Høgskolen i Gjøvik

Avdeling for elektro- og allmennfag

EKSAMEN

FAGNAVN:	**Algoritmiske metoder I**
FAGNUMMER:	**L 171 A**
EKSAMENSDATO:	**19. desember 1996**
KLASSE:	**95HINDA / 95HINDE (2AA / 2AE)**
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, 25 %)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.15, 11 og 22 i læreboka.

a) Legg teksten «H Ø G S K O L E S T 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 «H Ø G S K O L E S T 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.)

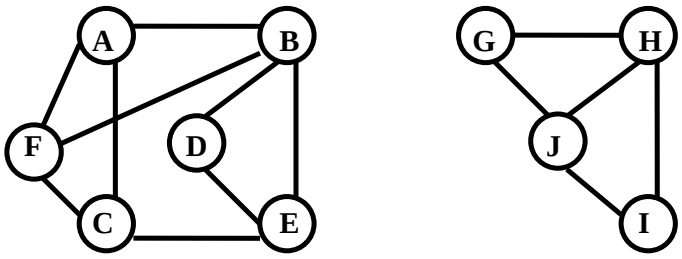
c) Vis **konstruksjonsprosessen** når bokas metode **for Huffman-koding** (s.324-330) brukes på teksten «MISSISIPPI IS STILL MISSING SHIPS» (inkludert de blanke).
Hvor mange bits trengs for å kode denne teksten ? Dvs. skriv/tegn opp:

- 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 bytes som brukes for å kode teksten.

Oppgave 2 (teori, 25 %)

Denne oppgaven inneholder fire uavhengige oppgaver fra kap.29, 30, 31 og 32 i læreboka.

a) Vi har følgende (ikke-vektede, ikke-rettede og ikke-sammenhengende) graf:



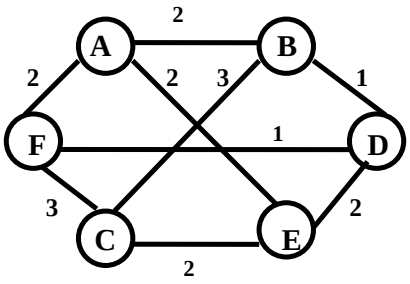
Tegn **dybde-først søketreet** for dette tilfellet (ved bruk av nabomatrise), når "Search" (s.424) og "Visit" (s.427) fungerer som angitt i læreboka.

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

AH AC DF GH IF FE BD EI JD CH DG JI

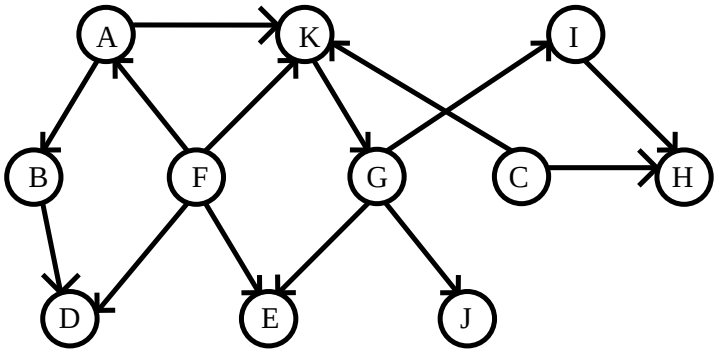
Vi skal nå utføre union-find på denne grafen. Tegn opp **arrayen «dad»s innhold etterhvert** som skogen bygges opp (jfr. fig.30.6) vha. «find»-funksjonen s.444 i læreboka. Ut fra dette: tegn også opp den resulterende **union-find skogen** (jfr. nedre høyre skog i fig.30.5).

c) Følgende 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.

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



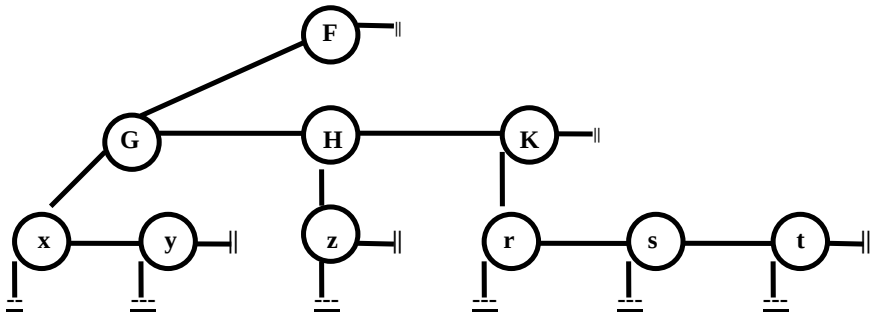
Angi **EN topologisk sorteringssekvens**.

Oppgave 3 (koding, 25%)

Vi har en trestruktur der hver node kan ha vilkårlig mange subtrær. Det er da naturlig å kjede sammen subtrærne til hver enkelt node i en liste ordnet etter rekkefølgen av subtrærne. Vi skal representere nodene slik:

```
class node  {
 char navn;           // Nodens navn/ID.
 node* forste_subtre; // Første subtre. Er NULL dersom ingen subtrær.
 node* neste_sosken;  // Neste node i listen man er subtre sammen med.
                      //  Er NULL dersom man selv er det siste subtreet.
};
```

Et tre etter dette prinsippet kan f.eks. se slik ut («forste_subtre» peker (på skrå) nedover, «neste_sosken» peker vannrett og NULL = —||):



Din oppgave er å lage en rekursiv funksjon «void skriv_ut(node* n)». Som parameter tar denne en peker til roten i et (sub)tre som beskrevet ovenfor. Den skriver ut nodenes navn i prefiks rekkefølge. For hver node, bortsett fra blad-nodene («forste_subtre» == NULL), skal utskriften av subtrærne omsluttes med paranteser, og subtrærne skal skilles med komma. For figuren ovenfor ville utskriften ha blitt (blanke ikke medregnet): F(G(x, y), H(z), K(r, s, t)).

Oppgave 4 (koding, 25 %)

N personer skal fordele M gjenstander (f.eks. malerier eller antikviteter) seg imellom. M er (noe) større enn N. Alle gjenstandene nummereres fra 1 til M. Alle N personene oppgir tre ulike gjenstander de ønsker seg aller mest, i prioritert rekkefølge. Ønskene ligger lagret i arrayen «int onsker[N+1][4]», der vi ikke bruker indeks nr.'0' i noen av «retningene». F.eks: «onsker[6][3]» inneholder nummeret på gjenstanden som person nr.6 har som 3.prioritet.
Den beste fordelingen er representert av arrayen «int beste_fordeling[M+1]». Der indeks nr.'i' inneholder nummeret til personen (av de N) som har blitt tildelt gjenstand nr.'i'. (Den inneholder '0' dersom ingen har blitt tildelt gjenstand nr.'i'.)
Den beste fordelingen er når **summen** av alle brukernes innfridde prioritet er minst mulig. (Det mest optimale er når alle får innfridd sitt første ønske, dvs. med en beste totalprioritet på N.)
Er det flere slike med lik minste totalprioritet, så skal den første beregnede sendes tilbake i arrayen «beste_fordeling».
Funksjonen skal også melde fra (via variabelen «bool losning_funnet») om den i det hele tatt klarte å la alle N personene få et av sine ønsker innfridd. (Det kan jo hende at for mange ønsker seg de samme gjenstandene, slik at en fordeling er umulig.)

Din oppgave er å skrive en rekursiv funksjon «void fordel_gjenstander(int n)», som (om mulig) finner en løsning på denne problemstillingen. Parameteren til funksjonen betyr at den nå skal fordele/finne (om mulig) en gjenstand til person nr.'n'. Funksjonens output skal altså være at variabelen «losning_funnet» er satt til 'false' eller 'true'. Dersom denne er satt til 'true', så vil i tillegg den globale arrayen «beste_fordeling» inneholde hvordan gjenstandene er fordelt til personene.

Hint: I tillegg til det tre ovenfor nevnte variablene/arrayene, så er det nok lurt å bruke:

- en hjelpe-array (f.eks: «int gjenstand[M+1]») som viser hvilke gjenstander som i øyeblikket er fordelt til hvilke personer. Når man eventuelt når fram til en ny minste totalprioritet, så kopieres arrayens innhold over i «beste_fordeling».
- to enkelt-variable som holder orden på den aller minste totalprioritet oppnådd, og på den for øyeblikket totale sum av alle personenes (fra 1 til 'n') prioritet.

Lykke til, og god jul !
FrodeH