Eksamen i 2ADS101 Algoritmer og datastrukturer for spill
17.10.17

Tillatte hjelpemidler: Alle trykte og håndskrevne notater og filer på egen pc kan benyttes.

Innlevering: Kildekode i zippet fil. Svar på teorispørsmål kan skrives som kommentar i kildekode eller i teksteditor (lever som pdf). Bruk kandidatnummeret i filnavnet.

Oppgave 1 (20%)
Skriv et program som genererer 50 tilfeldige tall mellom 0 og 10 og sorterer stigende. Begrunn valg av sorteringsfunksjon.

Oppgave 2 (20%)
Anta at du skal implementere hashing for et datasett med ca 50 objekter og en heltallig nøkkelverdi. Beskriv hvordan du vil gjøre dette. Angi hashfunksjonen du vil bruke, med begrunnelse. (Det er ikke meningen at du skal gjøre en ferdig implementering, men bruk gjerne pseudokode/kode-eksempler).

Oppgave 3 (20%)
Gitt tallene {11, 17, 4, 7, 9, 1, 22, 14}.

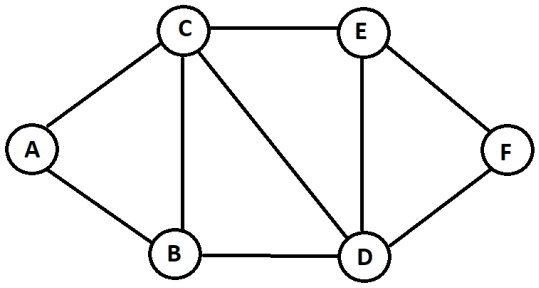
a) Tegn et binært søketre som du får når tallene settes inn i rekkefølge.

b) Tegn en stigende heap som du får når tallene settes inn i rekkefølge.

c) Hva er søketiden i
   1. Usortert array?
   2. Sortert arrray?
   3. Binært søketre?
   4. Hashtabell?
   5. std::vector?
   6. std::list?

d) Nevn anvendelser av binært søketre og heap i implementering av stl containere.

Oppgave 4 (40%)
Gitt en graf som vist på figuren nedenfor:



Klassene ***Node*** og ***Kant*** er gjengitt til slutt i oppgaveteksten. Klassene skal benyttes nøyaktig slik som de er skrevet.

a) Opprett et C++ prosjekt med filene ***kant.h***, ***node.h***, ***node.cpp*** og ***main.cpp***. Kopier kildekode fra oppgavefila (dette dokumentet) og lim inn i C++ filer. Test at du får kompilert med en tom ***main()*** funksjon.

b) Implementer ***Node::settinnkant()*** slik at den setter inn en kant med både vekt, peker til franode og peker til tilnode når den kalles av en gitt node.

c) Implementer ***Node::skrivrek()*** slik at den skriver ut alle nodene med tilhørende kanter, når den kalles av en peker til første node i grafen. Utskriften skal vise grafens topologi i et konsoll-vindu.

d) Bygg grafen på figuren i ***main()*** ved å sette inn noder og kanter, og skriv ut grafen til slutt. Kantene skal ha positiv vekt, og alle kantene skal ikke ha lik vekt. Du kan bruke hjelpevariabler etter behov, og du kan for eksempel begynne slik:

```
Node* graf = new Node('A', nullptr);
Node* a = graf; // bruker flere hjelpevariabler
```

e) La minst to forskjellige noder kalle funksjonen ***f()***. Forklar hva funksjonen gjør.

```
// Klassen kant med implementering
#ifndef KANT_H
#define KANT_H
#include "node.h"

struct Kant {
    double mVekt;
    Node* mTilnode;
    Node* mFranode;
    Kant* mNestekant;
    Kant() :
       mVekt(0.0), mFranode(nullptr), mTilnode(nullptr), mNestekant(nullptr)
       { }
    Kant(double vekt, Node* franode, Node* tilnode, Kant* nestekant) :
       mVekt(vekt), mFranode(franode), mTilnode(tilnode), mNestekant(nestekant)
       { }
    ~Kant() { }
    bool operator < (const Kant& k) const { return mVekt < k.mVekt; }
    bool operator > (const Kant& k) const { return mVekt > k.mVekt; }
    void skriv() const {
        if (mFranode) {
            std::cout << mFranode->mNavn;
            std::cout << mTilnode->mNavn;
            std::cout << "(" << mVekt << ") ";
        }
    }
};
#endif // KANT_H
```

```
// node.h
#ifndef NODE_H
#define NODE_H

struct Kant;
struct Node {
    char mNavn;
    Node* mNestenode;
    Kant* mForstekant;
    bool mBesoekt;

    Node();
    Node(char navn, Node* neste);
    ~Node() { }
    void settinnkant(double vekt, Node* til);
    void skrivrek() const;
    void skriv() const;
    double f();
};
#endif // NODE_H
```

```cpp
// Node.cpp
#include <iostream>
#include <queue>
#include <list>
#include "kant.h"
#include "node.h"

Node::Node()
    : mNavn(' '), mNestenode(nullptr), mForstekant(nullptr), mBesoekt(false)
{ }
Node::Node(char navn, Node* neste) :
    mNavn(navn), mNestenode(neste), mForstekant(nullptr), mBesoekt(false)
{ }

void Node::settinnkant(double vekt, Node* til) {
    // oppgave
}

void Node::skrivrek() const {
    // oppgave
}

double Node::f() {
    double sum = 0.0;
    std::priority_queue<Kant, std::vector<Kant>, std::greater<Kant> > apq;

    // Initialiserer
    Kant kant(0.0, nullptr, this, nullptr);
    apq.push(kant);

    do {
      kant = apq.top();
      apq.pop();
      Node* p = kant.mTilnode;

      if (!p->mBesoekt) {
          p->mBesoekt = true;
          kant.skriv();
          sum += kant.mVekt;
          for (auto* q=p->mForstekant; q; q=q->mNestekant)
             apq.push(*q);
      }
   } while (!apq.empty());
    return sum;
}
```