Data structures

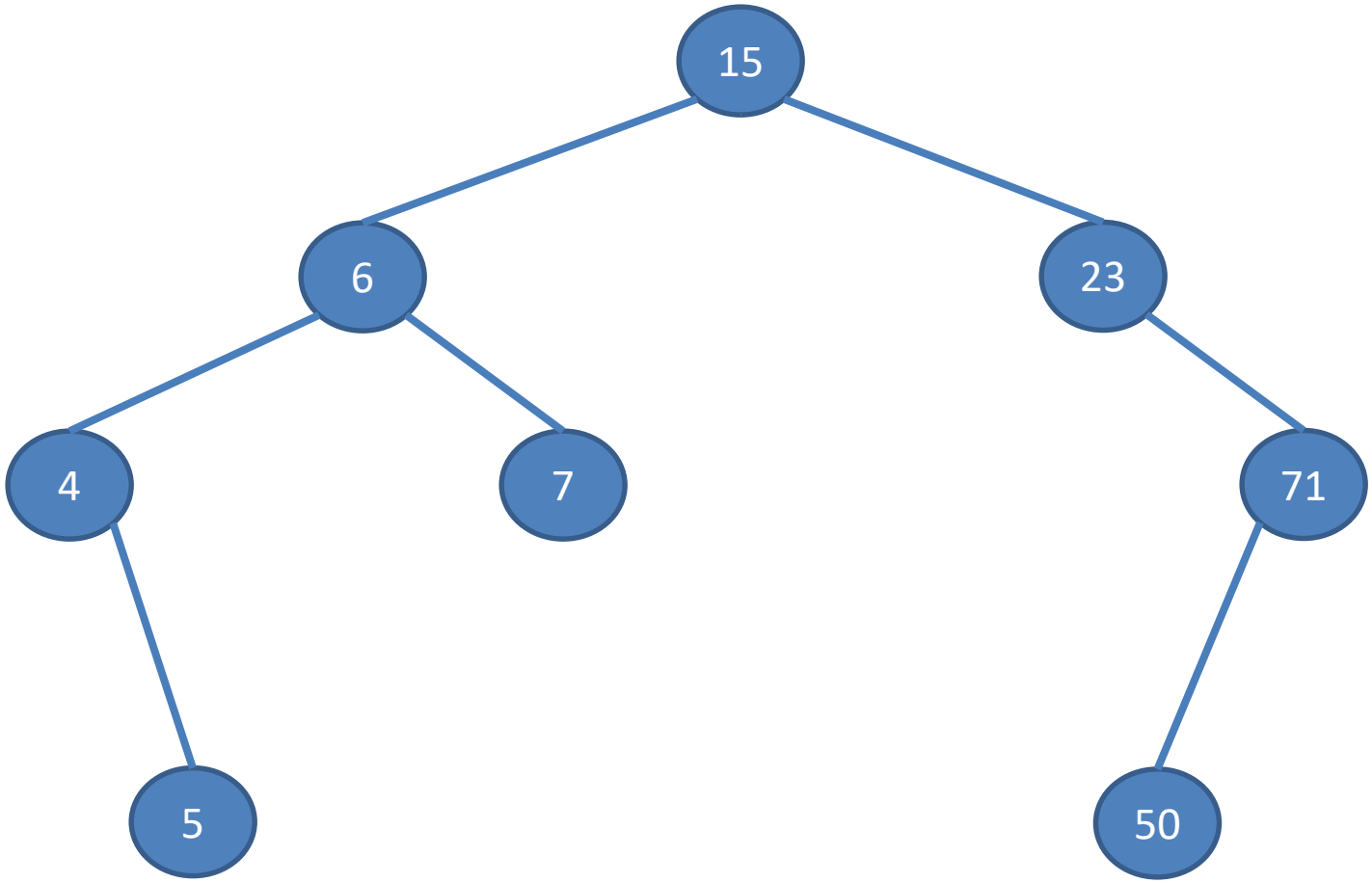
Non-linear data structures

- ในบางปัญหา linear storage ไม่ใช่วิธีที่ดีที่สุดในการจัดการข้อมูล
- ในหัวข้อนี้เราจะมาพิจารณา non-linear data structure ซึ่งจะทำให้เราสามารถจัดการข้อมูลได้รวดเร็วขึ้น ซึ่งจะทำให้ algorithm ของเราทำงานเร็วขึ้น
- ตัวอย่างเช่นถ้าเราต้องการกลุ่มของคู่อันดับที่เปลี่ยนแปลงขนาดได้(เช่น key->value) การใช้ map จะทำให้ประสิทธิภาพการทำงานเป็น O(logn) สำหรับการ insert/search/delete โดยเขียนเพียงไม่กี่บรรทัด ขณะที่หากเก็บโดยใช้ static array ของ struct จะต้องใช้เวลาเป็น O(n) ในการ insert/search/delete และต้องเขียน code ในการ traverse ยาวด้วย

Balanced Binary Search Tree

- Balanced Binary Search Tree(BST): ใน C++ STL map/set
- BST เป็นวิธีหนึ่งในการจัดการข้อมูลในโครงสร้าง tree ในแต่ละ subtree ที่มี root ที่ x คุณสมบัติของ BST คือ item ของ left subtree ของ x จะมีค่าน้อยกว่า x และ item ใน right subtree ของ x จะมากกว่าหรือเท่ากับ x สิ่งนี้จำเป็นใน application ที่ใช้เทคนิค divide and conquer
- การจัดการข้อมูลของ BST นั้น search(key) insert(key) findmin() findmax() successor(key)/predecessor(key) และ delete(key) ทำงานใน O(logn) เนื่องจากว่า worst case ระยะทางจาก root to leaf เป็น O(logn) ทั้งนี้เวลาจะได้เช่นนี้ BST ต้อง balance

ตัวอย่าง BST



- ในการเขียน balanced BST เช่น Adelson–Velskii Landis (AVL) หรือ Red–Black (RB) trees นั้นเป็นงานที่เสียเวลาและยากภายใต้เวลาที่จำกัด นอกจากเราจะเตรียม code ไปแข่งด้วย อย่างไรก็ตาม C++ มี map และ set ที่โดยทั่วไปแล้ว implement ด้วย RB tree ซึ่งรับประกันการดำเนินการหลักของ BST เช่น insert/search/delete ว่าทำงานใน O(logn)
- หากใช้ 2 โครงสร้างนี้คล่องก็ช่วยประหยัดเวลาในขณะแข่งได้ ความแตกต่างของ 2 โครงสร้างนี้คือ map เก็บคู่อันดับของ (key->data) ส่วน set เก็บเพียง key
- ปัญหาส่วนใหญ่เราจะใช้ map มากกว่า set โดยเอาไว้ใช้ map สิ่งของ ส่วน set เอาไว้ใช้ตัดสินว่ามี key นี้หรือไม่

Set structures

- Set เป็นโครงสร้างข้อมูลที่เก็บกลุ่มของสมาชิก(เก็บว่ามีหรือไม่มี) การดำเนินการ (operation) พื้นฐานของ set คือ insertion, search removal
- C++ standard library มีการ implement set 2 แบบ
  - โครงสร้างข้อมูล set อยู่บนพื้นฐานของ balanced binary search tree และการดำเนินการของทันทำงานใน O(logn)
  - โครงสร้างข้อมูล unordered_set ใช้ hashing และการดำเนินการทำงานใน O(1) โดยเฉลี่ย

- ทีนี้การจะเลือกใช้งาน set implement แบบไหนนั้นก็แล้วแต่ ประโยชน์ของ set คือมันเก็บลำดับของสมาชิกและมีฟังก์ชันที่ unorderer_set ไม่มี
- ส่วน unordered_set นั้นทำงานรวดเร็ว มีประสิทธิภาพ
- code ต่อไปเป็นการสร้าง set และเก็บ integer จากนั้นแสดงตัวอย่างการใช้ operation บางอัน
- function insert เป็นการเพิ่มข้อมูลให้กับ set
- function count คืนค่า 0 ถ้าไม่มีตัวที่สอบถาม คืนค่า 1 ถ้ามีตัวที่สอบถามใน set
- function erase จะลบสมาชิกออกจาก set

```
set<int> s;
s.insert(3);
s.insert(2);
s.insert(5);
cout << s.count(3) << "\n"; // 1
cout << s.count(4) << "\n"; // 0
s.erase(3);
s.insert(4);
cout << s.count(3) << "\n"; // 0
cout << s.count(4) << "\n"; // 1
```

- set สามารถถูกใช้คล้ายกับ vector แต่มันไม่สามารถเข้าถึงข้อมูลโดยการใช้ []
- ตัวอย่างต่อไปเป็นการสร้างเซต สอบถามจำนวนสมาชิกใน set และ print สมาชิกทุกตัว

```
set<int> s = {2,5,6,8};
cout << s.size() << "\n"; // 4
for (auto x : s) {
    cout << x << "\n";
}
```

- คุณสมบัติที่สำคัญอย่างหนึ่งของ set คือสมาชิกจะไม่มีตัวซ้ำ ดังนั้น ฟังก์ชัน count จะคืนค่า 0 (ถ้าไม่มีสมาชิกใน set) หรือ 1 (ถ้ามีสมาชิกใน set) ส่วนฟังก์ชัน insert จะไม่เพิ่มสมาชิกซ้ำเข้าไปใน set ถ้าสมาชิกตัวนั้นมีใน set แล้ว

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

http://www.cplusplus.com/reference/set/set/

- C++ ยังมี multiset และ unordered_multiset ที่ทำงานคล้าย set และ unordered_set แต่สามารถเก็บตัวซ้ำได้
- ตัวอย่างการใช้งาน multiset

```
multiset<int> s;
s.insert(5);
s.insert(5);
s.insert(5);
cout << s.count(5) << "\n"; // 3
```

- ฟังก์ชัน erase จะลบทุกตัวที่มีค่านั้นใน multiset

```
s.erase(5);
cout << s.count(5) << "\n"; // 0
```

หากต้องการลบตัวเดียว สามารถทำได้โดย

```
s.erase(s.find(5));
cout << s.count(5) << "\n"; // 2
```

http://www.cplusplus.com/reference/set/multiset/

Set iterators

- Iterators ถูกใช้ในการเข้าถึงข้อมูลใน set ตัวอย่างต่อไปเป็นการสร้าง iterator ชื่อ it ที่ชี้ไปยังสมาชิกตัวที่น้อยที่สุดใน set

```
set<int> s;

s.insert(3);

s.insert(2);

set<int>::iterator it = s.begin();
```

หากต้องการเขียนแบบสั้น ให้ใช้

```
auto it = s.begin();
```

- สมาชิกที่ iterator ชี้สามารถเข้าถึงได้โดยการใช้ * เนื่องจากว่าเป็น pointer

```
auto it = s.begin();
cout<<*it;
```

Iterator สามารถถูกย้ายได้โดยใช้ operator ++ (forward)และ –(backward) หมายความว่า iterator ย้ายไปสมาชิกตัวถัดไปหรือสมาชิกตัวก่อนหน้าใน set

- Code สำหรับการ print สมาชิกทุกตัวจากน้อยไปมาก

```
for (auto it = s.begin(); it != s.end(); it++) {
        cout << *it << "\n";
}
```

- code ในการหาตัวมากสุด

```
auto it = s.end(); it--;
cout << *it << "\n";
```

- ฟังก์ชัน find(x) คืน iterator ที่ชี้ไปยังสมาชิกที่มีค่าเท่ากับ x อย่างไรก็ตาม ถ้า set ไม่มี x iterator จะเป็น end
- auto it = s.find(x);
- if (it == s.end()) {
- // x is not found
- }
- ฟังก์ชัน lower_bound(x) คืน iterator ที่ชี้สมาชิกตัวที่น้อยที่สุดที่มีค่าอย่างน้อย x ส่วน upper_bound(x) คืน iterator ที่ชี้สมาชิกตัวที่น้อยที่สุดที่มีค่ามากกว่า x ทั้งสองฟังก์ชันถ้าไม่มีสมาชิกอยู่ใน set จะคืน end (ใช้กับ unordered_set ไม่ได้เพราะว่าไม่มีลำดับ)

- ตัวอย่าง code ในการหาสมาชิกตัวที่ใกล้กับ x

```
auto it = s.lower_bound(x);
if (it == s.begin()) {
        cout << *it << "\n";
} else if (it == s.end()) {
        it--;
        cout << *it << "\n";
} else {
        int a = *it; it--;
        int b = *it;
        if (x-b < a-x) cout << b << "\n";
        else cout << a << "\n";
}
```

Case นี้คือกรณีอะไร

Case นี้คือกรณีอะไร

Case นี้คือกรณีอะไร

Map structure

- Map เป็น generalized array ที่เก็บคู่ของ key–value ขณะที่ key ใน array ธรรมดานั้นเป็น integer ที่เรียงต่อกัน 0, 1, 2, ..., n–1 เมื่อ n เป็นขนาดของ array แต่ key ใน map สามารถเป็นชนิดข้อมูลใดๆ และไม่จำเป็นต้องเรียงต่อกัน
- C++ standard library มีการ implement map 2 แบบซึ่งสอดคล้องกับการ implement set นั่นคือ
  - map อยู่บนพื้นฐานของ balanced binary tree เข้าถึงข้อมูลใน O(logn)
  - unordered_map ใช้การ hash และเข้าถึงข้อมูลโดยใช้ O(1) โดยเฉลี่ย

- ต่อไปเป็นตัวอย่างการสร้าง map เมื่อ key เป็น string และ value เป็น integer

```
map<string,int> m;
m["monkey"] = 4;
m["banana"] = 3;
m["harpsichord"] = 9;
cout << m["banana"] << "\n"; // 3
```

- ถ้ามีการเรียก key นั้นแต่ map ไม่ได้เก็บค่า key นั้นจะถูกเพิ่มเข้าไปใน map ด้วยค่า default ตัวอย่างเช่น ถ้ามีการเรียก key "xxx" ค่า 0 จะถูกเพิ่มเข้าไปใน map

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

- ฟังก์ชัน count จะตรวจสอบว่า key นั้นอยู่ใน map หรือไม่

```
if (m.count("xxx")) {
    // key exists
}
```

- หากต้องการ print ทุก key และ value ใน map

```
for (auto x : m) {
    out << x.first << " " << x.second << "\n";
}
```

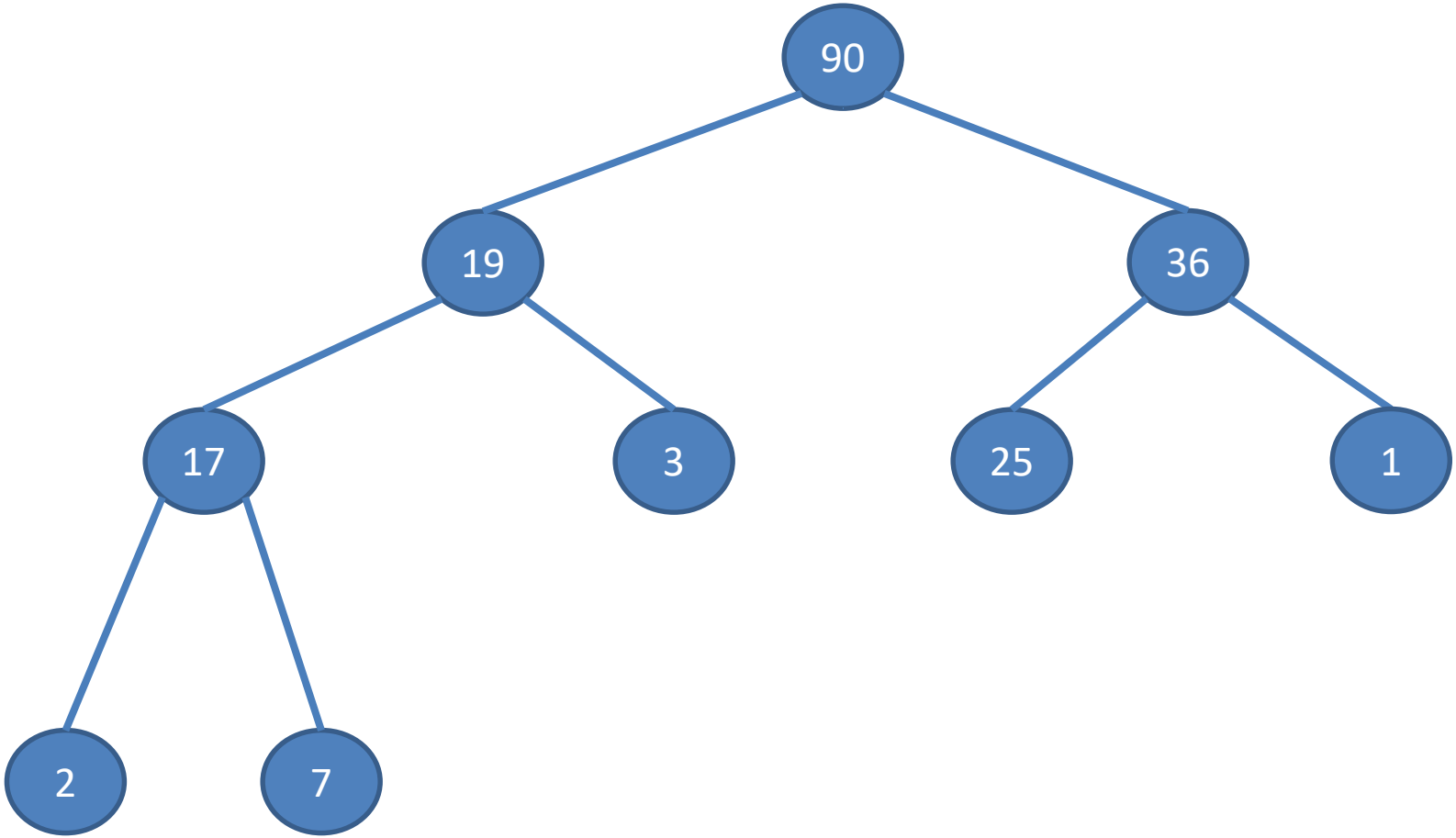
http://www.cplusplus.com/reference/map/map/

Heap

- Heap เป็นอีกวิธีหนึ่งในการเก็บข้อมูลใน tree
- (Binary) Heap คือ binary tree ซึ่งคล้ายกับ BST ยกเว้นมันจะต้องเป็น complete tree
- Complete Binary tree สามารถถูกเก็บได้อย่างมีประสิทธิภาพใน array 1 มิติที่ เริ่ม index ที่ 1 ที่มีขนาด n+1 ช่อง ซึ่งเราจะใช้ index ในการช่วยบอกว่าช่องไหนเป็นลูกทางซ้ายหรือขวาของช่องไหน
- index ที่ 0 ไม่ถูกใช้ เมื่อกำหนดช่องที่ i มาให้โหนดพ่อ โหนดลูกทางซ้าย โหนดลูกทางขวาสามารถคำนวณได้ตามลำดับนี้ $\left\lfloor \frac{i}{2} \right\rfloor, 2 * i, 2 * i + 1$
- การจัดการ index สามารถทำได้อย่างรวดเร็วโดยใช้ bit manipulation

i>>1, 1<<1, และ (i<<1)+1

- ตัวอย่างเช่น array A={N/A,90,19,36,17,3,25,1,2,7}



- การใช้โครงสร้าง (max) heap ทำให้มีคุณสมบัติของ heap นั่นคือในแต่ละ subtree ที่มี root ที่ x นั้น items ใน left และ right subtree ของ x จะมีค่าน้อยกว่าหรือเท่ากับ x
- คุณสมบัตินี้รับประกันว่า top หรือ root ของ heap นั้นจะเป็นตัวที่มีค่ามากที่สุดเสมอ
- ไม่รองรับการ search ใน heap ไม่เหมือน BST แต่รองรับการลบ (extract) ตัวที่มากที่สุด
- ExtractMax() และ insert item ใหม่(Insert v) ใช้เวลา O(logn) จากการเดินทาง root-to-leaf หรือ leaf-to-root และ สลับตำแหน่งกันเพื่อรักษาคุณสมบัติ heap

Priority queue

- Max heap เป็นโครงสร้างข้อมูลที่นำมาออกแบบ priority queue โดย item ที่มีความสำคัญมากจะถูก dequeue (ExtractMax()) และการนำของเข้า enqueue (Insert(v)) ทำงานใน O(logn)
- priority_queue ใน C++ อยู่ใน C++STL queue library
- priority queue ถูกใช้เป็นส่วนหนึ่งของหลาย algorithms เช่น Prim's MST, Dijkstra SSSP
- ขณะที่ ordered set นั้นรองรับทุกการดำเนินการของ priority queue แต่ประโยชน์ของการใช้ priority queue คือมันทำงานเร็วกว่า(ค่าคงที่น้อยกว่า) priority queue ส่วนใหญ่ถูก implement ด้วย heap ซึ่งอยู่ในรูปที่ง่ายกว่า balanced binary search ที่ถูกใช้ใน ordered set

- โดยทั่วไปสมาชิกใน C++ priority queue ถูกเก็บในลำดับที่ลดลง (decreasing) ลามารถหาหรือลบสมาชิกที่มีค่ามากที่สุดใน queue

```
priority_queue<int> q;
q.push(3);
q.push(5);
q.push(7);
q.push(2);
cout << q.top() << "\n"; // 7
q.pop();
cout << q.top() << "\n"; // 5
```

- ถ้าเราต้องการที่จะสร้าง priority queue ที่รองรับการค้นหาและลบข้อมูลตัวที่น้อยที่สุดสามารถทำได้
- priority_queue<int,vector<int>,greater<int>> q;
- http://www.cplusplus.com/reference/queue/priority_queue/

Hash table

- Hash Table ถูกใช้ใน unordered_map โดย hash table เป็นอีกโครงสร้างข้อมูลแบบ non-linear ไม่แนะนำที่จะนำไปใช้ในการแข่งเขียนโปรแกรมเท่าไรถ้าไม่จำเป็นจริงๆ
- ทั้งนี้ map/set ก็เร็วเพียงพอเมื่อขนาดข้อมูลเข้าโดยทั่วไปไม่เกิน 1M ประสิทธิภาพของ O(1) จาก hash table และ O(log1M) จาก BST ไม่ต่างกันมาก