Høgskolen i Gjøvik

Avdeling for Teknologi

E K S A M E N

FAGNAVN: **Algoritmiske metoder I**

FAGNUMMER: **L 189 A**

EKSAMENSDATO: **11. desember 2001**

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

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 fire uavhengige oppgaver fra kap. 3 (ab), 9 (d) og 11 (c) i læreboka.

a) Koden øverst side 28 i læreboka leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: (3 * (((7 * 3) + (4 + 12)) * (5 + 6)))
Hva blir dette skrevet på en postfix måte ?
Tegn opp innholdet på stakken etter hvert som koden side 28 leser tegn i infix-uttrykket.

b) Koden midt på side 27 i læreboka leser et postfix-uttrykk og regner ut svaret.
Vi har postfix-uttrykket: 7 4 6 * + 2 5 * 6 + +
Hva blir svaret ?
Tegn opp innholdet på stakken etter hvert som koden side 27 foretar beregningen.

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

36 24 27 20 21 19 24 18 19 19 18 18

Utfør etter tur følgende operasjoner på denne heap: insert(24), insert(36), remove(), remove() og replace(19). **Tegn opp heapen etter at hver av operasjonene er utført.**

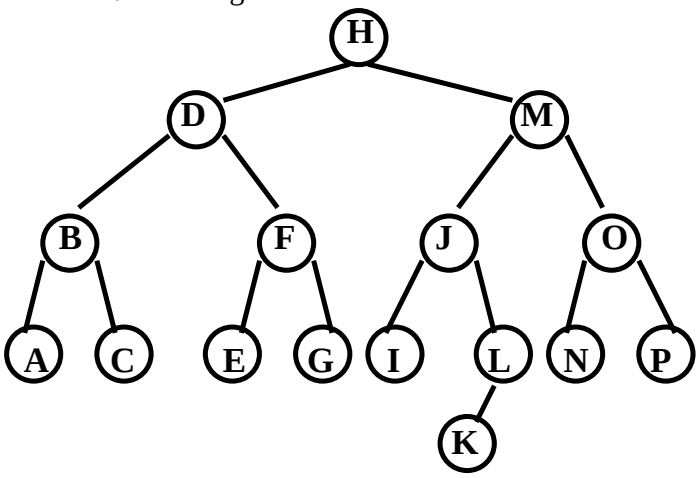
NB: For hver av operasjonene skal du operere videre på den heapen som ble resultatet av den forrige operasjonen.

d) Du skal utføre Quicksort på teksten «B U L L E K N O T T» (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. 14, 15 og 30 i lærebok.

a) 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 side 210 i læreboka som er aktuelle når du etter tur fjerner henholdsvis tegnene 'B', 'L' og 'D'.**

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.

b) Legg teksten «D R A G E H A L E S U P P E» (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.**

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

GA FG BD CD AC DF CE EG

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 side 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).

Oppgave 3 (teori og koding, 30%)

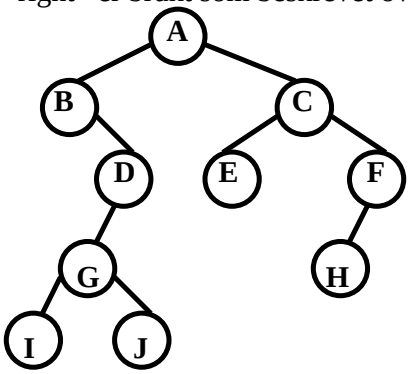
Denne oppgaven omhandler binære trær, der hver node i treet er definert på følgende måte:

```
struct node  {
   char id;             // Nodens ID/navn.
   node *left, *right;  // Pekere til venstre/høyre subtre, evt.
                        //   right peker til neste i preorder rekkefølge.
   bool neste_i_preorder;// = 'false' dersom høyre subtre finnes,
};                    // = 'true' dersom tomt høyre subtre og at right dermed
                        //     peker til neste node i preorder rekkefølge.
```

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

«Left» og «right» peker til henholdsvis venstre og høyre subtre. Om venstre subtre ikke finnes (er tomt), så peker «left» til z-noden. Om høyre subtre er tomt, så brukes «right» på en noe spesiell måte: I denne oppgaven er vi alltid interessert i å se på nodene i preorder rekkefølge i treet. Når vi evt. bare får en peker til en gitt node (kanskje midt inne i treet), så skal det uansett være lett og mulig å finne neste node i preorder rekkefølge i treet. For å enkelt kunne gjøre dette så skal «right», når den ellers hadde pekt til z-noden, peke til neste node i preorder rekkefølge. Datamedlemmet «neste_i_preorder» er 'true' dersom «right» er brukt på denne måten, og vil være 'false' dersom «right» peker til et reelt høyre subtre. (Dersom en node har tomt høyre subtre og er den aller siste i preorder rekkefølge, så vil «neste_i_preorder» være 'true' og «right» peke til z-noden.)

a) Vi har følgende tre, der «right» er brukt som beskrevet ovenfor.



Lag en tegning av treet 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 «neste_i_preorder» er lik 'true'.

b) Vi antar fortsatt at vi har et tre der «right» og «neste_i_preorder» er satt som beskrevet ovenfor. **Lag funksjonen** `node* neste(node* x)` **som for en gitt node 'x' (ulik z eller NULL) leverer en peker til den neste noden i preorder rekkefølge.** Dersom 'x' er den siste noden i preorder rekkefølge, skal funksjonen returnere z-noden. Legg vekt på at funksjonen blir effektiv!

c) Med den ovenfor angitte bruken av «right» og «neste_i_preorder» er det altså *alltid* mulig, ut fra en gitt node, å finne neste node i preorder rekkefølge. Dette gjelder faktisk selv om nodene ikke har peker opp til sin mor/far/foreldre, eller at vi kjenner roten i treet !
Vi vurderer nå å gjøre det tilsvarende for *postorder* rekkefølge. Pekeren «right» skal altså brukes til å peke på neste node, bare at nå dreier det seg om den neste i *postorder* rekkefølge. Det er *ikke* tillatt med noen ekstra peker i noden, f.eks. opp til mora. Vi får angitt en node i treet et sted, og skal finne neste node i postorder rekkefølge. **Vil denne teorien/algoritmen fungere? Begrunn (kort) svaret.**

d) Vi har et konkret binært tre som er laget vha. struct'en angitt i denne oppgavens innledning. Om høyre subtre er tomt, er treet foreløpig slik at «right»-pekerne referer til z-noden. Mens alle nodenes «neste_i_preorder» er i utgangspunktet satt til 'false'.
Lag en rekursiv funksjon: `void sett_neste(node* rot)` **som går gjennom hele (sub) treet under «rot» og gir korrekt verdi til nodenes «right» og «neste_i_preorder».**
Hint: Det kan være lurt å benytte seg av en global variabel «forrige». Denne peker til den *forrige* noden i preorder rekkefølge, og bør initielt inneholde referanse til z-noden.

Oppgave 4 (koding, 20%)

I denne oppgaven skal du skrive et program som genererer alle sekvenser av et gitt mønster/type (se nedenfor). Programmet *skal* bl.a. inneholde en rekursiv funksjon som vil gjør det meste av denne genereringen. Det som er generert ligger lagret i den globale arrayen «monster». Globalt har vi også verdien «n», som forteller oss hvor lang sekvensen er.
De sekvensene du skal generere skal alle være av lengde «n» (siden første tegn ligger i «monster[0]», så vil det siste tegnet ligge i «monster[n-1]»). De *eneste* tegnene som skal brukes er: '*', '(' og ')'. Det er viktig at den parentetiske strukturen er velformet (dvs. like mange '(' som '), og at parentesene inneslutter andre parenteser og/eller stjerner riktig). Tegnet '*' kan forekomme innimellom så mye det passer og er plass til. Når en ny sekvens er generert («n» lang) skal den skrives ut/displayes på skjermen.
For n = 3 vil de *eneste* lovlige sekvensene være:

```
* * *
* ( )
( * )
( ) *
```

Ved n = 3 vil *f.eks.* følgende sekvenser være ulovlige:

((*	Ikke velformet, da for mange '(' ift. ')'.
* *)	Ikke velformet, da for mange ')' ift. '('.
) (*	Ikke velformet, da parentesene ikke inneslutter (vender mot hverandre) riktig.

Bare for klargjøringens (?) del: For n = 4, vil de *eneste* lovlige sekvensene være:

```
* * * *      * * ( )      * ( * )
* ( ) *      ( * * )      ( * ) *
( ) * *      ( ) ( )      ( ( ) )
```

Hint1: I en halvferdig sekvens kan man holde orden på parentesstrukturen vha. *en eneste* int, som teller antall '(' ift. antall ')'.

Hint2: Når man skal sette inn en ny ')' kan man kun gjøre dette om det er nok '(' tidligere i sekvensen.

Hint3: Når man skal sette inn en ny '*' eller '(' kan man kun gjøre dette om det etterpå også vil være plass til alle nødvendige/påkrevde ')'.

NB: Legg vekt på god og effektiv avskjæring underveis i genereringen !

Lykke til !

frode@haug.com