NTNU

Eksamen

EMNENAVN:	**Algoritmiske metoder**
EMNENUMMER:	**IMT2021**
EKSAMENSDATO:	**19.desember 2016**
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 som gir gjennomslag på tre stk ark.
  Pass på så du ikke skriver på mer enn ett innføringsark om gangen
  (det blir uleselige gjennomslag når flere ark ligger oppå hverandre).
- Ved innlevering skilles hvit og gul besvarelse og legges i hvert sitt omslag.
- Oppgavetekst, kladd og blåkopi beholder kandidaten.
- Husk kandidatnummer på alle ark.

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder uavhengige oppgaver bl.a. fra kap.11, 14, 15 og 9 i læreboka.

a) **Hva blir utskriften fra følgende program**:

```
#include <iostream>
using namespace std;

void rekursiv(int n) {
     if (n < 1)  cout << '\n';
     else  {   cout << ' ' << n % 5;
               rekursiv(n / 4);  }
}

int main()  {  rekursiv(100);   return 0;  }
```

b) I de følgende deloppgaver er det key'ene "L E E D S U N I T E D"
(i denne rekkefølge fra venstre mot høyre, 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:**

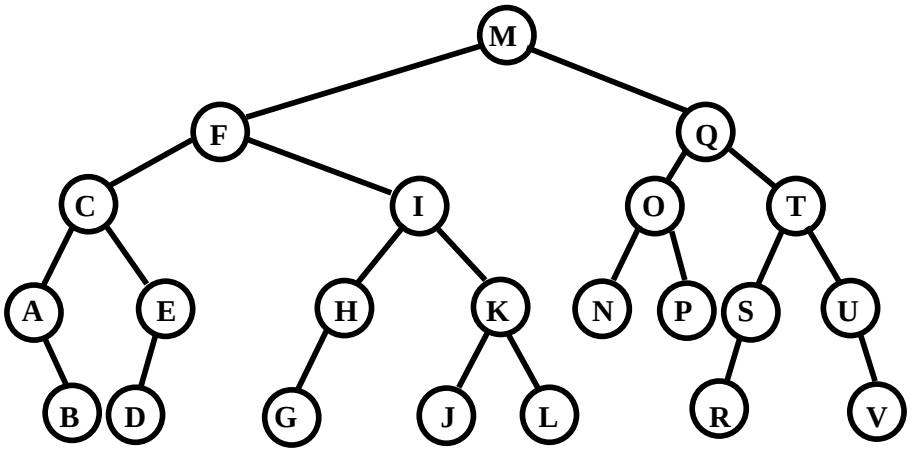
1) **en heap**
2) **et binært søketre**
3) **et 2-3-4 tre**
4) **et Red-Black tre**

c) Du skal utføre Quicksort på teksten "N E W C A S T L 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**.

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder uavhengige oppgaver fra kap.14, 22 og 30 i læreboka.

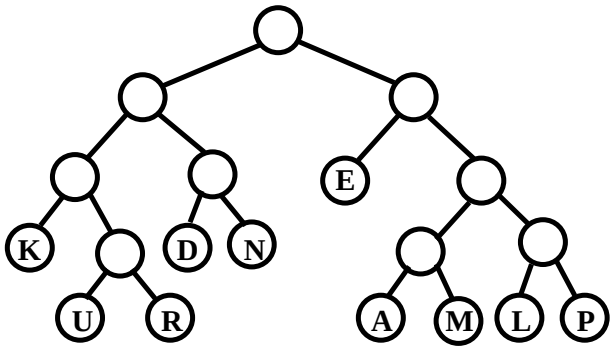
a)



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 'H', 'F' og 'T'.**

NB: Du skal for hver fjerning *på nytt* ta utgangspunkt i treet ovenfor.
Dvs. på intet tidspunkt skal det fra treet være fjernet flere bokstaver samtidig.

b) Vi har følgende ferdiglagde Huffman kodingstrie (dad'enes nr/indeks er uinteressant):



Vi har også følgende bitstrøm, som er kodet etter denne trien:
0100010011010100011000111000101101111110011 **<u>Hva er teksten i denne meldingen?</u>**

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

FD BA EC CD AF CA

Vi skal utføre *union-find m/weight balancing (WB) og path compression (PC)* på denne grafen. **<u>Tegn opp arrayen "dad"s innhold etterhvert</u>** som skogen bygges opp (jfr. fig.30.9) vha. "find"-funksjonen s.447 i læreboka. **<u>Bemerk når WB og PC er brukt.</u>** Ut fra dette: **<u>tegn også opp den resulterende union-find skogen</u>** (dvs. noe lignende til nedre høyre skog i fig.30.8).

Oppgave 3 (koding, 30%)

Vi har en toveis liste (alle har en peker til neste node, men også en til forrige node) bestående av nodene:

```
struct Node  {
  int   ID;              //   Nodens ID/key/nøkkel/navn (et tall).
  Node* prev_left;       //   Peker til forrige eller venstre subtre (evt. nullptr).
  Node* next_right;      //   Peker til neste eller høyre subtre (evt. nullptr).
  Node(int id) {  ID = id;  prev_left = next_right = nullptr;  }
};
```

Vi har også de globale dataene:

```
const int ANTNODER = 15;
Node* root     = nullptr;   //   Peker til listen eller til treets rot.
```

«Tomme pekere» peker *ikke* til en `z`-node, men til `nullptr/NULL`.

For oppgave 3a, så antar vi at det er bygd en toveis liste, med `ANTNODER` noder i. Deres `ID` er uvesentlig. `root` peker initielt til starten på denne listen.

a) **<u>Lag den *rekursive* funksjonen</u>** `Node* omformListeTilBBT(int len)`

Funksjonen skal omforme en toveis liste (med `len` noder i) til et mest mulig balansert binært tre, og returnere en peker til rota for dette (sub)treet. Den *skal* bruke nodene i den opprinnelige listen til å bygge dette treet (derfor heter også pekerne `prev_left` og `next_right`).

NB: Vi skal *ikke* finne midten av lista (som i eksamen 24.februar1997 oppgave 4), for deretter rekursivt bygge første halvdel som venstre subtre, og andre halvdel som høyre subtre. I stedet *skal* funksjonen bygge venstre subtre først, før dette kobles til (sub)treets rot, og tilslutt tilkobles høyre subtre. Dette medfører at nodene i lista blir tilkoblet treet i den samme sekvensielle rekkefølge de initielt ligger i (inorder rekkefølge). Dermed bygges treet på en måte nedenfra.

Funksjonen kalles fra `main` ved: `root = omformListeTilBBT(ANTNODER);`
Men, i det kallet skjer, peker altså den globale `root` til starten på toveis listen.

For oppgave 3b, så antar vi at det er bygd et binært tre (ikke nødvendigvis hverken balansert eller søketre). Nodenes `ID` er uvesentlig. `root` peker initielt til rota for dette treet.

b) **Lag den *rekursive* funksjonen** `Node* omformBTTilListe(Node* n)`
Funksjonen skal omforme et binært tre til en toveis liste. Nodene i listen skal komme i samme rekkefølge som når treet traverseres i inorder rekkefølge. Den *skal* bruke nodene i det opprinnelige treet til å bygge denne listen (derfor heter altså pekerne `prev_left` og `next_right`). Parameter `n` fortsetter hele tiden å peke til den samme node, og er også den som blir returnert, bare at dens `prev_left` og `next_right` som oftest er oppdatert.
Hint: En nodes forrige er den lengst til høyre i venstre subtre. En nodes neste er den lengst til venstre i høyre subtre.
Funksjonen kalles fra `main` ved: `root = omformBTTilListe(root);`
Dette medfører som regel at `root` altså peker et eller annet sted inni listen. En while-løkke blar `root` frem til listens start (ved å følge pekerne `prev_left`), før den brukes/skrives ut.
Vi forutsetter at `root` *ikke* er `nullptr/NULL` ved kallet i `main`.

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.

Oppgave 4 (koding, 20%)

Tallet 3247651089 inneholder alle sifrene fra 0-9, og kun én gang hver. De tre første sifrene utgjør tallet 324, de tre neste tallet 765, de fire siste tallet 1089. Vi har nettopp at: 324 + 765 = 1089. **Lag et komplett program som skriver ut *alle* løsninger på slike regnestykker for et 10-sifret tall**, der:

- alle sifrene er unike (dvs. alle sifrene 0-9 er i bruk)
- tallet brutt opp i: et 3-sifret tall + et 3-sifret tall = et 4-sifret tall
- det *virkelig* er et 4-sifret tall til slutt (dvs. starter ikke med '0')
- like regnestykker som f.eks. 324 + 765 og 765 + 324 blir silt vekk

Kan du gjenbruke noe kode fra læreboka eller eksempler, så bare bruk/henvis 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!

FrodeH