Tentamen - Datastrukturer, algoritmer och programkonstruktion.

DVA104

Akademin för innovation, design och teknik

Onsdag 2017-05-31

Skrivtid: 08.30-13.30
Hjälpmedel: Handskrivna anteckningar (obegränsad mängd) samt ordbok/lexikon
Lärare: Caroline Uppsäll

Betygsgränser

Betyg 3: 20p
Betyg 4: 28p
Betyg 5: 34p
Max: 37p

Allmänt

- Kod skriven i tentauppgifterna är skriven i C-kod.
- På uppgifter där du ska skriva kod ska koden skrivas i C.
- Markera tydligt vilken uppgift ditt svar avser.
- Skriv bara på ena sidan av pappret.
- Referera inte mellan olika svar.
- Om du är osäker på vad som avses i någon fråga, skriv då vad du gör för antagande.
- Oläsliga/Oförståeliga svar rättas inte.
- Kommentera din kod!
- Eventuellt intjänade bonuspoäng (4p) kan endast användas på ordinarie tentamenstillfälle och adderas till ett godkänt resultat.
- Tips: Läs igenom hela tentan innan du börjar skriva för att veta hur du ska disponera din tid.

Lycka till!

Uppgift 1: *(8p)*

a) Rekursion ger ofta eleganta och tydliga lösningar men kan också vara negativt. Ange minst en sak som är negativt med att använda rekursion (förutom att det kanske är krångligt). *(1p)*

Rekursion använder mer minne än iterativa lösningar
(nytt minne alloleras för varje nytt anrop)
stackhanteringen tar extra tid
Ibland gör vi med rekursion onödigt många anrop (t.ex fibonaccitalen)

b) Ange det enda alternativet nedan som är falskt när det gäller ett binärt sökträd. Motivera ditt svar med **en** mening – *för poäng krävs både korrekt svar och en korrekt motivering. (1p)*

1) Varje träd har minst en nod falskt - ett träd kan vara tomt (NULL)
2) Varje icke-tomt träd har exakt en root-nod sant
3) Varje nod i trädet har max 2 barn sant
4) Varje icke-root-nod har exakt en förälder sant

c) 32 lag är kvalificerade för en tävling. Om lagens namn är lagrade sorterat i en array, hur många lag måste då algoritmen binärsökning titta på för att leta reda på ett specifikt lag (i värsta fall). Motivera ditt svar med **en** mening **och/eller** illustration – *för poäng krävs både korrekt svar och en korrekt motivering. (1p)*

1) Max 32
2) Max 6
3) Max 16
4) Max 1

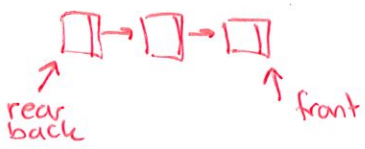
Binärsökning halverar mängden för varje ny kontroll.

kontroll 1: 32 data
2: 16
3: 8
4: 4
5: 2
6: 1 → här vet vi att datat inte finns - värsta fallet.

d) Förklara vad "krock" innebär i en hashtabell. *(1p)*

att två nycklar genererar samma hashindex
- ska läggas på samma plats i hashtabellen

e) Varför är det en dålig idé att implementera en kö (FIFO) som en länkad lista som använder listans head som köns rear/back/slut? *(1p)*



Om ingen tail finns blir både peek och dequeue O(n) - medan enqueue blir O(1)
Om front = head - vänt om.

f) Vad innebär det att en algoritm är stabil och vad innebär det att den är naturlig? *(1p)*

Stabil - behåller den relativa ordningen mellan två eller fler lika data
Naturlig - effektivare för en redan sorterad mängd

ta bort dubletter

g) Om tecknen 'D', 'E', 'B', 'A', 'C', 'D', 'F', 'C', 'G', 'F', 'E' läggs till i ett set. Vilka data kommer då setet att innehålla (oberoende av ordning)? Motivera ditt svar med **en** mening **och/eller** *illustration – för poäng krävs både korrekt svar och en korrekt motivering. (1p)*

1) D E B A C D F C G F E
2) D E B A C F G
3) D E C F
4) B A G

h) Ge tre anledningar till varför det i en ADT är viktigt att skilja på interface, implementation och användande. *(1p)*

underlättar
modifiering
återanvändning
felsökning
testning
konstruktion
vidareutveckling

Genom att kapsla in implementationen kan man undvika onödiga bindningar

Uppgift 2: *(4p)*

Antag följande lista av heltal:

```
#include <stdio.h>

typedef struct node
{
    int data;
    struct node *next;
}listNode;

typedef struct list
{
    listNode *head;
    listNode *tail;
}List;
```

Anta att head och tail pekar på första resp. sista nod, eller på NULL vid tom lista.

Och följande funktion createNewNode:

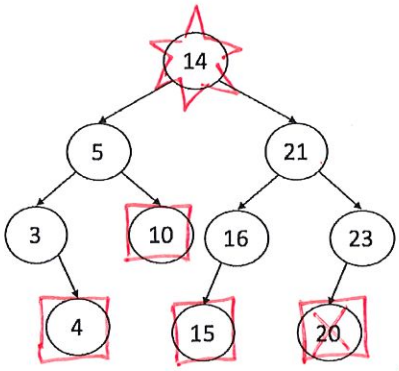
```
listNode* createNewNode(int data)
{
    listNode* newNode = (listNode*)malloc(sizeof(listNode));
    if(newNode)
    {
        newNode->data = data;
        newNode->next = NULL;
    }
    return newNode;
}
```

Din uppgift är att skriva funktionen som lägger till data sist i listan. Du måste använda dig av följande funktionshuvud:

```
List addLast(List *list, int data)
{
    listNode *newNode = createNewNode(data);
    if(newNode != NULL)
    {
        if (list->head == NULL)
            list->head = list->tail = newNode;
        else
        {
            list->tail->next = newNode;
            list->tail = newNode;
        }
    return *list;
}
```

Uppgift 3: *(9p)*

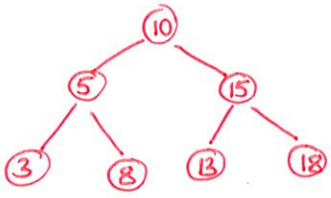
a) Antag följande binära sökträd *(2p)*



20 bör ligga till vänster om 21

- Rita av trädet på ditt svarspapper.
- Rita fyrkanter runt alla löv
- Rita en stjärna runt rooten
- Kryssa över den eller de noder som inte ligger sorterat (om någon alls)

b) Rita ett fullt binärt sökträd med minst 5 noder. Du kan låta noderna innehålla heltal men väljer själva vilka tal noderna ska innehålla. *(1p)*



fullt - alla nivåer är helt fyllda.

c) Antag implementationen:

```
struct treeNode
{
      int data;
      struct treeNode* left;
      struct treeNode* right;
};

typedef struct treeNode* BSTree;
```

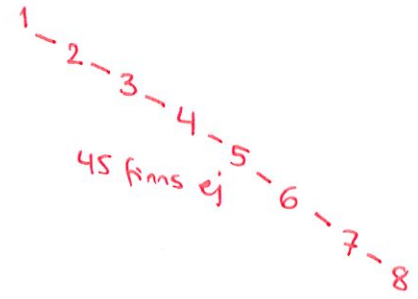
Skriv en funktion som summerar innehållet i alla noder i det binära sökträdet, du väljer själv om funktionen ska vara rekursiv eller iterativ men vilket du väljer ska anges på svarspappret ovanför själva lösningen. *(4p)*

```
int sum(BSTree tree)
{
    if (tree == NULL)
        return 0;
    return tree->data + sum(tree->left) + sum(tree->right);
}
```

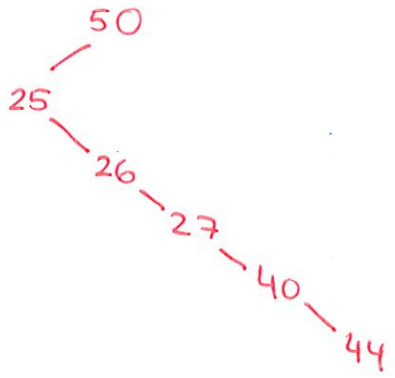
d) Antag att vi har ett binärt sökträd som innehåller tal mellan 1 och 100 och att vi vill söka efter 45. Vilken (möjligtvis vilka) av nedanstående sekvenser kan vara sekvensen av de noder som undersöks i jakten på 45. – *för poäng krävs både korrekt(a) svar och korrekt(a) motivering(ar). (2p)*

1) 5, 2, 1, 10, 39, 34, 77, 63 5 är root, vi går till 2 (V) men borde gå åt H
2) 1, 2, 3, 4, 5, 6, 7, 8 45 > än alla datan - ok - 45 finns ej 45 > 5
3) 9,8, 63, 0, 4, 3, 2, 1 9 root, går till 8 (V) men borde gå till H (45 > 9)
4) 8, 7, 6, 5, 4, 3, 2, 1 letar åt fel håll
5) 50, 25, 26, 27, 40, 44, 42 efter 44 bär vi gå åt H (45 > 44) men går åt V.
6) 50, 25, 26, 27, 40, 44 45 < 50 - gå vänster, 45 > än resten - gå höger

(2)



(6)



Uppgift 4: *(4p)*

För vardera av funktionerna nedan. Ange vilken komplexitetsklass de ska placeras i (med avseende på n).

De komplexitetsklasser du har till ditt förfogande är följande (ej sorterade i ordning):
$O(2^n)$, $O(n)$, $O(\log_2 n)$, $O(1)$, $O(n^3)$, $O(n \log_2 n)$, $O(n^2)$

a)

```
void sunny(int n, int x)
{
    for(int k = 0; k < n; ++k)
    {
        if(x < 50)
        {
            for(int i = 0; i < n; ++i)
                for(int j = 0; j < i; ++j)
                    printf("x = %d\n", x);
        }
    }
}
```

n varv

* n varv, n varv } n^2

vid x > 50 => O(n)

vid x < 50 => $O(n^3)$

↑ tre nästlade som alla beror på n

b)

```
int silly(int n, int m)
{
    if(n < 1)
        return m;
    else if(n < 10)
        return silly(n/2, m);
    else
        return silly(n-2, m);
}
```

vi tittar på stora n

stora n, -2 påverkar inte => O(n)

c)

```
void happy(int n)
{
    for(int i = n*n; i > 0; i--)
    {
        for(int j = 1000; j > 0; j = j/2)
            printf("j = %d\n", j);
        for(int m = 0; m < 5000; m++)
            printf("m = %d\n", m);
    }
}
```

n^2 varv

ej beroende av n

ej beroende av n

} $O(n^2)$

Uppgift 5: *(5p)*

a) Antag en delete-funktion för en hashtabell som löser krockar med öppen adressering. Funktionen ska anropa en hashfunktion för att få ett index genererat från den nyckel som ska lagras. Funktionen ska sedan leta upp rätt nyckel/värde baserat på indexet (genom att titta framåt ett steg i taget) och ta bort denna. För att funktionen ska fungera korrekt är det ytterligare två saker som behöver göras, vilka och varför? *(2p)*

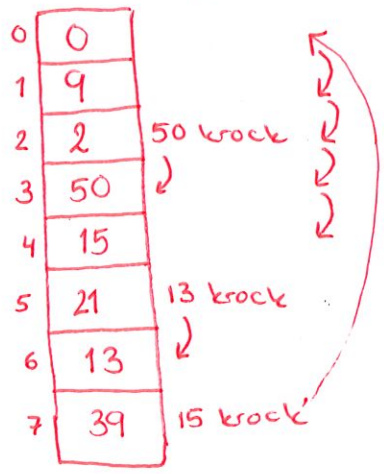
ta bort alla efterföljande buckets (fram till tom plats) och lägga till dem igen (så att de hamnar på rätt plats baserat på "nya" förutsättningar).

b) Antag att du har en hashtabell med 8 platser där hashfunktionen är key % 8 och du ska lägga in sekvensen 21, 13, 9, 2, 50, 0, 39, 15 Visa hur tabellen ser ut efter att sekvensen sats in. *(2p)*

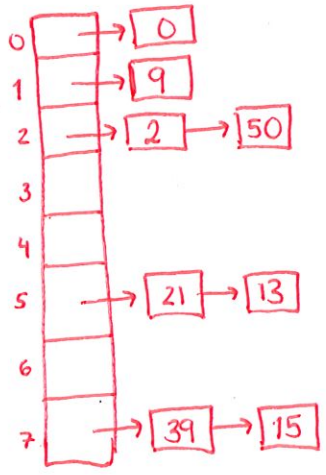
8:ans tabell: 0, 8, 16, 24, 32, 40, 48

key	%8
21	5
13	5
9	1
2	2
50	2
0	0
39	7
15	7

öppen adressering



länkad tabell



c) En länkad hashtabell har en tabellstorlek som är 512. Vilket är det totala antalet nycklar som kan läggas in i tabellen? Motivera ditt svar med **en** mening – *för poäng krävs både korrekt svar och en korrekt motivering. (1p)*

1) 256
2) 511
3) 512
4) 1024
5) Det finns inget max

det är endast det fysiska minnet som begränsar mängden data

Uppgift 6: *(7p)*

a) Följande array skickas som indata till en sorteringsalgoritm

8	7	6	5	4	3	2	1

Vid något tillfälle under sorteringen ser arrayen ut så här:

1	2	6	5	4	3	7	8

Kan algoritmen vara en Insertion Sort (insättningssortering), Selection Sort (Urvalssortering) eller Bubble Sort? Motivera ditt svar – *för poäng krävs både korrekt svar och en korrekt motivering. (2p)*

Bubble

8 7 6 5 4 3 2 1

efter varv 1: 7 6 5 4 3 2 1 8

efter varv 2: 6 5 4 3 2 1 7 8

ej rätt

Insättning

V H

8 | 7 6 5 4 3 2 1

7 8 | 6 5 4 3 2 1

6 7 8 | 5 4 3 2 1

ej rätt, vi sorterar bakifrån

Urval

V H

8 7 6 5 4 3 2 1 minst

byt

1 | 7 6 5 4 3 2 8

byt

1 2 | 6 5 4 3 7 8

RÄTT

b) Följande array har precis blivit uppdelad i första varvet av en Quick Sort:

0 2 3 4 5 6 7 8 9 ordning vid sortering

3	0	2	4	5	8	7	6	9

Vilka av dessa element kan ha varit pivot i det första varvet? Samtliga möjligheter ska anges. *(2p)*

Pivot ska nu ligga på rätt plats med mindre data till vänster och större data till höger.

Möjligt pivot: 4 - rätt plats, mindre till V, större till H

5 - - " -

9 - - " -

7 ligger på rätt plats men 8 och 6 ligger på fel sida om 7

c) Du ska nu jobba med Insertion Sort (Insättningssortering) och Merge Sort. För vardera av dessa, vilken är värsta fallets komplexitet om du vet att...

1) Indatat redan är sorterat
2) Indatat är bakåtsorterat
3) Indatat är en linjär sekvens innehållande *n* kopior av samma data

Du ska för varje case ovan och **för vardera** av algoritmerna motivera/förklara med **en** mening – *för poäng krävs både korrekt svar och en korrekt motivering. (3p)*

Insättning

1) $O(n^2)$ - om vi söker från vänster i den sorterade delen, vi måste söka igenom alla sorterade varje varv.
↳ annars $O(n)$

2) $O(n^2)$ - för alla data måste alla i sorterade delen flyttas för att skapa en lucka.

3) $O(n^2)$ - ej stabil - lika data läggs innan - flytta alla i v för att skapa lucka (byter relativ ordning) - samma som ②

$O(n^2)$ - samma som punkt ① (stabil)

Om algoritmen inser att alla är lika - $O(n)$ - men en sådan modifiering är inte beskriven

Merge

1) 2) 3) $O(n \log n)$ - merge sort går alltid samma arbete oavsett hur data ser ut.