----------LÖSNINGSFÖRSLAG----------

Tentamen - Datastrukturer, algoritmer och programkonstruktion.

DVA104

Akademin för innovation, design och teknik

Onsdag 2018-01-10

Skrivtid: 8.30-13.30
Hjälpmedel: Handskrivna anteckningar (obegränsad mängd) samt ordbok/lexikon
Lärare: Caroline Uppsäll (anträffbar på 0704616110)

Betygsgränser

Betyg 3: 17p
Betyg 4: 24p
Betyg 5: 29p
Max: 33p

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)*

Förklara med max 5 meningar hur rekursion använder sig av stack-beteendet

Lösningsförslag

Vid anrop läggs den nya funktionsinstansens variabler högst upp på stacken, när instansen terminerar/returnerar tas dess variabler (på toppen av stacken) bort och exekveringen återgår till anropande instans – vars instanser nu ligger på toppen av stacken.

Uppgift 2: *(1p)*

Vad bör man utgå ifrån när man skapar en ADT, svara med 1-2 meningar?

Lösningsförslag

Vad man vill kunna göra med datatypen – inte hur man gör det

Uppgift 3: *(2p)*

Antag att 530264 anges som inparameter till funktionen nedan. Vilket blir resultatet?

```
int funk(int n)
{
    if(n == 0)
        return 0;

    return ((n % 10) + funk(n / 10));
}
```

Att ange en förklaring till resultatet är inget krav men kan ge delpoäng om svaret inte är korrekt.

Lösningsförslag

Funktionen summerar delarna i ett tal, 5+3+0+2+6+4 = 20 kommer att returneras

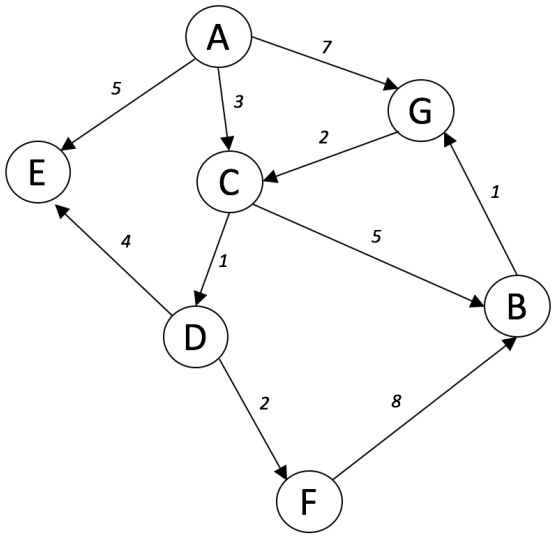
Anrop 1: n = 530264
Anrop 2: n = 53026
Anrop 3: n = 5302
Anrop 4: n = 530
Anrop 5: n = 53
Anrop 6: n = 5
Anrop 7: n = 0 BASFALL returnera 0 till anrop 6
Returnera 5%10 + 0 = 5 till anrop 5
Returnera 53%10 + 5 = 3 + 5 = 8 till anrop 4

Returnera 530%10 + 8 = 0 + 8 = 8 till anrop 3
Returnera 5302%10 + 8 = 2 + 8 = 10 till anrop 2
Returnera 53026%10 + 10 = 6 + 10 = 16 till anrop 1
Returnera 530264%10 + 16 = 4 + 16 = 20 till anrop 0

Helt korrekta anrop (men felaktig retur eller resultat) ger 1p.

Uppgift 4: *(2p)*

Vilka av nedanstående påståenden är sanna för följande graf



a) Grafen är riktad
b) Grafen är oriktad
c) Grafen är sammanhängande
d) C D E A C är en loop i grafen
e) Grafen innehåller inga loopar alls
f) Ingrad i nod A är 0
g) Grafen är en multigraf
h) Grafen är viktad

Uppgift 5: *(4p)*

Antag att du har en enkellänkad lista bestående av följande nodtyp och listtyp

```
typedef struct node
{
    int data;
    struct node *next;
}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. head och tail måste alltid peka på första respektive sista noden. För en tom lista är dessa satta till NULL.

Som hjälp har du en funktion
`Node* createNewNode(int data);`
som allokerar dynamiskt minne för en ny nod, om minne gick att allokera sätts data-delen av noden till det data som anges i parameterlistan och pekaren next sätts till NULL, den nya noden returneras. Om allokeringen misslyckas returneras NULL från funktionen.

Och en funktion
`List * createList();`
som allokerar minne för en lista samt sätter dess head och tail till NULL. Om allokeringen lyckades returneras den nya listan, om inte returneras NULL.

Din uppgift är att skriva en funktion som tar in en lista (`myList`), skapar en ny, tom, lista och kopiera hela innehållet från den inskickade listan (`myList`) till den nya listan samt returnera den nya listan. Om något problem skulle uppstå, som att det t.ex. inte går att allokera minne för en nod eller lista, så ska NULL returneras från funktionen. Ordningen på datat i den nya listan ska vara den samma som ordningen på datat i `myList`.

Använd följande funktionshuvud

```
List *copy(List *myList);
```

Lösningsförslag

```
List * copy(List *myList)
{
    Node *newNode;
    Node *cur = myList->head;
    List *newList = createList();
    if(newList == NULL)
        return NULL;
    while(cur != NULL) //gå igenom hela myList
    {
        newNode = createNewNode(cur->data);
        if(newNode == NULL)
            return NULL;
        if(newList->head == NULL) //om det är första noden
            newList->head = newList->tail = newNode;
        else //för alla andra noder, länka in sist (först går också bra)
        {
            newList->tail->next = newNode;
            newList->tail = newNode;
        }
        cur = cur->next;
    }
    return newList;
}
```

Uppgift 6: *(1p)*

Antag att du har en cirkulär arrayimplementation av en kö (FIFO) där nedanstående funktion lägger till data i kön

```
void enqueue(int queue[], int front, int back, int data, int arrSize)
{
    queue[back] = data;
    back = (back + 1) % arrSize;
}
```

Vad är det som saknas i implementationen ovan, det räcker att svara med text?

Lösningsförslag

Det är faktiskt två saker som saknas, men för poäng krävs det endast att man anger en sak.

- Test om arrayen är full innan insättning
- Retur av back

Uppgift 7: *(1p)*
Vilket av följande är ett korrekt binärt sökträd utskrivet i preorder (LR)?

a) 10 5 8 2 23 15 30
b) 2 8 5 15 30 23 10
c) 10 5 2 8 23 15 30
d) 10 23 5 30 15 8 2

Uppgift 8: *(2p)*
Antag att du har ett balanserat binärt sökträd med 4000 noder, hur många noder måste du i värsta fall söka igenom för att inse att det du söker efter inte finns i trädet? NULL är inte en nod och ska alltså inte räknas.
Ange antalet noder och en motivering på 1-2 meningar

Lösningsförslag

Maximalt 12 noder behöver sökas igenom, med 12 nivåer kan det få plats maximalt 4095 noder i trädet, vi behöver endast titta på en nod per nivå

Uppgift 9: *(5p)*
Antag följande trädnod

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

```
typedef struct treeNode* BSTree;
```

Själva trädet (rooten) är pekare till en trädnod (`BSTree root`), om trädet är tomt är rooten satt till NULL. När en ny nod skapas sätts dess left och right-pekare till NULL.

Du har en funktion
```
int numberOfNodes(const BSTree tree);
```
som returnerar antalet noder i trädet.

Skriv en funktion som avgör om ett binärt träd är ett sökträd eller inte. Funktionen ska returnera 1 om trädet är ett sökträd och 0 om det är inte är det, är trädet tomt kan -1 returneras.
Funktionshuvudet du ska använda är

```
int isSearchTree(BSTree root)
```

Det är fullt tillåtet att skriva hjälpfunktioner (vill du gå på en lösning med hjälp av tipset nedan så behöver du en hjälpfunktion)

Tips: du behöver nödvändigtvis inte kontrollera sorteringen i själva trädet, går det kanske att använda någon annan struktur, t.ex. en struktur där man kan testa datat linjärt?

Lösningsförslag
```
void writeInOrderToArray(BSTree tree, int arr[], int *index)
{
    if(tree != NULL)
    {
        writeInOrderToArray(tree->left, arr, index);
        arr[*index] = tree->data;
        (*index)++;
        writeInOrderToArray(tree->right, arr, index);
    }
}

int isSearchTree(BSTree root)
{
    int size = numberOfNodes(root);
    int arr[size], i = 0;
    writeInOrderToArray(root, arr, &i); //skriv hela trädet i inorder till en array
    for(i = 0; i < size-1; i++) //testa om arrayen är sorterad största till minsta
        if(arr[i] < arr[i+1]) //om något data ligger fel är trädet inte sorterat
            return 0;
    return 1;
}
```

Uppgift 10: *(2p)*
Antag att du har en hashtabell med 10 platser som löser krockar med hjälp av öppen adressering. Hashfunktionen gör beräkningen key%10 för att hitta rätt index, den linjära sökningen vid krock görs genom att titta ett steg framåt.

Följande nycklar har lagts till i angiven ordning:
498, 98, 398, 244, 139, 1244, 574, 31, 824, 94

Vilken komplexitet får sökning efter nyckel 94 i tabellen? Motivera ditt svar med 1-2 meningar

Lösningsförslag
O(n) när 94 ska läggas till är index 4, 5, 6, 7, 8, 9, 0, 1, 2 upptagna, alla dessa letas igenom innan 94 hittas på index 3.

Uppgift 11: *(2p)*
Antag samma tabell som i uppgiften innan

Du har en hashtabell med 10 platser som löser krockar med hjälp av öppen adressering. Hashfunktionen gör beräkningen key%10 för att hitta rätt index, den linjära sökningen vid krock görs genom att titta ett steg framåt.

Följande nycklar har lagts till i angiven ordning:
498, 98, 398, 244, 139, 1244, 574, 31, 824, 94

Hur ser tabellen ut efter att 574 har tagits bort?

Lösningsförslag

Index 0: 398
Index 1: 139
Index 2: 31
Index 3
Index 4: 244
Index 5: 1244
Index 6: 824
Index 7: 94
Index 8: 498
Index 9: 98
Alla efterföljande (fram till tom plats) – I det här fallet alla andra nycklar – måste tas bort (I ordning) och läggas till igen. Man gör detta för en nyckel i taget

Om man tar bort alla och sen ligger in dem i samma ordning som man tog bort dem i får man resultatet: 398, 139, 31, tom, 824, 94, 244, 1244, 498, 98 vilket också ger poäng på uppgiften

Uppgift 12: *(3p)*

Du vill hitta ett effektivt sätt att ta reda på om någon annan i rummet har samma födelsedag som du själv har och har kommit fram till tre olika algoritmer för att lösa problemet.

a) Du säger vilken födelsedag du har och frågor om någon annan i rummet har samma födelsedag. Om någon har samma födelsedag svarar hen ja.

b) Du berättar din födelsedag för den första personen och frågar om hen har samma födelsedag, om personen svarar nej går du vidare till nästa person och upprepar samma sak, du gör detta med samtliga personer i rummet eller tills du får svaret ja.
c) Du frågar endast person 1, person 1 frågar i sin tur person 2 som i sin tur frågar person 3 osv. Detta görs på följande sätt: Du berättar din födelsedag för person 1 och frågar om hen har samma födelsedag, om person 1 säger nej ber du hen att fråga person 2 och berätta svaret för dig, om svaret är nej ber du person 1 att tar reda på om person 3 har samma födelsedag, person 1 ber då person 2 att fråga person 3 osv.

I samtliga tre algoritmer är det antalet personer i rummet som bestämmer storleken på problemet.

I värsta fall, ange för var och en av algoritmerna ovan vilken komplexitetsklass de hamnar i. Möjliga komplexitetsklasser är O(1), O(log n), O(n), O(n log n), $O(n^2)$, $O(2^n)$
Motivera dina svar med 1-2 meningar vardera.

Lösningsförslag

a) O(1) – du behöver endast fråga en gång oavsett hur många personer som är i rummet
b) O(n) – du behöver i värsta fall fråga alla i rummet, en och en (värsta fallet är att ingen har samma födelsedag)
c) O(n2) – Värsta fallet är samma som i b men antalet frågor blir 1 + 2 + 3 + … + n-1 + n = n(n+1)/2

Uppgift 13: *(2p)*
Antag att följande sekvens skickas in till en sorteringsalgoritm

5	3	8	1	7	6	9	2	4

Någon gång under sorteringen ser sekvensen ut som nedan

4	3	1	2	5	6	9	8	7

Vilken sorteringsalgoritm har använts? Bubblesort, Insertionsort eller Quicksort (1:a elementet väljs som pivot)

Uppgift 14: *(2p)*
Ge två anledningar till varför QuickSort är effektivare än MergeSort vid ett val av pivot som hela tiden delar mängden på hälften (medelfall/average), de har ju båda samma komplexitet. Svara med ca 1-3 meningar per anledning.

MergeSort kräver mer minne – det gör inte quicksort – att allokera minne och kopiera ut delarrayer tar tid.
Om mängden innehåller delar som är relativt sorterade redan kommer quicksort inte behöver göra lika många byten, algoritmen behöver fortfarande gå igenom alla element men om de ligger relativt sorterat behöver den inte flytta på data. MergeSort gör samma jobb oavsett av hur mängden ser ut

Uppgift 15: *(3p)*
Nedan finns implementationen av en urvalssortering där olika kodsegment är numrerade, Din uppgift är att koppla ihop rätt kodsegment med rätt kommentar från listan med kommentar. Vissa av kommentarerna hör inte till den aktuella algoritmen och ska alltså lämnas utan koppling till kodsegment.

```
void selectionSort(int arr[], int n)
{
    int i, j, x;

    for (i = 0; i < n-1; i++) //A 3 eller 9
    {
        x = i; //B – 2
        for (j = i+1; j < n; j++) //C – 10
            if (arr[j] < arr[x]) //D – 5
                x = j; //E – 1

        int temp = arr[x];
        arr[x] = arr[i];          //F – 6
        arr[i] = temp;
    }
}
```

1) /*Byt vilket index som håller det minsta talet*/
2) /*sätt minsta värdet till första index i höger delarray*/
3) /*För alla element i höger delarray förutom ett*/
4) /*Plocka ut första elementet i höger delarray*/
5) /*Om ett nytt tal är minst*/
6) /*Byt plats på det minsta elementet och det första elementet i höger delarray*/
7) /*Gå igenom hela vänster delarray*/
8) /*Flytta element i vänster delarray för att skapa en lucka*/
9) /*Flytta gränsen mellan vänster och höger delarray*/
10) /*För alla element i höger delarray*/