Høgskolen i Gjøvik

Avdeling for Teknologi

KONTINUASJONSEKSAMEN

FAGNAVN: **Algoritmiske metoder I**

FAGNUMMER: **L 189 A**

EKSAMENSDATO: **13. august 2001**

KLASSE: **99HINDA / 99HINDB / 99HINEA / 00HDESY (2DA / 2DB / 2EA / DESY)**

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, evt. 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.

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 6 & 7, 11 og 16 i læreboka.

a) (5 %) En funksjon som beregner og returnerer svaret på -1^n kan skrives på følgende måte (Vi forutsetter at n er større eller lik 0. -1^0 er definert som 1, hvilket funksjonen ivaretar.):

```
int minus_en_i(int n)  {
  int svar = 1;
  for (int i = 1;  i <= n;  i++)
      svar *= -1;
  return svar;
}
```

Hvilken orden er denne funksjonen av ?
Funksjonens innmat kan også skrives slik at den faktisk blir av O(1) ! Gjør dette.

b) (10 %) Teksten "T O R S K E T U N G E" (i denne rekkefølge fra venstre til høyre, blanke regnes ikke med) skal heap-sorteres vha. bottom-up heap konstruksjon.
(Se fig.11.8, koden s.156 og fig.11.9 i læreboka.)
Tegn opp heapens innhold etterhvert som heapen konstrueres og deretter sorteres etter denne metoden (Dvs. lag en figur etter samme prinsipp som fig.11.9.)

c) (10 %) Koden s.237 og teksten s.239 i læreboka gir oss følgende kode ved dobbel hashing:

```
const int M = 11;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (5 - (k % 5));  }

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 11 lang (indeks 0-10). Keyene "TORSKETUNGE" 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.** (Innholdet i "info" trenger du ikke å ta hensyn til.)

Oppgave 2 (teori, 25%)

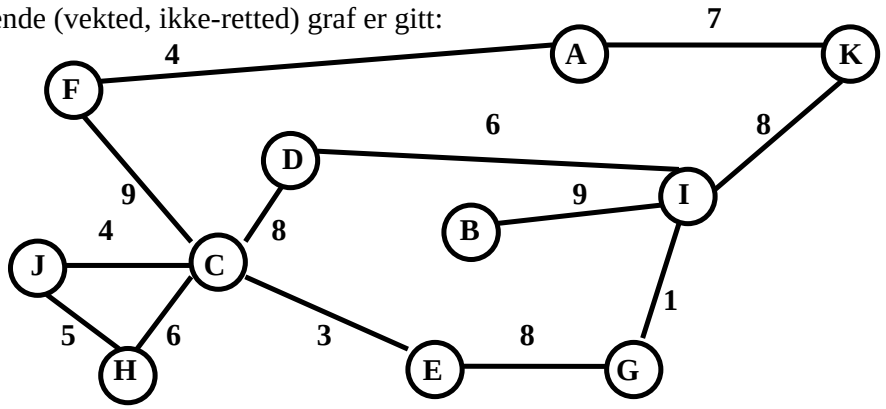
Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 30, 29 & 31 i lærebok og om FSM.

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

BC AE DB BF EC

Vi skal nå **utføre union-find m/path compression** (men *ikke* weight balancing) på denne grafen. Det er den første bokstaven som skal legges inn under den siste bokstaven i hver kantangivelse. Dvs. den *eneste* innmaten i if-setningen `if (doit && (i != j))` s.447 i læreboka blir: `{ dad[j] += dad[i]-1; dad[i] = j; }` **Tegn opp innholdet i arrayen "dad" 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).

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



1. Skriv en lovlig rekkefølge av nodene ut fra et Dybde-Først-Søk (DFS) som starter i node F.
2. Skriv en lovlig rekkefølge av nodene ut fra et Bredde-Først-Søk (BFS) som starter i node F.
3. Gi en kort forklaring på hvorfor kanten EG *kan* (men ikke er nødt til å) høre med i grafens minimums spenntre.
4. Tegn opp et minimums spenntre for grafen (du trenger ikke å angi "fringen" underveis).

c) Følgende deterministiske endelige tilstandsmaskin (FSM) er gitt:

$M = (\{ q_0, q_1, q_2, q_3 \}, \{ a, b \}, \delta, q_0, \{ q_0, q_1, q_2 \})$

Transisjonstabellen:

$\delta(q_0, a) = q_0$	$\delta(q_0, b) = q_1$
$\delta(q_1, a) = q_0$	$\delta(q_1, b) = q_2$
$\delta(q_2, a) = q_0$	$\delta(q_2, b) = q_3$
$\delta(q_3, a) = q_3$	$\delta(q_3, b) = q_3$

Tegn tilstandsmaskinen.
Er det mulig, på en deterministisk måte, å tegne FSM'en enklere ? I så fall: gjør det.
Hva slags setninger/språk godtar den ?

Oppgave 3 (koding, 30%)

Denne oppgaven omhandler binære trær, der hver node i treet er definert på følgende måte:

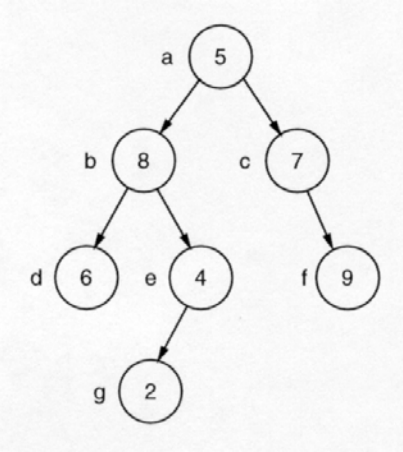
```
struct node  {
    int id;             //  Nodens ID/navn.
    int avstand;        //  Antall mellomliggende kanter til nærmeste åpning.
    node *left, *right; //  Pekere til venstre og høyre subtre/barn.
    node *sti;          //  Retning (node) mot nærmeste åpning.
};
```

Nodene i treet ligger *ikke* sortert etter noe spesielt kriterie, dvs. treet er *ikke* noe binært søketre.

Når nye noder skal settes inn i treet, er vi interessert i at disse plasseres så nære roten som mulig. Vi skal derfor se på en teknikk som gjør at vi fra hver node lett kan finne *nærmeste* ledige plass i det subtreet som noden er rot i.

Vi skal i det videre kalle noder som har ett eller ingen barn (dvs. at noden har et tomt venstre og/eller høyre subtre) for **"åpne noder"**. Noder som har to barn/subtrær, kaller vi for **"indre noder"**. For hver indre node, er vi interessert i å finne de åpne nodene i subtrærne som ligger så nære den indre noden som mulig. Disse nodene kaller vi for den indre nodens **"nærmeste åpninger"**.

Eksempel: I figuren under er a og b indre noder. c, d, e, f, og g er åpne noder. a's nærmeste åpninger er c, mens b's nærmeste åpninger er d og e.



Husk at nodenes "id" er et heltall, og at bokstavene a-g bare brukes for å entydig identifisere nodene på figuren. Dette er spesielt aktuelt om flere har lik "id".

Vi er interessert i å raskt kunne gå fra en indre node til en av dens nærmeste åpninger. I hver indre node i treet vil derfor variabelen "sti" angi retningen til en av de nærmeste åpningene, dvs. at den peker til det samme som enten "left" eller "right", avhengig av hvilken vei man må gå for å komme til nærmeste åpne node. Variabelen "avstand" angir antall mellomliggende kanter fra noden til en av de nærmeste åpningene. Dersom det finnes en likeverdig nærmeste åpning i både venstre og høyre subtre, skal "sti" være like "left". For åpne noder er "sti" lik NULL og "avstand" lik 0.

Når det skal settes inn en ny node i treet, skal den plasseres rett under den nærmeste åpningen vi kommer til ved å følge "sti"-pekerne nedover i treet. Om den åpne noden vi da kommer til ikke har noen barn, skal den nye noden settes inn som venstre subtre, ellers skal den settes inn der det er plass. Etter innsettingen må vanligvis variablene "sti" og "avstand" justeres i en del av nodene for at de skal stemme med kravene som er beskrevet ovenfor.

a) Vi skal sette inn en node (h) med ”id” lik 3 i treet som er tegnet i figuren ovenfor.
Tegn hvordan treet ser ut, inklusive variablene ”sti” og ”avstand” i *hver* node, både *før* og *etter* innsetting av en node med ”id” lik 3.

b) Anta at vi har et binært tre, der variablene ”sti” og ”avstand” allerede er riktig satt.
Lag funksjonen `void skriv_indre_og_aapne(node* rot)` **som først skriver ut id’ene til *alle* indre noder i treet, og *deretter* skriver ut id’ene til *alle* åpne noder.**
Hint: Funksjonen bør nok benytte seg av rekursive funksjoner som utfører de to oppgavene.

c) Anta at vi har et binært tre, der variablene ”sti” og ”avstand” i hver node *ikke* er satt riktig etter kravene som er beskrevet ovenfor. **Lag funksjonen** `void stifinner(node* rot)` **som rekursivt går gjennom et tre der roten pekes ut av parameteren ”rot”, og sørger for at ”sti” og ”avstand” settes til riktige verdier i alle nodene.**

d) **Lag funksjonen** `void sett_inn(node *rot, int v)` **som lager en ny node med ”id” lik ”v” og setter den inn i treet med rot i ”rot”, på nærmeste ledige plass slik som angitt ovenfor. Funksjonen skal også oppdatere andre noder i treet, slik at variablene ”sti” og ”avstand” er korrekte i alle noder, også etter innsettingen.**

Oppgave 4 (koding, 20%)

Studer følgende mønster av stjerner og innledende blanke:

```
*
* *
  *
* * * *
    *
    * *
      *
* * * * * * * *
        *
        * *
          *
        * * * *
            *
            * *
              *
```

Skriv et komplett program, bl.a. inneholdende en rekursiv funksjon, som genererer *eksakt* dette mønsteret.
NB 1: Legg merke til de innledende blanke på de fleste av linjene.
NB 2: De tilsynelatende blanke tegnene mellom stjernene (på hver linje) teller ikke med.
NB 3: Den rekursive funksjonen, som foretar utskriften, får *kun* lov til å skrive innholdet på *en* stjerne-linje!

Lykke til !

frode@haug.com