KTH CSC, Alexander Baltatzis

DD1320/1321 Lösningsförslag

Fredag 10 juni 2016 kl 8–12

Hjälpmedel: En algoritmbok (ej pythonkramaren) och ditt eget formelblad. För betyg E krävs att alla E-uppgifter är godkända, för betyg C krävs utöver detta betyg C på C-uppgifterna och för betyg A krävs högsta betyg på alla uppgifter. Lycka till!

1. *KMP*

Betyg E. Man kan behöva leta efter någonstans att slappa efter bollspel på stranden. Konstruera en KMP-automat som söker efter `BADBOLLSBAR`

Rita upp automaten och ange next-vektorn.

(10 min)

```
B A D B O L L S B A R
0 1 1 0 2 1 1 1 0 1 3
```

2. *Syntax*

Betyg E. Följande syntax

```
<RAMSA> ::= <VERB><SUBST>
<VERB>  ::= "HEJA"
<SUBST>   ::= "SVERIGE"
```

godkänner

```
HEJA SVERIGE
```

Modifiera syntaxen så att man kan säga 'heja' oändligt många gånger

```
HEJA HEJA HEJA HEJA HEJA HEJA SVERIGE
HEJA HEJA SVERIGE
```

(10 min)

Det finns flera lösningar. Det behövs rekursivitet i lösningen.

```
<VERB> ::= ‘‘HEJA’’ | ‘‘HEJA’’ <VERB>
```

eller

```
<RAMSA> ::= <VERB><SUBST> | <VERB><RAMSA>
```

3. *Storleksordning*

Betyg E. Lagledarna har börjat fundera på algoritmer för att besegra motståndarna. De undrar vad som gäller i allmänhet för vanliga algoritmer. Givet N element, vad är det för komplexitet för följande om man vill prestera så bra som möjligt och inte känner till några speciella omständigheter. Motivera kort.

a) Söka ett element i ett balanserat binärt sökträd.

b) Söka ett element i en osorterad vektor.

c) Ta bort första elementet i en osorterad vektor.

d) Ta bort första elementet i en länkad lista.

(10 min)

```
a) O(log N)
b) O(N)
c) O(N)    flyttning av elementen kostar
d) O(1)
```

4. *Komprimering*

Betyg E. Vi vill komprimera meningen nedan:

```
MER MÅL MERA MÅÅÅL
```

Komprimeringen ska göras med Huffmankodning enligt den givna frekvenstabellen. Rita Huffmanträdet och ange bitrepresentationen för varje tecken i ditt svar sorterat enligt tabellen.

Lösning:

Bokstav	Frekvens	Bitrepresentation
'Å'	22	10
'M'	22	00
' '	17	110
'R'	11	010
'E'	11	011
'L'	11	1110
'A'	6	1111

Trädet ska också ritas.

5. *Hashning*

Betyg E. En dictionary har *set*, *get* och *exists in* som gränssnitt.

Kan en dictionary med detta gränssnitt implementeras med ett bloomfilter? Motivera ditt svar.

(10 min)

Svar:

Nej, bloomfilter har ingen get. Även set är annorlunda, bloomfilter tar inget värde utan sätter bara nyckeln. Funktionen `set` i bloomfilter är mer som `insert`.

6. Diffie-Hellman

Zlatan använder sin privata nyckel och Johanna använder Zlatans publika nyckel. Zlatan autentiserar med sin privata nyckel och Johanna verifierar Zlatans identitet med Zlatans publika nyckel.

7. *Prioritetskö*

Betyg C. En prioritetskö (heap) brukar implementeras med en vektor som representerar ett träd. Barnen och föräldrar kommer man åt via matematiska funktioner (2*N, 2*N+1, N/2). Antag att man istället implementerar prioritetskön med ett pekarbaserat binärt träd.

```
class Nod:
   def __init__(self, d, p, l = None, r = None):
      self.data   = d
      self.parent = p     #föräldern
      self.left   = l     #vänsterbarnet
      self.right  = r     #högerbarnet
...


class Heap:
   def __init__(self):
      self.first = None   # pekar ut första (mest prioriterade) elementet
      self.last  = None   # pekar ut sista elementet
...
```

Antag en heap om 1000 element. Jämför implementationerna med avseende på:

- Prestanda
- Minnesåtgång

Alla operationer (tilldelning, multiplikation etc) kan antas kosta ett (1 i någon enhet) och alla enstaka minnesplatser kan antas vara lika stora.

(20 min)

minne

Trädet tar 4 gånger så mycket plats. Varje datavärde åtföljs av tre pekare. Om heapen är 1000 stor för vektorn så är den 4000 för trädet. Å andra sidan, om heapen inte är full och t.ex. bara 10 värden är inlagda tar trädet bara 40 platser.

prestanda

I båda datastrukturerna tar det log(N) att stoppa in ett värde. Algoritmen är i stort densamma.

Det sker fyra tilldelningar när man skapar en ny nod däremot sker det inte fyra tilldelningar när man trappar upp/ner elementen utan de byter plats precis som i vektorn.

Det är knepigt att hitta var `last` ska finnas, när sista raden är fylld ska ett nytt värde på början av nästa rad i trädet vilket är i den motsatta änden. Det går att lösa i log(N) men är lite knepigt. Man behöver inte ta hänsyn till last-problemet.

8. *To sort, or not to sort, that is the question*

Betyg C. Supporterklubben Hejarkaninerna har en medlemslista med 1000 medlemmar lagrad som en osorterad lista. Listan är för närvarande implementerad som en vektor.

Varje dag tillkommer 10 nya medlemmar som genast ska läggas till i listan. Varje månad slutar 20 medlemmar och de rensas ut samtidigt samma dag sista måndagen kl 15 i varje månad.

Du har till uppgift att bedöma hur effektivt det är att:

a) Lägga till medlemmar osorterat och söka i den osorterade listan när de som slutat ska tas bort (som man gör nu).

b) Använda en länkad lista. Lägga till medlemmar osorterat och söka i den osorterade listan när de som slutat ska tas bort (som man gör nu fast med länkad lista).

c) Använda en sorterad vektor. Sortera in den nya medlemmen i vektorn varje gång en ny medlem läggs till. Söka med binärsökning när medlemmar ska tas bort.

d) Använda en sorterad vektor vid behov. Lägga till medlemmar osorterat vartefter. Sortera listan (t.ex. med mergesort) i slutet på månaden så att man kan söka med binärsökning när medlemmar ska tas bort.

(20 min)

Notera att antal medlemmar som ska in (900) är i samma storleksordning som de som finns i vektorn.

Översiktligt lösningsförslag, operationer i slutet på första månaden.

	Insättning	Sökning	Bortagning	Sortering	Kommentar
a)	O(1)	O(N)	O (N)		
	900*1	20 * 850	20*850		Sök + flytta= N
b)	O(1)	O(N/2)	O(1)		
	900*1	20 * 850	20 * O(1)		bortagning íngen flytt
c)	O(N) +	O(log N)	O(N)		binärsök + flytt
	900*(850 + 10)	10	20 * (850 + 10)		
d)	(O1)	O(log N)	O(N)	O(N log N)	
	900	10	20 * (850 + 10)	1900 * 11	

Kommentar: Eftersom det bara sker insättningar och borttagningar borde listan b) vara bäst.

9. *Bollspel*

Betyg A. På väggen sitter ett bollspel med n bollar i en labyrint. Spelbrädet är täckt av en plastruta så det går inte att ta på bollarna. Målet är att få in alla bollar i labyrintens mitt samtidigt. För att flytta bollarna kan man rotera brädet 90 grader åt vänster eller höger. Vid en vridning påverkas bollarna av gravitationen och kan ramla ner om labyrinten så tillåter.

Konstruera en algoritm som ger en effektiv lösning av problemet. Berätta också vilka datastrukturer som behövs.

Rita gärna!

(25 min)

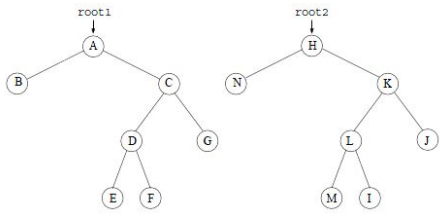
Mycket kortfattat

I varje steg kan man vrida antingen höger eller vänster. Rita. Det bildar ett problemträd. Man kan lösa med bredden eller djupet först. Spelbrädet har bara fyra tillstånd (fyra positioner). Bollarna har olika placering. Inför en datastruktur med spelbrädets position och bollarnas position. Inför en datastruktur för att hålla reda på tidigare positioner, t.ex. en dictionary med positionsstrukturen som nyckel. Dessa datastrukturer samt en stack (DFS) eller kö (BFS) behövs för att beskriva problemträdsalgoritmen.

Det behövs också en funktion som räknar ut hur bollarna rör sig när man vrider på spelplanen. Denna funktion behöver, förutom bollarnas och spelplanens position, mer detaljerad information hur spelplanen ser ut, vilken riktning gravitationen går och eventuellt hur vridningen sker i detalj (acceleration m.m.).

10 *Rekursion*

Betyg A. Anropet `samestructure(root1,root2)` ska ge `True` för två binärträd med exakt samma struktur, alltså lika om man bortser från innehållet i noderna. Ange en rekursiv algoritm för `samestructure` antingen i text eller i kod.
(25 min)




```
def same(a, b):
   if a == None and b == None:
      return True
   if a == None and b != None:
      return False
   if b == None and a != None:
      return False
   return same(a.right, b.right) and same(a.left, b, left)
```