Høgskolen i Gjøvik
Avdeling for informatikk og medieteknikk

E K S A M E N

EMNENAVN: **Algoritmiske metoder**

EMNENUMMER: **IMT2021**

EKSAMENSDATO: **20. desember 2011**

KLASSE(R): **10HB - INDA / PUA / DRA / ISA / SPA**

TID: **09.00-14.00**

EMNEANSVARLIG: **Frode Haug**

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

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

- Kontroller at alle oppgavearkene er til stede.
- 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.

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.11, 15 og 16 i læreboka.

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

98 87 72 49 73 70 70 45 40 46 42 31 39

Utfør etter tur følgende operasjoner på denne heap: insert(87), insert(86), remove(), remove() og replace(24). **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.

b) Legg key'ene "CAMBRIDGESTUDENT" (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.

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

```
const int M = 13;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (4 - (k % 4));  }

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;
}
```

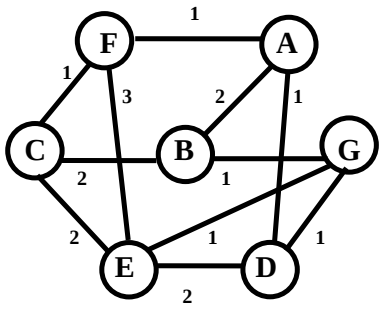
"k" står for bokstavens nummer i alfabetet (1-29). Vi har en array som er 18 lang (indeks 0-17). Keyene "CAMBRIDGESTUDENT" skal legges inn i denne arrayen. **Skriv opp hver enkelt key's returverdi fra både hash1 og hash2. Tegn også opp arrayen for hver gang en ny key legges inn.** (Innholdet i "info" trenger du ikke å ta hensyn til.)

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.22, 31 og 33.

a) LZW-teknikken skal brukes på følgende tekststreng: "SOSS SOSSER SOM SOMRE "
Katalogen inneholder allerede alle de standard ASCII-tegnene i indeksene 0-255.
Hva blir katalogens innhold f.o.m. indeks nr.256 og oppover?
Hva blir den kodede (output) strengen?

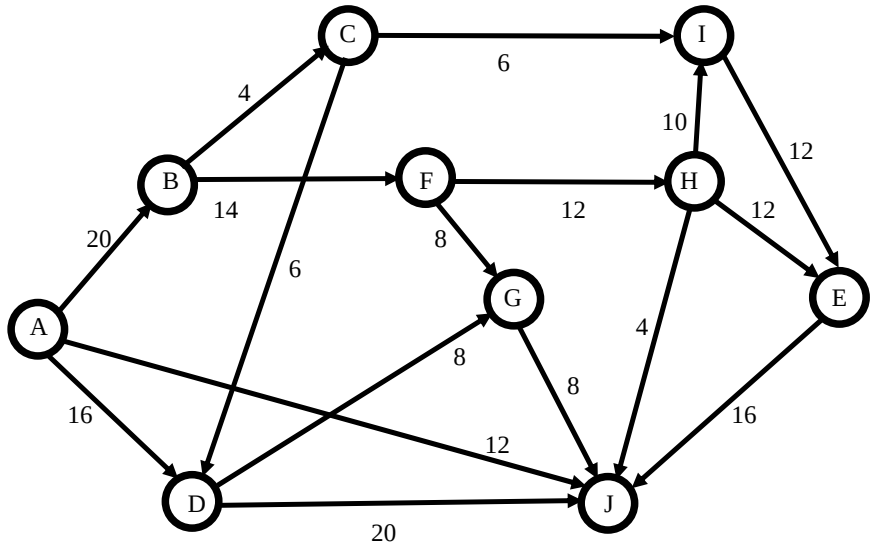
b) Følgende vektede (ikke-rettede) graf er gitt:



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.

c) Vi har følgende nettverk (dvs. en graf som er både vekted og retted):



'A' er kilden og 'J' er kummen. Kapasiteten for hvert enkelt rør/kant er skrevet på figuren, og flyten i røret går i pilretningen. **Tegn figuren opp igjen, men nå med ny verdi på hver kant, som forteller hva som er max. flyt langs hver av kantene (fra kilde til kum).**
Hva er max. flyt fra kilde til kum?

Oppgave 3 (koding, 30%)

Vi har et *binært søketre* bestående av nodene:

```
struct Node  {
  int   ID;       //  Nodens ID/key/nøkkel/navn (et tall).
  Node* left;     //  Peker til venstre subtre, evt. z når tomt.
  Node* right;    //  Peker til høyre subtre, evt. z når tomt.
  Node (int id, Node* l, Node* r)  //  Constructor:
      { ID = id; left = l; right = r; }
};
```

Vi har også de to globale variablene:

```
Node* z   = new Node(0, NULL, NULL);    // z-noden (ID = 0).
Node* rot = z;                          // Rot-peker.
```

I *hele* denne oppgaven skal det *ikke* innføres flere globale variable eller struct-medlemmer enn det gitt ovenfor. **Hint:** Tegn opp et litt større tilfeldig binært søketre, så er det lettere å studere/tenke på hvordan de ulike funksjonene skal operere. Treet *kan* inneholde duplikate noder (noder med lik ID).

a) **Lag den *ikke-rekursive* funksjonen** `void hoyreGren()`

Funksjonen skal skrive ut `ID`'en til *alle* nodene som ligger på stien fra rota og ned til den bladnoden (har to z-barn) som ligger *lengst til høyre av alle bladnoder*. Husk at en bladnode trenger *ikke* å være høyre-barn av sin mor.

b) **Lag den *ikke-rekursive* funksjonen** `void gren(Node* p)`

Parameteren `p` peker til en helt *vilkårlig bladnode ett eller annet sted i treet*. Funksjonen skal skrive ut `ID`'en til *alle* nodene som ligger på stien fra rota og ned til denne bladnoden. Husk at siden treet *kan* inneholde duplikate noder, så kan det finnes noder lengre opp i treet som har samme `ID` som den bladnoden vi leter etter. Funksjonen skal også håndtere dette.

c) **Lag den *rekursive* funksjonen** `void grener(Node* p)`

Funksjonen skal sørge for at stiene ned til *alle* bladnoder i *hele* treet blir skrevet ut. Lag et linjeskift i utskriften etter at hver sti er skrevet. Kalles fra `main` ved: `grener(rot);`

Hint: Sikkert lurt at den kaller minst en av funksjonene laget i oppgave 3a og/eller 3b.

Oppgave 4 (koding, 20%)

Vi har en "mengde" med positive heltall. Med følgende algoritme skal vi håndtere disse:

1. Er det to eller flere tall i mengden, **plukk ut** *to* av tallene. Vi kaller disse x og y. (Er det bare *ett* tall i mengden, gå til punkt nr.5.)
2. **Beregn (x + y) * 2** (dvs. summer tallene x og y, og multipliser svaret med to).
3. **Erstatt x og y** i mengden med dette totalsvaret (fra punkt nr.2).
4. Er det fortsatt **to eller flere tall** i mengden, **gjenta** fra punkt nr.1 igjen.
5. **Svaret/resultatet** er det *ene* tallet som nå er i mengden.

Vi er nå interessert i å **finne det maksimale tallet** som kan bli svaret etter denne algoritmen. Alle tallene er lagret i arrayen `arr`. I denne er `N` elementer i bruk (indeks 0 til `N`-1).

Det finnes ihvertfall to (helt uavhengige) måter vi kan tenke å løse dette på:

a) Vi kan permutere tallene på alle tenkelige måter (jfr. bl.a: EKS_06, oppgave nr.7 og 8 i heftet). Hver gang vi har funnet en ny permutasjon, dvs. ifm. kode-grenen `if (i == N-1)`, utfører vi algoritmen angitt ovenfor. Det maksimale svaret på alle disse beregningene, er svaret på oppgaven vår. **Definer global variabel og skriv koden inni if-setningen som besvarer/ løser oppgaven/problemet.** Du trenger *ikke* å tenke på at avskjæringer kanskje bør legges inn andre steder i koden (EKS_06). Men legg vekt på at den koden du skriver er rask og effektiv.

b) Tenk ut en **bedre, raskere og mer effektiv måte å velge ut elementer** fra mengden på, slik at resultatet blir maksimalt. **Beskriv måten/algoritmen du vil arbeide etter, og skriv kode for (implementer) denne.** Du kan forutsette at all kode i læreboka er ferdig skrevet og fritt kan brukes/kalles, spesielt gjelder dette ulike sorterings-algoritmer og klasser som: `Stack`, `Queue` og `PQ` (heap). *Om* du bruker slik kode, henvis til hvor den står i læreboka.

I *hele* dette oppgavesettet skal det *ikke* brukes ferdig kode fra (standard)biblioteker (STL).

Løkke tæll!

frode@haugianerne.no