Høgskolen i Gjøvik
Avdeling for informatikk og medieteknikk

Kontinuasjonseksamen

EMNENAVN: **Algoritmiske metoder**

EMNENUMMER: **IMT2021**

EKSAMENSDATO: **10. august 2010**

KLASSE(R): **08HB - IND* / 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.

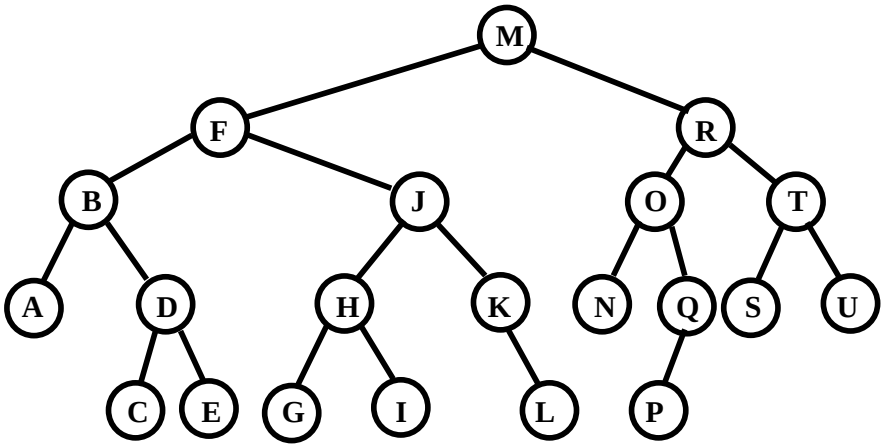
Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.9, 11 og 14 i læreboka.

a) Du skal utføre Quicksort på teksten ”R A N G E R S” (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.

b) Teksten ”E M I R A T E S C U P” (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.)

c)



Du skal nå fjerne («remove») noen noder fra dette treet. **Tegn opp treet for hver gang og fortell hvilken av «if ..... else if ..... else»-grenene i koden s.210 som er aktuelle når du etter tur fjerner henholdsvis tegnene ’J’, ’Q’ og ’F’.**

NB: Du skal for hver fjerning *på nytt* ta utgangspunkt i treet ovenfor.
Dvs. på intet tidspunkt skal det fra treet være fjernet to/tre av bokstavene samtidig.

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.22, 30 og 31 i læreboka.

a) Vis **konstruksjonsprosessen** når bokas metode **for Huffman-koding** (s.324-330) brukes på teksten ”SILDESALATEN SYNES SPESIELT SALT OG SYRLIG SIDEN DEN SMAKTE SLIK” (inkludert de blanke – husk den mellom ”DEN” og ”SMAKTE”).
Hvor mange bits trengs for å kode denne teksten? Dvs. skriv/tegn opp:

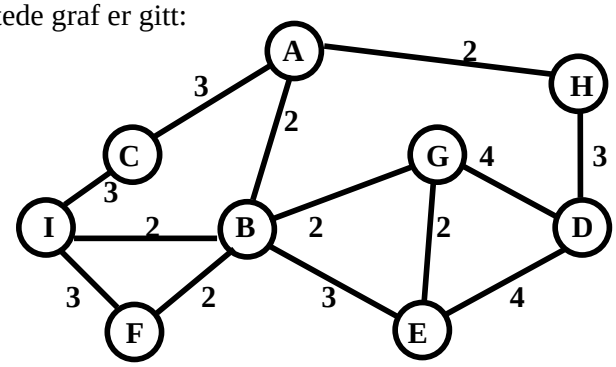
- tabellen for bokstavfrekvensen (jfr. fig.22.3).
- tabellen for ”dad”en (jfr. fig.22.6).
- Huffmans kodingstreet/-trien (jfr. fig.22.5 og koden øverst s.328).
- bokstavenes bitmønster og ”len” (jfr. fig.22.7 og koden øverst s.329).
- totalt antall bits som brukes for å kode teksten.

b) Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:

AE BC DF GA DB CE FE BF DG

Vi skal nå utføre *union-find m/weight balancing* (*ikke* 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 "I" 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, 32%)

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-noden om det er tomt.
  Node* right;    //  Peker til høyre subtre. Om tomt, peker den til
                  //     etterfølgende node/ID (i inorder rekkefølge)
                  //     evt. z-noden om noden selv er den aller siste.
  bool  harHoyreBarn;   //  'true' om har ekte høyrebarn.
  Node (int id, Node* l, Node* r, bool hHB)
      { ID = id;  left = l;  right = r;  harHoyreBarn = hHB;  }
};
```

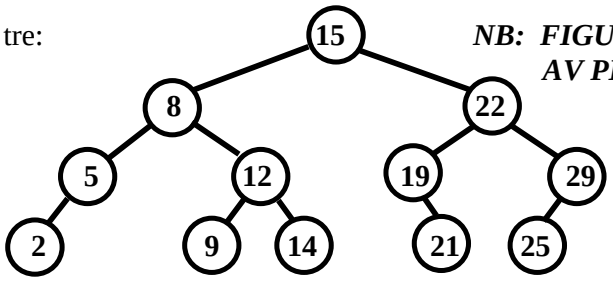
Vi har også de to globale variablene:

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

`left` og `right` peker til henholdsvis venstre og høyre subtre. Om venstre subtre ikke finnes (er tomt), så peker `left` til z-noden. Om høyre subtre er tomt, så brukes `right` på en noe spesiell måte: I denne oppgaven er vi alltid interessert i å se på nodene i *inorder* rekkefølge i treet. Når vi evt. bare får en peker til en gitt node (kanskje midt inne i treet), skal det uansett være lett å finne neste node i treet. For å enkelt kunne gjøre dette skal `right`, når den ellers hadde pekt til z-noden, peke til neste node i inorder rekkefølge. Dette vil *alltid* være oppover et sted i treet!

Datamedlemmet `harHoyreBarn` er 'false' dersom `right` er brukt på denne måten, og vil være 'true' dersom `right` peker til et reelt høyre subtre. Dersom en node har tomt høyre subtre og er den *aller* siste i treet, så vil `harHoyreBarn` være 'false' og `right` peke til z-noden.

Eksempel på et slikt tre:



NB: FIGUR MANGLER PÅTEGNING AV PILER FOR «HØYREBARN» IFT. NESTE I INORDER REKKEFØLGE!

a) Følgende verdier (i denne rekkefølge) settes inn i et binært søketre som følger prinsippene beskrevet ovenfor: 13, 21, 15, 5, 11, 8, 26, 14, 18, 20, 3
Lag en tegning av treet, der du også tegner inn hvordan `right` vil peke i de nodene som har tomt høyre subtre.

Lag funksjonen `Node* forste(Node* p)` **som returnerer en peker til den sekvensielt første noden (den med minst verdi) i subtreet under 'p'**.
Er treet tomt skal den returnere z-noden. (Du får her *ikke* bruk for bl.a. `harHoyreBarn`.)

b) **Lag funksjonen** `bool finn(int v)` **som returnerer true/false til om 'v' finnes i treet tilpekt av den globale `rot`.** (*Her* får du bruk for bl.a. `harHoyreBarn`.)

c) **Lag funksjonen** `void traverser(Node* p)` **som skriver ut *alle* verdiene i *hele* resten av treet, startende med 'p'.** 'p' kan altså peke nedi selve hovedtreet et sted. Til tross for dette skal *alle* sekvensielt etterfølgende verdier i *hele* treet skrives ut. Dette kan vi (selvsagt) få til (få tak i) da `right` og `harHoyreBarn` brukes som de gjør. Husk å håndtere tilfellet der 'p' initielt peker til z-noden. **Hint:** Du bør nok bl.a. bruke funksjonen du laget i oppgave 3a.
Funksjonen skal *ikke* være rekursiv!

d) **Lag funksjonen** `void legginn(int v)` **som legger inn en ny node** (med ID lik 'v') etter prinsippene beskrevet ovenfor. Husk å håndtere tilfellet der treet er helt tomt.
Hint: En ny node: 1) havner *alltid* inn nederst, 2) har *aldri* et ekte høyre-barn, 3) vil som oftest ha en annen eksisterende node som sin neste i inorder rekkefølge.

Oppgave 4 (koding, 18%)

Lag et komplett program (der det evt. er lov å henvise til annen ferdig kode i læreboka og/eller faglærers eksempler) **som finner alle mulige måter å representere tallet 100 på** ved å *alltid* sette inn *ett* addisjonstegn ('+') og *ett* divisjonstegn ('/') i omstokkingen av *alle* sifrene fra 1 til 9 ('+' kommer alltid før '/'). F.eks: 100 = 91 + 5742 / 638 eller 100 = 3 + 69258 / 714
Vektlegg effektivitet og avskjæringer.

Løkke tæll!

frode@haugianerne.no