E K S A M E N

FAGNAVN:	**Algoritmiske metoder**	
FAGNUMMER:	**LO 164 A**	
EKSAMENSDATO:	**15. desember 1994**	**TID: 09.00-14.00**
FAGLÆRER:	**Frode Haug**	**KLASSE: 2AA/AE**
ANTALL SIDER UTLEVERT:	**5 (inkludert denne forside)**	
TILLATTE HJELPEMIDLER:	**Alle trykte og skrevne.**	

NB! **- Det er tillatt å føre med blyant (men skriv pent).**

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

- Kontroller at alle oppgavearkene er tilstede.

- Ikke skriv noe av din besvarelse på oppgavearkene.

- Husk kandidatnummer på alle ark.

- DET ER <u>INGEN</u> SAMMENHENG MELLOM DE ULIKE DELENE I OPPGAVENE 1-4. DERMED KAN <u>ALLE</u> UNDERPUNKTER LØSES TOTALT UAVHENGIG.

Oppgave 1 (teori, 10 %)

Vi skal se på problemet med å velge riktig sorteringsalgoritme ut fra hvordan situasjonen er. Under er skissert et antall situasjoner, og du skal velge en fornuftig sorteringsmetode for hver av dem. Gi en kort begrunnelse for hvert valg. (Sammen med begrunnelsen er det altså nok å kun henvise til de syv metodene som er beskrevet i læreboka.)

Situasjon A:

Vi skal så fort som mulig utføre sortering av flere hundre enkeltarrayer, hver med over tusen heltall. Innholdet av de enkelte arrayene er "godt blandet" (slik at ingen av våre sorteringsalgoritmer får spesielle fordeler eller ulemper), og det er den totale tiden for alle sorteringene som er av betydning. NB: Arrayene skal ikke samsorteres (merges), men bare sorteres hver for seg.

Situasjon B:

Samme som situasjon A, bortsett fra at hver av arrayene nå bare inneholder omkring ti heltall.

Situasjon C:

Samme som situasjon A, bortsett fra at hver av arrayene nå inneholder omkring hundre heltall, og at alle arrayene på forhånd er "nesten sortert".

Situasjon D:

Som en sentral del av en operasjon i et interaktivt program, inngår én sortering av en array med omkring 5000 heltall. Hukommelsesplassen i programmet er knapp, og ikke minst er det viktig at man kan garantere at operasjonen er ferdig etter en viss rimelig tid.

Oppgave 2 (teori, 10 %)

I de følgende oppgaver er det key'ene "E K S A M E N S N E R V E R" (i denne rekkefølge fra venstre til høyre, og blanke regnes ikke med) som du skal bruke. For alle oppgavene så gjelder det at den initielle heap, tre eller array er tom før første innlegging ("Insert") utføres.

Tegn (skriv) den resulterende datastruktur når key'ene legges inn i:

a) et binært søketre.

b) en heap.

c) et 2-3-4 tre.

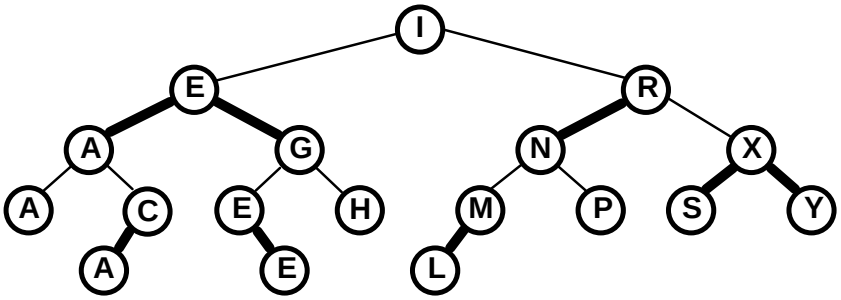
d) et Red-Black tre.

e) en array vha. linear probing. Arrayen har indeksene 0-13. Bruk hash-funksjonen: $h_1(k) = k \bmod 14$, der "k" står for bokstavens nummer i alfabetet (1-29).

Oppgave 3 (teori, 15 %)

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

a) Vi har et Red-Black tre med følgende utseende (jfr. fig.15.8 s.222 i læreboka.) :



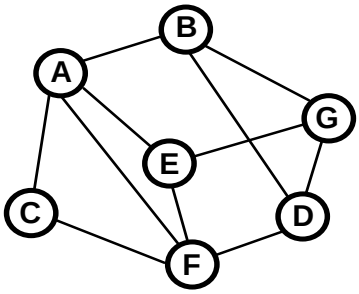
- Hvilke linker oppdateres og
- hvilke "farger" (Red/Black) får de

av "Split"- og "Rotate"-funksjonene (s.225 og 226 i læreboka) når bokstaven "J" og deretter "B" blir lagt inn i dette treet ? (Tegn opp.)

b) Vi har søkestrengen "EKSAMENSNERVER". Mønsteret vi skal søke med er "ERVE". Bokstavene som totalt kan forekomme i søkestrengen er tegnene A-Å.

- Angi skip-arrayen for dette tilfellet.
- Tegn opp hvordan mønsteret forflyttes 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.

c) Vi har grafen:



c1) Skriv opp nabomatrisen for dette tilfellet.

c2) Tegn dybde-først søketreet for dette tilfellet (dvs. ved bruk av nabomatrise), når "Search" (s.424) og "Visit" (s.427) fungerer som angitt i læreboka.

Oppgave 4 (koding, 10 + 20 %)

a) (10%) Funksjonen for å legge inn i binære søketrær ("Insert" s.206 i læreboka) vil alltid legge til nye noder aller nederst i treet. Modifiser denne funksjonen, slik at den holder noder med identisk key samlet i treet. (Dvs. dersom en annen node i treet har samme key som en gitt node, så skal entens dets "mor" eller en av dets "barn" ha lik key.)

NB: Funksjonene "Search" og "Remove" (s.204 og 210 i læreboka) skal fortsatt ha samme funksjonalitet (dvs. de skal ikke endres/omskrives p.g.a. modifiseringen av "Insert").

b) (20%) Vi skal i denne deloppgaven arbeide med en graf. Denne representerer et gatenettverk. Nodene er gatekryss, og kantene er gater. En del gater er enveiskjørte. Dette medfører at for disse kantene vil det være en referanse fra en node og til en annen, men ikke omvendt. (Toveiskjørte gater vil dermed ha referanse begge veier mellom nodene.) Til hver kant (gate) er det knyttet et reelt tall 'mh' ("maksimum høyde"), som er den maksimale høyden en bil kan ha i den tilsvarende gaten (p.g.a. lave broer, ledninger o.l). Maksimum antall gater ut fra et kryss setter vi til 6.

En node i grafen er representert ved følgende struct:

```
struct kryss  {
   char navn[80];          //  Gatekryssets navn.
   int ant_gater;          //  Antall utgående gater fra krysset (1-6 stk.).
   kryss* gate[7];         //  Referanse til andre kryss, dvs. gatene/kantene
                           //    til disse. Vi bruker indeksene 1-6.
   float mh[7];            //  Maksimum høyde for gatene ut fra krysset.
   int merke;              //  Hjelpevariabel for prosedyren du skal lage.
};
```

Skriv en funksjon: **bool muligvei(kryss* fra, kryss* til, float h);**

Denne funksjonen skal svare på ('false' eller 'true') om det finnes en vei fra kryss A til kryss B, som en bil med høyde 'h' kan kjøre. Funksjonen behøver ikke å lete etter noen spesielt "god" vei, og så fort prosedyren har funnet en mulig vei, skal den skrive ut (navnene til alle kryssene langs) denne, baklengs eller forlengs, og så terminere/avslutte. **LØSNINGEN SKAL VÆRE REKURSIV, DEN SKAL BESTÅ AV EN FUNKSJON OG DU SKAL IKKE BRUKE EN GLOBAL VARIABEL FOR Å INDIKERE AT DU ER FERDIG/ KOMMET FRAM** (men utnytte at den rekursive funksjonen returnerer en verdi).

Du kan anta følgende:

- Datastrukturen (grafen) er på en eller annen måte innlest og opprettet før 'muligvei' tilkalles. Dvs. alle nodene er opprettet, pekerne ('gate') mellom dem, høyden for gatene ('mh'), navnene og 'ant_gater' er satt. Dessuten er merke initiert til '0'.
- Grafen er sammenhengende.
- De aktuelle parametrene, som sendes med idet 'muligvei' tilkalles første gangen, er to pekere til start- og sluttnoden. (Hvordan man på det startende kallstedet har fått tak i disse to pekerne, trenger du ikke å bekymre deg med.)

Oppgave 5 (koding, 35 %)

Tallene fra 1-16 skal plasseres inn i et 4x4 rutenett. Dette skal gjøres på en slik måte at summen av alle linjer, kolonner og hoved-/bidiagonal samtidig er 34. (Hoveddiagonal går fra øvre venstre til nedre høyre hjørne. Bidiagonal går fra øvre høyre til nedre venstre hjørne.)

En løsning kan være:

16	**15**	**2**	**1**
4	**3**	**14**	**13**
5	**10**	**7**	**12**
9	**6**	**11**	**8**

Skriv et program som skriver ut alle løsninger på denne oppgaven. Den skal basere seg på en rekursiv løsning.

NB 1: Programmet trenger ikke å være komplett (med alle include'r, hele main, o.l), men angi nok kode til at sensor forstår at du vet hvordan programmet bør være.
Dvs. ta med kode som f.eks:
- globale variable.
- kallet (i main?) som starter det hele.
- nødvendige/aktuelle funksjoner som:
  - finner løsninger (rekursiv).
  - tester om er en lovlig løsning.
  - skriver ut en løsning til skjermen.

NB 2: Legg vekt på å skrive programkode som er effektiv, elegant og som foretar avskjæringer når det man er i ferd med å bygge opp aldri kan bli en lovlig løsning (da deler av den allerede har en sum som er ulik 34).

NB 3: Du trenger ikke å avskjære/hoppe over løsninger som bare er rotasjoner/speilinger av andre allerede funne løsninger.

(Digresjon: Det er 416 løsninger når '1' står fast i øvre venstre hjørne.).

Lykke til og glad jol !

(Og har du vært riktig flinkt barn, så har du stått til jul i Alg.Svett.)

FrodeH