Høgskolen i Gjøvik
Avdeling for informatikk og medieteknikk

E K S A M E N

EMNENAVN: **Algoritmiske metoder**

EMNENUMMER: **IMT2021**

EKSAMENSDATO: **15. desember 2014**

KLASSE(R): **13HB - 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.8, 9 og 15 i læreboka.

a) Vi har koden (s.112) for distribuert telling. Teksten som skal sorteres (blanke regnes ikke med): "A A B C D D D B C A B B C A" (dvs. M = 4, N = 14). **Angi innholdet i arrayen `count`:**

- **etter at 2.for-løkke er ferdig utført**
- **etter at 3.for-løkke er ferdig utført**
- ***pluss* innholdet i arrayen `b` i det 4.for-løkke har gått halvveis**

b) Du skal utføre Quicksort på teksten "P R O S E D Y R E" (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) Legg key'ene "PROSEDYREKONKURRANSE" (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, 25%)

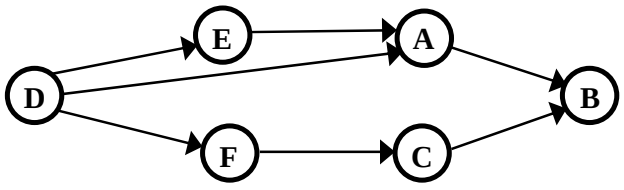
Denne oppgaven inneholder tre uavhengige oppgaver fra kap.22, 30 og 32 i læreboka.

a) Vis **konstruksjonsprosessen** når bokas metode **for Huffman-koding** (s.324-330) brukes på teksten "IPSWICH MOT LEEDS SAMT WEST HAM MOT SWANSEA ER MORSOMT" (inkludert 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 og "len" (jfr. fig.22.7 og koden øverst s.329).
- totalt antall bits som brukes for å kode teksten.

b) Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:
BA CB EF FD AF CD BF
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 rettede (ikke-vektede) asykliske graf («dag») er gitt:



Angi ***alle* mulige topologiske sorteringssekvenser** av nodene i denne grafen.

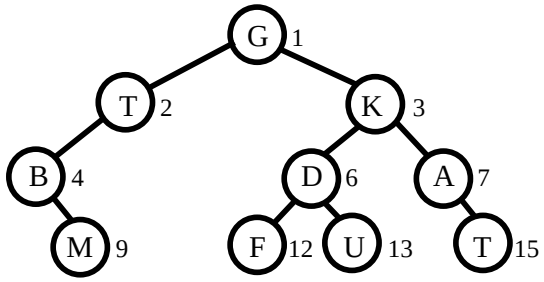
Oppgave 3 (koding, 30%)

Vi har et *binært* tre (*ikke* nødvendigvis *søketre*) bestående av nodene:

```
struct Node  {
  char  ID;    //  Nodens ID/key/nøkkel/navn (et tegn).
  int   pos;   //  Posisjon i treet.
  Node* left;  //  Peker til venstre subtre, evt. NULL når tomt.
  Node* right; //  Peker til høyre subtre, evt. NULL når tomt.
  Node (char id, int p)  //  Constructor:
     { ID = id;  pos = p;  left = right = NULL; }
};
```

Vi har også den globale variabelen:
`Node* rot = NULL; // Rot-peker` (har altså *ikke* at `head->right` er rota).
«Tomme pekere» peker altså *ikke* til en `z`-node, men til `NULL`.

`pos` er nodens posisjon i treet. Rota er nr.1. Om en node har pos lik `i`, vil et evt. venstre barn ha `pos` lik `2*i`, mens et evt. høyre barn ha `pos` lik `2*i + 1`. Altså etter samme prinsipp som i en heap (representert som en array), bare at subtrær/barn kan mangle (treet trenger ikke være fullt eller komplett). Eksempel på et slikt tre, der `pos`-verdier (etter reglene ovenfor) er skrevet til høyre for noden (noder må gjerne ha duplikat `ID`):



a) **Lag de to *rekursive* funksjonene** `void speilvend(Node* p)` og `void settPosisjoner(Node* p, int pos)`

Den første funksjonen skal speilvende noden `p` og alle dens subtrær/barn. Dvs. bytte om på sine venstre og høyre subtrær/barn, og at dette skjer rekursivt videre nedover. Kalles fra `main` med: `speilvend(rot);` Når dette har skjedd vil ikke `pos` lengre være korrekt i nodene. Den andre funksjonen skal derfor rekursivt gå gjennom noden med alle dets subtrær/barn, og sette rett verdi for `pos` i alle nodene igjen. Kalles fra `main` med: `settPosisjoner(rot, 1);`

b) **Lag den *rekursive* funksjonen** `void speilvend(Node* p, int pos)`

Funksjonen skal *alene* utføre det samme som *begge* funksjonene ovenfor til sammen utfører. Kalles fra `main` med: `speilvend(rot, 1);`

c) **Lag den *ikke-rekursive* funksjonen** `Node* finnNode(int pos)`

Funksjonen starter å lete i rota. Den returnerer en peker til noden som har klassemedlemmet `pos` lik parameteren `pos`. Finner den ikke en slik node, returneres `NULL`.
Forklar/redegjør klart for hvordan funksjonen din virker (dens tankegang/algoritme).
Hint: Binærkoden for `pos` er vesentlig.

NB: I *hele* oppgave 3:

- *kan* `p` inn til funksjonen og `rot` være `NULL`.
- skal det *ikke* innføres flere globale data eller `struct`-medlemmer enn angitt ovenfor.

Oppgave 4 (koding, 20%)

Et *perfekt tall* er et tall som selv er summen av alle tall det heltallig er delelig med (når vi ser bort fra tallet selv). Det første perfekte tallet er 6 (= 1 + 2 + 3). Det andre er 28 (= 1 + 2 + 4 + 7 + 14). Det tredje er 496 (= 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248). De tre neste er: 8128, 33.550.336 og 8.589.869.056 Mao: det blir fort en stor verdi, selv om det bare er snakk om det sjette slike tallet.

Skriv et komplett program som beregner/finner de fem første slike perfekte tall.
(Det 6.tallet tar det altfor lang tid for dagens PCer å beregne.)

Vektlegg kode som er: kort, elegant, effektiv/avskjærende.

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 brukt i 1.klasse er tilgjengelig.

Løkke tæll!

frode@haug.es