Institutionen för data- och elektroteknik
Uno Holmer
uno.holmer@chl.chalmers.se

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. Varje korrekt svar ger två poäng.

1. Vilken/vilka av bitföljderna är möjliga Huffman-kompressioner av strängen ”tillplattad”?
   a. 10100110110001001101101110000
   b. 01001101010100101101101101010
   c. 01011001001110110010010001111
   d. 10100111111001101101101010000
   e. två av ovanstående
   f. inget av ovanstående

2. Vilken datastruktur lämpar sig bäst om man vill beräkna medianen i en datamängd
   a. binärt sökträd
   b. graf
   c. prioritetskö
   d. hashtabell
   e. stack
   f. ingen av ovanstående

3. Skillnaden i antalet noder i två binära träd av samma höjd är 11. Det ena trädet är fullt och det andra har ett löv. Hur höga är träden? (ett träd med 1 nod har höjden 1)
   a. 2
   b. 3
   c. 4
   d. 5
   e. inget av ovanstående

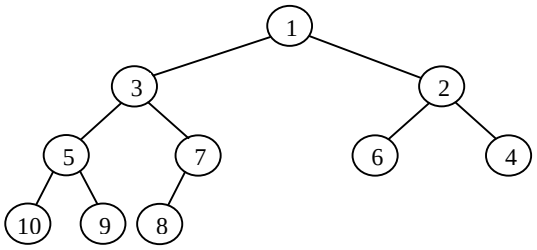
4. Vilket uttryck ger den snävaste övre begränsningen för tidskomplexiteten hos kodavsnittet

```
for ( I = 0; I < N; I++ )
    for ( J = I; J > 0; J /= 2 )
        M++;
for ( I = 1; I < N; I *= 2 )
    for ( J = I; J > 0; J-- )
        M++;
```

   a. $O(N)$
   b. $O(\log N)$
   c. $O(N \log N)$
   d. $O(\log^2 N)$
   e. $O(N^2)$
   f. inget av ovanstående

forts.

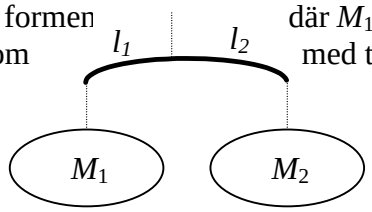
5. I vilken följd lagras talen i en fältrepresentation av den binära högen



a. 10 5 9 3 8 7 1 6 2 4
b. 1 3 2 5 7 6 4 10 9 8
c. 10 9 5 8 7 3 6 4 2 1
d. 1 3 5 10 9 7 8 2 6 4
e. inget av ovanstående

(10 p)

Uppgift 2

En mobil är antingen
1. ett löv med en massa m:  *m*
2. eller på formen  där M_1 och M_2 är mobiler och l_1 och l_2 är dellängderna på en stång som med två snören förbinder de två delmobilerna med varandra.

Mobiler kan modelleras som dataobjekt, en klassdefinition finns i bilaga.

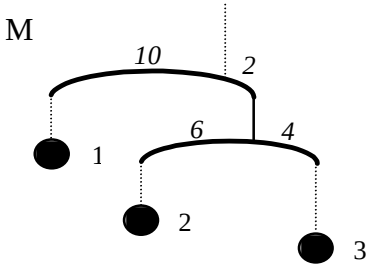
a) Konstruera en rekursiv medlemsfunktion i mobilklassen som skriver ut mobilen ”tillplattad”, men med strukturen visad med parenteser:

```
void PrettyPrint() const;
```

Utskriftsformatet för en enkel mobil är ’**(**’ *vikt* ’**)**’ och ’**[**’ *mobil* ’**,**’ *längd* ’**,**’ *längd* ’**,**’ *mobil* ’**]**’ för en sammansatt .

Exempel: M.PrettyPrint() skall ge utskriften

```
[(1),10,2,[(2),6,4,(3)]]
```



(6 p)

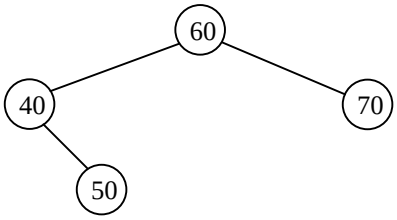
b) Konstruera en rekursiv medlemsfunktion i mobilklassen som avgör om en mobil är balanserad. En enkel mobil är alltid balanserad, och en sammansatt om stängerna på alla nivåer är vågräta. *Ledning:* en hävstång med dellängderna l_1, l_2 och massorna m_1 och m_2 är balanserad om $l_1\, m_1 = l_2\, m_2$. Eventuell hjälpfunktion får införas om behov föreligger.

```
bool Balanced() const;
```

(6 p)

Uppgift 3

Tre tal skall sättas in i det binära sökträdet



a. Visa hur trädet ser ut efter den insättningsföljd av talen 10, 20, 30 som ger det mest höjdbalanserade trädet.

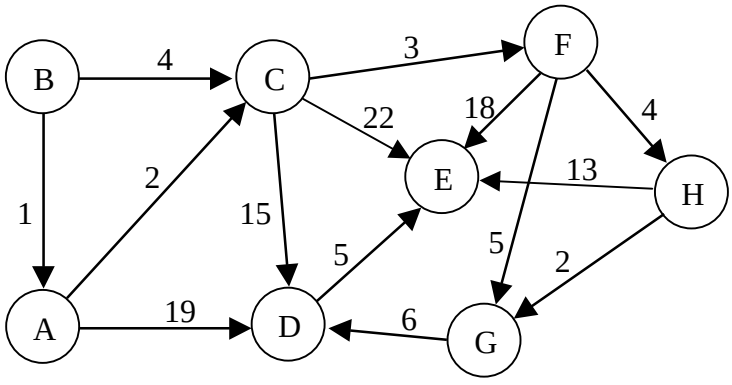
(2 p)

b. En av de möjliga insättningssekvenserna i a ger ett träd som kan roteras i ett steg så att det uppfyller AVL-villkoret. Beskriv med en figur hur denna rotation går till i det aktuella fallet.

(4 p)

Uppgift 4

Betrakta följande riktade graf



a) Ange grafens grannmatris

(3 p)

b) Ange kortaste oviktade vägar, resp. de kortaste viktade vägarna från nod B till samtliga övriga noder.

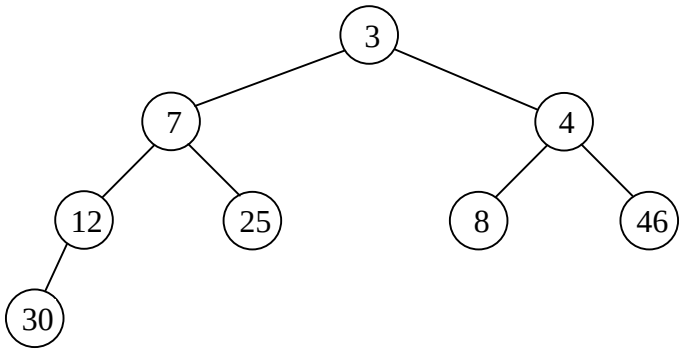
(3 p)

c) Konstruera och rita en sammanhängande riktad ickecyklisk graf med 5 noder a,b,c,d och e sådan att de två följderna bacde och abced är möjliga topologiska ordningar av grafens noder.

(3 p)

Uppgift 5

Om *H* är den binära högen



a) Hur ser *H* ut efter tre anrop av DeleteMin?

(3 p)

b) Hur ser *H* ut efter insättning av talen 15, 20, 6, 2 med operationen Insert?

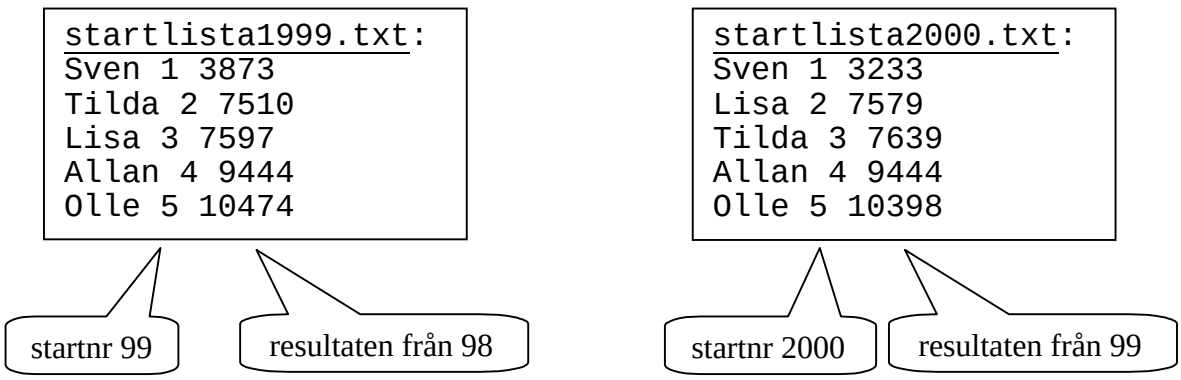
(3 p)

Uppgift 6

I ett årligt motionslopp får deltagarna startnummer i nästa års tävling efter prestation, kortare tid ger lägre startnummer. Vinnaren ett visst år får alltså starta med startnummer 1 året därpå. Deltagarlistan finns lagrad i en textfil sorterad efter startnummer. Varje rad innehåller

Namn Startnr Tid

där Tid (sekunder) är *föreg.* års tid (0 för nya deltagare) och Startnr är startnumret i årets tävling. Skriv ett program som simulerar en tävling genom att slumpa fram resultattider för deltagarna. Deltagare som inte sprungit tidigare har tiden 0 i indatafilen och kan antas springa i mål på en slumpmässig tid mellan 1 och 3 timmar. Deltagare som sprungit tidigare får sin fjolårstid, varierad slumpmässigt ±15 minuter.
Ett påbörjat program finns i bilaga. Uppgiften består i att implementera funktionen Run i klassen Simulator. Funktionen skall läsa föregående års startlista från inströmmen, samt producera årets startlista på utströmmen. Använd lämplig datastruktur.
Exempel: År 1999 hade Tilda startnummer 2 och Lisa 3, baserat på resultaten från 1998. 1999 sprang Lisa in på tiden 7579 mot Tildas 7639. De båda byter alltså startnummer år 2000.



(10 p)

Uppgift 7

En hashtabell innehåller följande heltal

0	
1	
2	
3	
4	
5	165
6	26
7	
8	
9	

Hashfunktionen definieras *Hash(x,M) = x mod M*, där *M* är antalet platser i tabellen.

a) Ange tabellens utseende vid linjär resp. kvadratisk sondering efter operationerna: [1]

Insert(95); Remove(165); Insert(7); insert(634); Insert(15); Insert(337); Insert(49); Remove(15); Insert(79);

(4 p)

b) Vad händer om man efter insättningssekvensen med kvadratisk sondering i a dessutom försöker sätta in 36124? Förklara!

(3 p)

[1] sondering = probing

Bilagor till tentamen

Bilaga till uppgift 6

```
class Deltagare {
public:
    Deltagare() {}
    Deltagare( const char *N, int Nr ) : Startnr(Nr) { strcpy(Namn,N); }
    friend ostream & operator<< ( ostream & Out, const Deltagare & D );
    friend istream & operator>> ( istream & In, Deltagare & D );

    char Namn[ 100 ];
    int Startnr;
};

class Resultat {
public:
    Resultat() : Tid(0) {}
    Resultat( const Deltagare & D, unsigned long T ) : Delt(D), Tid(T) {}
    bool operator< ( const Resultat & R ) const { return Tid < R.Tid; }
    friend ostream & operator<< ( ostream & Out, const Resultat & R );
    friend istream & operator>> ( istream & In, Resultat & R );

    Deltagare Delt;
    unsigned long Tid;
};

class Simulator {
public:
    // Öppnar filerna
    Simulator( const char *StartLst, const char *NyStartLst );

    void Run();         // implementera denna!

private:
    ifstream Startlista;
    ofstream NyStartlista;
};

void main() {
    char Startlista[ 100 ], NyStartlista[ 100 ];
    cout << "Förra årets startlista: ";
    cin.getline( Startlista, 100 );
    cout << "Spara årets startlista i: ";
    cin.getline( NyStartlista, 100 );
    Simulator S( Startlista, NyStartlista );
    S.Run();
}
```

Bilaga till uppgift 2

```
class Mobile {
public:
    Mobile( float Mass );                        // Simple case
    Mobile( const Mobile *L, float LLength,    // Composite case
            const Mobile *R, float RLength );
    ~Mobile();
    float Mass() const;       // Return total mass of mobile
    int Height() const;      // Return the max height of mobile
    void PrettyPrint() const; // Print a flat view of mobile
    bool Balanced() const;    // Determine if mobile is balanced
private:
    enum MobileType { Simple, Composite };
    MobileType Type;
    float theMass;                  // Simple
    float LeftLength, RightLength;  // Composite
    const Mobile *Left, *Right;     //    -" -
    int IsSimple() const { return Type == Simple; }
};

// make a leaf mobile
Mobile::Mobile( float M )
    : Type(Simple), theMass(M), Left(NULL), Right(NULL) { }

// make a composite mobile
Mobile::Mobile( const Mobile *L,  float LLength,
                const Mobile *R, float RLength )
    : Type(Composite), Left(L), Right(R),
      LeftLength(LLength), RightLength(RLength) { }
```

```
// Set class interface
//
// CONSTRUCTION: with (a) size of maximal integer to be stored
//      in the set, (b) initialization with existing set
// ************************  public operations ************************
// Set operator+= (int)     --> Add an integer to the set
// Set operator= (Set)      --> Copying assignment
// bool operator< (int,Set) --> Membership relation
// bool operator<= (Set)    --> Subset relation
// bool operator== (Set)    --> Equality relation
// Set operator|| (Set)     --> Set union
// Set operator&& (Set)     --> Set intersection
// Set operator- (Set)      --> Set difference
// int size()               --> Number of distinct elements in the set
// istream operator>> (istream,Set) --> Reads a set from the keyboard
// ostream operator<< (ostream,Set) --> Prints a set on the screen

class Set {
public:
    Set(int maxNum);                              // constructor
    Set(const Set & s);                           // copy constructor
    ~Set();                                       // destructor
    Set & operator += (int x);                    // add an integer
    const Set & operator = (const Set & Rhs);     // copying assignment
    friend bool operator < (int x, const Set & Rhs); // membership
    bool operator <= (const Set & Rhs) const;     // subset
    bool operator == (const Set & Rhs) const;     // equality
    Set operator || (const Set & Rhs) const;      // union
    Set operator && (const Set & Rhs) const;      // intersection
    Set operator - (const Set & Rhs) const;       // difference
    int Size() const { return theSize; }          // number of elements
    friend istream & operator >> (istream & in,Set & value);
    friend ostream & operator << (ostream & out,const Set & value);
private:
    int *theSet;  // array pointer
    int max;      // max size of stored numbers (= array size - 1)
    int theSize;  // number of distinct elements in the set
};
```

```
// Queue class interface
//
// Etype: must have zero-parameter and constructor
// CONSTRUCTION: with (a) no initializer;
//     copy construction of Queue objects is DISALLOWED
// Deep copy is supported
//
// ******************PUBLIC OPERATIONS*********************
// void Enqueue( Etype X )--> Insert X
// void Dequeue( )        --> Remove least recently inserted item
// Etype Front( )         --> Return least recently inserted item
// int IsEmpty( )         --> Return 1 if empty; else return 0
// int IsFull( )          --> Return 1 if full; else return 0
// void MakeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// Predefined exception is propogated if new fails
// EXCEPTION is called for Front or Dequeue on empty queue

template <class Etype>
class Queue
{
  public:
    Queue( );
    ~Queue( );

    const Queue & operator=( const Queue & Rhs );

    void Enqueue( const Etype & X );    // Insert
    void Dequeue( );                    // Remove
    const Etype & Front( ) const;       // Find
    int IsEmpty( ) const;
    int IsFull( ) const;
    void MakeEmpty( );
  private:
    Queue( const Queue & ) { }    // Disable copy constructor
    // Data representation ...
 };
```

```
// BinaryHeap class interface
//
// Etype: must have zero-parameter constructor and operator=;
//     must have operator<
// CONSTRUCTION: with (a) Etype representing negative infinity
// Copy construction of BinaryHeap objects is DISALLOWED
// Deep copy is supported
//
// ******************PUBLIC OPERATIONS******************
// void Insert( Etype X ) --> Insert X
// Etype FindMin( )       --> Return smallest item
// void DeleteMin( )      --> Remove smallest item
// void DeleteMin( Etype & X ) --> Same, but put it in X
// int IsEmpty( )         --> Return 1 if empty; else return 0
// int IsFull( )          --> Return 1 if full; else return 0
// void MakeEmpty( )      --> Remove all items
// void Toss( Etype X )   --> Insert X (lazily)
// void FixHeap( )        --> Reestablish heap order property
// ******************ERRORS*****************************
// Predefined exception is propogated if new fails
// EXCEPTION is thrown for FindMin or DeleteMin when empty

template <class Etype>
class BinaryHeap
{
  public:
        // Constructor, destructor, and copy assignment
    BinaryHeap( const Etype & MinVal );
    ~BinaryHeap( ) { delete [ ] Array; }

    const BinaryHeap & operator=( const BinaryHeap & Rhs );

        // Add an item maintaining heap order
    void Insert( const Etype & X );

        // Add an item but do not maintain order
    void Toss( const Etype & X );

        // Return minimum item in heap
    const Etype & FindMin( );

        // Delete minimum item in heap
    void DeleteMin( );
    void DeleteMin( Etype & X );

        // Reestablish heap order
    void FixHeap( );

    int IsEmpty( ) const;
    int IsFull( ) const;
    void MakeEmpty( );
private:
    // Data representation
};
```