NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Kontinuasjonseksamensoppgave i IDATG2102 – Algoritmiske metoder

Faglig kontakt under eksamen:	**Frode Haug**
Tlf:	**950 55 636**
Eksamensdato:	**8.august 2022**
Eksamenstid (fra-til):	**09:00-13:00 (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) 8 Et av eksemplene i pensum leser og omgjør et infix-uttrykk til et postfix-uttrykk.
Vi har infix-uttrykket: (((5 + 7) * ((4 * 3) * (8 + 2))) * (3 + 6))
Hva blir dette skrevet på en postfix måte?
Skriv/tegn stakkens innhold etter hvert som koden leser tegnene i infix-uttrykket.

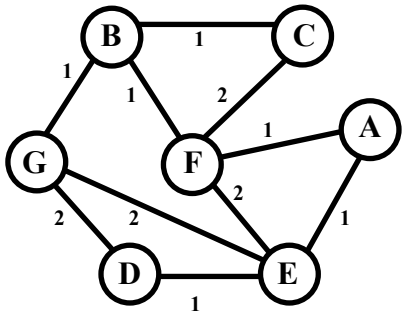
b) 8 Quicksort skal utføres på bokstavene/keyene "B A N A N B I L D E" (blanke regnes *ikke* med).
Lag en oversikt/tabell der du for hver rekursive sortering skriver de involverte bokstavene og markerer/uthever hva som er partisjonselementet.

c) 9 Heapsort (vha. bottom-up heap konstruksjon) skal utføres på bokstavene/keyene "B A N A N B I L D E" (blanke regnes *ikke* med).
Skriv/tegn opp heapens innhold etterhvert som heapen konstrueres og deretter sorteres.

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre totalt uavhengige oppgaver fra pensum.

a) 9



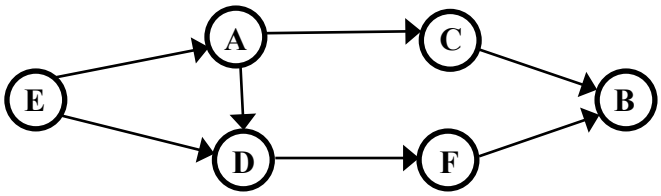
Vi bruker nabomatrise, og starter i node E. **Skriv/tegn opp minimums spenntreet (MST)** for denne grafen, etter at koden i EKS_31_MST.cpp er utført/kjørt.
Skriv også opp innholdet i/på fringen etterhvert som konstruksjonen av MST pågår.
NB: *Husk at ved lik vekt så vil noden sist oppdatert (nyinnlagt eller endret) havne først på fringen ift. andre med den samme vekten.*

b) 8 Vi har rutenettet med **S**(tart)- og **M**(ål)-ruter:

1	2 **S**	3	4 **x**	5	6
7			10	11	12
13 **x**	14	15 **x**	16		18
19		21		23	24
25 **x**	26	27		29	
	32	33 **x**	34	35 **M**	36

Hva vil den minste f-verdien i rutene 4, 13, 15, 25 og 33 (x) kunne være når det *kun* kan gås opp/ned/høyre/venstre (*ikke* på skrå) med en vekt på 1 (en), og som heuristikk brukes Manhatten distanse (summen av antall ruter horisontalt og vertikalt til og inkludert målet)

c) 8 Angi ***alle* mulige topologiske sorteringssekvenser** av nodene for den rettede (ikke-vektede) asykliske grafen («dag»):



Oppgave 3 (koding, 36%)

Vi har et binært tre bestående av:

```
struct Node  {
  int ID;                  // Nodens ID/key/nøkkel/navn (et tall).
  Node *left, *right;      // Referanse til begge subtrærne (evt. nullptr/NULL).
  Node(int id) {  ID = id; left = right = nullptr;  }
};
```

Vi har den globale variabelen:

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

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.

Det skal her lages/kodes to *helt* uavhengige funksjoner:

a) Lag den *rekursive* funksjonen

```
bool skrivNivaa(const Node* t, const int n, const int nivaa)
```

Funksjonen sørger (om mulig) rekursivt for at *alle* noder på nivå `nivaa` blir skrevet ut (på skjermen) fra venstre mot høyre. Dette gjøres om `n` er lik `nivaa` (og da returneres det `true`). Ellers prøver den rekursivt å få dette gjort for enda et høyere nivå (da `n` foreløpig er mindre enn `nivaa`), ved rekursivt å tilkalle seg selv med `n` *en* høyere, og for hver av barna/subtrærne (ved å sende med disse som parametre). `nivaa` forblir hele tiden den samme. Det returneres om det ønskede nivået blir funnet/oppnådd (hos *minst* ett av barna/subtrærne). Husk å håndtere at `t` også kan være `nullptr/NULL`. Rota er på nivå nr.1.

Funksjonen kalles (flere ganger) fra `main` vha. koden:

```
int aktueltNivaa = 1;
while (skrivNivaa(gRoot, 1, aktueltNivaa)) aktueltNivaa++;
```

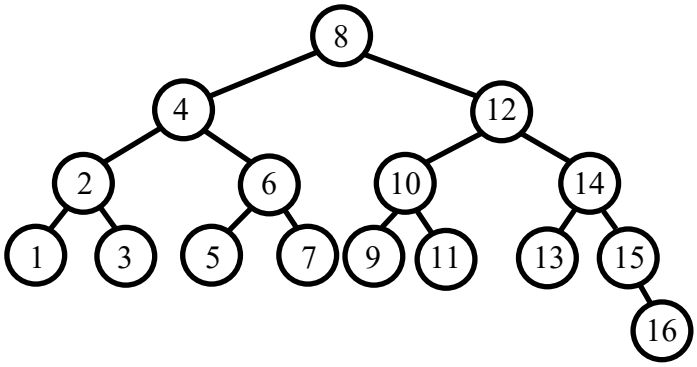
Totalt blir treet altså skrevet ut level order. Husk **NB**-punktet ovenfor.
Hvordan et tre har blitt bygd/satt opp, og `gRoot` satt til å peke på dets rot, trenger du ikke å tenke på.

b) **Lag den *rekursive* funksjonen** `Node* byggBalansertBST(const int arr[], const int start, const int slutt)`

Vi har en sortert array med `int`'er. Det skal rekursivt bygges (med `Node`'r) et mest mulig balansert binært søketre (BST) ut fra dets elementer. Input til funksjonen er hele tiden den samme arrayen `arr`, samt grensene/indeksene (`start` og `slutt`) for intervallet det skal bygges ut fra. Funksjonen kalles/brukes i `main` vha. koden:

```
int IDer[] {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
gRoot = byggBalansertBST(IDer, 0, 15);
```

Treet (under `gRoot`) vil da bli seende slik ut:



Oppgave 4 (koding, 14%)

Vi har `int tallene[1000000];` som inneholder 1 million tilfeldige *positive* heltall.
Skriv et *komplett* program som går igjennom denne arrayen, og summer sammen alle tallene som er *større eller lik det største tallet hittil funnet/registrert* tidligere i arrayen.
Dvs. vi oppsummerer *kun* tall som vil ha utgjort en *stigende (eller lik) sortert sekvens* av tall.
Skriv til slutt ut totalsummen, og antall tall som er summert sammen.

Eksempel (med bare ti tall):

5	4	3	6	7	5	6	7	5	1	4	10

Totalsum: 35 (da 5 + 6 + 7 + 7 + 10 = 35, altså 5 tall summert)
Antall summert: 5

For full score vektlegges også kort og effektiv kode.

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