NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

Kontinuasjonseksamensoppgave i IMT2021 – Algoritmiske metoder

Faglig kontakt under eksamen:	**Frode Haug**
Tlf:	**950 55 636**
Eksamensdato:	**7.august 2017**
Eksamenstid (fra-til):	**09:00-14:00 (5 timer)**
Hjelpemiddelkode/Tillatte hjelpemidler:	**F - Alle trykte og skrevne.** (kalkulator er *ikke* tillatt)
Annen informasjon:	
Målform/språk:	**Norsk**
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 uavhengige oppgaver bl.a. fra kap.11, 14, 15 og 8 i læreboka.

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

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

void rekursiv(int n, int m) {
  if (n > m) {  rekursiv(n - 2, m + 2);
                cout << ' ' << n - m;  }
}

int main()  {  rekursiv(15, 1);   return 0;  }
```

b) I de følgende deloppgaver er det key'ene "K A R A V O S T A M O"
(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) Vi skal utføre Shellsort på key'ene "KARKINAGRI". Jfr. kode s.109 i læreboka.
For hver gang indre for-løkke er ferdig (etter: `a[j] = v;`):
Tegn opp arrayen og skriv verdiene til 'h' (4 og 1) og 'i' underveis i sorteringen. Marker spesielt de key'ene som har vært involvert i sorteringen (jfr. fig.8.7 s.108).

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder uavhengige oppgaver fra kap.16, xx og xx i læreboka.

a) Koden s.237 og teksten s.239 i læreboka gir oss følgende kode ved dobbel hashing:

```
const int M = 17;
int hash1(int k)   {  return  (k % M);        }
int hash2(int k)   {  return  (4 - (k % 4));  }

void insert(itemType v, infoType info)  {
   int x = hash1(v);
   int u = hash2(v);
   while (a[x].info != infoNIL)  x = (x+u) % M;
   a[x].key = v;  a[x].info = info;
}
```

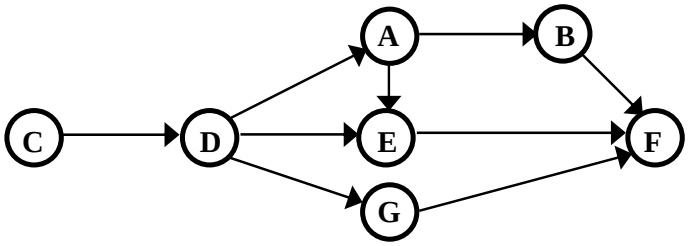
"k" står for bokstavens nummer i alfabetet (1-29). Vi har en array som er 17 lang (indeks 0-16). Keyene "CHRISTOSRACHES" skal legges inn i denne arrayen. **Skriv opp hver enkelt key's returverdi fra både hash1 og hash2. Tegn også opp arrayen for hver gang en ny key legges inn.** (Innholdet i "info" trenger du ikke å ta hensyn til.)

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

FA EB DC AB CA CB

Vi skal utføre *union-find m/weight balancing (WB) og path compression (PC)* 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. **Bemerk når WB og PC er brukt.** Ut fra dette: **tegn også opp den resulterende union-find skogen** (dvs. noe lignende til nedre høyre skog i fig.30.8).

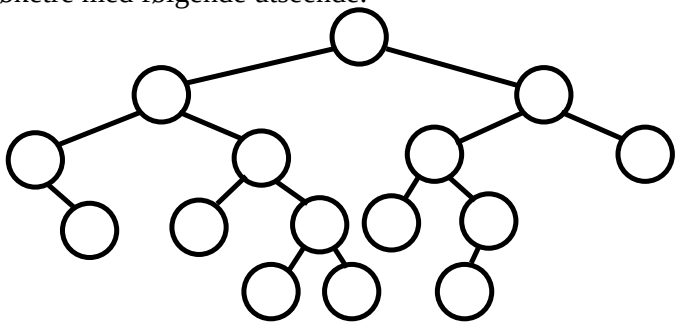
c) Følgende rettede (ikke-vektede) asykliske graf («dag») er gitt:



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

Oppgave 3 (koding, 30%)

Vi har et binært søketre med følgende utseende:



Disse nodene er bygd opp omkring følgende kode:

```
struct Node  {
  int   ID;          //  Nodens ID/key/nøkkel/navn (et tall).
  Node* left;        //  Peker til venstre subtre (evt. nullptr).
  Node* right;       //  Peker til høyre subtre (evt. nullptr).
  int   leftNumber; //  Antall noder totalt i venstre subtre.
  Node(int id) { ID=id;  left=right=nullptr;  leftNumber=0; }
};
```

Vi har også den globale variabelen:

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

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

a)

10% 3% **Del 1: Skriv tallene 1-15 i en sekvens slik at hvis noder med disse IDen settes inn i denne rekkefølge, i et på forhånd tomt tre, så vil treet få samme form som i figuren ovenfor.**

3% **Del 2:** Alle noder i et binært tre har et bestemt antall noder i sitt venstre subtre (null eller flere). **Tegn av treet ovenfor, og angi i forbindelse med hver node antallet i venstre subtre.**

4% **Del 3: Lag den *ikke-rekursive* funksjonen** `int minst()` **som returnerer verdien til den aller minst noden (første i inorder rekkefølge) i et vilkårlig binært søketre. Er treet tomt returneres 0 (null).**

b) Koden (litt omskrevet ift. side 206 i læreboka) for å sette inn i et binært søketre er:
6%

```
void settInn(int id) {
  Node *p = nullptr, *x = root;
  while (x)
    {  p = x;  if (id < x->ID) x = x->left;
               else            x = x->right;  }
  x = new Node(id);
  if (!p)  root = x;
  else if (id < p->ID) p->left = x;
  else p->right = x;
}
```

`leftNumber` inneholder altså totalt antall noder hver node har i sitt i venstre subtre.
Endre/utvid koden ovenfor, slik at dette ivaretas for *alle* nodene som, ved innsettelse av *en* ny node, får *en* mer i sitt venstre subtre. Det holder at du angir hvor ny kode skal inn.

c) **Lag den *ikke-rekursive* funksjonen** `int hent(int n)` **som returnerer verdien til noden**
14% **som er nr.'n' i inorder rekkefølge i treet.** Den første noden er nr.0, den andre er nr.1, osv. Du kan forutsette at `leftNumber` er korrekt satt i alle nodene, samt at `n` er et gyldig tall ift. totalt antall noder i treet.

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%)

Et tall A 'kappes'/deles i to deler: B og C. Deretter kan det lages regnestykket: $A = (B + C)^2$.
Noen eksempler på slike tall er: $81 = (8 + 1)^2 = 9^2$ $88209 = (88 + 209)^2 = 297^2$
$100 = (10 + 0)^2 = 10^2$ $998001 = (998 + 1)^2 = 999^2$

Lag et komplett program som skriver ut *alle* løsninger på slike regnestykker for tall *mindre enn* 1.000.000 (en million).

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!

Frode