NTNU

Kontinuasjonseksamen

EMNENAVN: **Algoritmiske metoder**

EMNENUMMER: **IMT2021**

EKSAMENSDATO: **15. august 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 tre uavhengige oppgaver fra kap.3, 11, 12, 14 og 15 i læreboka.

a) Koden midt på side 27 i læreboka leser et postfix-uttrykk og regner ut svaret.
Vi har postfix-uttrykket: 3 4 + 3 2 * + 2 + 5 3 * 4 2 + * + **Hva blir svaret ?**
Tegn opp innholdet på stakken etter hvert som koden side 27 foretar beregningen.

b) I de følgende deloppgaver er det key'ene "M Y S U S E T E R"
(i denne rekkefølge fra venstre mot, 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**

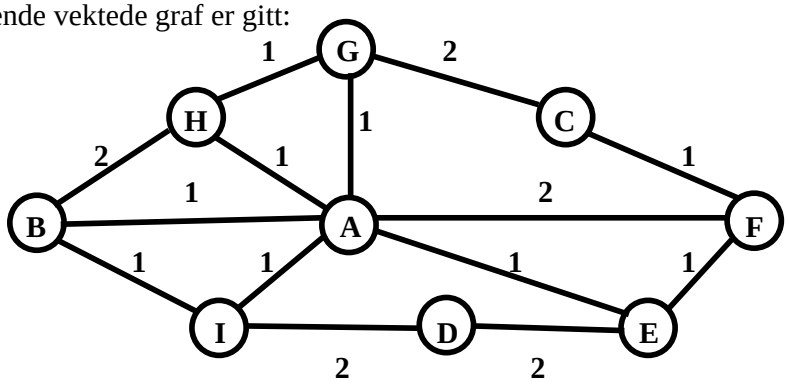
c) Vi skal utføre *rekursiv* Mergesort på key'ene "MYSUSETER". Jfr. kode s.166 i læreboka.
For hver gang tredje og siste for-løkke i koden s.166 er ferdig:
Tegn opp arrayen med de key'ene som har vært involvert i sorteringen (jfr. fig.12.1 s.167)

Oppgave 2 (teori, 25%)

Denne oppgaven inneholder tre uavhengige oppgaver fra kap.30, 31 og 32.

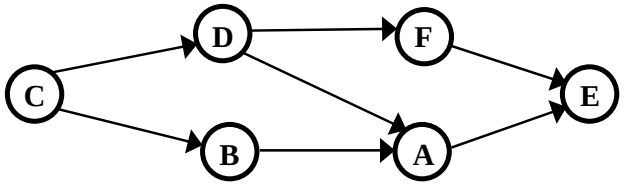
a) Følgende kanter i en (ikke-retted, ikke-vekted) graf er gitt:
FC DA EF AE CA BF
Vi skal utføre *union-find m/weight balancing (WB) og path compression (PC)* på denne grafen.
Tegn opp arrayen "dad"s innhold etterhvert som skogen bygges opp (jfr. fig.30.9) vha. "find"-funksjonen s.447 i læreboka. **Bemerk 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).

b) 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 "H" og til *alle* de andre nodene?
Tegn også opp innholdet i prioritetskøen etterhvert som koden utføres (jfr. fig.31.13).
NB: Husk at ved lik prioritet så vil noden sist oppdatert (nyinnlagt/endret) havne først i køen.

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 tre (*ikke* nødvendigvis binært søketre (BST)) bestående av nodene:

```
struct Node  {
  int   ID;          //   Nodens ID/key/nøkkel/navn (et tall).
  Node* left;        //   Peker til venstre subtre, evt. NULL/nullptr når tomt.
  Node* right;       //   Peker til høyre subtre, evt. NULL/nullptr når tomt.
  Node (int id, Node* l, Node* r)     //   Constructor.
       { ID = id;  left = l, right = r; }
};
```

Vi har også de tre globale variablene:

```
Node* rot     = NULL;  //   Rot-peker (har altså ikke at head->right er rota).
const int MIN = 0, MAX = 9999;  //   Verdier mindre/større enn alle i treet.
```

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

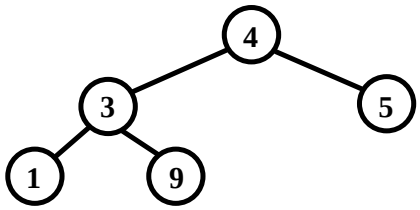
NB: I *hele* oppgave 3:

- *kan* `n` inn til funksjonene og `rot` være `NULL/nullptr`.
- startes/kalles funksjonene fra `main` med `rot` som parameter.
- 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.

a) **Lag den *rekursive* funksjonen** `int finnMin(Node* n)`
Funksjonen returnerer den *minste* verdien i treet tilpekt av og under `n`.
Er treet tomt returneres `MAX`.

Lag også den *rekursive* funksjonen `int finnMax(Node* n)`
Funksjonen returnerer den *største* verdien i treet tilpekt av og under `n`.
Er treet tomt returneres `MIN`.

b) **Lag den *rekursive* funksjonen** `bool erBST(Node* n)`
Funksjonen returnerer `true/false` til om treet tilpekt av og under `n` er et binært *søketre* (BST) eller ei. Du bør bruke begge funksjonene fra oppgave 3a (du kan bare tilkalle/bruke dem, selv om du evt. ikke klarte å implementere/kode dem).
NB: For det holder nemlig *ikke* bare å sjekke om en nodes venstre barn er mindre enn noden selv, og om høyre barn er større eller lik. En slik algoritme/tankegang vil medføre at f.eks. følgende tre blir vurdert som BST – hvilket det *ikke* er, da '9' er til venstre for '4':



Oppgave 4 (koding, 20%)

Maksimum snømengde (i cm) i målt en eller annen dag i løpet av vinteren på et ukjent sted de N siste årene er lagret i arrayen `mengde[N]`. I arrayen `sist[N]` skal det i indeks nr.`i` lagres *antall år siden sist det ble målt minst like mye snømengde*. Finnes ikke noe slikt foregående år, lagres det -1.

Om f.eks. N = 20 og arrayen `mengde` inneholder:

```
50, 32, 74, 29, 61, 42, 53, 29, 78, 39, 51, 18, 27, 64, 25, 42, 48, 37, 44, 48
```

vil `sist` i de fem første 'skuffene' inneholde:

```
-1   1  -1   1   2  . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Hva blir innholdet i de 15 neste 'skuffene' i `sist`?

Gitt N, en fylt `mengde`-array og en tom `sist`-array.
Skriv et komplett program som beregner `sist` sine verdier i *alle* de ulike 'skuffene'.

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@haug.es