# K O N T I N U A S J O N S E K S A M E N

| | | |
|---|---|---|
| **FAGNAVN:** | **Algoritmiske metoder** | |
| **FAGNUMMER:** | **LO 164A** | |
| **EKSAMENSDATO:** | **6. september 1996** | **TID: 09.00-14.00** |
| **FAGLÆRER:** | **Frode Haug** | **KLASSE: 2 AA / AE** |
| **ANTALL SIDER UTLEVERT:** | **4 (inkludert denne forside)** | |
| **TILLATTE HJELPEMIDLER:** | **Alle trykte og skrevne** | |

**NB: - Kontroller at alle oppgavearkene er tilstede.**

**- LES HELE OPPGAVETEKSTEN NØYE, FØR DU BEGYNNER Å BESVARE NOE SOM HELST.**

**- DET ER INGEN SAMMENHENG MELLOM DE ULIKE DELENE I OPPGAVENE 2 OG 3. DERMED KAN ALLE UNDERPUNKTER LØSES TOTALT UAVHENGIG.**

**- Ikke skriv noe av din besvarelse på oppgavearkene.**

**- Kladd og oppgavearkene leveres sammen med besvarelsen. Kladd merkes med "KLADD".**

**- Husk kandidatnummer på alle ark (IKKE oppgavearkene).**

# Oppgave 1 (teori, 10 %)

Følgende programkode er gitt:

```
#include <iostream>
using namespace std;


void rekursiv(int n)  {
            if (n < 10)  cout << n;
            else  {
                  rekursiv(n / 10);
                  cout << n % 10;
            }
}

void main()  {
            rekursiv(123);
}
```

Hva blir utskriften fra programmet ?

# Oppgave 2 (teori, 15 %)

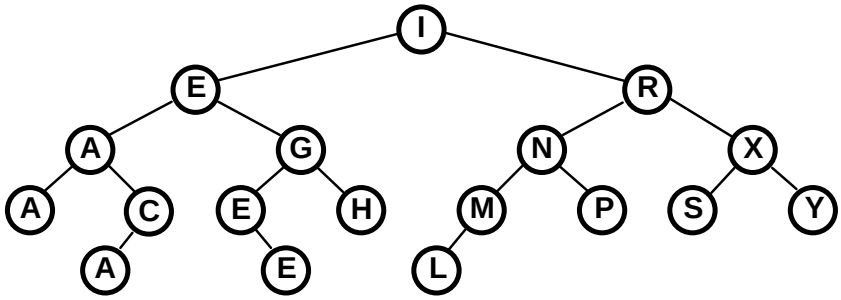
Denne oppgaven inneholder fire uavhengige oppgaver fra kap.3 og 4 i læreboka.

**a)** Koden øverst s.28 i læreboka leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: ( ( ( ( 7 * 14 ) * ( 16 + 3 ) ) + 74 ) * 12 )
Hva blir dette skrevet på en postfix måte ?
Tegn opp innholdet på stakken etter hvert som koden leser tegn i infix-uttrykket.

**b)** Koden midt på s.27 i læreboka leser et postfix-uttrykk og regner ut svaret.
Vi har postfix-uttrykket: 3 7 + 4 18 * + 2 15 * * 7 +
Hva blir svaret ?
Tegn opp innholdet på stakken etter hvert som koden foretar beregningen.

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

**d)** Følgende tre er gitt:




Angi nodenes sekvens når dette treet traverseres på en: preorder, inorder, postorder og level-order måte.

# Oppgave 3 (teori, 20 %)

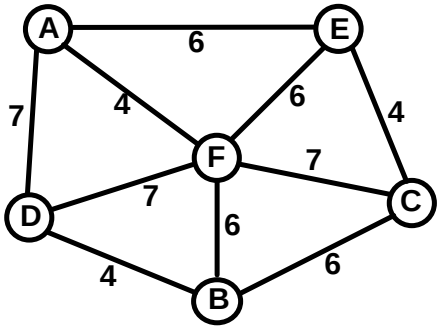
Denne oppgaven inneholder tre totalt uavhengige oppgaver, som er basert på ulike deler av pensum.

I deloppgave a) og b) er det key'ene "T A N N R E G U L E R I N G" (i denne rekkefølge fra venstre mot høyre, og blanke regnes ikke med) som du skal bruke.

**a)** Vi skal utføre Shellsort på key'ene. Jfr. kode s.109 i læreboka. For hver gang indre for-løkke er ferdig (etter: `a[j] = v;` ): Tegn opp arrayen og skriv verdiene til 'h' og 'i'. Marker spesielt de key'ene som har vært involvert i sorteringen (jfr. fig.8.7 s.108).

**b)** Tegn (skriv) arrayen etter hvert som hver av key'ene legges inn i en heap.

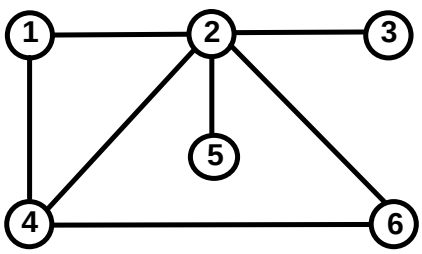
**c)** Følgende vektede graf er gitt:




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.

# Oppgave 4 (koding, 30 %)

Vi skal i denne oppgaven se på en ikke-vekted graf. For en gitt slik graf skal vi si at en ikke-tom undermengde (et utplukk) av nodene er "uavhengige" dersom det ikke går noen kant mellom noder i utplukket. Dersom vi f.eks. har følgende graf:



Her utgjør f.eks. både { 1, 3, 5, 6 } og { 3, 4 } uavhengige nodemengder, mens f.eks. mengden { 4, 5, 6 } ikke er uavhengig (pga. kant mellom 4 og 6). Merk at alle mengder med bare en node er uavhengige, mens den tomme nodemengde altså ikke regnes som uavhengig. For en gitt graf vil vi ha skrevet ut alle mulige uavhengige nodemengder (selv om dette kan bli nokså mange!). Vi antar at nodene er identifisert med tallene fra 1 til 'n', og at grafen er angitt ved nabomatrisen 'G', som er globalt deklarert. Du skal lage en rekursiv prosedyre som finner og skriver ut alle slike uavhengige utplukk. Dvs. du skal lage innmaten til den nedenfor angitte funksjonen. (Studer kommentaren inne i funksjonen og hintet nedenfor nøye, før du begynner å kode.)

```
.....
int G[n+1][n+1];        //    Nabomatrisen.
int utplukk[n+1];       //    Nåværende utplukk.
int ant = 0;            //    Nåværende antall noder utplukket.

void genresten(int k)  {
      // Når denne kalles er det gjort et uavhengig utplukk blant
      // nodene 1,2,...,k-1, og antallet i dette utplukket ligger i
      // "ant", og de utplukkede nodene er angitt i arrayen "utplukk"
      // fra indeks 1 til "ant". Denne rekursive funksjonen skal
      // sørge for å få laget alle mulige (lovlige) utvidelser av
      // dette utplukket (også den tomme utvidelse!), og å få skrevet
      // ut hver av de resulterende utplukkene.
}
.....
int main(void)  {
   genresten(1);
   return 0;
}
```

Hint: En mulig organisering er at hvert rekursivt kall bare ser på alle muligheter for å utvide utplukket med en node, og å overlate videre utplukk til nye rekursive kall. Tenk nøye over hvor i funksjonen det skal gis utskrift.

# Oppgave 5 (koding, 25 %)

La oss tenke oss følgende regnestykke: AB * C = DE * F = GHI , der hver bokstav står for et unikt siffer fra '1' til '9'. Ett av svarene på dette regnestykket vil f.eks. kunne være:
39 * 4 = 78 * 2 = 156 Her ser vi at hvert av de ni sifrene kun er brukt en gang.
Skriv rekursiv kode som genererer alle regnestykkene som tilfredsstiller bokstav-regnestykket ovenfor, og der hvert av sifrene fra '1' til '9' kun er brukt en gang. Effektiviser, ved å avskjære genereringen av mulige løsninger, når f.eks:

- AB > DE , f.eks: 78 * 2 = 39 * 4 = 156 som er likt eksemplet ovenfor.
- AB * C < DE , det 1.produktet er allerede mindre enn 1.faktor i det 2.uttrykket.
- AB * C != DE * F , de to første produktene er ikke like hverandre.

Lykke til i den rekursive verden !

FrodeH