NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Eksamensoppgave i IDATG2102 – Algoritmiske metoder

Faglig kontakt under eksamen:	**Frode Haug**
Tlf:	**950 55 636**

Eksamensdato:	**26.november 2020 - HJEMME**
Eksamenstid (fra-til):	**15:00-19:00 (4 timer)**
Hjelpemiddelkode/Tillatte hjelpemidler:	**F - Alle trykte og skrevne.** (kalkulator er *ikke* tillatt)
Annen informasjon:	

Målform/språk:	**Bokmål**
Antall sider (inkl. forside):	**4**

Informasjon om trykking av eksamensoppgaven

Originalen er:

1-sidig X **2-sidig** ☐

sort/hvit X **farger** ☐

Skal ha flervalgskjema ☐

Kontrollert av:

_____

Dato Sign

Oppgave 1 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

a) Et av eksemplene i pensum leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: (((((4 * 6) + (5 * 3)) * (5 + 6)) + 8) * 3)
Hva blir dette skrevet på en postfix måte?
Skriv/tegn stakkens innhold etter hvert som koden leser tegnene i infix-uttrykket.

b) I de følgende deloppgaver er det key'ene "A L L E R S I S T E"
(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. **Skriv/tegn den *resulterende* datastruktur når key'ene legges inn i:**

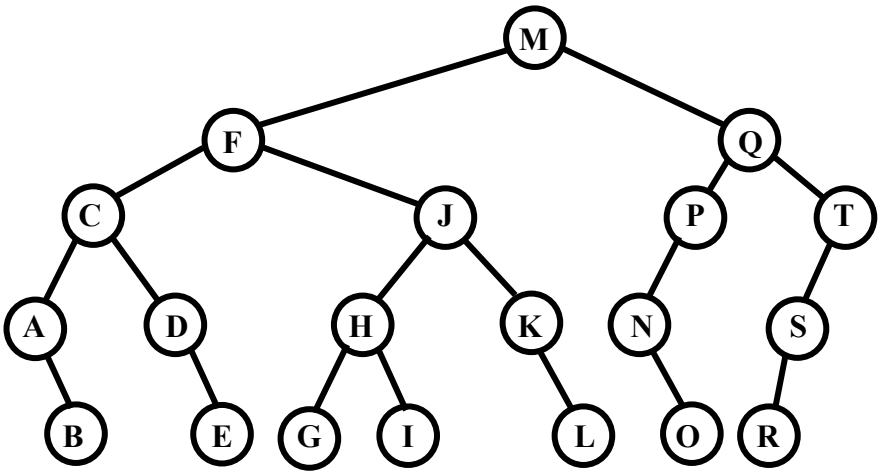
1) **en heap**
2) **et binært søketre**
3) **et 2-3-4 tre**
4) **et Red-Black tre**

c) Quicksort skal utføres på bokstavene/keyene "A L L E R S I S T E" (blanke regnes *ikke* med).
Lag en oversikt/tabell 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 totalt uavhengige oppgaver fra pensum.

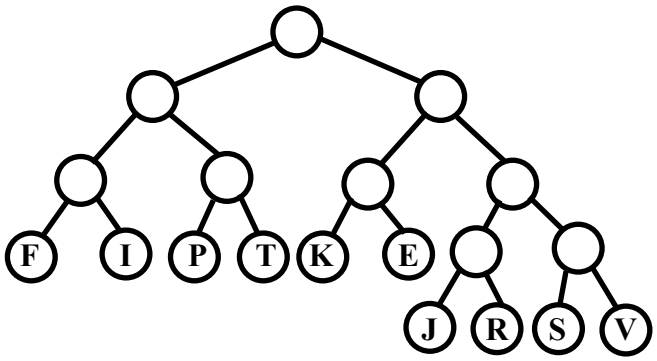
a)



Det skal fjernes («remove») noen noder fra det ovenfor gitte *binære søketreet*.
Skriv/tegn treet for hver gang, og fortell hvilken av «if ..... else if ..... else»-grenene i `EKS_28_BinertSokeTre.cpp` (dvs. Case 1, Case 2, Case 3) som er aktuelle når det etter tur fjernes henholdsvis bokstavene 'C', 'Q' og 'M'.

NB: For hver fjerning skal det *på nytt* tas utgangspunkt i *hele* det aktuelle treet.
Dvs. *på intet tidspunkt* skal det fra treet være fjernet *mer enn en* bokstav.

b) Vi har følgende ferdiglagde Huffman kodingstrie (innholdet i `gForeldre` er uinteressant):



Vi har også følgende bitstrøm (melding), som er kodet etter denne trien:
00010101101100111101100101110111111001
<u>Hva er bokstavenes bitmønster *og* hva er teksten i denne meldingen?</u>

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

AC AE DB BC EB

Utfør Union-Find *m/weight balancing (WB) og path compression (PC)* på denne grafen.
<u>Skriv/tegn opp innholdet i `gForeldre` etter hvert som `unionerOgFinn2` kjøres/utføres.</u> **<u>Bemerk hvor WB og PC er brukt.</u>**
<u>Skriv/tegn også opp den resulterende union-find skogen.</u>

Oppgave 3 (koding, 28%)

Vi har et binært tre bestående av:

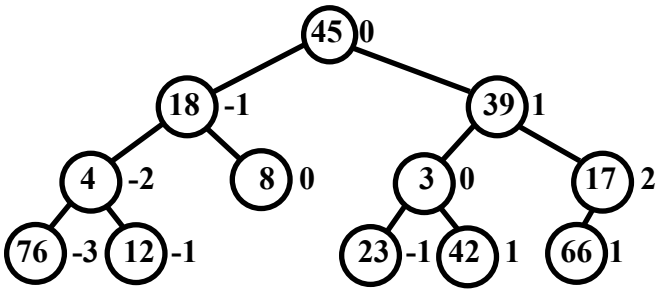
```
struct Node  {
  int ID;                 //  Nodens ID/key/nøkkel/navn (et tall).
  int avstand;            //  Nodens "vertikale avstand" ift. rota (brukes kun i 3A).
  Node *left, *right;     //  Referanse til begge subtrærne (evt. 'nullptr/NULL').
  Node(int id) { ID = id; left = right = nullptr; avstand = 0; }
};
```

Vi har *kun* den globale variabelen:

```
Node* gRoot = nullptr;   //  Rot-peker (har altså ikke at head->right er rota).
```

Det skal her lages/kodes to *helt* uavhengige funksjoner.
Hvordan et tre har blitt bygd/satt opp, trenger du ikke å tenke på.

a) **<u>Lag den *rekursive* funksjonen</u>** `void settVertikalAvstand(const Node* t)`
Funksjonen skal oppdatere *alle* nodene i treet med deres *relative vertikale avstand* ift. rota.
Dvs. `avstand` inni hver node er *en* mindre enn moren om den er et venstre barn,
mens den er *en* høyere enn mora om den er et høyre barn.
Eksempeltre, påskrevet vertikal avstand utenfor noden til høyre:



b) **Lag den *rekursive* funksjonen** `void skrivNoderUtenSosken(const Node* t)`
Funksjonen skal skrive ut `ID` for *alle* nodene i treet som *ikke* har søsken.
Husk å spesialbehandle selve rota (for *den* har jo ingen søsken).
For treet ovenfor, ville utskriften ha blitt: 45 66

Begge funksjonene kalles initielt fra `main` med `gRoot` som parameter.

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 f.eks. array, stakk, kø eller liste.

Oppgave 4 (koding, 22%)

Lag den *ikke-rekursive* funksjonen `int fjernDuplikater(int a[], const int n)`

I arrayen `a` er indeksene 1 til `n` i bruk. `a` inneholder kun positive heltall, og er sortert stigende.
Men, den kan inneholde duplikate/like verdier (som pga sorteringen ligger rett etter hverandre).
Funksjonen skal fjerne eventuelle slike duplikate verdier fra arrayen. Den skal sørge for at alle de da unike/ulike verdiene blir liggende rett etter hverandre (og fortsatt sortert) i første del av arrayen. Siste del av arrayen skal, om det fantes duplikater, fylles med 0'er (nuller).
Funksjonen skal også returnere antall forskjellige/unike/ulike verdier i arrayen.

Vi forutsetter at `n >= 0`, samt at `a[0] != a[1]`.
Antagelig bør koden spesialbehandle når `n` er 0 (null) eller 1.

For full score vektlegges effektivitet, og at ingen ekstra array brukes.

NB: I *hele* dette oppgavesettet skal du *ikke* bruke kode fra (standard-)biblioteker (slik som bl.a. STL og Java-biblioteket). Men de vanligste includer/import du brukte i 1.klasse er tilgjengelig.
Koden kan skrives valgfritt i C++ eller Java.

Løkke tæll!
FrodeH