Eksamen H2019 ADS101
Algoritmer og dastrukturer for spill

9/12/19

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

1

(teori) Gitt tallene 17, 14, 5, 7, 12, 1, 16, 29, 13.

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

b) Tegn opp treet etter at tallet 14 er slettet.

c) Tegn opp et AVL-tre med de samme tallene satt inn i rekkefølge.

2

(teori) Gitt en array med tallene 17, 14, 5, 7, 12, 1, 16, 29, 13, 4, 8, 18, 22, 2.

a) Anta at tallene skal sorteres med en umodifisert rekursiv quicksort, hvor pivotelementet velges ved (left+right)/2. Her er left og right indeks til henholdsvis første og siste element i array. Sett opp en tabell som viser alle bytter i første kall på den rekursive quicksort-funksjonen. Forklar hva som er oppnådd i dette funksjonskallet.

b) Anta nå at tallene skal sortertes med mergesort. Sett opp en tabell som viser hvordan hele sorteringen foregår, helt til du har en ferdig sortert array. Forklar med tekst i tillegg.

c) Hva er kompleksiteten til disse algoritmene? Regn ut for begge tilfellene ovenfor.

3

Følgende C++ kode er gitt:

```
using namespace std;

struct Test {
    int key;
    std::string s;
    bool operator == (const Test& t2) const { return key == t2.key; }
};
namespace std {
    template<>
    class hash<Test> {
    public:
        size_t operator() (const Test& t) const {
            return t.key % 7;
        }
        bool operator() (const Test& t1, const Test& t2) {
            return t1.key == t2.key;
        }
    };
}
int main(int argc, char *argv[])
{
    hash<Test> hashtabell;
    unordered_set<Test> uordnet_sett;
```

a) (programmering) Skriv kode for å sette inn poster med nøkler 12, 13, 20, 21, 1, 2, 3, 4, 5, 6 i uordnet_sett i opplistet rekkefølge.

b) (teori) Tegn en figur som viser hvordan postene blir plassert i uordnet_sett.

c) (teori) Hvor mange buckets er det i uordnet_sett etter at postene er satt inn? Hva blir fyllingsgraden?

4

(programmering) I denne oppgaven skal du bruke kode som er gitt til slutt i oppgaven, uten endringer.

a) Implementer funksjonene *Node::settinn_kant(), Graf::settinn_node()* og *Graf::finn_node()*. I sistnevnte funksjon skal du benytte den interne strukturen til å søke etter en node med gitt navn. Funksjonen skal returnere en peker til noden dersom den fins, og nullptr ellers.

b) 1. Implementer funksjonen *Graf::settinn_kant()*
   2. Lag en testgraf i *main()* med noder {**A, B, C, D** og **E**} og kanter {**AB(1.0), AC(2.0), BC(2.0), CD(3.0), DE(1.0), AE(5.0), CE(4.0**}

c) implementer funksjonen *mst()* slik at den returnerer kostnaden for et minste spenntre i grafen.

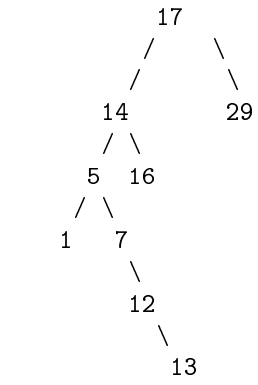
```
#include <iostream>
#include <list>
#include <queue>
using namespace std;

struct Kant;
struct Node {
    char m_navn;
    bool m_besokt;
    std::list<Kant> m_kanter;
    Node(char navn) : m_navn(navn), m_besokt(false) { }
    void settinn_kant(const Kant &kant);
};

struct Kant {
    float m_vekt;
    Node* m_tilnode;
    Kant(float vekt, Node* tilnode) : m_vekt(vekt),  m_tilnode(tilnode) { }
    bool operator > (const Kant& k) const { return m_vekt > k.m_vekt; }
};

struct Graf {
    std::list<Node*> noder;
    Graf() { }
    Node* finn_node(char navn);
    void settinn_node(char navn);
    void settinn_kant(char fra_navn, char til_navn, float vekt);
    float mst();
};
```

Slutt på oppgaven.

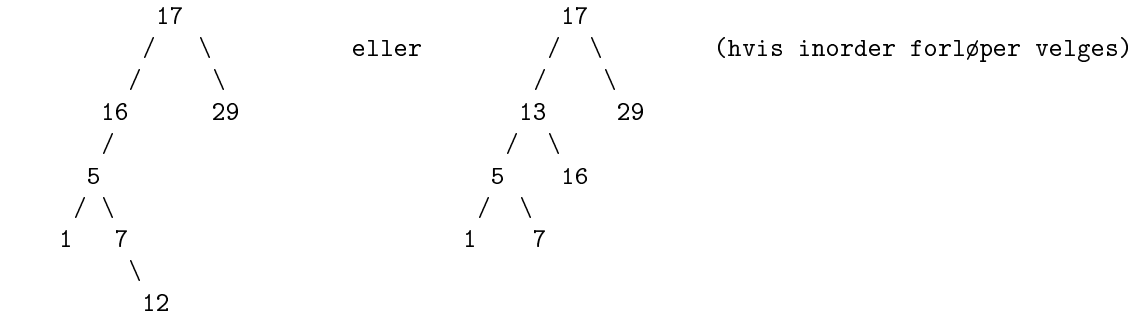
5 Fasit

5.1 Binærtre fasit

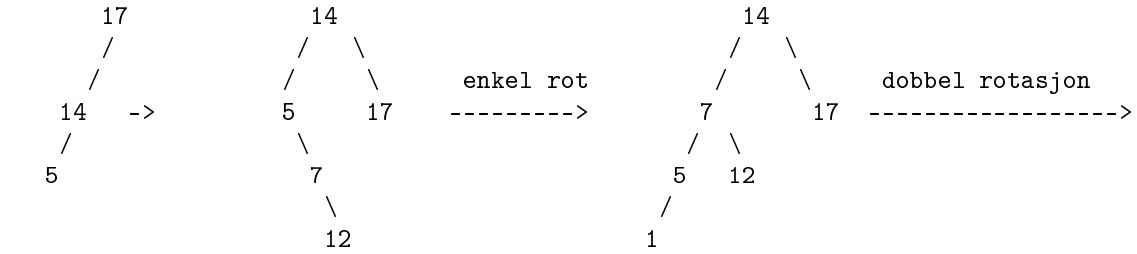
a) Tallene innsatt i binært søketre:

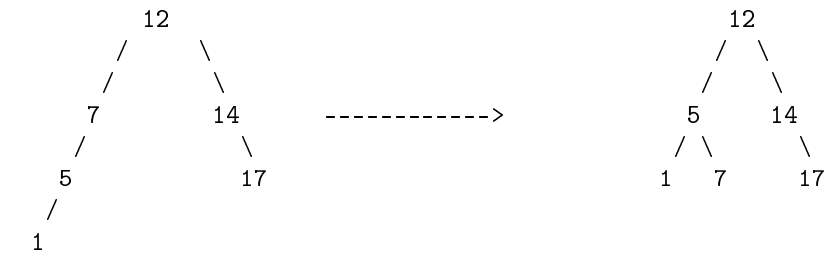


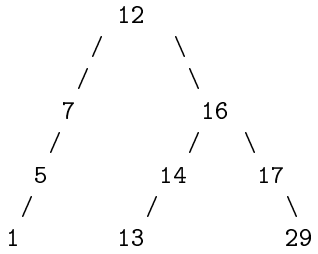
b) Tallet 14 slettet:



c)







Siste tall 13 er uteglemt, det går inn i treet som vsub til 14 uten rotasjoner.

5.2 Sortering fasit

Quicksort:

17	14	5	7	12	1	16	29	13	4	8	18	22	2
2	14	5	7	12	1	16	29	13	4	8	18	22	**17**
2	14	5	7	12	1	**8**	29	13	4	**16**	18	22	17
2	14	5	7	12	1	8	**16**	13	4	**29**	18	22	17
2	14	5	7	12	1	8	**4**	13	**16**	29	18	22	17

Pivotelementet blir riktig plassert. Sorteringen kan nå gjentas for venstre og høyre del av tabell (unntatt pivotelementet). Vi får $n + \frac{n}{2} + \frac{n}{4} + \ldots$ sammenligninger/bytter når tabellen deles i like store halvdeler. Verste tilfelle ved 1 element i den ene delen.

Mergesort:

17	14	5	7	12	1	16	29	13	4	8	18	22	2
14	17	5	7	1	12	16	29	4	13	8	18	2	22
5	7	14	17	1	12	16	29	4	8	13	18	2	22
1	5	7	12	14	16	17	29	4	8	2	13	18	22
1	2	4	5	7	8	12	13	14	16	17	18	22	29

Første rad viser original array. I rad to flettes to og to tall, deretter i rad tre fire og fire. I rad fire flettes de 8 tallene i venstre del og de 6 tallene i høyre del, og i siste rad flettes de to delene sammen.

5.3 Hashing fasit

Et lignende eksempel er vist på forelesningsnotater. **a)**

```
Test t;

t.key = 12; t.s = "tolv"; us.insert(t);
t.key = 13; t.s = "tretten"; us.insert(t);
t.key = 20; t.s = "tjuesju"; us.insert(t);
t.key = 21; t.s = "tjueen"; us.insert(t);
for(int i=1; i<7; i++) {
    t.key = i; t.s = "blabla"; us.insert(t);
}
```

b)
Figuren nedenfor viser postene ved nøkkelverdier satt inn i std::unordered_set.

0	1	2	3	4	5	6
21	1	2	3	4	12	13
–	–	–	–	–	5	20
–	–	–	–	–	–	6

c)
insert() genererer i tillegg 4 nye buckets som ikke er vist, slik at totalt antall buckets er 11 etter at postene er satt inn.
Fyllingsgraden beregnet av uordnet_sett.load_factor()=$\frac{10}{11} = 0.91$.

5.4 Grafer fasit

mst() for en lignende datastruktur er gjengitt i en tidligere gjennomgått oppgave (eksamen H2017).

```
void Node::settinn_kant(const Kant& kant) // Oppgave 4a
{
    m_kanter.push_back(kant);
}

Node* Graf::finn_node(char navn) // Oppgave 4a
{
    for (auto it=noder.begin(); it!=noder.end(); it++ )
        if ((*it)->m_navn == navn)
            return *it;
    return nullptr;
}

void Graf::settinn_node(char navn) // Oppgave 4a
{
    noder.push_back(new Node(navn));
}

void Graf::settinn_kant(char fra_navn, char til_navn, float vekt) // Oppgave 4b
{
    Node* franode = finn_node(fra_navn);
    Node* tilnode = finn_node(til_navn);
    if (franode && tilnode)
        franode->settinn_kant(Kant{vekt, tilnode});
}

float Graf::mst() // Oppgave 4c
{
    float sum = 0.0;
    std::priority_queue<Kant, std::vector<Kant>, std::greater<Kant> > apq;
    // Initialiserer
    Kant kant(0.0, noder.front());
    apq.push(kant);
```

```
    do {
        kant = apq.top();
        apq.pop();
        Node* p = kant.m_tilnode;
        if (!p->m_besokt) {
            p->m_besokt = true;

            sum += kant.m_vekt;
            for (auto it=p->m_kanter.begin(); it!= p->m_kanter.end(); it++)
            apq.push(*it);
        }
    } while (!apq.empty());
    return sum;
}

// Lage en testgraf G={V,E}
// hvor V={A, B, C, D, E}
// og E = {AB(1.0), AC(2.0), BC(2.0), CD(3.0), DE(1.0), AE(5.0), CE(4.0)
int main() // Oppgave 4b
{
    Graf graf;
    for (auto ch='a'; ch<'f'; ch++)
        graf.settinn_node(ch);

    graf.settinn_kant('a', 'b', 1.0f);
    graf.settinn_kant('a', 'c', 2.0f);
    graf.settinn_kant('b', 'c', 2.0f);
    graf.settinn_kant('c', 'd', 3.0f);
    graf.settinn_kant('d', 'e', 1.0f);
    graf.settinn_kant('a', 'e', 5.0f);
    graf.settinn_kant('c', 'e', 1.0f);

    cout << graf.mst();
    return 0;
}
```