Høgskolen i Gjøvik
Avdeling for teknologi

KONTINUASJONSEKSAMEN

FAGNAVN: **Algoritmiske metoder I**

FAGKODE: **L 189 A**

EKSAMENSDATO: **15. august 2002**

KLASSE(R): **00HINDA / 00HINDB / 00HINEA**
(2DA / 2DB / 2EA)

TID: **09.00-14.00**

FAGLÆRER: **Frode Haug**

ANTALL SIDER UTLEVERT: **5 (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 (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. 16, 11 og 9 i læreboka.

a) Vi har hash-funksjonen: `hash(k) = k mod 13`
der "k" står for bokstavens nummer i alfabetet (1-29). Vi har også en array med indeksene 0-12. **Tegn opp arrayen hver gang key'ene "D U N D R E K L U M P E N" (i denne rekkefølge fra venstre til høyre, blanke regnes ikke med) legges inn i den vha. linear probing.**

b) Følgende prioritetskø, organisert som en heap, er gitt:

38 29 26 27 28 25 26 23 22 21 26 25

Utfør etter tur følgende operasjoner på denne heap: insert(29), insert(39), remove(), remove() og replace(25). **Skriv 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.

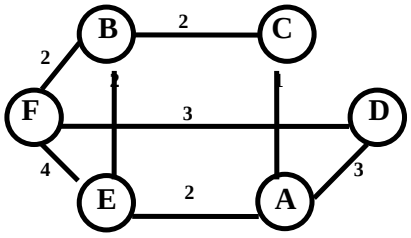
c) Du skal utføre Quicksort på teksten "S O P P K O K E R" (blanke regnes ikke med).
Lag en figur tilsvarende fig. 9.3 side 119 i læreboka, der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 15 & 31 i lærebok og om FSM.

a) Legg teksten "D U N D R E K L U M P E N" (i denne rekkefølge fra venstre til høyre, blanke regnes ikke med) inn i et 2-3-4 tre. **Tegn opp treet etterhvert som bokstavene legges inn. Gjør også om sluttresultatet til et Red-Black tre.**

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 *ikke*-deterministiske endelige tilstandsmaskin (FSM) er gitt:

$M = (\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{a, b\}, \delta, q_0, \{q_0\})$

Transisjonstabellen:

$\delta(q_0, a) = q_1$		
$\delta(q_1, a) = q_4$	$\delta(q_1, b) = q_4$	$\delta(q_1, b) = q_2$
$\delta(q_2, a) = q_3$	$\delta(q_2, b) = q_4$	
$\delta(q_4, b) = q_0$	$\delta(q_4, b) = q_5$	
$\delta(q_5, a) = q_0$		

Tegn tilstandsmaskinen.

Hvilke av følgende fem ulike uttrykk vil den godta:

1) **aba**
2) **abba**
3) **abbba**
4) **abbaab**
5) **aabaabbb**

Oppgave 3 (koding, 20%)

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

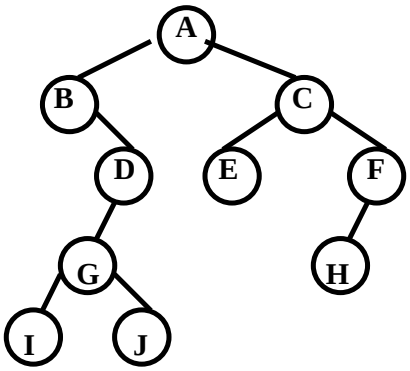
```
struct node  {
    char id;            //  Nodens ID/navn.
    node *left, *right; //  Pekere til venstre og høyre subtre/barn.
    int  rightref;      //  = 0 dersom høyre subtre finnes.
                        //  = 1 dersom tomt høyre subtre.
                        //      'right' peker da til noe annet (se oppg.3b).
};
```

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

En algoritme som besøker hver node to ganger (dobbeltorder) *kan* skrives på følgende måte:

```
void dobbeltorder(node* t)  {
   if (t != z)  {            //  Treet er ikke tomt.
      cout << t->id << "1 ";  //  Besøker/skriver noden for første gang.
      dobbeltorder(t->left);  //  Traverserer venstre subtre dobbeltorder.
      cout << t->id << "2 ";  //  Besøker/skriver noden for andre gang.
      dobbeltorder(t->right); //  Traverserer høyre subtre dobbeltorder.
   }
}
```

Vi har følgende tre:



Starten på utskriften fra en dobbeltorder-traversering av dette treet vil være:

A1 B1 B2 D1 G1 I1 I2 ……

a) **Fullfør utskrifts-sekvensen ved dobbeltorder-traverseringen av treet ovenfor.**
Hva kalles ordningen av nodene du får hvis du fjerner alle utskriftene inneholdende ”2”?
Hva kalles ordningen av nodene du får hvis du fjerner alle utskriftene inneholdende ”1”?

b) **I denne deloppgaven skal du lage funksjonen** `node* neste(node *x, int d)` **som for en gitt node ’x’ og en indeks ’d’ (som *alltid* har verdien 1 eller 2), returnerer med *’x’ besøkt for d’te gang sin etterfølger* i dobbeltorder-sekvensen.** (Anta at ’x’ aldri peker til z.)
Eksempel: Dersom noden er ’D’ og ’d’=1 vil den etterfølgende noden være ’G’.
Dersom noden er ’B’ og ’d’=1 vil den etterfølgende noden være ’B ’ selv.
For noden ’H’ og ’d’=2 vil den etterfølgende noden være ’F’.

Forklar først kort hvorfor det er umulig for en kodebit å sette opp datastrukturen korrekt ved å bruke en representasjon av nodene *uten* ”rightref” inni.

For å realisere dette bruker vi ”right” og ”rightref” på følgende måte:

- Medlemmet ”rightref” har verdien ’1’ for *alle* noder i treet som *ikke* har noe ekte høyre subtre, og verdien ’0’ for *alle andre* noder (som altså *har* et ekte høyre subtre).
- For alle noder som *ikke* har noe ekte høyre subtre, vil deres ”right” peke på den neste noden vi skal gå til etter å ha besøkt noden for *andre* gang i dobbeltorder-traverseringen.
- ”right” peker til z-noden i den aller siste noden i dobbeltorder-traverseringen.

Lag en tegning av treet ovenfor der du også tegner inn hvordan ”right” vil peke i de nodene som har tomt høyre subtre. Sett også et kryss i de nodene der ”rightref” er lik 1.

Vi forutsetter nå at alt dette er korrekt satt i alle nodene. **Programmer ”neste”- funksjonen.**

Oppgave 4 (koding, 30%)

I en rettet graf definerer vi ”til-alle-kjernen” (TA-kjernen) og ”fra-alle-kjernen” (FA-kjernen) som følgende nodemengder:

- En node A er med i *TA-kjernen* til en rettet graf hvis og bare hvis det går rettede veier/stier *fra A til alle andre noder* i grafen.
- En node A er med i *FA-kjernen* til en rettet graf hvis og bare hvis det går rettede veier/stier *til A fra alle andre noder* i grafen.

Hver node er representert på følgende måte:

```
struct node  {
   char id;         //  Nodens ID/navn.
   int  ant_naboer; //  Antall reelle naboer (et tall mellom 0 og 10 stk.).
   node* nabo[11];  //  Peker til naboene. Vi antar at 10 stk. holder!
   int  merke;      //  1 eller 0, alt ettersom om noden er SEEN/UNSEEN.
};
```

Vi har også en global array med pekere direkte til hver av nodene i hele grafen: `node* G[101];`

a) **Lag funksjonen** **`void skriv_TAK();`**
Funksjonen skal, ved hjelp av den *rekursive* funksjonen (som du også skal/må lage) `void visit(node* x)`, gjøre et dybde-først-søk fra hver av nodene i grafen for å avgjøre, og *skrive ut*, hvilke noder som er med i TA-kjernen for grafen. Funksjonene får bare lov til å endre på medlemmet ”merke” i hver node. Dennes verdi er ukjent/udefinert ved oppstart av ”skriv_TAK”-funksjonen. **Angi også tidsforbruket til funksjonene ved O-notasjon i forhold til antall noder V og antall kanter E i grafen. Begrunn/forklar kort ditt svar.**

b) Anta at en node A er med i *både* TA- *og* FA-kjernen.
Forklar kort hva som da mer kan sies om de to kjernene og hele grafen.
Anta nå i stedet at grafen er løkkefri/uten cykler.
Forklar kort hva som da kan sies om størrelsen på TA- og FA-kjernen.

c) Vi antar nå at grafen er løkkefri. **Lag funksjonen** **`bool er_i_FAK(node* x);`**
Funksjonen skal undersøke om noden ”x” er med i FA-kjernen for grafen. I så fall skal den returnere ”true” (ellers ”false”). Som i oppgave 4a, skal denne funksjonen også fungere ved at den gjør dybde-først-søk gjennom grafen vha. en *rekursiv* funksjon (som du også skal/må lage): `bool visit2(node* t, node* x);` Denne funksjonen sjekker og returnerer ”true”/”false” til om det finnes en vei/sti fra akkurat node ”t” til node ”x”. Funksjonene bør avslutte søket så fort det eventuelt blir klart at svaret blir negativt (”false”). Funksjonene får også her bare lov til å endre på medlemmet ”merke” i hver node. Dennes verdi er også ukjent/udefinert ved oppstart.

Lykke til !

frode@haug.com