Høgskolen i Gjøvik
Avdeling for teknologi

E K S A M E N

FAGNAVN: **Algoritmiske metoder**

FAGNUMMER: **IMT2021 og IMT2001**

EKSAMENSDATO: **12. desember 2003**

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. 16, 9 og 11 i læreboka.

a) Koden s.237 og teksten s.239 i læreboka gir oss følgende kode ved dobbel hashing:

```
const int M = 11;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (7 - (k % 7));  }

void insert(itemType v, infoType info)  {
   int x = hash1(v);
   int u = hash2(v);
   while (a[x].info != infoNIL)  x = (x+u) % M;
   a[x].key = v;  a[x].info = info;
}
```

"k" står for bokstavens nummer i alfabetet (1-29). Vi har en array som er 11 lang (indeks 0-10). Keyene "TEGNESERIER" skal legges inn i denne arrayen. **Skriv opp hver enkelt key's returverdi fra både hash1 og hash2. Tegn også opp arrayen for hver gang en ny key legges inn.** (Innholdet i "info" trenger du ikke å ta hensyn til.)

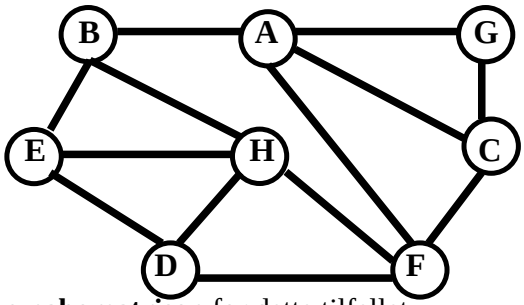
b) Du skal utføre Quicksort på teksten "S L I M B I L L E N" (blanke regnes ikke med). **Lag en figur tilsvarende fig. 9.3 side 119 i læreboka**, der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

c) Teksten "S L I M B I L L E N" (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.)

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 lærebok.

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 BF DC EH GF HG FH EF

Vi skal nå utføre union-find m/path compression og weight balancing på denne grafen. **Tegn opp arrayen "dad"s innhold etterhvert** som skogen bygges opp (jfr. fig.30.9) vha. "find"-funksjonen s.447 i læreboka. Ut fra dette: **tegn også opp den resulterende union-find skogen** (dvs. noe lignende til nedre høyre skog i fig.30.8).

c) Vis **konstruksjonsprosessen** når bokas metode **for Huffman-koding** (s.324-330) brukes på teksten "TORSKEN ER DORSK TOSKEN ER SVORSK OG TOKEN ER IKKE NORSK" (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.

Oppgave 3 (koding)

Denne oppgaven går ut på å gjøre om et *binært søketrær* til en *toveis sirkulær liste*. I en toveis liste vil en gitt node ha pekere både til den som kommer etter den i listen, men også til den som kommer foran den i listen. At den er sirkulær betyr at den "første sin forrige" er den siste noden i listen, og at den "siste sin neste" er den første noden i listen. Hver node i treet/listen er definert på følgende måte:

```
struct node  {
  char key;                  //  Nodens navn/ID.
  node *mindre, *storre;     //  Pekere til venstre/høyre subtre evt.den som
};                           //    kommer før/etter i toveis sirkulær liste.
```

I utgangspunktet har vi altså et binært søketre, der 'root' peker til rota i dette treet, og der alle tomme subtrær består av at det pekes til 'z'-noden. Hvordan dette treet har blitt bygd opp trenger du ikke å bekymre deg med/tenke på. For å kunne gjøre om dette binære søketreet til en toveis sirkulær liste, så trengs det spesielt to funksjoner. Det er disse to du her skal lage/kode:

a) **(teller 40% av oppgave 3)**

Lag funksjonen `node* skjot_sammen(node* liste1, node* liste2)`

Funksjonen får *to* sirkulære lister inn som parametre. Den skal sørge for at disse blir skjøtet sammen til *en* sirkulær liste, og at det returneres en peker til starten på denne sammenslåtte listen.

NB: Husk også på å ta hensyn til at listene kan være tomme (at det bare pekes til 'z'-noden).

b) (teller 60% av oppgave 3)

Lag den *rekursive* funksjonen `node* lag_liste(node* rot)`

Funksjonen gjør om treet med rota i 'rot' til en toveis sirkulær liste, og returnerer en peker til starten på denne listen. **NB:** Husk å ta hensyn til at 'rot' bare peker til 'z'-noden.

Hint: Den gjør rekursivt om henholdsvis venstre og høyre subtre til to toveis sirkulære lister. Etter den venstre listen skjøtes rota (gjør denne om til en en-nodes toveis sirkulær liste). Skjøt så den høyre toveis sirkulære listen etter dette igjen. Til alt dette siste bruker du funksjonen fra oppgave 3a.

Oppgave 4 (koding)

Et palindrom er en tekst som er symmetrisk om midten. Hvis vi deler den i to halvparter, så vil den siste delen være lik den første bare i motsatt rekkefølge. Eller sagt med helt andre ord: et palindrom er likt forlengs som baklengs. Noen eksempler på slike norske ord kan være guttenavnet "otto" eller fuglen "stillits" (det finnes faktisk en type som heter *det*!). Det lengste norske enkeltordet er "regninger". (Verdens lengste palindrome enkeltord er det finske ordet for luthandler ; "saippuakivikauppias".)

Men, når vi i en lengre sammensatt tekst kun tar hensyn til bokstavene, dvs. alle blanke og andre spesialtegn ignoreres, samt at vi lar store og små bokstaver telle likt, så vil faktisk også følgende norske tekster være palindrom: "Det regna vel i Levanger, Ted?" og "Rolf Are vurderer om Arons ni drag i gardinsnora morer edru Vera Flor".

Vi har gitt følgende program:

```
#include <iostream>     //  cout, cin
#include <cstring>      //  strlen
#include <cctype>       //  isalpha, toupper
using namespace std;

bool palindrom(char txt[], int start, int slutt)
  {   // Lag innmaten   }

int main()  {
  char tekst[80];
  cout << "Tekst:  ";  cin.getline(tekst, 80);
  cout << "\n\nDette er "
       << (!palindrom(tekst, 0, strlen(tekst)-1) ? "IKKE " : "")
       << "et palindrom!\n\n\n";
  return 0;
}
```

Din oppgave er å skrive innmaten til den *rekursive* funksjonen `bool palindrom(…)` ovenfor.

Funksjonen må finne ut om det som er i indeksene 'start' og 'slutt' i arrayen 'txt' er likt, eller om et av dem skal ignoreres (da det ikke er en bokstav), og må rekursivt sjekke dette for de andre indeksene.

Husk at:

- det er nok å sjekke de to "halvdelene" ift. til hverandre.
- alle tegn som *ikke* er bokstavene A-Z eller a-z skal ignoreres (dette gjøres vha. funksjonen `bool isalpha(c)`. Denne returnerer true/false til om 'c' er blant bokstavene A-Z eller a-z).
- store og små versjoner av samme bokstav skal telle likt (bruk funksjonen `char toupper(c)`).

NB1: Teksten som sendes inn av main ('tekst') skal være *identisk* når den kommer tilbake. Dvs. det er ikke lov å fjerne tegn som ikke er bokstaver, eller å gjøre om alt til store/små bokstaver.

NB2: Om du vil ha *full* uttelling/score på denne oppgaven (da den faktisk er "litt lett"), så må du inne i `palindrom` *ikke* bla forbi ikke-bokstaver vha. løkker, men sørge for at rekursive kall gjør dette.

Lykke til !

frode@haug.com