TENTAMEN: Algoritmer och datastrukturer

Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv ditt namn och personnummer på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programmen skall skrivas i C++, vara indenterade och kommenterade, och i övrigt vara utformade enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tentamenstesen och förbered ev. frågor.

I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa.

Lycka till!

Uppgift 1

Välj **ett** svarsalternativ på varje fråga. Motivering krävs ej. Varje korrekt svar ger två poäng. Garderingar ger noll poäng.

1. Vilken av metoderna a-d lämpar sig bäst för BFS-genomlöpning av noderna i ett träd?
   a. rekursion
   b. rekursion + FIFO-kö
   c. iteration + FIFO-kö
   d. iteration + LIFO-stack

2. Vilken av sorteringsmetoderna a-d kräver *O*(N) extra minne
   a. shellsort
   b. quicksort
   c. mergesort
   d. heapsort

3. Ett av sambanden 1-3 beskriver tidskomplexiteten hos f.

   1. $\begin{cases} T(1) = 1 \\ T(N) = T(N-1) + N \end{cases}$

   2. $\begin{cases} T(1) = 1 \\ T(N) = 2T(N/2) + c \end{cases}$

   3. $\begin{cases} T(1) = 1 \\ T(N) = T(N/2) + N \end{cases}$

```
int f( int n ) {
    if ( n == 0 )
        return 1;
    else {
        int s = f( n/2 );
        for ( int i = 0; i < n; i++ )
            s += 1;
        return s;
    }
}
```

   Vilket av a-d är en minsta övre begränsning för lösningen till sambandet i fråga?
   a. *O*(log N)
   b. *O*(N)
   c. *O*(N·log N)
   d. *O*(N^2)

4. Antag att du skall konstruera en algoritm som med en textfil av ord som indata rapporterar radnumret i filen för den andra förekomsten av varje ord som påträffas minst två gånger. Vilken STL-behållare a-f är mest användbar för en effektiv lösning av problemet?
   a. vector
   b. queue
   c. set
   d. multiset
   e. map
   f. multimap

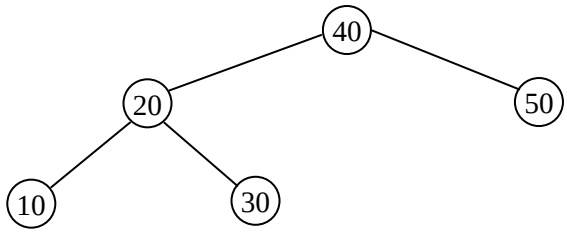
forts.

5. Vilket av a-e uttrycker bäst hur många bågar det finns i en tät graf (eng. dense graph) med N noder om det mellan två noder *v* och *w* får finnas högst två bågar, en från *v* till *w* resp. en från *w* till *v*?
   a. $O(N)$
   b. $O(N^2)$
   c. $\Omega(N)$
   d. $\Omega(N^2)$
   e. $\Theta(N^2)$

(10 p)

Uppgift 2

a) Visa de tre olika AVL-träden som fås genom insättning av ett av talen 5, 35 respektive 45 i trädet



Nödvändiga rotationer som krävs för aև upprätthålla AVL-villkoret skall beaktas i lösningen.

(5 p)

b) Vad är det minimala resp. maximala antalet noder i ett AVL-träd av höjd 4?

(1 p)

c) Definiera ett generellt rekursivt samband för det minimala antalet noder i ett AVL-träd.

(2 p)

Uppgift 3

a) Hur undviker man primär klusterbildning i en sluten hashtabell?

(2 p)

b) En hashtabell innehåller

0	
1	92
2	58
3	24
4	
5	
6	

Ange tabellens utseende efter insättning av 126 om tabellen implementeras som i Weiss. Hashfunktionen definieras *hash*(*x*) = *x mod M*, där *M* är antalet platser i tabellen..

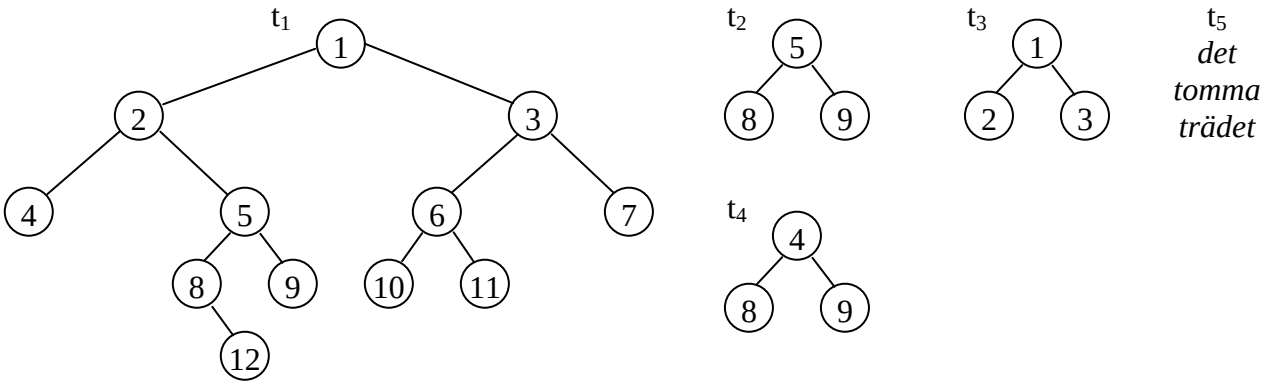
(3 p)

Uppgift 4

Binära träd kan representeras med noder av typen

```
struct TreeNode {
    int element;
    TreeNode *left, *right;
};
```

Betrakta de binära träden



Vi skall införa begreppen *matchar* resp. *inkluderar* för träd. Ett träd matchar ett annat om det överensstämmer med det andra från roten och nedåt, men det andra kan vara djupare och innehålla ytterligare noder. Ett träd inkluderar ett annat om det andra ingår i ett delträd i det första.

- Alla träd matchar, resp. inkluderar sig själva.
- Det tomma trädet matchar alla träd, men inga icketomma träd matchar det tomma trädet.
- Alla träd inkluderar det tomma trädet, men det tomma trädet inkluderar inga icketomma träd.

För träden ovan gäller dessutom att

t_3 matchar t_1	t_1 inkluderar t_2, t_3
t_1 matchar ej t_2, t_3, t_4	t_1 inkluderar ej t_4
t_2 matchar ej t_1, t_3, t_4	t_2 inkluderar ej t_1, t_3, t_4
t_3 matchar ej t_2, t_4	t_3 inkluderar ej t_1, t_2, t_4
t_4 matchar ej t_1, t_2, t_3	t_4 inkluderar ej t_1, t_2, t_3

Uppgiften är att generalisera exemplet och definiera de båda begreppen som rekursiva funktioner:

a) Implementera

```
bool matches( const TreeNode *t1, const TreeNode *t2 );
```

så att den returnerar sant om t1 matchar t2, falskt annars.

(5 p)

b) Implementera

```
bool includes( const TreeNode *t1, const TreeNode *t2 );
```

så att den returnerar sant om t1 inkluderar t2, falskt annars. *Ledning*: utnyttja funktionen i a.

(5 p)

Uppgift 5

a) Vilka två egenskaper karakteriserar en binär hög?

(2 p)

b) Hur ser den binära högen

	9	12	11	21	22	33	34	26	23	37	39	98	36	97	35
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

ut efter ett anrop av `deleteMin`? Rita resultatet som ett träd.

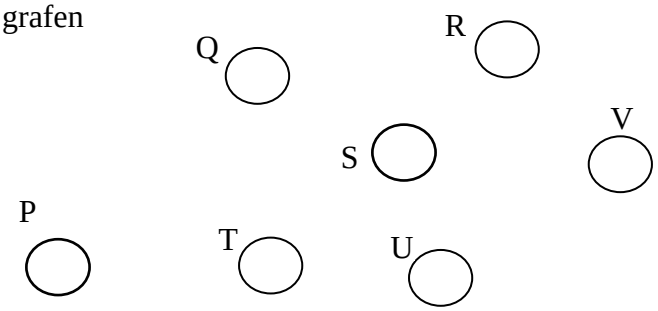
(2 p)

Uppgift 6

a) Nämn två villkor som måste vara uppfyllda för att noderna i en graf skall kunna ordnas topologiskt.

(2 p)

b) Rita in bågar i grafen

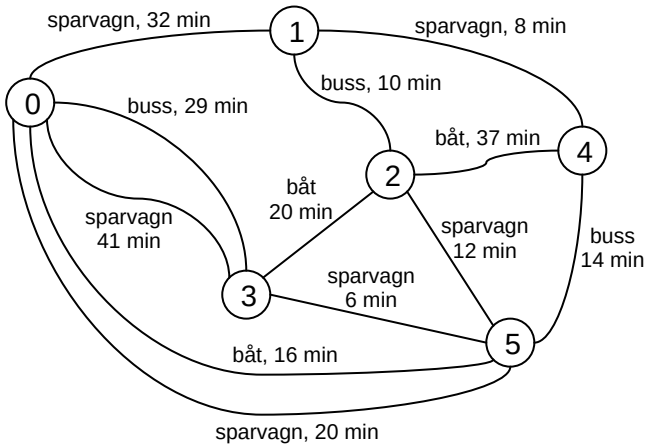


så att SUVTPQR, SUTVPQR, SUTPVQR och SUTPQVR blir de enda möjliga topologiska ordningarna av noderna.

(5 p)

Uppgift 7 *Denna uppgift kan lösas i två svårighetsgrader som ger 5 resp. 16 poäng*

Trafikbolaget Värsttrafik har ekonomiska problem och måste spara. En viss inbesparing har redan uppnåtts genom att ersätta spårvagnar med sparvagnar, och nu blir du anlitad som konsult för att ta fram underlag för en konsekvensanalys av nedläggning av linjer. Som framgår av följande karta är platserna 0 – 5 trafikerade av ett antal olika färdmedel, i vissa fall flera på samma sträcka.



Besparingsidén är att begränsa antalet linjesträckor till ett absolut minimum, så att alla platser är fortfarande är nåbara, men inte nödvändigtvis med direktförbindelse. Ett tidskrav måste uppfyllas: Den sammanlagda restiden på de valda linjesträckorna skall vara minimal. Ett linjenät som uppfyller dessa krav ges i den vänstra tabellen nedan.

```
Optimerat linjenät:
0 --- 5 båt (16 minuter)
1 --- 4 sparvagn (8 minuter)
1 --- 2 buss (10 minuter)
2 --- 5 sparvagn (12 minuter)
3 --- 5 sparvagn (6 minuter)
```

Tabellen till höger ger total restid i det nya nätet från varje plats till alla övriga. Problemet är nu att konstruera ett generellt program som givet ett ooptimerat linjenät av ovanstående typ konstruerar ett optimerat nät och skriver ut två tabeller som i exemplet.

```
Restider:
från 0 till 1: 38 minuter
från 0 till 2: 28 minuter
från 0 till 3: 22 minuter
från 0 till 4: 46 minuter
från 0 till 5: 16 minuter
från 1 till 2: 10 minuter
från 1 till 3: 28 minuter
från 1 till 4: 8 minuter
från 1 till 5: 22 minuter
från 2 till 3: 18 minuter
från 2 till 4: 18 minuter
från 2 till 5: 12 minuter
från 3 till 4: 36 minuter
från 3 till 5: 6 minuter
från 4 till 5: 30 minuter
```

Uppgiften kan lösas i två svårighetsgrader.
Gör antingen a eller b.

a) Beskriv principiellt i ord och/eller pseudokod lämpliga algoritmer som löser ovanstående problem, samt vilka datastrukturer som behövs och deras roll i lösningen.

(5 p)

b) Genomför en fullständig implementering av klassen `TrafikAnalys` som finns sist i bilagan. Använd lämpliga datastrukturer. Det ooptimerade linjenätet finns i en textfil där första raden är ett heltal som anger antalet orter och resterande rader en linjesträcka per rad. Varje rad har formen: *ort1 ort2 trafikslag restid* (samtliga är ickenegativa heltal). Orterna numreras från 0 och uppåt.

(16 p)

BILAGOR TILL TENTAMEN

```
// Disjoint set class
//
// CONSTRUCTION: with int representing initial number of sets
//
// ******************PUBLIC OPERATIONS*********************
// void union( root1, root2 ) --> Merge two sets
// int find( x )              --> Return set containing x
// ******************ERRORS********************************
// BadArgument exception thrown as needed.
// Elements in the set are numbered starting at 0.

class DisjSets {
public:
    DisjSets( int numElements );

    int find( int x ) const;
    int find( int x );
    void unionSets( int root1, int root2 );

private:
    // Data representation ...
};


// BinaryHeap class interface.
//
// CONSTRUCTION: with no parameters or vector containing items.
//
// ******************PUBLIC OPERATIONS*********************
// void insert( x )       --> Insert x
// void deleteMin( )      --> Remove smallest item
// void deleteMin( min )  --> Remove and send back smallest item
// Comparable findMin( )  --> Return smallest item
// bool isEmpty( )        --> Return true if empty; else false
// void makeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// Throws UnderflowException as warranted.

template <class Comparable>
class BinaryHeap {
public:
    BinaryHeap( );
    BinaryHeap( const vector<Comparable> & v );

    bool isEmpty( ) const;
    const Comparable & findMin( ) const;

    void insert( const Comparable & x );
    void deleteMin( );
    void deleteMin( Comparable & minItem );
    void makeEmpty( );
private:
    // Data representation ...
};
```

Bilaga till uppgift 7

```
enum Trafikslag { Buss, Baat, Sparvagn };

struct Linjestracka {
    Linjestracka() {}
    Linjestracka( int a, int b, Trafikslag ts, float t )
         : fran(a), till(b), trafikslag(ts), tid(t) {}

    int fran, till;
    Trafikslag trafikslag;
    float tid;

    bool operator>( const Linjestracka & ls ) const {
        return tid > ls.tid;
    }
    bool operator<( const Linjestracka & ls ) const {
        return tid < ls.tid;
    }
    void print() const;   // ort1 --- ort2 ts (x minuter)
};                        // ex. 1 --- 2 buss (10 minuter)

// Graph class interface
//
// ******************PUBLIC OPERATIONS*********************
// void addEdge( int v, int w, double cvw, Trafikslag ts )
//                              --> Add additional edge
// void unweighted( int s )     --> Single-source unweighted
// void dijkstra( int s )       --> Single-source weighted
// void printEdges( )           --> Print all edges after alg is run
// double getDistance( int s ) --> Get distance to s after alg is run
// ******************************************************
class Graph{
public:
    Graph( ) { }
    ~Graph( );
    void addEdge( int sourceName, int destName, double cost,
                                                Trafikslag ts );
    void unweighted( int startName );
    void dijkstra( int startName );
    double getDistance( int destName ) const;
    void printEdges() const;
private:
    // Data representation ...
};

// Uppgift
class TrafikAnalys {
public:
    TrafikAnalys( istream &in ) : inStream(in) {}
    void readFile();                          // implement this
    void create();                            // implement this
    void search();                            // implement this
private:
    // Data representation ...                // implement this
};
```