Høgskolen i Gjøvik
Avdeling for teknologi

E K S A M E N

FAGNAVN: **Algoritmiske metoder**

FAGNUMMER: **IMT2021**

EKSAMENSDATO: **3. desember 2004**

KLASSE(R): **03HINDA / 03HBINFA / 03HBMETEA / div. andre**

TID: **09.00-14.00**

FAGLÆRER: **Frode Haug**

ANTALL SIDER UTLEVERT: **4 (inkludert denne forside)**

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

- Kontroller at alle oppgavearkene er tilstede.
- Innføring med penn, eventuelt trykkblyant som gir gjennomslag. Pass på så du ikke skriver på mer enn ett innføringsark om gangen (da det blir uleselige gjennomslag når flere ark ligger oppå hverandre).
- Ved innlevering skilles hvit og gul besvarelse, som legges i hvert sitt omslag.
- Oppgavetekst, kladd og blå kopi beholder kandidaten til klagefristen er over.
- Ikke skriv noe av din besvarelse på oppgavearkene. Men, i oppgavetekst der du skal fylle ut svar i tegning/tabell/kurve, skal selvsagt dette innleveres sammen med hvit besvarelse.
- Husk kandidatnummer på alle ark. Ta vare på dette nummeret til sensuren faller.

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 8, 12 og 15 i læreboka.

a) Vi skal utføre Shellsort på key'ene "SUMPSVAMP". Jfr. kode s.109 i læreboka.
For hver gang indre for-løkke er ferdig (etter: `a[j] = v;`):
Tegn opp arrayen og skriv verdiene til 'h' (4 og 1) og 'i' underveis i sorteringen.
Marker spesielt de key'ene som har vært involvert i sorteringen (jfr. fig.8.7 s.108).

b) Vi skal utføre rekursiv Mergesort på key'ene "SUMPSVAMP". Jfr. kode s.166 i læreboka.
For hver gang tredje og siste for-løkke i koden s.166 er ferdig:
Tegn opp arrayen med de key'ene som har vært involvert i sorteringen (jfr. fig.12.1 s.167).

c) Legg key'ene "SUMPSVAMPENES" (i denne rekkefølge fra venstre til høyre) inn i et 2-3-4 tre.
Tegn opp treet etterhvert som bokstavene legges inn.
Gjør også om sluttresultatet til et Red-Black tre.

Oppgave 2 (teori, 25%)

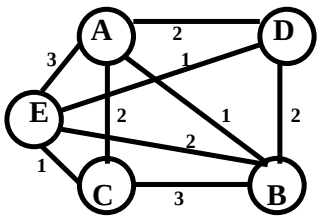
Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 30 & 31 i læreboka og om FSM.

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

AC BF DE BG EF GE FD AD CE EB

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).

b) 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.

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

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

Transisjonstabellen:

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

Tegn tilstandsmaskinen.
Hva slags setninger/språk godtar den ?

Oppgave 3 (koding, 36%)

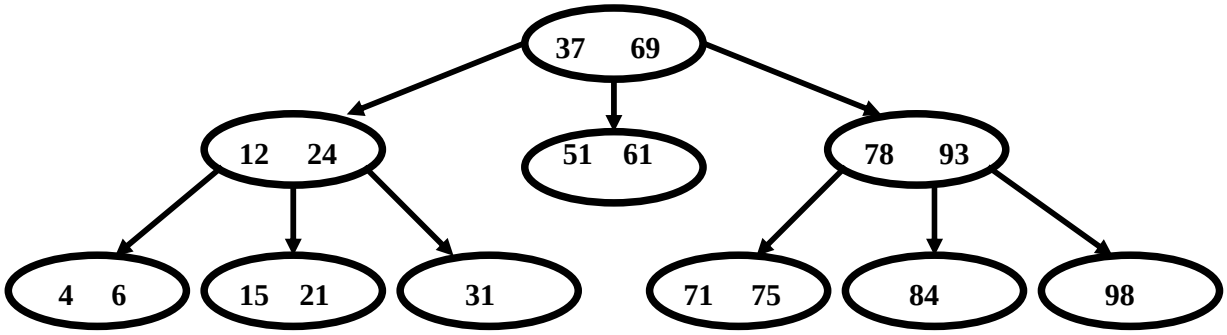
Vi skal i denne oppgaven se på *trinære* trær, dvs. trær som har opptil to key'er i og tre subtrær til hver node. Til dette skal vi bruke node-definisjonen:

```
struct node  {
      int key1, key2;           //  Verdiene for key1 og key2.
      bool key2filled;          //  = 'true' om key2 har en reell verdi.
      node *lsub, *msub, *rsub; //  De evt. opptil tre subtrærne.
      node(int v)               //  "Constructor":
      {  key1 = v;  key2filled = false;  lsub = msub = rsub = z;  }
};
```

og de globale variablene:

```
node* z = NULL;
node* root = z;
```

Om en node har to verdier, så er det *alltid* slik at 'key1' < 'key2'. 'lsub' peker evt. til et tre der *alle* verdiene i treet er < 'key1'. 'msub' peker evt. til et tre der *alle* verdiene i treet er > 'key1' men < 'key2'. Og tilsvarende: alle verdiene i treet evt. tilpekt av 'rsub' er > 'key2'. Dvs. treet inneholder ikke like verdier (duplikater).
Alle *interne* noder (dvs. noder med *minst* ett ikke-tomt subtre) har *alltid* to verdier. *Bladnodene* (noder med *bare* tomme subtrær) kan ha enten en eller to verdier ('key1' har *alltid* verdi, og 'key2' *kan* ha verdi eller ikke være i bruk. Dette siste kan sjekkes via 'key2filled'). Eksempel på et slikt tre kan være:



a) **Lag funksjonen** `void set_key2(node* t, int v)`

Denne funksjonen skal gjøre *alt* som er nødvendig for å sette den andre verdien 'key2' i *bladnoden* tilpekt av 't'. **NB:** 'v' *kan* også være mindre enn den nåværende i 'key1'!

b) **Lag funksjonen** `void display_sorted(node* t)`

Funksjonen *skal* rekursivt traversere hele treet og skrive ut *alle* verdiene i *sortert stigende rekkefølge*. **NB:** Husk at i enkelte (blad)noder *kan* det være at 'key2' *ikke* er i bruk!

c) **Lag funksjonen** `node* find(node* t, int v)`

I treet tilpekt av 't' så leter/søker funksjonen etter og returnerer en peker til noden der enten 'key1' eller 'key2' har verdien 'v'. Om denne *ikke* er å finne, returneres en peker til 'z'-noden. Funksjonen kan valgfritt kodes iterativt eller rekursivt.

d) **Lag funksjonen** `bool insert(node* t, int v)`

Funksjonen legger verdien 'v' inn i treet tilpekt av 't'. Om dette lykkes, så returneres 'true'. Ved forsøk på innleggelse av en allerede eksisterende verdi (duplikat), så returneres 'false'. Nye verdier vil *alltid* legges inn nederst i treet. Dette gjøres da enten i en bladnode som har plass ('key2' ikke er i bruk), eller som en ny bladnode som et passende subtre til sin "mor". Funksjonen kan valgfritt kodes iterativt eller rekursivt. Husk også å håndtere at treet kan være tomt ('root' peker til 'z').

Oppgave 4 (algoritmer, 14%)

Vi har en fil med følgende format:

<Filmtittel> <Antall skuespillere> <Navn skuespiller 1> ……. <Navn skuespiller N>

Hver slik linje på filen inneholder opplysninger om navnene til *hoved*skuespillerne i den aktuelle filmen.

Om to skuespillere har spilt sammen i en film, så sier vi at de har en "filmgrad på 1". Om skuespiller C har spilt sammen med skuespiller K, og skuespiller K har spilt sammen med skuespiller T, men C og T har aldri spilt sammen, så sier vi at skuespillerne C og T har en "filmgrad på 2". Og tilsvarende, når det er flere ledd/skuespillere mellom C og T: C har spilt med K, som igjen har spilt med P, som igjen har spilt med R, som igjen har spilt med T, dvs. det er "filmgrad 4" melllom C og T i dette tilfellet.

Din oppgave her blir å bestemme minimums filmgraden mellom skuespillerne X og Y. Du skal faktisk *ikke* skrive noe kode, men via pseudokode og/eller prosatekst besvare følgende spørsmål/ problemstillinger:

- **Beskriv/tegn hensiktsmessig(e) datastruktur(er) for å løse dette problemet.**
- **Hvordan vil du bygge opp denne datastrukturen ved å lese inn fra fila?**
- **Finnes det noen algoritme(r)/"kodesnutte(r)" i læreboka som du, med eller uten små modifikasjoner, kan bruke? Henvis til sidenummeret for *den* / *alle* de aktuelle.**
- **Hvilken av disse algoritmene må *minst* modifiseres/tilpasses ifm. dette problemet?**
- **Hvilken av disse algoritmene vil generelt være *mest effektiv* (finne svaret raskest)?**

Den satt, tenker jeg, og: Lykke til !

frode@haug.com