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.

1. En eller flera av strängarna i a-e kan omöjligen ge upphov till Huffmankodningen code('a')="1", code('b')="00", code('c')="01", vilken/vilka?
   a. abacbca
   b. caba
   c. baacab
   d. cbabcab
   e. bacaca
   f. minst två
   g. inget av ovanstående

2. En sorteringsalgoritm som baseras på jämförelser av två element i taget gör högst
   a. $O(\log N)$ jämförelser
   b. $O(N)$ jämförelser
   c. $O(N \log N)$ jämförelser
   d. $O(N^2)$ jämförelser
   e. ingen av ovanstående

3. Hur stor maximal skillnad kan det vara i antalet noder i två AVL-träd av höjd 4 (höjd enl. Weiss)?
   a. 17
   b. 19
   c. 21
   d. 23
   e. inget av ovanstående

4. Vilket påstående stämmer om T(N)

   $T(1) = c$
   $T(N) = T(N/2) + N$

   a. $T(N) = \Theta(N)$
   b. $T(N) = O(\log N)$
   c. $T(N) = \Omega(N \log N)$
   d. $T(N) = \Theta(N^2)$
   e. inget av ovanstående

5. I ett visst fiktivt språk består uttryck av variabler a,b,c,… och de binära operatorerna &, @ och $. Operatorerna har olika precedens, & lägst och $ högst. $ är vänsterassociativ medan & och @ är högerassociativa. Infixuttrycket "b $ c $ d @ e @ f & g & h" har postfixformen
   a. b c $ d $ @ e f g h & & @
   b. b c d $ $ e f @ @ g & h &
   c. b c $ d $ e @ f @ g & h &
   d. b c $ d $ e f @ @ g h & &
   e. inget av ovanstående

(10 p)

Uppgift 2

En enkellänkad lista kan representeras med följande nodtyp:

```
struct Node {
    int Data;
    Node *Next;
    Node( int D, Node *N ) : Data(D), Next(N) {}
};
```

a) Definiera *rekursivt* funktionen `Copy` som returnerar en kopia av argumentlistan

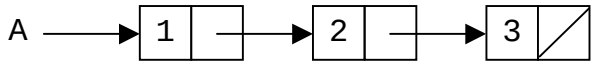
```
Node *Copy( const Node *L );
```

(4 p)

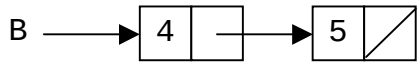
b) Definiera *rekursivt* funktionen `Append` som returnerar en sammansättning av argumentlistorna

```
Node *Append( const Node *L1, const Node *L2 );
```

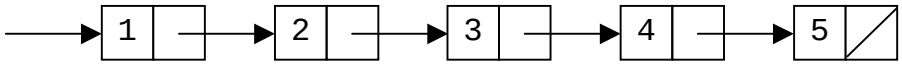
Exempel: Om A pekar ut listan



och B listan



så skall Append(A,B) returnera den nya unika listan



(6 p)

Listorna saknar huvud.

Uppgift 3

a) Vid inorder genomlöpning av ett binärt träd med fem noder märkta A-E besöktes noderna i ordningen C B A D E och vid postorder genomlöpning av samma träd C A E D B. Rita trädet!

(2 p)

b) Skriv en rekursiv funktion `IsAVL` som avgör om ett binärt träd är ett AVL-träd. Antag för enkelhets skull att trädets noder representeras med typen

```
struct Node {
    Node *Left, *Right;
};
```

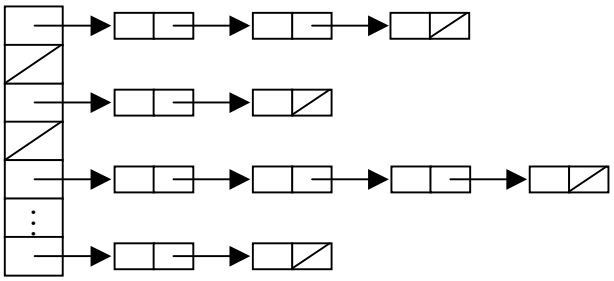
Ett tomt träd representeras med en NULL-pekare. Funktionen skall ta en pekare till trädet som argument och returnera `true` om den pekar på ett AVL-träd och `false` annars. Ev. extra hjälpfunktion får införas.

(6 p)

Uppgift 4

a)
Nummerupplysningen har en tjänst för upplysningar om abonnent som har ett visst nummer. Antag att abonnentinformationen lagras i en öppen hashtabell med kollisionslistor [1]:

```
SubscriberNode *SubscriberTable[ 1000 ];
```



där listnoderna definieras

```
struct SubscriberNode {
    unsigned long Phone;
    String Name;
    ...
    SubscriberNode *Next;
};
```

Hashfunktionen definieras

```
unsigned int
Hash( const SubscriberNode *S, unsigned int TableSize ) {
    return S->Phone % TableSize;
}
```

Antag att fältstorleken av någon anledning är fixerad till 1000 och att fältet därför inte kan expanderas vidare. Tabellen skall användas för att hantera abonnentinformation för ett stort antal stora myndigheter och organisationer. Dessa har i allmänhet nummer på formen *dddd*000 (om numret är sjusiffrigt). Uppskatta genomsnittligt antal jämförelser för lyckad sökning för en tabell med N sådana abonnenter. Ange söktiden på formen *O*(f(N)). Motivera svaret!

(2 p)

b) En sluten hashtabell har följande innehåll

0	28
1	78
2	
3	
4	130
5	
6	

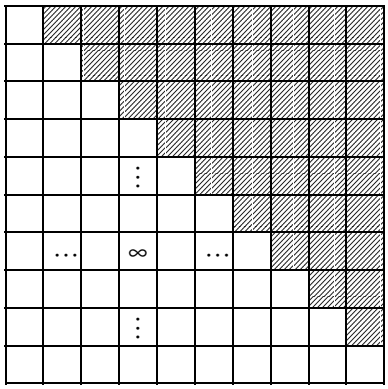
Hashfunktionen definieras *Hash(x,M) = x mod M*, där *M* är antalet platser i tabellen. Ange tabellens utseende efter Insert(112); Kvadratisk sondering används λ får ej överstiga 0.5. [2]

(4 p)

[1] Separate chaining
[2] sondering = probing

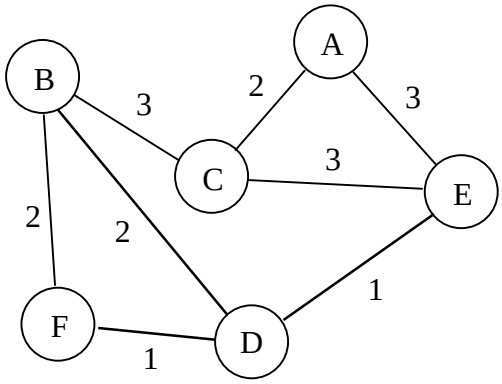
Uppgift 5

a) Går det att ordna noderna topologiskt för en riktad graf som har en grannmatris av följande form? Samtliga blanka celler innehåller ∞, de streckade cellerna innehåller tal. Motivera!



(2 p)

b) Betaltidningen "Metrix" planerar ett framtida tryckeri som skall förse orterna A-F med tidningar. Grafen nedan beskriver vägnätet som förbinder orterna. Grafen är oriktad [3].Talen på bågarna anger maximala körtider i timmar mellan resp. orter. Om tryckeriet placeras i en ort som har flera utfartsvägar sänds flera bilar samtidigt och i orter utefter rutten som har flera utfartsvägar kompletteras med ytterligare bilar. Utkörning från tryckeriet börjar senast klockan 03:00. I vilken av orterna A-F bör tryckeriet ligga så att alla lokala distributörer hinner få sina tidningar senast 06:00? Hur många bilar behövs minst?



(4 p)

Uppgift 6

En binär hög *H* representeras av fältet

-∞	3	7	5	11	23	18		
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

a) Rita *H* som ett träd.

(1 p)

b) Ange *H* efter sekvensen Insert(2); Insert(38); DeleteMin();

(4 p)

[3] det finns en riktad båge i vardera riktningen med samma kostnad

Uppgift 7

En väderstation samlar in en stor mängd temperaturavläsningar från ett antal temperaturgivare. Varje avläsning representeras som ett dataobjekt av typen `SensorData` (se bilaga) bestående av givarens namn, samt avläst temperaturvärde. Mätdata finns i en osorterad binärfil av sådana poster. Antalet poster kan överstiga en miljon million och det kan finnas mätdata från tusentals olika givare. Periodvis finns behov av att producera en lista med min/max-temperaturen för varje givare. Listan skall vara ordnad i bokstavsordning m.a.p. givarnamn (se nedan). I bilagan finns bl.a. en påbörjad definition av klassen WheatherReport som är ett embryo till implementering av ett program som skapar en sådan lista. Tanken är att ett objekt av klassen skall innehålla min/max-informationen för alla givarna. Det finns också operationer för att bygga upp informationen och skriva ut den.

- Konstruktorn tar som argument namnet på filen med givardata.
- `ComputeMinMax` bygger upp en intern representation av informationen i filen i en lämplig datastruktur i objektet.
- `ReportMinMax` skriver ut en tabell enligt nedan.

Välj en lämplig datarepresentation och implementera medlemsfunktionerna i klassen. För att ge full poäng krävs att lösningen uppfyller följande krav:

- Det får ej antas att indatafilen ryms i primärminnet.
- Sorteringsalgoritm får ej användas.
- Det får ej antas att indatafilen ryms i primärminnet.
- En för ändamålet lämplig datastruktur skall användas.
- Om G antalet olika givarnamn får algoritmens primärminnesbehovet vara högst $O(G)$.

Exempel:

Binärfilen med temperaturavläsningar

Kiruna	.3690000E01
Falun	.1320000E02
Kairo	.1870000E02
Kiruna	-.3710000E02
Falun	.1890000E02
Boden	-.4300000E01
Luleå	.2540000E02
Boden	-.1410000E02
Falun	.1930000E02

Utskrift

```
Arvika     min:  10.7  max:  10.7
Boden      min: -38.4  max:  28.2
Borås      min:  14.8  max:  15.2
Falun      min: -37.3  max:  19.7
Irkutsk    min: -44.2  max: -43.8
Kairo      min:  13.9  max:  41.3
Karesuando min: -42.5  max:  14.6
Kiruna     min: -37.1  max:  19.1
Lindholmen min:  11.1  max:  11.2
Luleå      min: -14.4  max:  25.4
Malmö      min:  -0.4  max:  12.8
Nordpolen  min: -47.5  max: -16.8
Torneå     min: -32.4  max:  47.3
```

(15 p)

BILAGOR TILL TENTAMEN

Bilaga till uppgift 7

```
struct SensorData {
    char Id[20];
    float Temperature;
    SensorData() {}
    SensorData( const char *I, float T )
        : Temperature(T) { strcpy(Id,I); }
};

struct SensorMinMax {
    char Id[20];
    float *MinTemp, *MaxTemp;

    SensorMinMax() : MinTemp(NULL), MaxTemp(NULL) {}

    SensorMinMax( const char *I )
        : MinTemp(NULL), MaxTemp(NULL) { strcpy(Id,I); }

    SensorMinMax( const char *I, float MaxT, float MinT )
        : MinTemp(new float(MinT)), MaxTemp(new float(MaxT))
            { strcpy(Id,I); }

    SensorMinMax( const SensorMinMax & Rhs )
        : MinTemp(new float(*Rhs.MinTemp)),
          MaxTemp(new float(*Rhs.MaxTemp))
            { strcpy(Id,Rhs.Id); }

    ~SensorMinMax() {
        if ( MinTemp ) delete MinTemp;
        if ( MaxTemp ) delete MaxTemp;
    }

    bool operator< ( const SensorMinMax & Rhs ) {
        return strcmp( Id, Rhs.Id ) < 0;
    }
};

class WeatherReport {
public:
    // implement these!

 ☞ WeatherReport( const char *File );

 ☞ ~WeatherReport();     // if needed

 ☞ void ComputeMinMax(); // Compute and store min/max temps

 ☞ void ReportMinMax();  // Print a min/max report

private:
 ☞ // your choice of data representation!
};
```

```
// 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
};
```

```
// SearchTree class interface
// Etype: must have zero-parameter and copy constructor,
//     and must have operator<
// CONSTRUCTION: with (a) no initializer;
// All copying of SearchTree objects is DISALLOWED
// ******************PUBLIC OPERATIONS*********************
// int Insert( Etype X )  --> Insert X
// int Remove( Etype X )  --> Remove X
// Etype Find( Etype X )  --> Return item that matches X
// int WasFound( )        --> Return 1 if last Find succeeded
// int IsFound( Etype X ) --> Return 1 if X would be found
// Etype FindMin( )       --> Return smallest item
// Etype FindMax( )       --> Return largest item
// int IsEmpty( )         --> Return 1 if empty; else return 0
// void MakeEmpty( )      --> Remove all items
// ******************ERRORS********************************
// Predefined exception is propogated if new fails
// ItemNotFound returned on various degenerate conditions

template <class Etype>
class SearchTree {
  public:
    SearchTree();
    ~SearchTree();
    // Add X into the tree. If X already present, do nothing.
    // Return true if successful
    bool Insert( const Etype & X );

    // Remove X from the tree. Return true if successful.
    bool Remove( const Etype & X );

    // Remove minimum item from the tree. Return true if successful.
    bool RemoveMin( );

    // Return minimum item in tree. If tree is empty,
    // return ItemNotFound.
    const Etype & FindMin( ) const;

    // Return maximum item in tree. If tree is empty,
    // return ItemNotFund.
    const Etype & FindMax( ) const;

    // Return item X in tree. If X is not found, return ItemNotFound.
    // Result can be checked by calling WasFound.
    const Etype & Find( const Etype & X );

    // Return true if X is in tree.
    bool IsFound( const Etype & X );

    // Return true if last call to Find was successful.
    bool WasFound( ) const;

    // MakeEmpty tree, and test if tree is empty.
    void MakeEmpty( )
    bool IsEmpty( ) const;
  protected: // private data representation
};
```

```
// PreOrder class interface; maintains "current position"
//          in Preorder Tree Traversal
//
// Etype: same restrictions as for BinaryTree
// CONSTRUCTION: with (a) Tree to which iterator is bound
// All copying of PreOrder objects is DISALLOWED
//
// ******************PUBLIC OPERATIONS**********************
// int operator+( )     --> True if at valid position in tree
// Etype operator( )    --> Return item in current position
// void First( )        --> Set current position to first
// void operator++      --> Advance (prefix)
// ******************ERRORS*********************************
// EXCEPTION is thrown for illegal access or advance


// PostOrder class interface; maintains "current position"
//           in Postorder Tree Traversal
//
// Etype: same restrictions as for BinaryTree
// CONSTRUCTION: with (a) Tree to which iterator is bound
// All copying of PostOrder objects is DISALLOWED
//
// ******************PUBLIC OPERATIONS**********************
// int operator+( )     --> True if at valid position in tree
// Etype operator( )    --> Return item in current position
// void First( )        --> Set current position to first
// void operator++      --> Advance (prefix)
// ******************ERRORS*********************************
// EXCEPTION is thrown for illegal access or advance


// InOrder class interface; maintains "current position"
//          in Inorder Tree Traversal
//
// Etype: same restrictions as for BinaryTree
// CONSTRUCTION: with (a) Tree to which iterator is bound
// All copying of InOrder objects is DISALLOWED
//
// ******************PUBLIC OPERATIONS**********************
// int operator+( )     --> True if at valid position in tree
// Etype operator( )    --> Return item in current position
// void First( )        --> Set current position to first
// void operator++      --> Advance (prefix)
// ******************ERRORS*********************************
// EXCEPTION is thrown for illegal access or advance


Exempel:
    SearchTree<int> T;
    PreOrder<int> It(T);
```