teori

(20p) Nedan finns fem frågor om algoritmer och datastrukturer. Varje fråga kan ge upp till fyra poäng. Motivering krävs!

a. Under vilka förhållanden kan sortering gå i tid O(n)?

b. Kommer en postorder-utskrift av noderna i ett binärträd att ge noderna i omvänd ordning mot en preorderutskrift?

c. Visa hur heapsort fungerar med följande data: 7 8 3 6 2

d. Hur många nivåer kan Huffmanträdet ha som mest vid kodning av alfabetets 29 bokstäver?

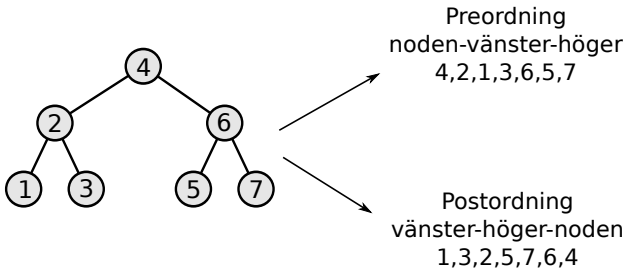
e. En text är krypterad med transpositionschiffer eller Caesarchiffer. Hur ser du vilket det är?

a) Sortering

Räknesortering (distribution count) kan ge O(n). Räknesortering går att använda om man vet att det finns få nyckelvärden i förhållande till antal data som ska sorteras. Med räknesortering behöver man endast gå igenom filen 2 gånger för att sortera vilket ger O(n) i komplexitet.

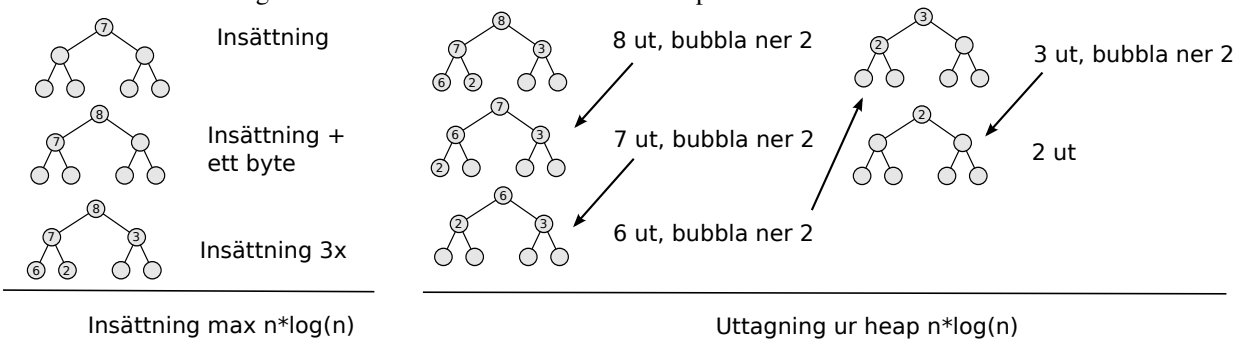
b) Pre- och postordning

Nej, så här blir det för ett träd med talen 4 6 5 7 2 1 3:



c) Heapsort

Talen 7 8 3 6 2 sorteras genom att först sätta in dem i en max-heap och sedan ta ut dem ur den.



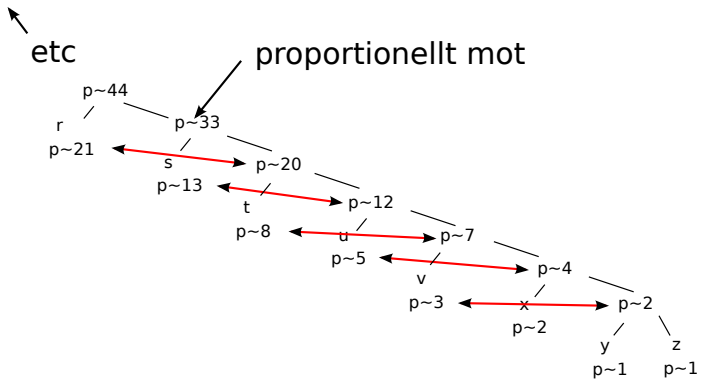
d) Huffmanträd

28 om vi har riktigt otur. Detta sker om sannolikheten för varje delträd som byggs summerar till mindre eller samma sannolikhet för det tecken med minst sannolikhet.

Detta träd kan vi till exempel få om sannolikheten för bokstäverna är distribuerade på följande sätt:

Givet Fibonaccis tal följd.
fib(0)=0
fib(1)=1
fib(n)=fib(n-1)+fib(n-2)

Får vi sannolikheterna:
$p_i = fib(i) / \sum_{i=1}^{29} fib(i)$
$p_1 \Rightarrow p_a, \ldots, p_{26} \Rightarrow p_z, \ldots$
Så här blir Huffmanträdet:



e) Transposition eller Ceasar?

Vi testar först om det är krypterat med Ceasar Det tar max 28 iterationer. Om det inte stämmer kan vi sluta oss till att meddelandet är kodat med transpositionschiffer

5. Spelkomprimering

(10p) Vi vill komprimera ett konsolspel (t ex Super Mario Bros.). Vilka komprimerings- algoritmer kan komma ifråga? Motivera ditt svar!

Vi har lärt oss om tre komprimeringsalgoritmer:
1. Huffmankodning
2. Lempel-Ziv (LZW)
3. Run-Length-Encoding (RLE)

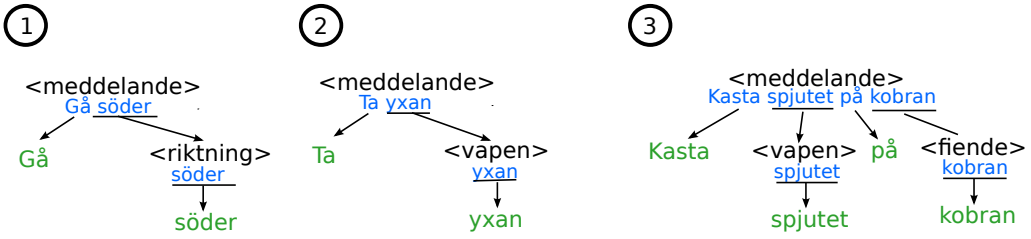
Dataspelet Super Mario Bros är lagrat i binärkod. Därför passar inte huffman eftersom den algoritmen har som mål att skapa binärkod.
LZW och RLE kan vi använda. LZW fungerar bra om vi har många kombinationer av ettor och nollor som upprepar sig och med RLE fungera bra om vi har långa repetitioner av ettor eller nollor.

6. Spelsyntax

(10p) Du vill skriva ett textbaserat äventyrsspel till mobilen. Den som spelar ska kunna skriva kommandon av typen:
GÅ ÖSTER
TA SVÄRDET
KASTA SVÄRDET PÅ PUMAN
Skriv en BNF-grammatik för sådana kommandon. Visa också med de tre exemplen ovan hur din syntax fungerar.
BNF-grammatik:

<meddelande> ::= Gå <riktning>|
Ta <vapen> |
Kasta <vapen> på <fiende>
<riktning> ::= öster| väster | nord | söder
<vapen> ::= svärdet | yxan | spjutet | ...
<fiende> ::= puman | lejonet | kobran | ...

Med exempel:



7. Patiens

(16p) I patiensen "Rutan" använder man bara ess, kungar, damer och knektar, alltså sexton kort. Det gäller att lägga ut korten i en 4x4-matris så att två kort av samma färg eller valör aldrig finns i samma rad, kolumn eller någon av de båda diagonalerna.

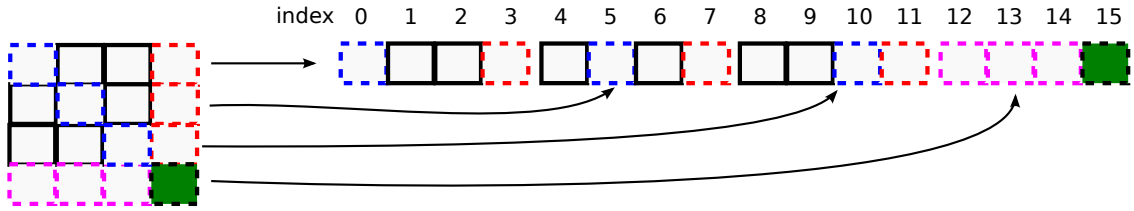
Man vill att ett program ska skriva ut alla lösningar till patiensen på nedanstående sätt.

hj E	sp D	kl Kn	ru K
kl K	ru Kn	hj D	sp E
ru D	kl E	sp K	hj Kn
sp Kn	hj K	ru E	kl D

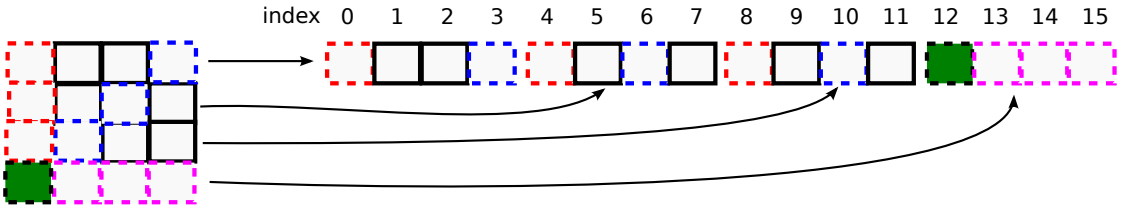
Du behöver inte skriva programkod, men du ska rita och förklara algoritmen utförligt samt beskriva datastrukturer, metoder och klasser.

Hm, vi ska alltså komma på en algoritm som hittar alla lösningar. Låt oss börja med att läga in ett kort i matrisen. Sen får vi försöka komma på regler för hur vi kan lägga in fler kort i matrisen. Vi kan tolka vara ny matris som barn till den ursprungsmatrisen. Då går det att tolka det hela som ett problemträd där lösningarna kommer finnas i löven. För att komma på reglerna för hur nya barn i problemträdet skapas kan vi kan vi först strukturera om matrisen till en vektor genom att lägga raderna efter varandra. För att sen få ett hum om hur vår algoritm bör se ut kan vi studera ett par exempel. Vi ritar därför upp två scenarion, ett där vi bara har sista positionen kvar att fylla in och ett där vi ska fylla fjärde sista. Utifrån dessa två scenarion bör vi kunna komma upp med reglerna för hur vi ska kolla bakåt i vektorn för att lägga in ett kort på position i.

Scenario 1

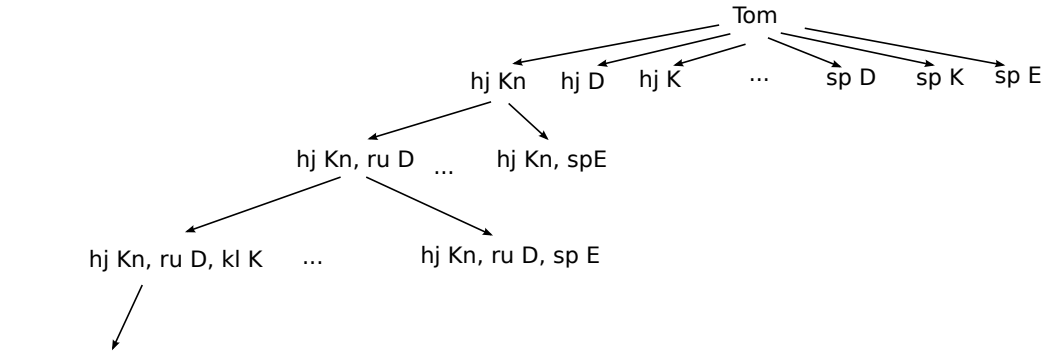


Scenario 2



Lodrätt: kolla bakåt $i=i-4$ tills $i<0$
Vågrätt: om $i \bmod 4=0 > 0$ kolla bakåt ett steg i taget tills $i \bmod 4=0$
Diagonalt:
1. om $i \bmod 4=0$ kolla bakåt $i=i-3$ tills $i<0$
2. om $i \bmod 4=3$ kolla bakåt $i=i-5$ tills $i<0$

Vi kan rita upp följande problemträd:



Datastrukturer:
1) Lista för korten.
2) Lista för lösningarna.

Algortimen:
Djupetförst sökning i problemträdet. En rekursiv funktion som får referens till *i* och sen kollar för varje kort i leken om kortfärg och kortvalör skiljer sig på

```
279 #4 position: \\
```

1. lodrätt: i=i-4 tills i<0
2. vågrätt: i=i-1 tills i mod 4 = 0
3. diagonalt:
    om i mod 3 = 0 kolla i=i-3 tills i<0
    om i mod 5 = 0 kolla i=i-5 tills i<0

Basfall: Om vi funnit en lösning eller 1, 2 eller 3 inte är uppfyllt för nått kort.
När vi finner en lösning spara undan den.

Metoder:
skapa_matris(i, vektor, losningar) - rekursiv funktion för att söka igenom problemträdet

Klasser
LankadLista
Nod