# Fasit til
# Eksamen H2020 ADS101
# Algoritmer og dastrukturer for spill

10/12/20

**Tillatte hjelpemidler:** Filer og programmer på egen pc, lærebok og notater, kursets Canvas-rom med lenker. **Alle deloppgaver teller like mye**.

## 1

(teori, 30%)
Gitt tallene 6, 8, 11, 4, 3, 9, 7. I hvert av punktene nedenfor skal du sette inn tallene i denne rekkefølgen.

a) Tegn opp et binært søketre etter at tallene er satt inn.

b) Tegn opp treet etter at tallet 6 er slettet.

c) Tegn opp array representasjonen til treet fra punkt b.

d) Tegn og vis hvordan du bygger en min-heap med de samme tallene.

e) Tegn heapen etter en pop operasjon.

f) Gitt en hashtabell med n=7 og hashfunksjonene
```
int hash(int key) { return (key % n); }
int dobbelthash(int key) {return (1 + key%(n-1)); }
```

Tegn opp hashtabellen etter at de samme tallene er satt inn.

## 2

(programmering, 15%)
Følgende C++ kode er gitt:

```
class BinaryNode {
public:
    BinaryNode(double data=0.0);
    void insert(double data);

private:
    int m_data;
    BinaryNode* m_left;
    BinaryNode* m_right;
};

BinaryNode::BinaryNode(int data)
   : m_data{data}, m_left{nullptr}, m_right{nullptr} { }

void BinaryNode::insert(int data) {
    if (data < m_data) {
        if (m_left)
            m_left->insert(data);
        else
            m_left = new BinaryNode(data);
    } else if (data > m_data) {
        if (m_right)
            m_right->insert(data);
        else
            m_right = new BinaryNode(data);
    }
}
```

a) Deklarer og skriv koden til en funksjon void stigende() slik at den skriver ut nodenes verdier i stigende rekkefølge når den kalles av roten i et tre.

b) Deklarer og skriv koden til en funksjon void avtakende() slik at den skriver ut nodenes verdier i avtakende rekkefølge når den kalles av roten i et tre.

c) Skriv et testprogram hvor du setter inn verdiene 14, 33, 12, 4, 7, 19, 29, 11, 21, 17 i opplistet rekkefølge og deretter skriver ut stigende og avtakende.

## 3

(teori og programmering, 15%) Linjene nedenfor viser sorteringen av noen tall.

```
14, 33, 12, 4, 7, 19, 29, 11, 21, 17,
14, 33, 4, 12, 7, 19, 11, 29, 17, 21,
4, 12, 14, 33, 7, 11, 19, 29, 17, 21,
4, 7, 11, 12, 14, 19, 29, 33, 17, 21,
4, 7, 11, 12, 14, 17, 19, 21, 29, 33,
```

a) Forklar hvordan tallene er sortert, og hvilken algoritme som er benyttet.

b) Skriv kode hvor du bruker std::priority_queue til å sortere de samme tallene (14, 33, 12, 4, 7, 19, 29, 11, 21, 17) i avtakende rekkefølge.

## 4

(programmering, 25%)
I denne oppgaven skal du bruke kildekode fra listing nedenfor (Node, Edge og Graph kode).

a) Implementer en konstruktør Graph::Graph() som legger inn nodene til grafen på figur 1.

b) Implementer Node* Graph::find_node(char navn) slik at den returnerer en peker til noden med gitt navn hvis den fins i grafen, og en nullpeker ellers.

c) Implementer void Graph::make_edges(char x, char y) slik at den setter inn kanter mellom xy og yx med vekt 1, gitt at noder med navn x og y fins i grafen.

d) Implementer void make_all_edges() slik at funksjonen setter inn alle kantene på grafen på figur 1.

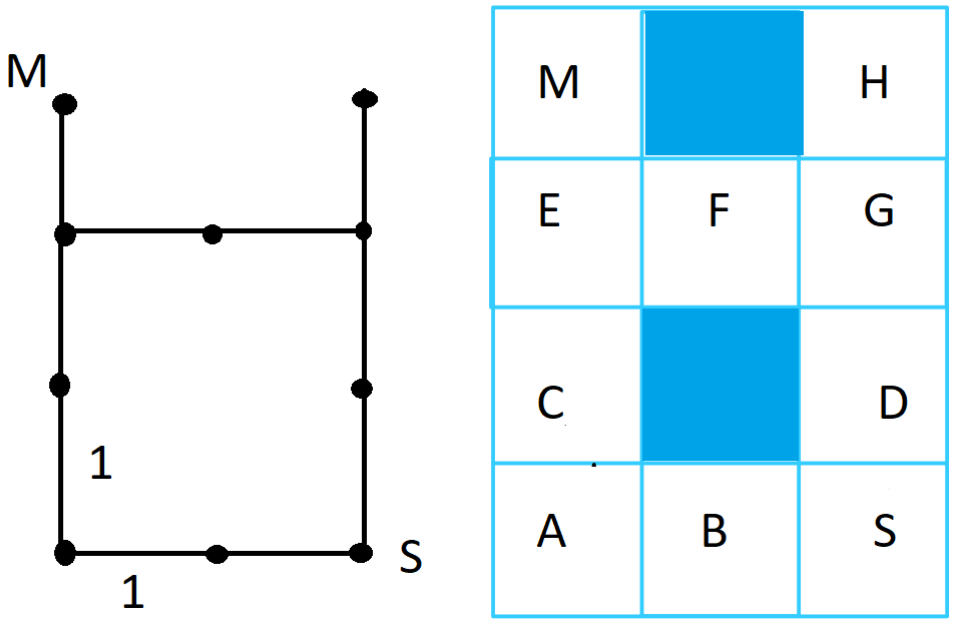
e) Implementer void Graph::print() const slik at den skriver ut en oversikt over alle noder og tilhørende kanter i grafen. Skriv deretter main() hvor grafen på figur 1 opprettes og skrives ut.

Listing 1: Node, Edge og Graph kode

```
struct Node;
struct Edge {
    Node* m_fra;
    Node* m_til;
    double m_vekt;
    Edge();
    Edge(Node* fra, Node* til);
};
Edge::Edge() : m_fra{nullptr}, m_til{nullptr}, m_vekt{0.0} { }
Edge::Edge(Node* fra, Node* til) : m_fra{fra}, m_til{til}, m_vekt{1.0} { }

struct Node {
    char m_navn;
    std::vector<Edge*> m_kanter;
    void push_edge(Edge* kant) { m_kanter.push_back(kant); }
};

struct Graph {
    std::vector<Node*> m_noder;
    Graph();
    Node* find_node(char navn);
    void make_edges(char x, char y);
    void make_all_edges();
    void print() const;
};
```




Figur 1: Graf

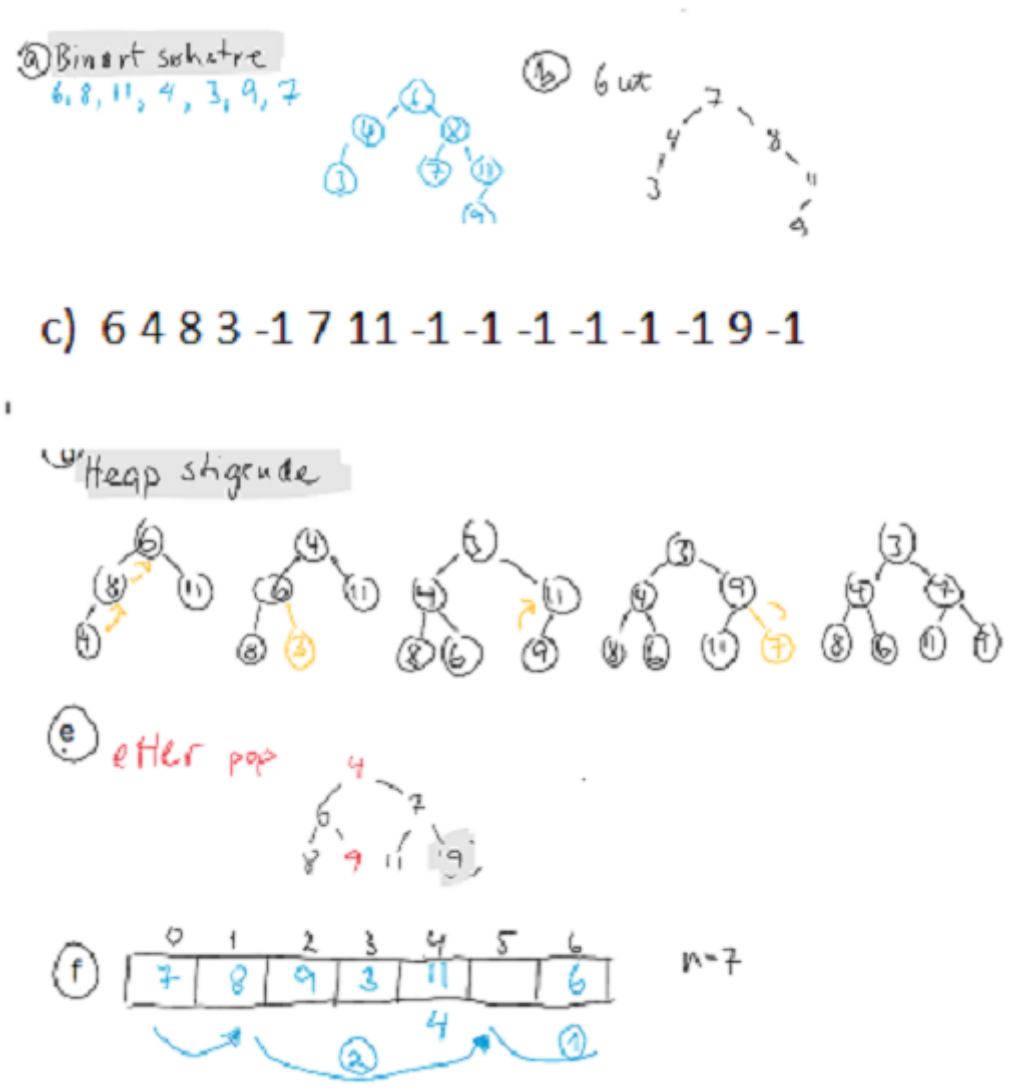
## 5

(teori, 15%)
Figur 1 viser samme graf tegnet på to forskjellige måter, med navngitte noder og hvor alle kantene har vekt = lengde = 1. La S være en startnode og M en sluttnode. Bruk A* algoritmen og bestem korteste vei fra startnode til sluttnode. Skriv opp prioritetskøa med vekt/lengde underveis.

*Slutt på oppgaven.*

# 6 Fasit

## 6.1




Figur 2: Fasit 1. I c) blir treet fra a) [7, 4 8, 3 -1 -1 11, -1 -1 -1 -1 -1 -1 9 -1]

## 6.2

Listing 2: Oppgave 2abc fasit

```
// a
void BinaryNode::stigende() const {
    if (m_left)
        m_left->stigende();
    std::cout << m_data << std::endl;
    if (m_right)
        m_right->stigende();
}
// b
void BinaryNode::avtakende() const {
    if (m_right)
        m_right->avtakende();
    std::cout << m_data << std::endl;
    if (m_left)
        m_left->avtakende();
}
```

Listing 3: Oppgave 2c fasit

```
int main(int argc, char *argv[])
{
    double a[]{14,33,12,4,7,19,29,11,21,17};
    BinaryNode tre{a[0]};
    for (int i=1; i<10; i++)
        tre.insert(a[i]);

    tre.stigende();
    tre.avtakende();
}
```

## 6.3

**a)** Det er flettesortering som er brukt. Til andre linje er to og to tall flettet sammen, slik at alle subarrayer av lengde 2 er sortert. Deretter flettes disse sammen til subarrayer av lengde 4, og dette gjentas slik at vi har en subarray av lengde 8 (og en av lengde 2) i nest siste linje, siden arraylengde 10 ikke er en potens av 2. Til slutt flettes disse to.

Listing 4: Oppgave 3b fasit

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

int main()
{
    int a[]{14,33,12,4,7,19,29,11,21,17};
```

```
    std::priority_queue<int> pq;
    for (int i=0; i<10; i++)
    {
        pq.push(a[i]);
    }
    while (!pq.empty())
    {
        cout << pq.top() << ", ";
        pq.pop();
    }
}
```

## 6.4

Listing 5: Oppgave 4 fasit

```
Graph::Graph() // a
{
    for (char ch='A'; ch<'I'; ch++)
        m_noder.push_back(new Node{ch});
    m_noder.push_back(new Node{'S'});
    m_noder.push_back(new Node{'M'});
}

Node* Graph::find_node(char navn) // b
{
    for (auto it:m_noder) {
       if (it->m_navn == navn)
           return it;
    }
    return nullptr;
}

void Graph::make_edges(char x, char y) // c
{
    Node* fra = find_node(x);
    Node* til = find_node(y);
    if (fra && til)
    {
        fra->push_edge(new Edge{fra, til});
        til->push_edge(new Edge{til, fra});
    }
}

void Graph::print() const // d
{
    for (auto it:m_noder)
    {
```

```
        std::cout << it->m_navn << ": ";
        for (auto k:it->m_kanter)
            std::cout << k->m_fra->m_navn << k->m_til->m_navn << ", ";
        std::cout << std::endl;
    }
}

void Graph::make_all_edges() // e
 {
    make_edges('A','B');
    make_edges('A','C');
    make_edges('E','C');
    make_edges('E','F');
    make_edges('E','M');
    make_edges('G','F');
    make_edges('G','H');
    make_edges('G','D');
    make_edges('S','D');
    make_edges('S','B');
}
int main() // e
{
    Graph g;
    g.make_all_edges();
    g.print();
    return 0;
}
```

### 6.5 A*

```
   D       B       G        F         A          H           E         C             M

                           SDGF 4.41 SBA   5.00 SDGH  5.00
SD 3.82 SB  4.16 SDG 4.24 SBA  5.00 SDGH  5.00 SDGFE 5.00 SDGFE 5.00 SBAC   5.00 SDGFEM 5.
SB 4.16 SDG 4.24 SBA 5.00 SDGH 5.00 SDGFE 5.00 SBAC  5.00 SBAC  5.00 SDGFEM 5.00 SBACE  5.
```