Høgskolen i Gjøvik
Avdeling for informatikk og medieteknikk

E K S A M E N

EMNENAVN: **Algoritmiske metoder**

EMNENUMMER: **IMT2021**

EKSAMENSDATO: **14. desember 2015**

KLASSE(R): **14HB - IDATA / 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**
(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, 9, 11, 14 og 15 i læreboka.

a) Koden øverst side 28 i læreboka leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: (((4 + 2) + (3 * 2)) + ((3 + 4) * (2 * 5)))
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 "L I G A M E S T E R"
(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**

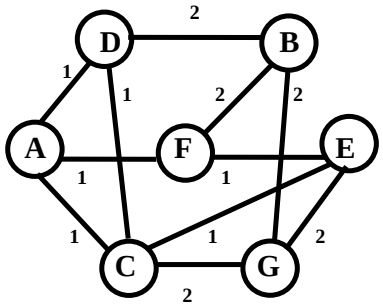
c) Du skal utføre Quicksort på teksten "L I G A M E S T E R" (blanke regnes ikke med).
Lag en figur tilsvarende fig. 9.3 side 119 i læreboka, der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

Oppgave 2 (teori, 25%)

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

a) LZW-teknikken skal brukes på følgende tekststreng: "MO MOS MOSE MOST "
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?

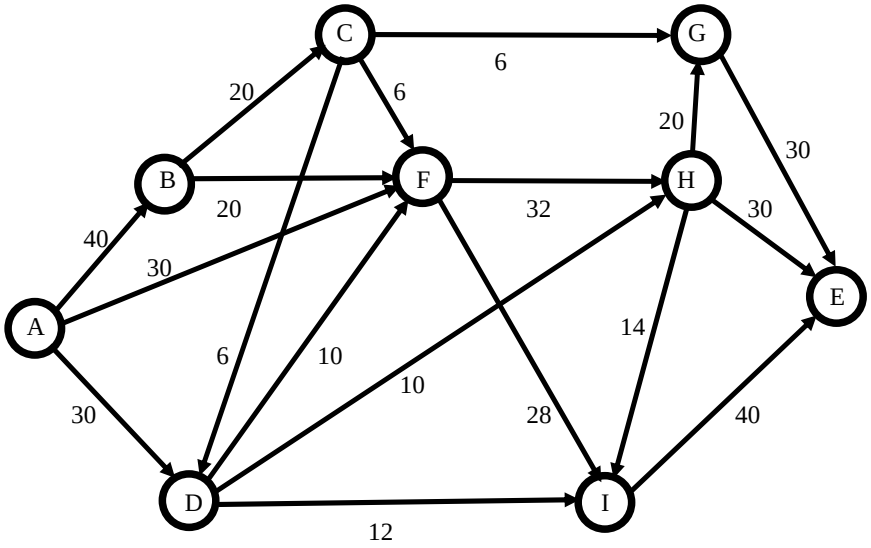
b) Følgende vektede (ikke-rettede) graf er gitt:



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", og det startes i node 'A').

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 'E' 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. NULL når tomt.
  Node* right;       //   Peker til høyre subtre, evt. NULL når tomt.
  Node (int id) { ID = id;  left = right = NULL; }
};
```

Vi har også de to globale variablene:

```
Node* rot   = NULL;    //   Rot-peker (har altså ikke at head->right er rota).
int   hoyde = 0;       //   Treets nåværende høyde.
```

«Tomme pekere» peker altså *ikke* til en `z`-node, men til `NULL`.
Rota ligger på *nivå* nr.0, dets barn er på nivå nr.1, dets barnebarn er på nivå nr.2, osv.
Treets *høyde* er definert til å være nivået + 1 for den eller de fjernest/dypest liggende noden(e) ift. rota. Dvs. et tomt tre har høyde lik 0. Inneholder det *kun* rota (altså på nivå nr.0), er høyden 1. Er det *kun* rota som har ett eller to barn (på nivå nr.1), er høyden 2, osv.

NB: I *hele* oppgave 3 skal det *ikke* innføres flere globale data eller `struct`-medlemmer enn angitt ovenfor. Det skal heller *ikke* brukes andre hjelpestrukturer, som stakk, kø eller liste.

a) **Lag den *ikke-rekursive* funksjonen** `void insert(int id)`
Funksjonen setter inn en ny node (med `ID` lik `id`) i treet tilpekt av `rot` (husk at den også *kan* peke til `NULL`). *Om* den nye noden legges inn på et nivå som er *en* høyere enn høyeste nivå hittil brukt, *må* funksjonen også oppdatere den globale variabelen `hoyde`.

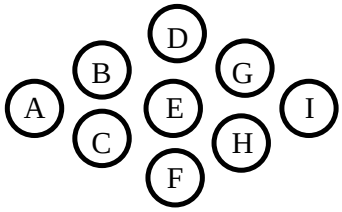
b) **Lag den *ikke-rekursive* funksjonen** `Node* mindreEnn(int v)`

Funksjonen skal returnere en peker til den noden som har den siste `ID`en i treet som er *mindre enn* `v`. Dvs. *den siste* som kommer *før* `v` i inorder rekkefølge. Parameteren `v` *kan*, men *behøver ikke* å være i treet. Er treet tomt, eller finnes det ingen verdier mindre enn `v`, returneres `NULL`. **Kommenter koden din godt, slik at det kommer klart frem tankegangen bak/for koden.**

c) **Lag den *ikke-rekursive* funksjonen** `void verdierPaaNivaa(int niva)`
og den *rekursive* funksjonen `void nivaaIDer(Node* p, int k, int niva)`

Den første funksjonen skriver bare en melding om `niva` er utenfor relevant område for det nåværende treet. I motsatt fall kalles den andre rekursive funksjonen med relevante parametre. `p` er aktuell node å besøke/visitere. `k` er nåværende nivå. `niva` er den samme som parameteren inn til den første funksjonen. Denne funksjonen skal derfor, om `p` *ikke* er `NULL`, rekursivt være med å besørge at hele treet blir besøkt, dog ikke lengre ned enn til `niva`. Den sørger også for at *alle* nodene som ligger på `niva` får skrevet ut sin ID. **NB:** Det er altså *kun* den første funksjonen som kalles fra `main` med ønsket nivå som parameter.

Oppgave 4 (koding, 20%)



Bokstavene A-I skal erstattes med tallene 1-9 (*alle* tallene skal brukes *en* gang hver), slik at *summen av*
alle en nodes naboer er heltallig delelig med nodens eget tall/verdi (dvs. ingen rest når summen av naboene deles med nodens tall/verdi).
Eks. på naboer: A har naboene B og C. B har naboene A, C, E og D. C har naboene A, B, E og F. D har naboene B, E og G. E har naboene B, D, G, H, F og C. Tilsvarende for nodene F, G, H og I.
Skriv et komplett program som finner *alle* løsninger på problemstillingen.
Legg vekt på å ikke gå videre med og lete etter løsninger når en nodes verdi ikke tilfredsstiller kravet. F.eks: har B verdien 3, og summen av dens naboer er 16, er dette ikke heltallig delelig. Det er da ikke noe poeng å prøve og sette verdier i nodene F, G, H og I.

En del løsninger programmet finner vil egentlig bare være speilinger omkring den vertikale (D, E, F) eller den horisontale (A, E, I) aksen. **Hvordan kan slike siles vekk, *uten* å lagre unna tidligere funne løsninger? Forklar kort** (du trenger *ikke* å skrive kode for dette).

NB: Kan du gjøre bruk av allerede eksisterende kode i lærebok, eksempler fra pensum (EKS_xx.CPP) og/eller løsningsforslag på oppgaver i læreboka/det grønne heftet, er det bare å henvise til dette.

NB: I *hele* dette oppgavesettet skal du *ikke* bruke string-klassen eller ferdig kode fra (standard-) biblioteker (slik som bl.a. STL). Men, de vanligste includer og funksjonene vi brukte i hele 1.klasse er tilgjengelig. Kode skal skrives i C++.

Løkke tæll!

frode@haug.es