Høgskolen i Gjøvik
Avdeling for informatikk og medieteknikk

E K S A M E N

EMNENAVN: **Algoritmiske metoder**

EMNENUMMER: **IMT2021**

EKSAMENSDATO: **18. desember 2013**

KLASSE(R): **12HB - IDATA / PUA / DRA / ISA / SPA**

TID: **13.00-18.00**

EMNEANSVARLIG: **Frode Haug**

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

TILLATTE HJELPEMIDLER: **Alle trykte og skrevne**
(kalkulator er *ikke* tillatt)

- Kontroller at alle oppgavearkene er til stede.
- Innføring med penn, eventuelt blyant 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.
- Husk kandidatnummer på alle ark.

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.3, 11, 14, 15 og 12 i læreboka.

a) Koden øverst side 28 i læreboka leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: ((((3 + 2) * (4 * 3)) * (5 + 3)) * (2 + 3))
Hva blir dette skrevet på en postfix måte ?
Tegn opp innholdet på stakken etter hvert som koden side 28 leser tegn i infix-uttrykket.

b) I de følgende deloppgaver er det key'ene "D I R E S T R A I T S"
(i denne rekkefølge fra venstre mot, og blanke regnes ikke med) som du skal bruke.
For alle deloppgavene gjelder det at den initielle heap/tre er *tom* før første innlegging ("Insert") utføres. **Tegn (skriv) den *resulterende* datastruktur når key'ene legges inn i:**

1) **en heap**
2) **et binært søketre**
3) **et 2-3-4 tre**
4) **et Red-Black tre**

c) Vi skal utføre *rekursiv* Mergesort på key'ene "DIRESTRAITS". Jfr. kode s.166 i læreboka.
For hver gang tredje og siste for-løkke i koden s.166 er ferdig:
Tegn opp arrayen med de key'ene som har vært involvert i sorteringen (jfr. fig.12.1 s.167)

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.16, 30 og 31.

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

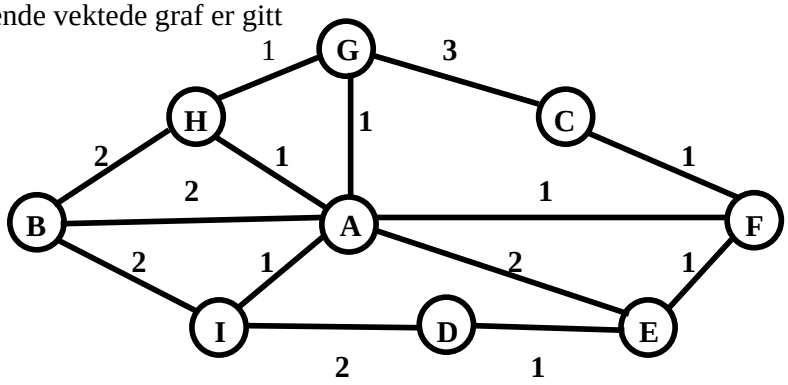
```
const int M = 17;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (4 - (k % 4));  }

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 17 lang (indeks 0-16). Keyene "GARRICKTHEATRE" 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) Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:
FG CB DB EA AG DF GB
Vi skal nå utføre *union-find m/weight balancing og path compression* 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) Følgende vektede graf er gitt



Hver nodes naboer er representert i en *naboliste*, der disse er *sortert alfabetisk*.
Koden s.455 i læreboka utføres på denne grafen – der "priority" er lik "val[k] + t->w".
Hvilke kanter er involvert i korteste-sti spenntreet fra "C" og til *alle* de andre nodene?
Tegn også opp innholdet i prioritetskøen etterhvert som koden utføres (jfr. fig.31.13 i læreboka). **NB:** Husk at ved lik prioritet så vil noden sist oppdatert (nyinnlagt eller endret) havne først i køen.

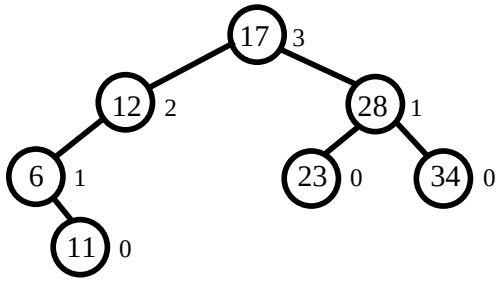
Oppgave 3 (koding, 30%)

Vi har et *binært søketre* bestående av nodene:

```
struct Node  {
  int   ID;      //  Nodens ID/key/nøkkel/navn (et tall).
  int   height;  //  Nodens høyde ift. dypeste subtre/lengste sti.
  Node* left;    //  Peker til venstre subtre, evt. NULL når tomt.
  Node* right;   //  Peker til høyre subtre, evt. NULL når tomt.
  Node* parent;  //  Peker oppover igjen til forelder/mor,
                 //           evt. NULL om er rota.
  Node (int id, Node* p)    //  Constructor:
     { ID = id;  parent = p;  height = 0;  left = right = NULL; }
};
```

Vi har også den globale variabelen:
`Node* rot = NULL; // Rot-peker` (har altså *ikke* at `head->right` er rota).

`parent` peker *alltid* til nodens mor/forelder (`rot->parent` er `NULL`) . `height` er nodens høyde ift. fjernest nedenforliggende bladnode. Dvs. stilengden ned til denne fjerneste bladnoden under.
I *hele* denne oppgaven skal det *ikke* innføres flere globale data eller struct-medlemmer enn det gitt ovenfor. Eksempeltre (der hver kant/link representerer *både* `parent` og `left/right`):



Tallet rett til høyre for hver node er dens `height` (bladnoder har høyde lik 0 (null)). Legger vi f.eks. inn en node med verdien '31' vil nodene ovenfor ('34' og'28', men *ikke* '17') øke `height` med +1. Legger vi derimot inn en node med verdien '4', vil bare denne få `height` lik 0, da de ovenfor allerede har `height` ift. en annen node som allerede er minst på samme nivå. Det samme gjelder for en node med verdi lik '15' (denne får `height` lik 0, mens mora allerede har `height` lik 2).

a) **Lag den *ikke-rekursive* funksjonen** `void insert(int id)`

`id` er `ID` for den nye noden som skal legges inn i treet (duplikate noder må gjerne forekomme). Funksjonen legger inn den nye noden (*alltid* som ny bladnode), og (om nødvendig) øker *aktuelle* noders `height` oppover i treet med +1. Husk å spesialbehandle det tomme tre.

b) **Lag den *ikke-rekursive* funksjonen** `Node* distance(Node* p, int dist)`

Funksjonen finner (og returnerer) en node som det er en stilengde ned til som er *eksakt* `dist` lang. Er det flere slike noder (med `dist` i stilengde ned til), returneres den *første i preorder rekkefølge under* `p`. Du kan forutsette at `p` *ikke* peker til NULL, samt at det *alltid* finnes minst en node nedenfor som det minst er denne stilengden til.

c) **Lag den *ikke-rekursive* funksjonen** `void longestPath(Node* p)`

Funksjonen starter hos `p`, går gjennom den lengste stien under denne og skriver nodenes `ID`. Har en nodes to subtrær lik `height`, velges subtreet som besøkes *sist i preorder rekkefølge*.

Oppgave 4 (koding, 20%)

I Norge har vi primært 8-sifrede telefonnumre. Men, vi har også både 3-, 4- og 5-sifrede. Det er da selvsagt viktig at ingen av disse numrene er prefikser av hverandre. F.eks. så lenge vi har nødnummeret
113, så kan *ingen* andre norske telefonnumre starte med disse tre sifrene.
Vi har en *meget* lang array (dog er det plass til den i maskinens primærhukommelse) med slike numre. Du skal lage en funksjon som returnerer `true` med en gang den oppdager at arrayen (som kommer inn som parameter) inneholder minst ett prefiks-tilfelle. Før den returnerer skriver den ut de to numrene som var involvert i at noe er prefiks av noe annet. Er det ingen numre i *hele* arrayen som er prefikser av hverandre, returnerer den (selvsagt) `false`.

Lag altså funksjonen `bool prefix( <en array> )`

Bestem selv, og angi som forutsetning i besvarelsen din:

1) **hvilken datatype arrayen er av (`int` eller `char`)**
2) **etter hvilke kriterie(r) den er ferdig sortert i det den kommer inn som parameter**

Forklar også hvilken metode/strategi/algoritme du har brukt under kodingen.

I *hele* dette oppgavesettet skal du *ikke* bruke string-klassen eller ferdig kode fra (standard)biblioteker (slik som bl.a. STL).

Løkke tæll! (Ikke ring meg, jeg ringer dere ☺)

frode@haug.es