Høgskolen i Gjøvik

Avdeling for elektro- og allmennfag

E K S A M E N

FAGNAVN:	**Algoritmiske metoder**	
FAGNUMMER:	**LO 164A**	
EKSAMENSDATO:	**12. desember 1995**	**TID: 09.00-14.00**
FAGLÆRER:	**Frode Haug**	**KLASSE: 2 AA / AE**
ANTALL SIDER UTLEVERT:	**5 (inkludert denne forside)**	
TILLATTE HJELPEMIDLER:	**Alle trykte og skrevne**	

NB: - Kontroller at alle oppgavearkene er tilstede.

- LES HELE OPPGAVETEKSTEN NØYE, FØR DU BEGYNNER Å BESVARE NOE SOM HELST.

- DET ER INGEN SAMMENHENG MELLOM DE ULIKE DELENE I OPPGAVENE 2. DERMED KAN ALLE UNDERPUNKTER LØSES TOTALT UAVHENGIG.

- Ikke skriv noe av din besvarelse på oppgavearkene.

- Kladd og oppgavearkene leveres sammen med besvarelsen. Kladd merkes med "KLADD".

- Husk kandidatnummer på alle ark (IKKE oppgavearkene).

Oppgave 1 (teori, 10 %)

Følgende programkode er gitt:

```
#include <iostream>
using namespace std;


int g(int n, int a, int b)  {
   if ( n == 0)return a;
   else        return g(n-1, b, a+b);
}

int f(int n)  {  return g(n, 0, 1);  }

void main()  {
   for (int i = 0;  i <= 5; i++)
cout << '\n' << f(i);
}
```

Hva blir utskriften fra programmet ? Dvs. hva returnerer f(0), f(1), f(2), f(3), f(4) og f(5) ?

(Legg merke til at alt arbeidet blir utført av den rekursive funksjonen "g", og at "f" eksisterer kun for å sette initielle verdier for parametrene 'a' og 'b'.)

Oppgave 2 (teori, 25 %)

Denne oppgaven inneholder fem totalt uavhengige oppgaver, som er basert på ulike deler av pensum.

a)

Følgende prioritetskø, organisert som en heap, er gitt:

97 84 56 22 72 42 8 17 14 7 12 37

Utfør etter tur følgende operasjoner på denne heap: insert(63), insert(40), remove() og replace(4). **Tegn opp heapen etter at hver av operasjonene er utført.**

NB: For hver av operasjonene skal du operere videre på den heapen som ble resultatet av den forrige operasjonen.

b)

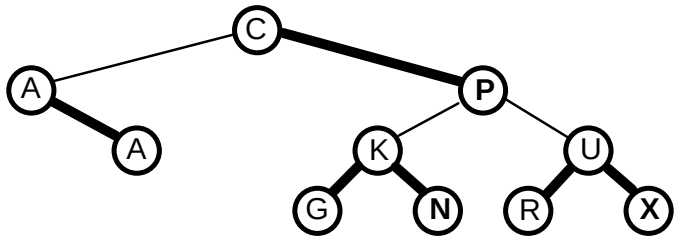
Koden s.237 og teksten s.239 i læreboka gir oss følgende kode ved dobbel hashing:

```
const int M = 7;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (4 - (k % 4));  }

void insert(itemType v, infoType info)  {
    int x = hash1(v);
    int u = hash2(v);
   while (a[x].info != infoNIL)  x = (x+u) % M;
   a[x].key = v;  a[x].info = info;
}
```

Vi har en array som er 7 lang (indeks 0-6). Keyene 71, 45, 15, 29 og 87 skal legges inn i denne arrayen. **Skriv opp hver enkelt key's returverdi fra både hash1 og hash2. Tegn også opp arrayen for hver gang en ny key legges inn.** ("Info"s innehold trenger du ikke å ta hensyn til.) **Beskriv også hva som skjer, dersom enten "hash1" eller "hash2" ved en feil alltid returnerer 0.**

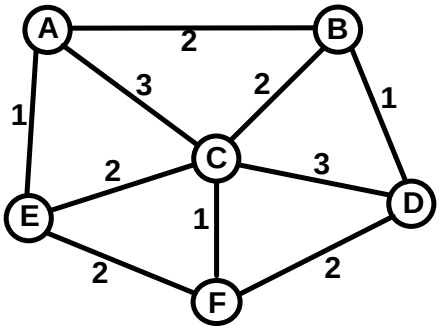
c) Følgende Red (tykke streker)-Black (tynne streker) tre er gitt:



Tegn opp resultatet når bokstavene "B", "L" og "T" legges inn i treet ovenfor.

NB: For hver gang (bokstav) skal du på nytt ta utgangspunkt i treet ovenfor. Dvs. bokstavene "B" "L" og "T" skal ikke til slutt befinne seg i det samme treet.

d) Følgende vektede graf er gitt:



Hver nodes naboer er representert i en naboliste. Alle listene er sortert alfabetisk.
Tegn opp minimums spenntreet for denne grafen, etter at koden s.455 i læreboka er utført. (Der "priority" er lik "t->w".)
Tegn også opp innholdet i prioritetskøen etterhvert som konstruksjonen av minimums spenntreet pågår (jfr. fig.31.4 i læreboka). NB: Husk at ved lik prioritet så vil noden sist oppdatert (nyinnlagt eller endret) havne først i køen.

e) Ta utgangspunkt i den samme grafen, og de samme nabolistene som i oppgave **d)**. Dersom vi utfører koden s.455, bare at "priority" nå er lik "val[k] + t->w" (jfr. skrivefeilen s.463 i læreboka), **hvilke kanter vil være involvert i korteste-sti spenntreet fra "D" og til alle de andre nodene ?**

Oppgave 3 (koding, 30 %)

En graf består av noder med følgende innhold:

```
const int MAX_NODER = 101;
const int MAX_NABO  =  11;

struct node  {
  int   navn;                //  Nodens navn.
  int   ant_kant;            //  Antall naboer/kanter
                                   ut fra node.
  node* neste[MAX_NABO];     //  Pekere til naboene.
  node* kopi;                //  Peker til evnt.
                                  duplikat av en selv.
  node(int id, int ant)
     {  navn = id;   ant_kant = ant;   kopi = NULL; }
};
```

I hele denne oppgaven skal vi se på to ulike funksjoner som hjelper oss med å lage en kopi av denne grafen. Dvs. den opprinnelige grafens "kopi" skal etterpå peke til en node som er en kopi av den selv, der "navn" og "ant_kant" er identiske, "neste[k]" i kopien peker til "neste[k]" sin kopi. Pekeren "kopi" i kopien skal fortsatt være NULL.

Hvordan den opprinnelige grafen (G) er blitt opprettet trenger du ikke å bekymre deg med.

a) Antall noder i G er gitt ved den globale variabelen: `int V;`
Alle nodene i G er tilgjengelig gjennom: `node* noder[MAX_NODER];`
(Der altså indeksene nr.1-"V" av "noder" er i bruk.)

Skriv en ikke-rekursiv funksjon "void kopier_1()" som lager en kopi-graf (G') av G som beskrevet ovenfor. Funksjonen bør gå gjennom "noder" to ganger. Først for å opprette en kopi av hver enkelt node, og deretter for å opprette pekere til naboene i G' på lik linje med de i G.

b) Du har nå ingen hjelpearray ("noder") eller informasjon om antall noder ("V") tilgjengelig. Vi antar i stedet at hele grafen G kan nås gjennom: `node* S;`

Skriv en rekursiv funksjon "void kopier_2(node* x)" som kopierer hele grafen som beskrevet ovenfor. (Startkallet til denne blir: "kopier_2(S);")

Hint: Bruk pekeren "kopi" til å sjekke om en node allerede er "SEEN/UNSEEN".

Oppgave 4 (koding, 35 %)

En labyrint består av ulike veier man kan følge for om mulig å komme til målet. Noen veier fører i retning av målet, noen går i sirkel, og noen er rene blindveier (stopper bare). Et eksempel er tegnet nedenfor. Man starter i "S"(tart) og skal lete seg fram til man finner "M"(ål). **Din oppgave er å skrive en rekursiv funksjon "bool los_labyrint(int x, int y)" som (om mulig) finner en løsning på en slik problemstilling.**

Du kan anta at hele labyrinten er innlest i en to-dimensjonal array:

```
char labyrint[75][20];
```

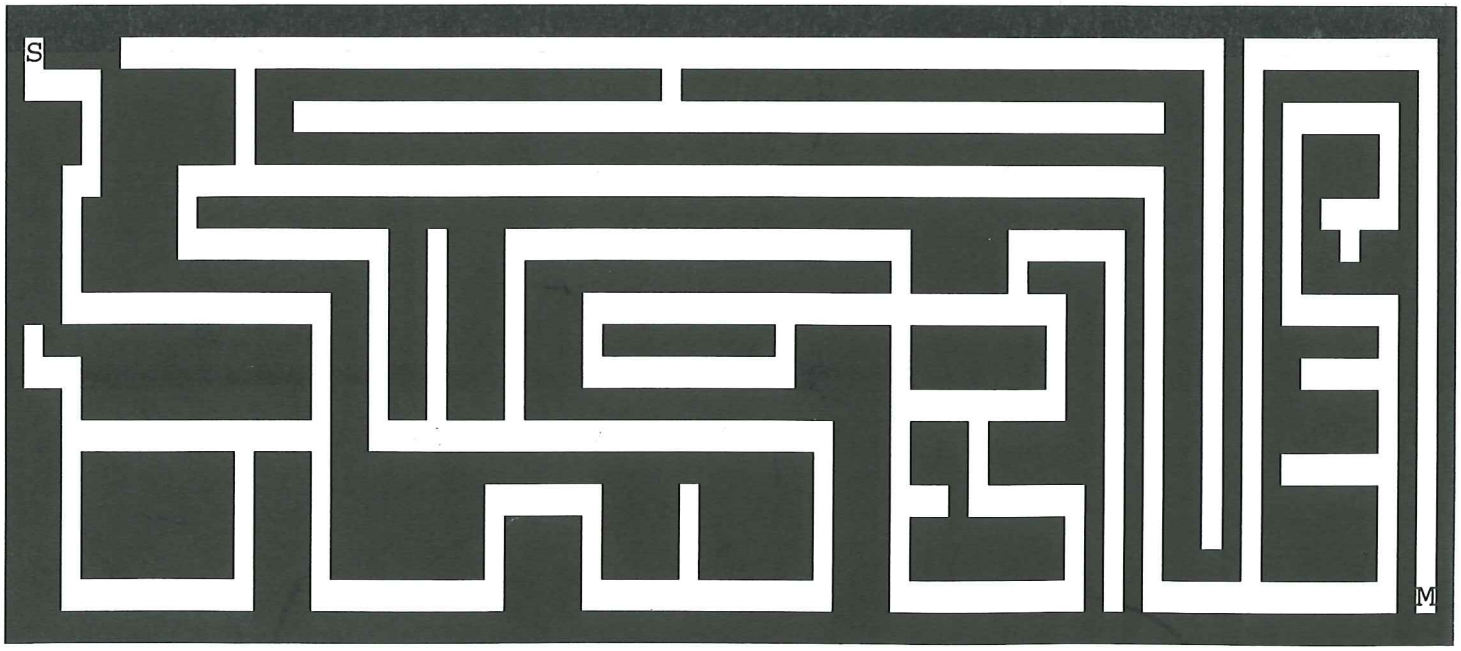
Hver "skuff" i denne arrayen inneholder til enhver tid ett av følgende fire tegn:

- 'M' - for mål
- '<blank>' - vei, ledig
- '•' - vei, opptatt
- '■' - utenfor veien (umulig å bevege seg her)

Bokstaven 'S' byttes ut med en '<blank>' etter at dens posisjon i arrayen er avlest.
Disse koordinatene brukes som input til "los_labyrint(.....)" ved det første kallet fra "main".

Hint:
- Når man står i en rute, så kan man kun bevege seg i de retninger som det er en '<blank>'. Bevegelsene kan ikke være diagonale, kun vannrett og loddrett.
- I det man går i en retning, så bør man merke den ruten man forlater med en '•'. Dermed vil man, når man møter på '•', ha funnet en sirkel, og man kan trekke seg minst ett hakk tilbake i rekursjonen.

NB:
- Legg merke til at rundt hele labyrinten er det en svart kant. Dermed vil du aldri havne utenfor brettet, dersom du stadig sjekker mot '■'er.
- Du trenger heller ikke å lage noen utskrift etterhvert som den rekursive funksjonen prøver ulike veier. Arrayen "labyrint" vil jo tilslutt inneholde løsningen, der løsningsveien er merket med '•'er. Denne kan så skrives ut av en annen funksjon (som du altså ikke skal lage).



28