Tentamen - Datastrukturer, algoritmer och programkonstruktion.

DVA104

Akademin för innovation, design och teknik

Måndag 2018-06-04

Skrivtid: 14.30-19.30
Hjälpmedel: Handskrivna anteckningar (obegränsad mängd) samt ordbok/lexikon och kortlek
Lärare: Caroline Uppsäll (anträffbar på 0704616110)

Preliminära betygsgränser

Betyg 3: 19p
Betyg 4: 27p
Betyg 5: 33p
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!
- Tips: Läs igenom hela tentan innan du börjar skriva för att veta hur du ska disponera din tid.

Lycka till!

Uppgift 1: *(1p)*

Hur många nivåer krävs i ett balanserat binärt sökträd för att lagra 100 noder, rootnoden ligger på nivå 1.

a) 100
b) 50
c) 10
d) 7
e) 6
f) Inget av ovanstående (ange då hur många nivåer som krävs).

Uppgift 2: *(1p)*

För att göra en bredden först traversering av en graf krävs det att man använder en underliggande datastruktur (förutom den struktur som beskriver själva grafen). Vilken?

a) Länkad lista
b) Stack
c) Set
d) Binärt sökträd
e) Kö
f) Array

Uppgift 3: *(3p)*

Antag att vi har implementerat vårt träd på följande vis:

```
struct treenode
{
      struct treenode* left;
      struct treenode* right;
      int data;
};
typedef struct treenode* Tree;
```

Skriv nu en funktion som räknar ihop (data)värdet av alla noder i trädet och returnerar resultatet. Om inga noder finns i trädet ska funktionen returnera värdet 0. Funktionen **måste vara rekursiv** och ha följande huvud:

```
int sumOfNodes(const Tree tree)
```

Förtydligande: du ska alltså inte räkna antalet noder, utan summan av alla noders värden.

Uppgift 4: *(5p)*

Betrakta följande program:

```
int foo(int n)
{
      if(n == 1)
            return 1;
      else
            return 2*foo(n-1) + foo(n-1);
}
```

1) Hur många anrop till foo() görs totalt när man gör anropet foo(3)? [1p]
2) Vad har foo() för tidskomplexitet (klass) med avseende på n? (svara i Ordo) [2p]
3) Kan du förenkla en rad i programmet så att programmet får bättre tidskomplexitet (klass) men fortfarande returnerar samma resultat för varje n? Skriv om en rad och skriv den nya tidskomplexiteten (som Ordo). [2p]

Uppgift 5: *(3p)*
Nedan finns två lösningar på ett problem. Du ska…
… ange komplexiteten (klass) i Ordo för vardera lösning [2p]
… diskutera utifrån komplexiteterna vilken lösning som är att föredra (effektivas) [1p]

A

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int strLen(char *str)
{
    int i;
    for(i = 0; str[i] != '\0'; i++);
    return i;
}

int main(void)
{
    char str[25];
    printf("Enter string: ");
    fgets(str, 25, stdin);
    for(int i = 0; i < strLen(str); i++)
        if(isalpha(str[i]))
            printf("%c", str[i]);
    return 0;
}
```

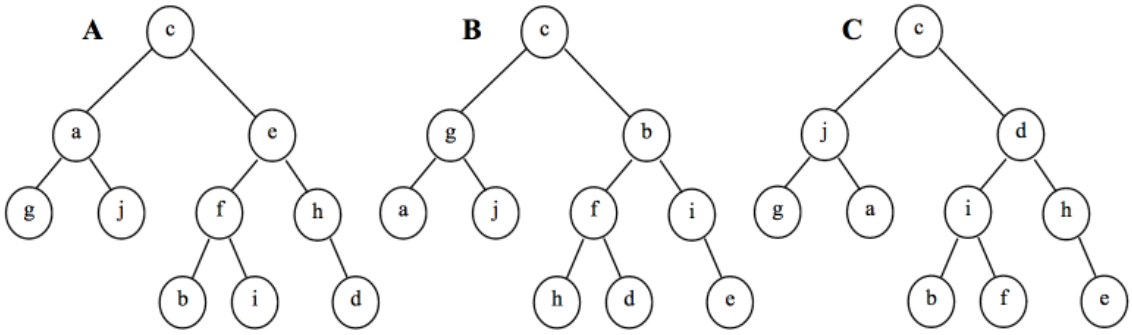
B

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

int strLen(char *str)
{
    int i;
    for(i = 0; str[i] != '\0'; i++);
    return i;
}

int main(void)
{
    char str[25];
    printf("Enter string: ");
    fgets(str, 25, stdin);
    int len = strLen(str);
    for(int i = 0; i < len; i++)
        if(isalpha(str[i]))
            printf("%c", str[i]);
    return 0;
}
```

Uppgift 6: *(2p)*
Vilket av träden nedan har
postorder (LR) traverseringen (genomlöpningen) g, a, j, b, f, i, e, h, d, c och
inorder (LR) traverseringen (genomlöpningen) g, j, a, c, b, i, f, d, h, e?



Uppgift 7: *(4p)*

Utgå ifrån följande mängd:

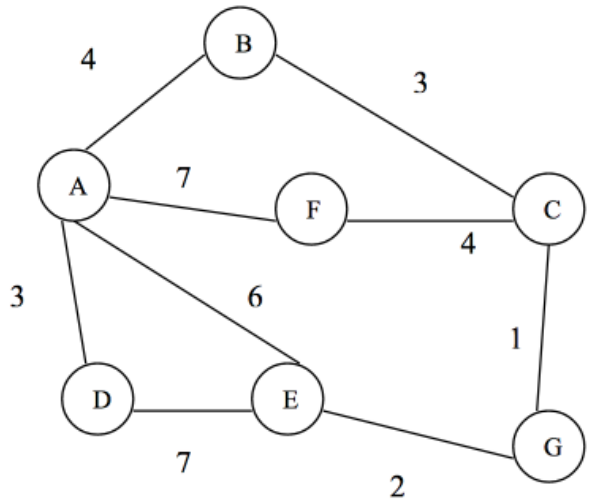
3	8	7	2	5	1	9	4	6

Denna mängd har sorterats (till 1, 2, 3, 4, 5, 6, 7, 8, 9) med hjälp av 4 olika algoritmer. a-d nedan är någonstans i vardera av dessa. Din uppgift är att koppla ihop rätt mängd med rätt sorteringsalgoritm. De algoritmer du har att välja på är Bubbelsortering, Insättningssortering (insertion sort), Urvalssortering (selction sort), Merge sort och Quick sort (där sista elementet valts som pivot). En av algoritmerna har inte använts alls.

a) 1 2 3 8 5 7 9 4 6
b) 3 2 5 1 7 4 6 8 9
c) 3 2 5 1 4 6 9 7 8
d) 2 3 5 7 8 1 9 4 6

Det är inte lönt att chansa, felaktigt angivet svar ger avdrag, frågan som helhet kan dock inte ge minuspoäng.

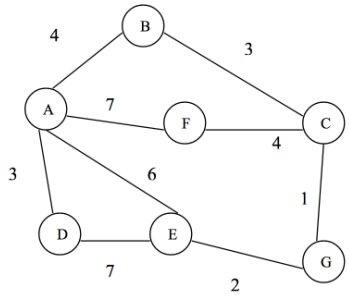
Uppgift 8: *(3p)*
Vilken/vilka av nedanstående påstående är sanna för följande graf?



a) Den är riktad
b) Den är viktad [0.5p]
c) Den är starkt sammanhängande
d) Graden i nod A är 3
e) Den är oriktad [0.5p]
f) Den är sammanhängande [0.5p]
g) Det är en multigraf
h) Den innehåller cykler [0.5p]
i) A B C G D E F är en djupet först traversering av grafen med start i A. [1p]

Det är inte lönt att chansa, felaktigt angivet svar ger avdrag, frågan som helhet kan dock inte ge minuspoäng.

Uppgift 9: *(4p)*
Fortsätt jobba med samma graf som i uppgift 8.



1) Förklara varför man skulle vilja ta ut ett minimum spanning tree (Prim's algoritm) ur en graf (ge gärna ett exempel) (max 5 meningar). [1p]

Grafen kan beskrivas i termer av nodpar (x, n, y) där x och y är två noder som delar en båge (ordningen är oviktig) och n är siffran på bågen. Grafen nedan kan alltså beskrivas som:
(A, 4, B)(A, 3, D)(A, 6, E)(A, 7, F)(B, 3, C)(C, 1, G)(C, 4, F)(D, 7, E)(E, 2, G)

2) Du ska ta fram ett minimum spanning tree med start i A av trädet nedan och presentera det som en lista av nodpar (enligt exemplet ovan). [3p]

Uppgift 10: *(4p)*
Antag att du har en enkellänkad lista bestående av följande nodtyp och listtyp

```
typedef struct node
{
    int data;
    struct node *next;
    struct node *prev;
}Node;
```

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

Listan består av en head-pekare till första noden samt en tail-pekare till sista noden, den sista nodens next är satta till NULL och likaså den första nodens prev. head och tail måste alltid peka på första respektive sista noden. För en tom lista är dessa satta till NULL.
När en nod skapas sätts alltid dess next och prev till NULL.

Antag nedan (ofullständiga) lösning på en funktion som tar bort ett angivet data ur listan genom att länka ur noden och frigöra minnet för den, funktionen ska returnera 1 om datat finns (tas bort) och annars 0. head och tail ska efter borttagningen peka på första respektive sista noden alternativt vara NULL (om listan nu är tom). (Notera att lösningen inte nödvändigtvis är den effektivaste lösningen på problemet utan är skriven för att ge så tydliga fall som möjligt).

Din uppgift är att fylla i koden som saknas (markerade med A-G).

```
int remove(List *myList, int dataToRemove)
{
      if(A)
            return 0;
      Node *cur = B;
      while(cur != NULL)
      {
            if(cur->data == dataToRemove)
            {
                  if(cur == myList->head && cur == myList->tail)
                        C
                  else if(cur == myList->head)
                        D
                  else if(cur == myList->tail)
                        E
                  else
                  {
                        F
                  }
                  free(cur);
                  return 1;
            }
            G;
      }
      return 0;
}
```

Uppgift 11: *(3p)*
Om öppen adressering används vid krockar i en hashtabell så kan man t.ex. använda en så kallad linjär sondering (linear probing) för att lösa dem, detta innebär att man vid krock hela tiden tittar ett steg framåt tills en tom plats hittas eller man inser att tabellen är full. Problemet med denna lösning är att krockar lätt leder till kluster av nycklar, vilket i sin tur ökar komplexiteten på sökning i tabellen.

Ett alternativ till linjär sondering (för öppen adressering) är att använda kvadratisk sondering (quadratic probing) för att hitta en ledig plats vid krock. Denna metod tenderar att sprida nycklarna mer än linjär sondering och därmed också minska klusterbildningen vid lika nycklar. Kvadratisk sondering går till så att man vid krock letar framåt i kvadrat. Vid första krocken så letar man 1^2 steg framåt (alltså 1 steg) vid andra krocken letar man 2^2 steg framåt (alltså 4 steg) från originalindexet, vid tredje krocken letar man 3^2 steg framåt från originalindexet, vid fjärde krocken letar man 4^2 steg framåt från originalindexet osv. Man cirkulerar i hashtabellen på samma vis som vid linjär sondering.

Tabellen till höger har använt just öppen adressering med kvadratisk sondering för att lösa krockar. Följande nycklar har lagts till (i angiven ordning):
35, 16, 25, 99, 23, 85, 19, 10.

Storleken på hashtabellen är 10 platser och hashfunktionen gör nyckel % 10.

0	99
1	10
2	
3	23
4	85
5	35
6	16
7	
8	19
9	25

1) Är tabellen korrekt? [2p]
2) Hur skulle en linjär hashtabell av samma storlek, och med samma hashfunktion, se ut vid insättning av ovan nämnt data i samma ordning. [1p]
   Det går bra att rita svaret eller att beskriva det mer i ”text”, t.ex:

   Index i --> 5 --> 15 --> 25
   Index j --> tom
   Index k -->17 --> 27

Uppgift 12: *(4p)*
Ett alternativ till linjär sondering (för öppen adressering) är (som beskrivits i uppgift 11) att använda kvadratisk sondering (quadratic probing) för att hitta en ledig plats vid krock. Denna metod tenderar att sprida nycklarna mer än linjär sondering och därmed också minska klusterbildningen vid lika nycklar. Kvadratisk sondering går till så att man vid krock letar framåt i kvadrat. Vid första krocken så letar man 1^2 steg framåt (alltså 1 steg) vid andra krocken letar man 2^2 steg framåt (alltså 4 steg) från originalindexet, vid tredje krocken letar man 3^2 steg framåt från originalindexet, vid fjärde krocken letar man 4^2 steg framåt från originalindexet osv. Man cirkulerar i hashtabellen på samma vis som vid linjär sondering.

Du ska nu implementera funktionen `quadraticProbe` som baserat på hashtabellen och en nyckel ska leta reda på korrekt index för nyckeln att placeras på enligt reglerna för kvadratisk sondering beskrivna ovan. Det du behöver veta om resterande implementation finns nedan, en bucket för nyckel-värde-paret, en typ för själva hashtabellen som innehåller tabellen (pekare till bucket) samt tabellens storlek samt hashfunktionen. Alla lediga platser är markerade med ”UNUSED”.

```
struct Bucket
{
      int key;
      int value;
};

typedef struct
{
      struct Bucket* table;
      unsigned int size;
} HashTable;


int hash(int key, int tablesize)
{
      return key % tablesize;
}
```

Din implementation av `quadraticProbe` ska använda sig av funktionshuvudet:

```
int quadraticProbe(HashTable* htable, int key);
```

Du behöver börja med att anropa hashfunktionen för att få ett startindex. Därefter behöver du kontrollera platsen (indexet) tills du hittar en ledig plats eller exakt samma nyckel (då ska ju värdet associerat med nyckeln uppdateras). Du behöver också hålla koll på om tabellen är full. Varje gång du behöver leta framåt ska du göra det kvadratiskt (du måste då veta vilket försök du är på). Indexet ska returneras ur funktionen, är tabellen full så att det inte går att lägga till den nya nyckeln ska -1 returneras.

Funktionen ska alltså inte lägga till någonting utan endast hitta rätt plats för nyckeln.