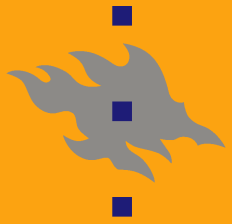
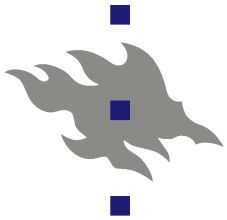


HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

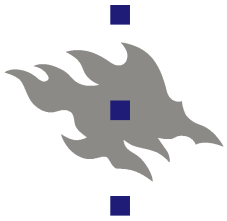
C++-pikakurssi, osa 3/3

Algoritmit ongelmanratkaisussa, kevät 2018



Tietorakenteita

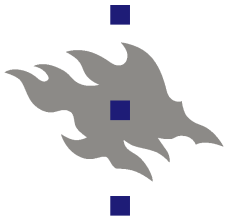
C++	Java
`set`	`TreeSet`
`multiset`	(ei suoraa vastinetta)
`unordered_set`	`HashSet`
`map`	`TreeMap`
`unordered_map`	`HashMap`
`priority_queue`	`PriorityQueue`



set-rakenne

```
set<int> s;
s.insert(2);
s.insert(5);
cout << s.count(2) << "\n"; // 1
s.erase(2);
cout << s.count(2) << "\n"; // 0
```

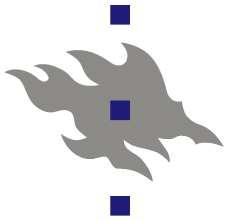
- set-rakenne pitää yllä joukkoa alkioista
- perusoperaatiot: lisäys (`insert`), poisto (`erase`) ja lukumäärä (`count`)
- jokainen alkio voi esiintyä enintään kerran (eli lukumäärä on aina joko 0 tai 1)



Pienin alkio

```
set<int> s = {1,2,3};
auto it = s.begin();
cout << *it << "\n"; // 1
```

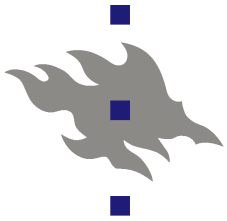
- iteraattori `begin` osoittaa joukon ensimmäiseen alkioon
- joukko on järjestyksessä, joten tämä kertoo pienimmän alkion
- alkion arvon saa selville `*`-syntaksilla



Suurin alkio

```
set<int> s = {1,2,3};
auto it = s.end();
it--;
cout << *it << "\n"; // 3
```

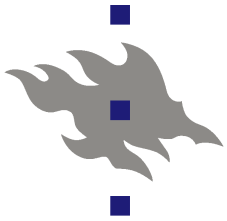
- iteraattori `end` osoittaa joukon viimeisen alkion *jälkeiseen* alkioon
- vähentämällä iteraattorin arvoa yhdellä päästään joukon suurimpaan alkioon



Alkion etsiminen

```
set<int> s = {1,2,3};
auto it = s.find(2);
if (it == s.end()) cout << ":(";
```

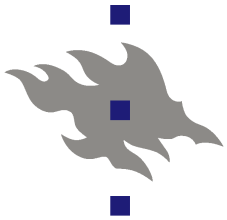
- `find`(x) antaa iteraattorin alkioon x
- jos alkiota ei löydy, tuloksena on `end()`, joka osoittaa joukon ulkopuolelle



Alkion etsiminen

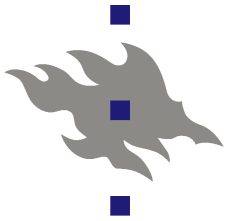
```
set<int> s = {1,2,3};
auto it1 = s.lower_bound(2);
cout << it1 << "\n"; // 2
auto it2 = s.upper_bound(2);
cout << it1 << "\n"; // 3
```

- `lower_bound`(*x*) etsii pienimmän alkion, joka on vähintään *x*
- `upper_bound`(*x*) etsii pienimmän alkion, joka on suurempi kuin *x*
- myös näissä `end()` kertoo, ettei alkiota ole



Tehokkuus

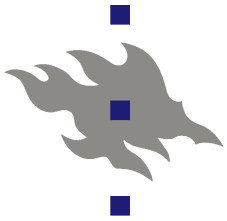
- oleellista set-rakenteessa on, että kaikki edellä esitetyt operaatiot toimivat tehokkaasti $O(\log n)$-ajassa
- rakenteen toteutus perustuu tasapainoiseen binääripuuhun (esim. punamusta puu)



multiset-rakenne

```
multiset<int> s;
s.insert(1);
s.insert(1);
s.insert(1);
cout << s.count(1) << "\n"; // 3
```

- multiset on kuin set, mutta sama alkio voi esiintyä monta kertaa rakenteessa
- `count` voi siis palauttaa muutakin kuin 0 tai 1

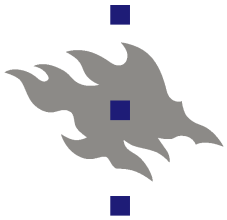


multiset-rakenne

```
// poista kaikki x:n kopiot
s.erase(x);
```

```
// poista yksi x:n kopio
s.erase(s.find(x));
```

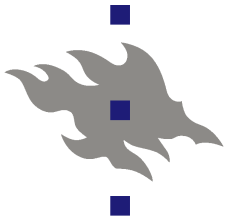
- funktio `erase` poistaa kaikki alkion kopiot, jos sille annetaan alkio
- yhden kopion voi kuitenkin poistaa antamalla iteraattorin (jonka voi etsiä `find`-funktiolla)



unordered_set-rakenne

```
unordered_set<int> s;
s.insert(1);
s.insert(2);
s.insert(3);
```

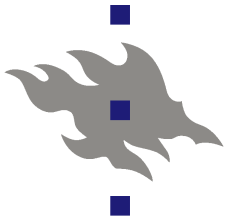
- unordered_set on kuin set, mutta se perustuu hajautustauluun eikä binääripuuhun
- operaatiot toimivat keskimäärin $O(1)$-ajassa
- alkiot eivät ole järjestyksessä, minkä vuoksi ei voi etsiä pienintä ja suurinta alkiota eikä käyttää funktioita `lower_bound` ja `upper_bound`



map-rakenne

```
map<string,int> x;
x["apina"] = 1;
x["banaani"] = 2;
x["cembalo"] = 3;
```

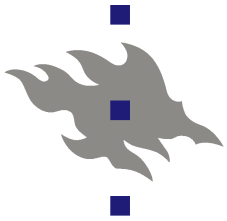
- map on taulukon yleistys, jossa avaimina voi olla mitä tahansa alkioita
- map perustuu tasapainoiseen binääripuuhun, vastaavasti unordered_map perustuu hajautustauluun



map-rakenne

```
map<string,int> x;
cout << x["aybabtu"] << "\n"; // 0
```

- jos avainta ei ole olemassa, se lisätään automaattisesti oletusarvolla
- esimerkiksi int-tyypin oletusarvo on 0 ja string-tyypin oletusarvo on tyhjä merkkijono



priority_queue-rakenne

```
priority_queue<int> q;
q.push(3);
q.push(1);
q.push(2);
cout << q.top() << "\n"; 3
q.pop();
cout << q.top() << "\n"; 2
```

- `push` lisää alkion, `top` hakee suurimman alkion ja `pop` poistaa suurimman alkion
- toimii tehokkaammin kuin set, mutta sisältää vähemmän ominaisuuksia