Høgskolen i Gjøvik

Avdeling for elektro- og allmennfag

KONTINUASJONSEKSAMEN

FAGNAVN:	**Algoritmiske metoder (2 vekttall)** **Algoritmiske metoder I (3 vekttall)**
FAGNUMMER:	**LO 164 A (2 vektall)** **L 171 A (3 vekttall)**
EKSAMENSDATO:	**Mandag 24. februar 1997**
KLASSE:	**xxHINDA / xxHINDE**
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 (ikke oppgavearkene).

NB: Dette eksamenssettet er laget felles for både de som tar 2 (siste gang forelest høsten 1995) og 3 (nytt kurs høsten 1996) vektalls eksamen i algoritmiske metoder (I). Husk derfor på:

1. **Skriv på <u>første arket i besvarelsen OG på omslaget</u> hvilken eksamen du går opp til (2 eller 3 vekttall).**
2. **Legg godt merke til hvilke oppgaver <u>du</u> skal besvare.**
   2 vekttall: Oppgave 1.1, 2, 3 og 4.
   3 vekttall: Oppgave 1.2, 2, 3 og 4.

Oppgave 1.1 (teori, 25 %)

NB: Løses <u>kun</u> av de som tar 2 vekttalls-eksamen.

Denne oppgaven inneholder fire uavhengige oppgaver fra kap. 3, 4, 19 og 31 i læreboka.

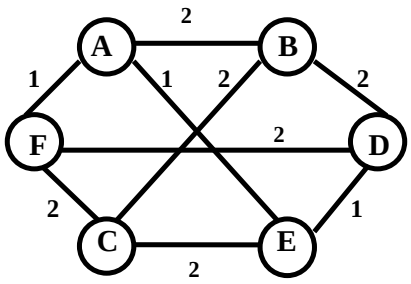
a) Koden øverst s.28 i læreboka leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: ((7 + 11) * (((8 * 2) * (3 + 7)) + (3 * (4 + 2))))
Hva blir dette **<u>skrevet på en postfix</u>** måte ?
Tegn opp **<u>innholdet på stakken etter hvert</u>** som koden leser tegn i infix-uttrykket.

b) **<u>Tegn opp parse-treet</u>** for uttrykket: ((A * B) * (C + D)) + (E * (F + G)))
(Hint: Figur 4.4 s.41 i læreboka.)

c) Vi har søkestrengen "HØGSKOLEKANDIDATSTUDIET". Mønsteret vi skal søke med er "DIE". Bokstavene som totalt kan forekomme i søkestrengen er tegnene A-Å.

- **<u>Angi skip-arrayen</u>** for dette tilfellet.
- **<u>Tegn opp hvordan mønsteret forflyttes</u>** bortover i søkestrengen når søkealgoritmen s.288 i læreboka utføres. Dvs. lag en figur etter samme prinsipp som fig.19.7 s.287 i læreboka.

d) Følgende vektede (ikke-rettede) graf er gitt:



Hver nodes naboer er representert i en naboliste. Alle listene er <u>sortert alfabetisk</u>.
Tegn opp **<u>minimums spenntreet</u>** for denne grafen, etter at koden s.455 i læreboka er utført (der "priority" er lik "t->w").
Tegn også opp **<u>innholdet i prioritetskøen etterhvert</u>** 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.

Oppgave 1.2 (teori, 25 %)

NB: Løses kun av de som tar 3 vekttalls-eksamen.

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 «INTERNET ER INTERNT ANARKISTISK» (inkludert de blanke).

Hvor mange bits trengs for å kode denne teksten ? Dvs. skriv/tegn opp:

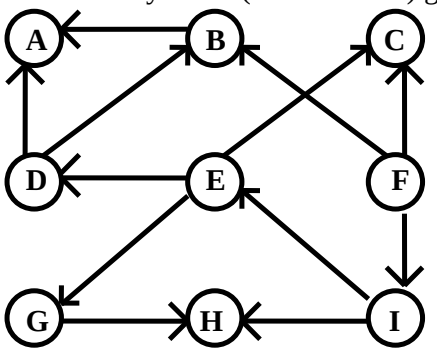
- tabellen for bokstavfrekvensen (jfr. fig.22.3).
- tabellen for «dad»en (jfr. fig.22.6).
- Huffmans kodingstreet/-trien (jfr. fig.22.5 og koden øverst s.328).
- bokstavenes bitmønster, med «code» og «len» (jfr. fig.22.7 og koden øverst s.329).
- totalt antall bits som brukes for å kode teksten.

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

AC BF AE CH HJ BH DI FH AI EG FI CD EI

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 **EN topologisk sorteringssekvens**.

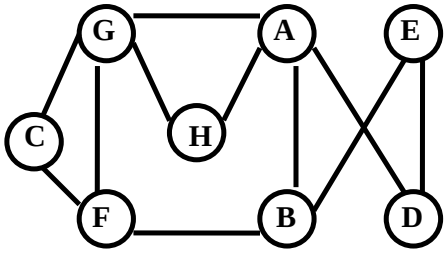
Oppgave 2 (teori, 25 %) NB: Løses av alle.

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 15, 11 og 29 i læreboka.

a) Legg teksten «I N T E R N E T U H Y R E T» (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 (prøv å få alt inn på ett A4-ark). Gjør også om slutt-resultatet til et Red-Black tre.

b) Teksten «I N T E R N E T U H Y R E T» (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) Vi har følgende (ikke-vektede, ikke-rettede og ikke-sammenhengende) graf:



Tegn **dybde-først søketreet** for dette tilfellet (ved bruk av nabomatrise), når "Search" (s.424) og "Visit" (s.427) fungerer som angitt i læreboka.

Oppgave 3 (koding, 30%) NB: Løses av alle.

Vi har følgende deklarasjon:
```
struct element {
    int id;
    element* next;
};
```
Vi har en liste som er bygd opp av slike elementer. (Hvordan denne er initiert og hektet sammen trenger ikke du å tenke på.) Listen har ingen hode eller hale, og det siste elementets next-peker refererer til NULL. Hele listen tilpekes av den globale variabelen «start». I denne oppgaven skal **du lage fire ulike REKURSIVE funksjoner** som bl.a. tar en peker til en liste som input, og som deretter utfører følgende operasjoner:

a) Skriver listens innhold ut baklengs: **void skriv_baklengs(element* liste);**
Kalles første gang ved: skriv_baklengs(start);

b) Sletter listen bakfra: **void slett_bakfra(element* & liste);**
Kalles første gang ved: slett_bakfra(start);
For å bl.a. ivareta at den globale variabelen «start» får verdien 'NULL' etter at funksjonkallet er ferdig utført, så blir parameteren «liste» referansesoverført.
NB: Husk at listens elementer er opprettet vha. «new».

c) Kopierer listen: **element* kopier_liste(element* liste);**
Kalles første gang ved: kopi = kopier_liste(start);
Der «kopi» er deklarert som: element* kopi;

d) Inverterer/snur listen:
void snu_liste(element* & rot, element* liste, element* forrige);
Kalles første gang ved: snu_liste(start, start, NULL);
For å ivareta at den globale variabelen «start» får verdien til orginal-listens siste element
etter at funksjonkallet er ferdig utført, så blir parameteren «rot» referansesoverført.
For å kunne snu «next»-pekerne, så er også parametrene «liste» og «forrige» brukt:
«Liste» peker til resten av lista, mens «forrige» peker til listens forrige element.

Oppgave 4 (koding, 20 %) NB: Løses av alle.

Vi har følgende deklarasjoner:

```
struct element  {
    int id;
    element * forrige, * neste;
};

element* toveisliste, * rot, * z;
```

Vi har en toveis-liste som er bygd opp av slike elementer. (Hvordan denne er initiert og hektet sammen trenger ikke du å tenke på.) Listen har ingen hode eller hale. Det siste elementets neste-peker og det første elementets forrige-peker refererer til z-noden. (Z-noden er som i boka, der dens forrige- og neste-peker refererer til z selv.) Hele listen tilpekes initielt av den globale variabelen «toveisliste». I denne oppgaven skal **du lage EN rekursiv funksjon** som sørger for at en slik liste gjøres om til et mest mulig balansert binært tre.

Funksjonen kalles ved: rot = lag_binaert_tre(toveisliste);

Tegn også opp hvordan treet blir seende ut dersom den intielle listen består av elementer med id'ene:

a) 1, 2, 3, 4 og 5
b) 1, 2, 3, 4, 5, 6, 7, 8, 9 og 10

Hint: Funksjonen bør lete seg fram til midten av lista den får som input og la dette elementet bli roten i et (sub)tre. (Består lista av et partall antall elementer, så tas det første elementet etter midten som rot.) Deretter tilkaller den seg selv rekursivt for hver av de to listene på hver side av seg selv, for å få på plass sine to subtrær/«barn». NB: Husk at disse subtrærne ikke tilpekes av «left» og «right», men av «forrige» og «neste». Husk også på å la relevante pekere referere til z-noden etterhvert.

Lykke til !! Og god vinterferie ??

FrodeH