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

Lösningsförslag till tentamen * *Preliminär*

Kursnamn	**Algoritmer och datastrukturer, 4p**
Tentamensdatum	**2001-08-22**
Program	**DAI 2**
Läsår	**2000/2001, lp I/II**
Examinator	**Uno Holmer**

*** se även `http://www.chl.chalmers.se/~holmer/` där finns det mesta av kursmaterialet.**

Uppgift 1 (10 p)

Ingen lösning ges. Se kurslitteraturen.

Uppgift 2 (10 p)

```
template <class T,class U>
Node< Pair<T,U> > *zip( Node<T> *l1, Node<U> *l2 )
                  throw (std::length_error)
{
    if ( l1 == NULL && l2 == NULL )
        return NULL;
    else if ( l1 != NULL && l2 != NULL )
        return new Node< Pair<T,U> >
                       ( Pair<T,U>( l1->data, l2->data ),
                         zip( l1->next, l2->next ) );
    else
        throw std::length_error(
                "zip called with lists of different lengths" );
}
```

Uppgift 3 (8+2 p)

a) (8 p)

```
template <class T>
List<T> fringe( const TreeNode<T> *aTree ) {
    if ( aTree->isLeaf() ) {
        List<T> aLeaf;
        ListItr<T> itr(aLeaf);
        itr.Insert( aTree->data );
        return aLeaf;
    } else
        return append( fringe( aTree->left ),
                       fringe( aTree->right ) );
}
```

b) (2 p)

```
template <class T>
bool sameFringe( const TreeNode<T> *t1, const TreeNode<T> *t2 )
{
    return fringe( t1 ) == fringe( t2 );
}
```

Uppgift 4 (3+3 p)

Kursbokens implementering av operationen Insert ger resultatet till höger för kvadratisk sondering. Linjär sondering efter samma princip ger resultatet till vänster. Observera att ett struket element (39) kan skrivas över igen med *samma* element, men för ett annat (t.ex. 12) som skulle kunna ligga på samma position väljs en tom cell.

a) (3 p)

	linj.
0	39
1	64
2	41
3	12
4	
5	
6	
7	
8	60
9	
10	
11	
12	77

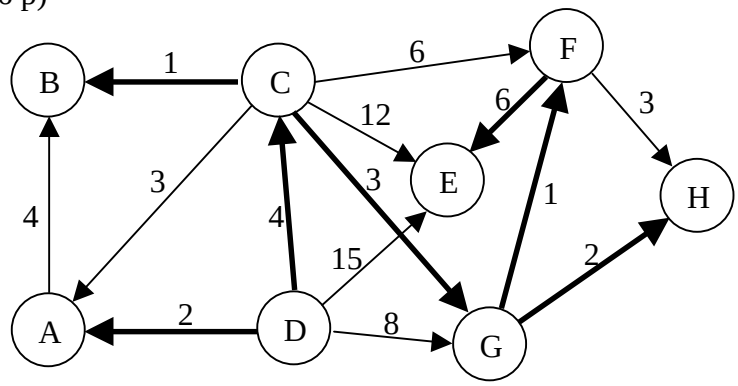
	kvad.
0	39
1	
2	41
3	64
4	
5	
6	
7	
8	60
9	
10	
11	12
12	77

b) (3 p)

De tre första elementen kan sättas in i tabellen, men ej det fjärde trots att $\lambda \leq 0.5$. Orsaken är att tabellstorleken ej är ett primtal.

Uppgift 5 (6 p)

a) (2 p)



b) (2 p)

Kortaste vägarna från D till övriga noder, samt kostnaden för resp väg inom parentes är

	A	**B**	**C**	**E**	**F**	**G**	**H**
*** oviktad**	DA (1)	DAB (2)	DC (1)	DE (1)	DGF (2)	DG (1)	DGH (2)
viktad	DA (2)	DCB (5)	DC (4)	DCGFE (14)	DCGF (8)	DCG (7)	DCGH (9)

*) I vissa fall finns alternativa vägar av samma längd.

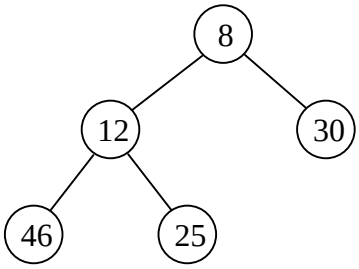
c) (2 p)

En topologisk ordning av noderna är D C A G B F E H, en annan D C G F H A E B.

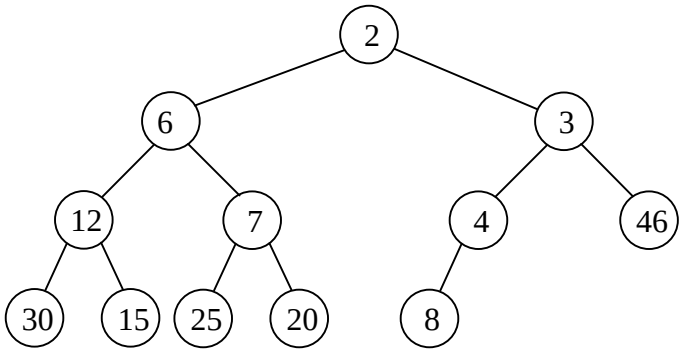
De möjliga ordningarna ges av $DC\begin{cases} AB \\ GF\begin{cases} E \\ H \end{cases} \end{cases}$ där $\begin{matrix} X \\ Y \end{matrix}$ betyder att ordningen av noderna i X i utdata är oberoende av ordningen av noderna i Y i utdata, A kan t.ex. förekomma var som helst i förhållande till G, F, E och H, men kommer alltid efter C och före B.

Uppgift 6 (p)

a) (3 p) *H* efter tre anrop av DeleteMin:



b) (3 p) *H* efter insättning av talen 15, 20, 6, 2 med Insert:



Uppgift 7 (12 p)

```
class Konto {
public:
    Konto( int startSaldo,  // kontots startsaldo
           int period,      // löpande period i timmar
           int maxUtt,      // max antal uttag per period
           int maxBel );    // max uttag under löpande period
    int saldo() const { return _saldo; }
    int transaktion( int belopp );
private:
    const time_t period;
    const int maxUttag;
    const int maxBelopp;
    int _saldo, ackumuleradeUttag, antalUttag;
    Queue<Uttag> uttagsLogg;
    void uppdateraLogg();
};
```

```
Konto::Konto( int startSaldo, int per, int maxUtt, int maxBel )
    :  period(per*3600), maxUttag(maxUtt), maxBelopp(maxBel),
       _saldo(startSaldo), antalUttag(0), ackumuleradeUttag(0)
{ }

void Konto::uppdateraLogg() {
    // Tag bort alla uttag äldre än period
    while ( ! uttagsLogg.IsEmpty() &&
                time(0) - uttagsLogg.Front().tidpunkt > period )
    {
        ackumuleradeUttag -= uttagsLogg.Front().belopp;
        antalUttag--;
        uttagsLogg.Dequeue();
    }
}

int Konto::transaktion( int belopp ) {
    // insättning
    if ( belopp >= 0 ) {
        saldo += belopp;
        return belopp;
    }
    // uttag
    belopp = -belopp;
    uppdateraLogg();
    // Uttag ok?
    if ( antalUttag >= maxUttag ||
            ackumuleradeUttag + belopp > maxBelopp )
        return 0;
    else { // Uttag ok!
        // Registrera och tidsstämpla uttaget
        ackumuleradeUttag += belopp;
        uttagsLogg.Enqueue( Uttag( belopp ) );
        antalUttag++;
        saldo -= belopp;
        // småuttagsavgift
        if ( belopp < 300 )
            _saldo -= 10;
        // övertrasseringsavgift
        if ( _saldo < 0 )
            _saldo -= 100;
        return -belopp;
    }
}
```