Høgskolen i Gjøvik

Avdeling for elektro- og allmennfag

EKSAMEN

FAGNAVN: **Algoritmiske metoder I**

FAGNUMMER: **L 171 A**

EKSAMENSDATO: **18. desember 1997**

KLASSE: **96HINDA / 96HINDE (2DA / 2DB)**

TID: **09.00-14.00**

FAGLÆRER: **Frode Haug**

ANT. SIDER UTLEVERT: **4 (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 (ikke oppgavearkene).

Oppgave 1 (teori, 20 %)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 11 (a og b) og 15 (c) i læreboka.

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

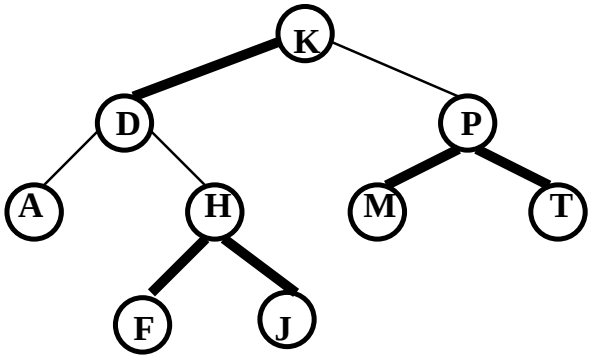
87 78 54 31 31 27 13 17 15 17 16 17

Utfør etter tur følgende operasjoner på denne heap: insert(67), insert(78), remove(), remove() og replace(31). **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.

b) Teksten «A L G M E T E R G Ø Y» 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) Følgende Red (tykke streker)-Black (tynne streker) tre er gitt:



Tegn opp resultatet når bokstavene "N" og "I" legges inn i treet ovenfor.

NB: For hver gang (bokstav) skal du på nytt ta utgangspunkt i treet ovenfor.
Dvs. bokstavene "N og "I" skal ikke til slutt befinne seg i det samme treet.

Oppgave 2 (teori, 25 %)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 22, 30 og 32 i læreboka.

a) Vis **konstruksjonsprosessen** når bokas metode **for Huffman-koding** (s.324-330) brukes på teksten «HIPP HIPP HIPP HURRA HURRA HURRAAAAAAA FOR HIG» (inkludert de blanke).
Hvor mange bits trengs for å kode denne teksten ? Dvs. skriv/tegn opp:

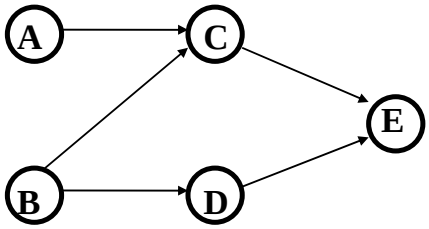
- tabellen for bokstavfrekvensen (jfr. fig.22.3).
- tabellen for «dad»en (jfr. fig.22.6).
- Huffmans kodingstreet/-trien (jfr. fig.22.5 og koden øverst s.328).
- bokstavenes bitmønster, med «code» og «len»
  (jfr. fig.22.7 og koden øverst s.329).
- totalt antall bits som brukes for å kode teksten.

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

AF BF BD BC CD FG AG EG AE CE AC DE

Vi skal nå utføre union-find på denne grafen. Tegn opp **arrayen «dad»s innhold etterhvert** som skogen bygges opp (jfr. fig.30.6) vha. «find»-funksjonen s.444 i læreboka. Ut fra dette: tegn også opp den resulterende **union-find skogen** (jfr. nedre høyre skog i fig.30.5).

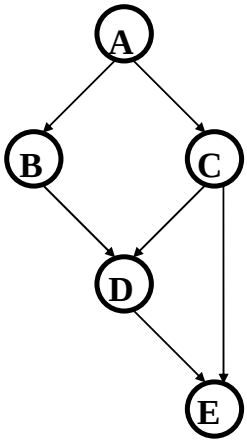
c) Følgende rettede asykliske (ikke-vektede) graf («dag») er gitt:



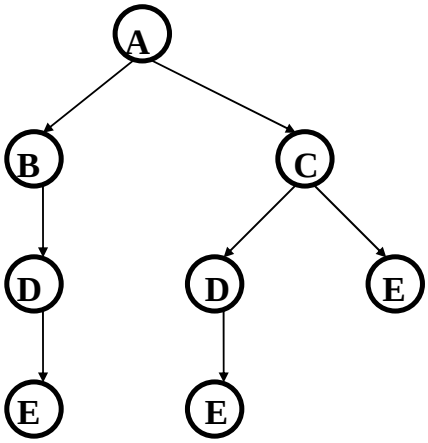
Angi **ALLE mulige topologiske sorteringssekvenser** av nodene i denne grafen.

Oppgave 3 (koding, 20 %)

Gitt en rettet graf uten løkker/sykler. Alle mulige veier gjennom grafen, fra en bestemt node, kan beskrives som et tre der nodene er kopier av nodene i grafen. I eksemplet under er det vist hvordan en graf kan «brettes» ut til et slikt tre:



Opprinnelig graf med nodene A, B, C, D, E.



Treet som representerer alle veier fra noden A.

Vi skal ta utgangspunkt i følgende grafrepresentasjonen av en node og dens naboer:

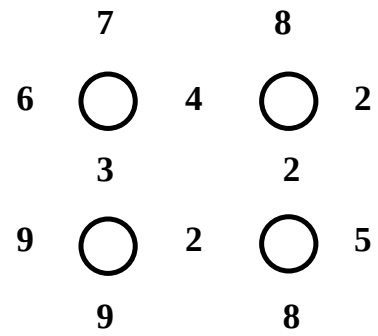
```
struct node  {
      char id;
      node* naboer[10];
      int ant_naboer, merke;
};
```

Skriv en rekursiv funksjon: `void brettut(node* start);`
som omformer grafen med utgangspunkt i node «start» til et slikt tre som forklart ovenfor.

Du kan forutsette at alle nodene i grafen kan nåes fra noden «start» (dvs. at grafen er sammenhengende). Du kan anta at «merke» er lik 0 («UNSEEN») i alle nodene når funksjonen din starter opp. Du kan også anta at det finnes en funksjon: `node* kopier(node* n);` som returnerer en peker til en kopi av den delen av en graf som består av noden «n» og alle nodene som kan nåes fra denne noden «n».

Oppgave 4 (koding, 35 %)

a) **(20%)** Dersom du plasserer en av de fire aritmetiske regneoperasjonene (+, -, *, /) inni hver av sirklene nedenfor, så vil summen på hver linje og kolonne kunne bli den samme.



<u>Skriv et rekursivt program som finner alle mulige løsninger på dette problemet.</u>

b) **(15%)** Vi har f.eks. følgende matematiske uttrykk, skrevet på en inorder måte:

12 * (3 + 17) .
(5 + 6) * (9 - 3) .
5 * ((7 + 6) + (12 / 2)) .
((6 / 3) + (8 - 6)) + (4 * ((2 + 11) - (18 - 6))) .

Lærebokas måte å beregne verdien til slike uttrykk på (når kun '+' og '*'), er å benytte seg av parse-trær (jfr. figur og kode s.41). I denne oppgaven derimot, skal du **<u>skrive EN rekursiv funksjon som leser et slikt uttrykk via cin-streamen, og beregner dets verdi.</u>** **NB:** Programmet skal **ikke** benytte seg av Stack eller Queue !

Du kan anta at:

- for å forenkle innlesningen (vha. "cin >> ch;"), så avsluttes uttrykket **alltid** med '.' .
- alle deluttrykk **alltid** består av to operander og en operator, omsluttet av paranteser (unntak er eventuelt ytterste nivå: se 1. og 3. eksempel ovenfor).
- det ikke forekommer noen skrivefeil i det matematiske uttrykket, f.eks. ikke matchende antall paranteser, bokstaver istedet for tall, tre eller flere operander inni **en** parantes eller at uttrykket ikke avsluttes med '.' .

Om det skulle være til noen hjelp (?), husk at:

- Hele regneuttrykket **alltid** starter med et tall eller '(' .
- Et tall etterfølges **alltid** av operator, ')' eller '.' .
- En operator etterfølges **alltid** av et tall eller '(' .
- En '(' etterfølges **alltid** av et tall eller en ny '(' .
- En ')' etterfølges **alltid** av en operator, en ny ')' eller '.' .

Lykke til, og god jul !
FrodeH