Høgskolen i Gjøvik
Avdeling for teknologi

Kontinuasjonseksamen

FAGNAVN: **Algoritmiske metoder**

FAGNUMMER: **IMT2021 og IMT2001**

EKSAMENSDATO: **13. august 2004**

KLASSE(R): **02HIND* / 02HINE* / div. andre**

TID: **IMT2021: 09.00-14.00**
IMT2001: 09.00-13.00

FAGLÆRER: **Frode Haug**

ANTALL SIDER UTLEVERT: **4 (inkludert denne forside)**

TILLATTE HJELPEMIDLER: **Alle trykte og skrevne.**

- Kontroller at alle oppgavearkene er tilstede.
- Innføring med penn, eventuelt trykkblyant som gir gjennomslag. Pass på så du ikke skriver på mer enn ett innføringsark om gangen (da det blir uleselige gjennomslag når flere ark ligger oppå hverandre).
- Ved innlevering skilles hvit og gul besvarelse, som legges i hvert sitt omslag.
- Oppgavetekst, kladd og blå kopi beholder kandidaten til klagefristen er over.
- Ikke skriv noe av din besvarelse på oppgavearkene. Men, i oppgavetekst der du skal fylle ut svar i tegning/tabell/kurve, skal selvsagt dette innleveres sammen med hvit besvarelse.
- Husk kandidatnummer på alle ark. Ta vare på dette nummeret til sensuren faller.

NB: De som tar IMT2021 (10 studiepoeng) skal *løse alle* oppgavene.
Oppgavene er da vektlagt med: 25%, 25%, 30% og 20%
De som tar IMT2001 (6 studiepoeng) skal *ikke løse* oppgave nr. 2.
Oppgavene er da vektlagt med: 40%, 35% og 25%

Oppgave 1 (teori)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 9, 11 og 15 i læreboka.

a) Du skal utføre Quicksort på teksten "R O S I N B O L L E N" (blanke regnes ikke med). **Lag en figur tilsvarende fig. 9.3 side 119 i læreboka**, der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

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

71 58 49 32 31 37 32 22 21 23 22 21

Utfør etter tur følgende operasjoner på denne heap: insert(39), insert(59), remove(), remove() og replace(21). **Skriv opp heapen etter at hver av operasjonene er utført.**

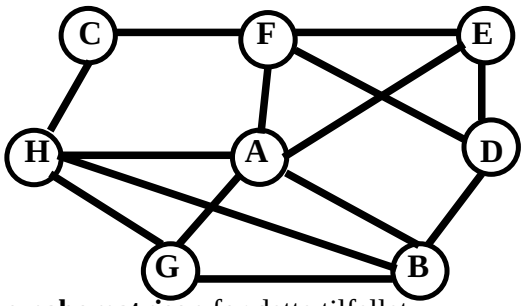
NB: For hver av operasjonene skal du operere videre på den heapen som ble resultatet av den forrige operasjonen.

c) Legg key'ene "BILSUPPEBOLLE" (i denne rekkefølge fra venstre til høyre) inn i et 2-3-4 tre. **Tegn opp treet etterhvert som bokstavene legges inn.**
Gjør også om sluttresultatet til et Red-Black tre.

Oppgave 2 (teori) - Skal *ikke* løses av de som tar IMT2001 (6 studiepoeng)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap. 29, 30 og 22 i læreboka.

a) Vi har grafen :



a1) **Skriv opp nabomatrisen** for dette tilfellet.

a2) **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.

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

AG GC CB BA FG BF AD DB AC EG

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) Vis **konstruksjonsprosessen** når bokas metode **for Huffman-koding** (s.324-330) brukes på teksten "DET ER KLASSE OVER DEN KLASSISKE SAMLINGEN TIL KLASSEN MED STJERNEKLASSIKERE" (inkludert de blanke – husk den etter "TIL"). 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.

Oppgave 3 (teori og koding)

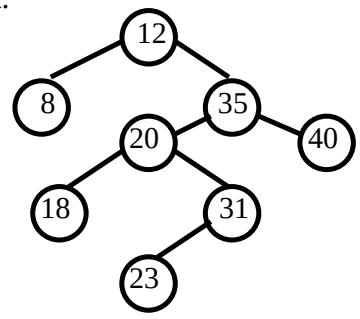
Et binære søketre kan bl.a. representeres ved hjelp av en tabell (`int tab[N][3]`), der en rad/linje i tabellen inneholder dataene om en bestemt node. Tabellen er indeksert fra 0 og oppover, og tabellens indeks er "adressen" til noden. N er max antall noder som kan være i treet. Hver rad inneholder tre dataelementer: Først (2.kolonne fra venstre) nodens verdi, så indeksen/adressen til venstre barn og sist (helt til høyre) indeksen/adressen til høyre barn. Hvis indeksen/adressen er -1 så betyr det at det ikke finnes noe slikt barn. I tillegg til tabellen `tab` har vi en enkelt-variabel `rot` som inneholder indeksen/adressen til rotnoden. Hvis den er -1, så betyr det at treet tomt.

Eks: Følgende tabell:

Indeks	Node	Venstre	Høyre
0	12	3	1
1	35	2	5
2	20	6	4
3	8	-1	-1
4	31	7	-1
5	40	-1	-1
6	18	-1	-1
7	23	-1	-1

Global variabel: `rot` 0

gir treet:



NB: Legg merke til at nodenes verdi (2.kolonne) bare kommer i en eller annen tilfeldig rekkefølge.

a) **(teller 40% av oppgave 3 NB: Oppgave 3a består av to deloppgaver!)**

Et slikt tre er representert ved tabellen nedenfor, og `rot` er 0. Verdien/noden 19 skal settes inn i dette treet. **Skriv det totale innholdet på de radene som blir endret/kommer i tillegg i denne forbindelse.** **Hint:** Det *kan* være til hjelp å tegne opp søketreet (ut fra tabellen) først.

Indeks	Node	Venstre	Høyre
0	10	3	1
1	25	2	7
2	15	-1	4
3	7	6	-1
4	21	5	-1
5	18	8	-1
6	4	-1	-1
7	30	-1	9
8	16	-1	-1
9	33	-1	-1

------- *slutt 1. deloppgave* -------

Tallene 16, 10, 15, 4, 33, 18, 30, 7, 21 og 25 er gitt. Disse skal legges inn i et binært søketre i denne rekkefølgen. **Skriv/tegn tabellrepresentasjon av dette treet** (slik som beskrevet ovenfor, og der *nodenes verdi blir liggende i denne rekkefølge* fortløpende fra indeks nr.0 og oppover).
Hint: Her kan det også være til hjelp å tegne opp søketreet først, før du lager selve tabellen.

b) **(teller 60% av oppgave 3)**

Lag funksjonen `void legg_inn(int v)`

Funksjonen skal legge den nye verdien 'v' (*alltid* større enn 0) inn i et slikt binært søketre, representert vha. en tabell som beskrevet ovenfor. Den må derfor finne ut hva som er foreldre-/mor-noden til den nye noden/verdien, oppdatere dennes opplysninger om sitt nye barn, samt legge inn dataene om den nye noden i tabellen (på første ledige plass/rad). Funksjonen må også passe på om det i det hele tatt er plass til mer i tabellen. Den kan fritt bruke konstanten 'N' og de globale variablene `tab` og `rot`, men får *ikke* lov til å innføre flere globale variable.

Oppgave 4 (koding)

Lag funksjonen

```
void lengste_sorterte_sekvens(int arr[], int N, int & start, int & len)
```

Funksjonen skal finne den lengste stigende sorterte delsekvensen av arrayen 'arr' (som har indekser fra 0 til N-1. Den oppdaterer de referanseoverførte variablene 'start' og 'len' med henholdsvis indeksen for starten på denne sekvensen og dets lengde. Om to eller flere slike delsekvenser er like lange, så returneres det med indeksen og lengden for den som ligger lengst til venstre (først i arrayen). Like elementer/verdier regnes som å være i stigende rekkefølge.

Eks: Vi har: int a[] = { 1, 3, 5, 2, 7, 9, 3, 8, 1, 10, 5, 2, 5, 7, 9, 11, 4, 5, 5, 10, 15 }
Lengste stigende sorterte delsekvens vil da være: 2 5 7 9 11 (indeks fra 11 og lengde 5)
Men, 4 5 5 10 15 vil også være på fem elementer, men den er *ikke lengst til venstre*.

NB 1: Løsningen trenger (og bør nok) *ikke* være rekursiv!

(Om arrayen er *baklengs* sortert, med ingen like elementer, så vil altså funksjonen (om du har laget/kodet den rett) returnere med 'start' satt til 0 (null) og 'len' lik 1.)

Lykke til !
frode@haug.com