Høgskolen i Gjøvik

Avdeling for elektro- og allmennfag

EKSAMEN

FAGNAVN: **Algoritmiske metoder I**

FAGNUMMER: **L 171 A**

EKSAMENSDATO: **17. desember 1998**

KLASSE: **97HINDA / 97HINDB (2DA / 2DB)**

TID: **09.00-14.00**

FAGLÆRER: **Frode Haug**

ANT. SIDER UTLEVERT: **5 (inkludert denne forside)**

TILLATTE HJELPEMIDLER: **Alle trykte og skrevne.**

- Kontroller at alle oppgavearkene er tilstede.

- INNFØRING MED PENN, evnt. trykkblyant som gir gjennomslag.
  Pass på at du ikke skriver på mer enn ett innføringsark om gangen
  (da det blir uleselige gjennomslag om flere ark ligger oppå hverandre når du skriver).

- Ved innlevering skilles hvit og gul besvarelse og legges i hvert sitt omslag.
  Oppgavetekst, kladd og blå kopi beholder kandidaten.

- Ikke skriv noe av din besvarelse på oppgavearkene.

- Husk kandidatnummer på alle ark (ikke oppgavearkene).

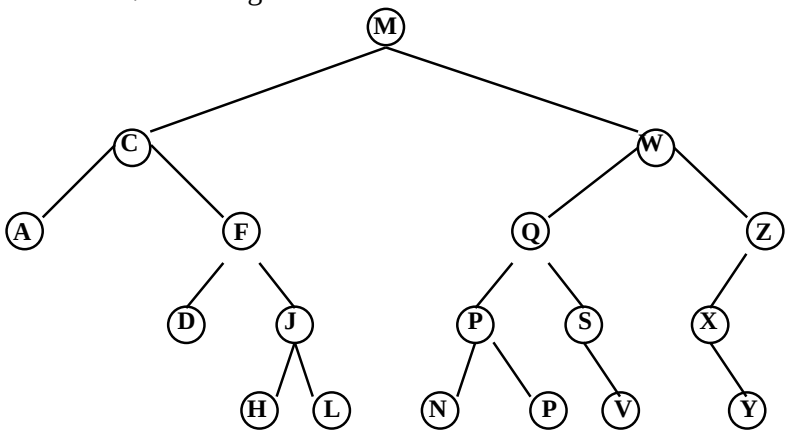
NB: Alle oppgavene teller likt, dvs. 25% vekt på hver av dem !

Oppgave 1 (teori)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 9, 14 og 16 i læreboka.

a) Du skal utføre Quicksort på teksten «PØLSELIM». **Lag en figur tilsvarende fig. 9.3 s.119 i læreboka**, der du for hver rekursive sortering skriver de involverte bokstavene og markerer/ uthever hva som er partisjonselementet.

b) Følgende binære søketre er gitt :



Du skal nå fjerne («remove») noen noder fra dette treet. **Tegn opp treet for hver gang og fortell hvilken av «if ..... else if ..... else»-grenene i koden s.210 som er aktuelle når du etter tur fjerner henholdsvis tegnene 'Z', 'Q' og 'W'.**

NB: Du skal for hver fjerning på nytt ta utgangspunkt i treet tegnet ovenfor. Dvs. på intet tidspunkt skal du, fra treet, ha fjernet to eller tre av de ovenfor skrevne bokstavene samtidig.

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

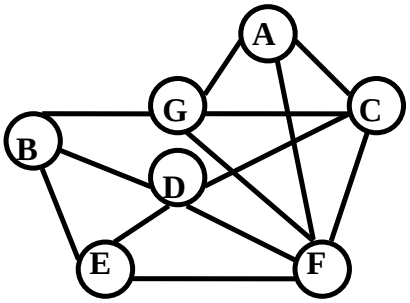
```
const int M = 9;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (4 - (k % 4));  }

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;
}
```

«k» står for bokstavens nummer i alfabetet (1-29). Vi har en array som er 9 lang (indeks 0-8). Keyene «PØLSELIM» 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 innhold trenger du ikke å ta hensyn til.)

Oppgave 2 (teori)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 29, 30 og 31 i læreboka.

a) Vi har grafen:



a1) **Skriv opp nabomatrisen** for dette tilfellet.

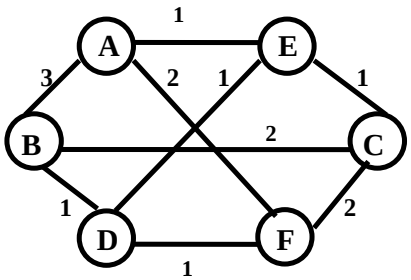
a2) **Tegn dybde-først søketreet** for dette tilfellet (dvs. ved bruk av nabomatrise), når "Search" (s.424) og "Visit" (s.427) fungerer som angitt i læreboka.

b) Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:

AF AC ED BD BA CE BE BF FC AD

Vi skal nå utføre union-find m/path compression og weight balancing på denne grafen. **Tegn opp arrayen «dad»s innhold etterhvert** som skogen bygges opp (jfr. fig.30.9) vha. «find»-funksjonen s.447 i læreboka. Ut fra dette: **tegn også opp den resulterende union-find skogen** (dvs. noe lignende til nedre høyre skog i fig.30.8).

c) Følgende vektede (ikke-rettede) 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.

Oppgave 3 (koding)

Anta at vi har en rettet graf G. Dersom G ikke har løkker/sykler, vet vi at vi kan finne en topologisk sortering av nodene ved følgende algoritme: Velg noder en etter en, men hele tiden slik at alle forgjengere (om det er noen) til den som velges allerede er valgt.

Anta nå videre at grafen har N noder, og at den er gitt ved følgende datastruktur
(les kommentarene nøye):

```
struct Node  {
   Kant* forste;   //  Den første i en liste av utgående kanter.
   int   ant;      //  Hjelpe-variabel.
   char  ID;       //  Nodens id/navn (en bokstav).
};                 //  NB: Denne trenger ikke du å bruke. Den brukes
                   //      kun til utskrift, for å kunne identifisere noden.

struct Kant  {
   Kant* neste;    //  Neste utgående kant. Er NULL om selv er siste kanten.
   Node* noden;    //  Noden som denne kanten går til.
};
```

a) La oss også anta at vi har en array:

```
Node* G[N+1];     //    Bruker indeks nr.1-N som peker til hver
                  //          av de N nodene.
```

Fra denne har vi altså en direkte peker til alle nodene i grafen.

Anta at variabelen «ant» i de forskjellige nodene har helt tilfeldige (søppel)verdier. **Skriv en programbit som i hver node setter variabelen «ant» lik antall innkommende kanter (dvs. antall forgjengere) til noden.**

NB: Hvordan grafen er bygd opp vha. noder, kantelister og G trenger du ikke å bekymre deg med i denne oppgaven. Forutsett bare at dette er korrekt utført allerede.

b) Nå antar vi at vi ikke lenger har arrayen G. I stedet vet vi at grafen har en node «S». Ved å starte i denne noden kan vi følge stier som gjør at vi når alle de andre nodene i grafen. Denne noden angis ved variabelen «Node* S».

Skriv en rekursiv funksjon: `void settant(Node* rn);` **som, når den kalles med «settant(S)», gjør et dybde-først-søk gjennom hele grafen, og setter antall forgjengere til hver node inn i variabelen «ant».** Funksjonen skal ikke bruke andre variable enn «ant» i hver node.

NB: Du kan her gå ut fra at variabelen «ant» initielt er «0» i alle noder.

Oppgave 4 (koding)

a)

(18%)

Dette er en nesten kynisk og lett mobbende oppgave. Men, alt er tillatt i Alg.met.I !
Ja, det er ikke det spørsmål vi ikke kan få stille oss ! Men, legg merke til at det her ihvertfall ikke er noen diskriminering eller mannssjåvinisme.

Det skal være fest i klasse 1A på Storhamar (!!!) videregående skole. I denne klassen går det like mange gutter som jenter. Dvs. det er N jenter og N gutter. Alle de N jentene er spurt om i hvilken grad de liker hver av de N guttene. Deres 'gradering' er et tall mellom 1 og 10, der 10 er høyest. En jente kan gjerne gi flere gutter den samme graderingen. Alle jentenes svar er lagret i arrayen «jente[N+1][N+1]». (Vi bruker ikke element nr.0 i noen av retningene.) I element «jente[i][j]» ligger altså tallet som angir hvor mye jente nr.'i' liker gutt nr.'j'. Tilsvarende er guttene spurt om, og deres svar er på samme måten lagret i arrayen «gutt[N+1][N+1]».

På festen skal det settes sammen par (bordplassering og fast dansepartner) på grunnlag av de innsamlede dataene ovenfor. **Din oppgave er i skrive et komplett program som beregner den parkombinasjonen av jenter og gutter, som gir at TOTALSUMMEN for hele klassen av parenes gradering av hverandre er MAKSIMAL.**

NB: Husk det holder ikke å kun sjekke hvor mye jente nr.'i' liker gutt nr.'j'. Det må også tas hensyn til det motsatte, altså hvor mye gutt nr.'j' liker jente nr.'i'.

Hint: Du må sikkert lage deg noen hjelpevariable/-arrayer for å kunne løse denne oppgaven.

b)

(7%)

På et hotell er det N rom med til sammen M senger. Det kommer N familier, med til sammen K mennesker, som skal forlegges på hotellet. Hvert menneske skal ha en egen seng å sove i. Dette medfører altså at alle rommene blir i bruk, men siden M er godt større enn K, så vil dette totalt sett gå greit.

Hvert rom inneholder et ulikt antall senger, og familiene har også ulikt antall medlemmer. Men, totalt sett finnes det store nok rom til alle familiene. Dvs. man trenger hverken å splitte familiene eller å flytte om på senger mellom rommene.

Vi ønsker at alle familiene skal oppleve en mest mulig bekvemmelighet. I dette tilfellet går dette på at hver familie skal plasseres slik at de får et størst mulig rom ift. hvor mange de er i familien.
Eksempel: I stedet for å plassere en 4-manns familie på et 4-manns rom (som det selvsagt er plass til), og en 2-manns familie på et 6-manns rom, så er det mer bekvemmelig (ihvertfall for 4-manns familien) at disse to bytter rom.

Skriv detaljert pseudokode for algoritmen som løser problemet med at hver av de N familiene skal få det mest mulig bekvemmelig.

NB: Husk at enhver familie kan ikke plasseres på ethvert rom (eks: 6-manns familie får jo ikke plass på et 4-manns rom.) Men totalt sett så finnes det rom som er store nok til å huse alle familiene.

Algoritmisk lykke til, og god jul !
FrodeH