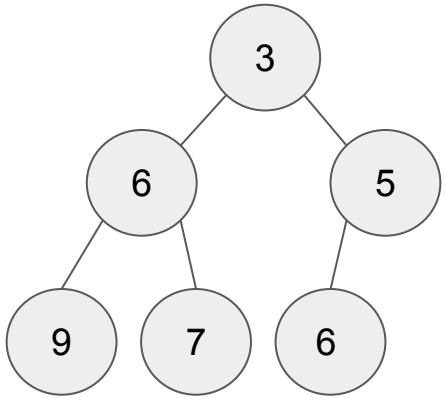
Heap

TA 黃頂軒
TA 李佳樺

slides: https://reurl.cc/QRM97p
records: https://reurl.cc/vaMqdy

Heap 結構

- 二元樹結構
- 每個節點為子樹的最小值 (Min Heap)
- Complete Binary Tree



Heap 操作

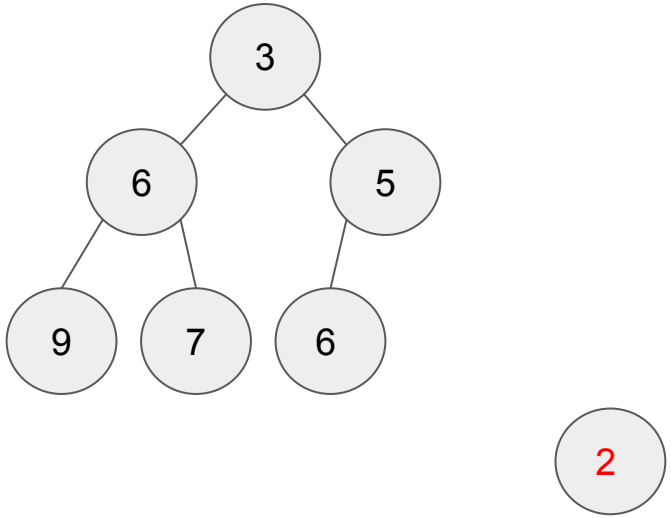
- `push(x)`：將 `x` 加入 heap 中
- `top()`：得到 heap 中最小值
- `pop()`：將 heap 中的最小值移除

push(x)

- 將 x 放到最後一層的第一個空位
- 調整結構維持 heap 性質

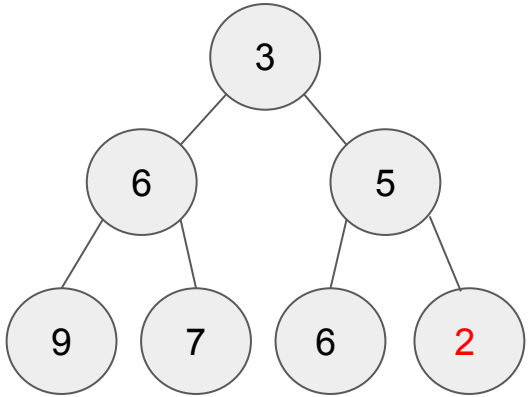
push(x)

以 push(2) 為例



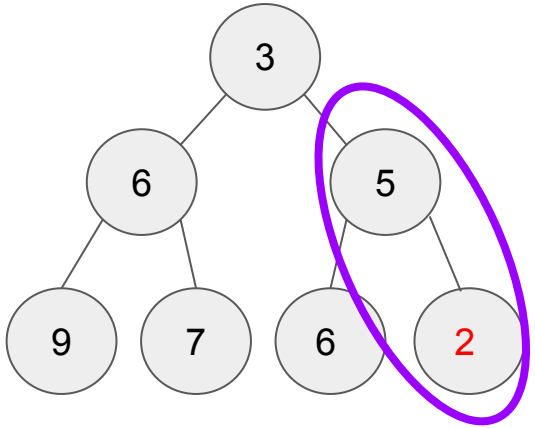
push(x)

以 push(2) 為例



push(x)

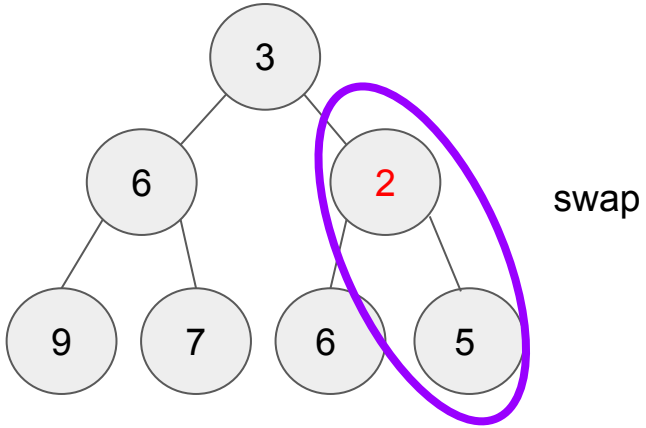
以 push(2) 為例



不滿足 heap 性質

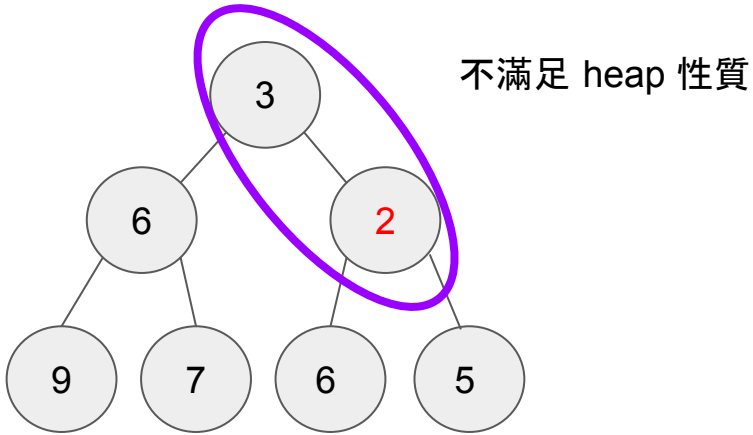
push(x)

以 push(2) 為例



push(x)

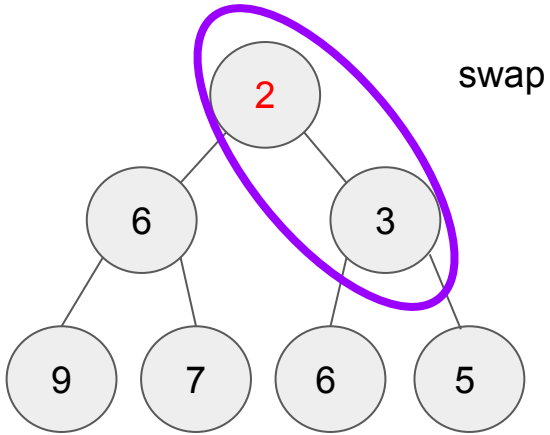
以 push(2) 為例



不滿足 heap 性質

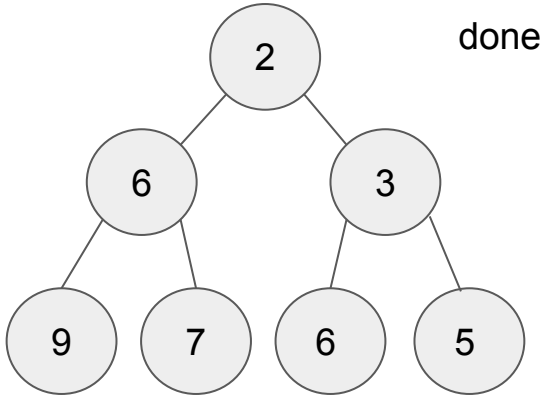
push(x)

以 push(2) 為例



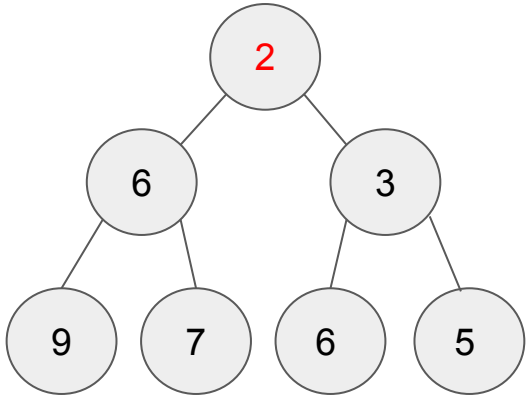
push(x)

- 檢查次數 = 樹高
  - O(log n)



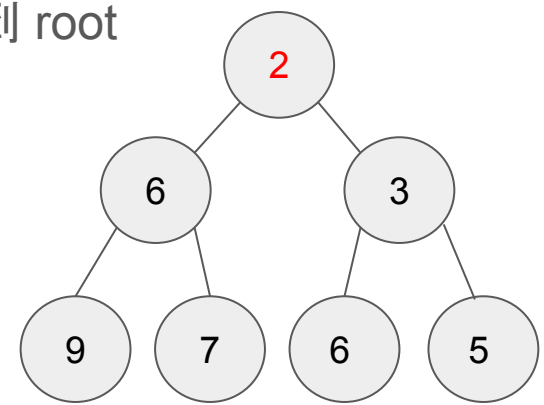
top()

- 二元樹的樹根就是最小值 (heap property)
- O(1)



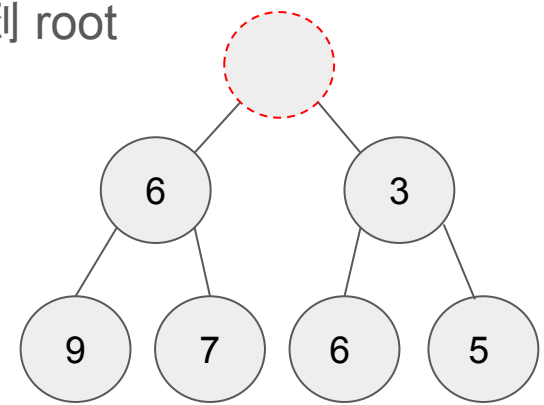
pop()

- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質



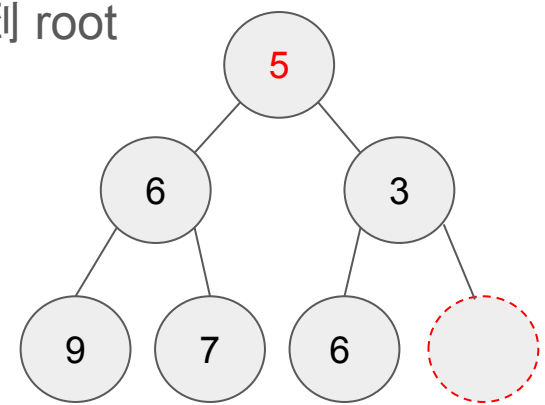
pop()

- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質



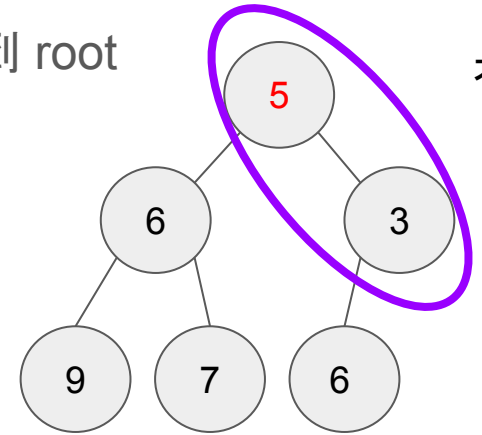
pop()

- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質



pop()

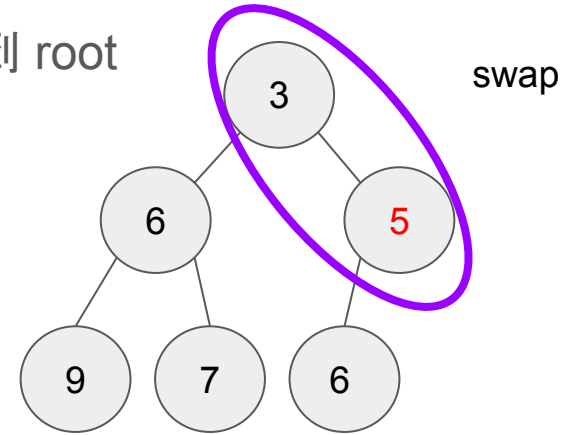
- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質



不滿足 heap 性質

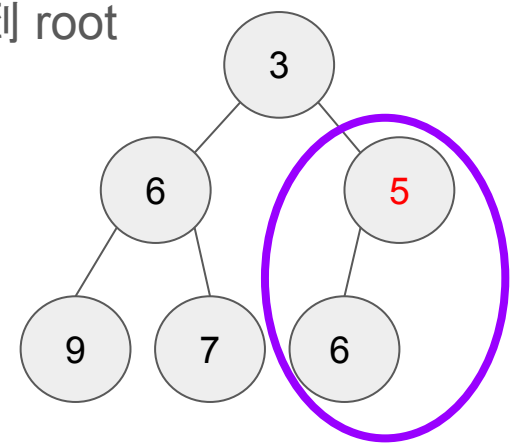
pop()

- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質



pop()

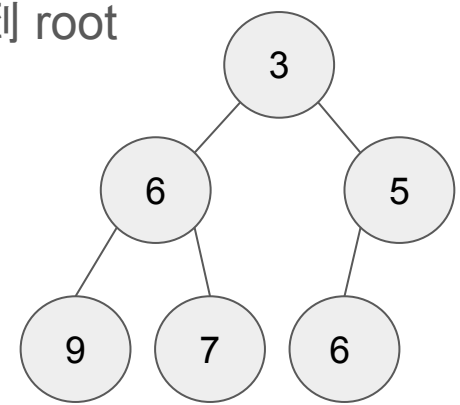
- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質



滿足 heap 性質

pop( )

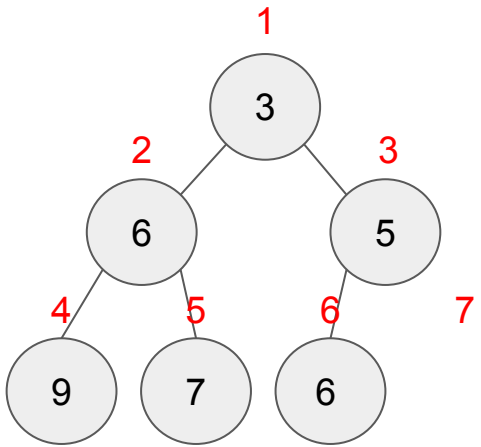
- 把最小值移除
- 把最後一層最後一個數字搬到 root
- 調整結構維持 heap 性質
- 檢查次數 = 樹高
  - O(log n)



done

實作

- 用陣列模擬二元樹結構
- 樹根編號為 1
  - v 的左孩子編號為 v * 2 + 0
  - v 的右孩子編號為 v * 2 + 1



實作

```
// min heap
struct Heap {
    std::vector<int> data;

    Heap() : data(1) {}
```

```
void push(int x) {
    int v = data.size();
    data.push_back(x);
    while(v > 1 && data[v] < data[v / 2]) {
        std::swap(data[v], data[v / 2]);
        v /= 2;
    }
}
```

實作

```
void pop() {
    data[1] = data.back();
    data.pop_back();
    int v = 1;
    while(v * 2 < (int) data.size()) {
        int L = data[v * 2];
        int R = (v * 2 + 1 < (int) data.size()) ? data[v * 2 + 1] : INT_MAX;
        if(data[v] <= L && data[v] <= R) {
            return;
        }
        if(L < R) {
            std::swap(data[v], data[v * 2]);
            v = v * 2;
        } else {
            std::swap(data[v], data[v * 2 + 1]);
            v = v * 2 + 1;
        }
    }
}
```

實作

```
int top() {
    return data[1];
}
```

```
int size() {
    return data.size() - 1;
}

bool empty() {
    return data.size() == 1;
}
```

Priority Queue

Priority Queue 是一種維護集合的想法，它需要支援以下幾種操作：

- Insert(x)：將 x 放入集合
- Minimum：輸出集合內的最小
- Extract-Min：將集合內最小移除
- Increase-Key(i, key)：將 i 的值增加為 key（key 必須比原本的值大！）

Priority Queue != Binary Heap (Priority Queue 是精神，Binary Heap 是實作)

Priority Queue 可以不用 Binary Heap 實作（但複雜度可能比較差）

Priority Queue vs Binary Heap

Priority Queue	Binary Heap	Pure Array
Build	O(n) Why not O(n log n)?	O(n)
Insert	O(log n)	O(1)
Minimum	O(1)	O(n)
Extract-Min	O(log n)	O(n)
Increase-Key	O(log n)	O(1)

Binary Heap

O(n log n) 也是對的，但是太悲觀了。我們有更好分析 upper bound 的方法。

一樣用 n 次 Insert 操作來 Build Binary Heap：

1. 第 i 層最多往 parent 檢查 i 次
2. 第 i 層最多有 2^i 個節點

$$\sum_{i=1}^{\lceil \log n \rceil} i \cdot 2^i = O(n)$$

內建 STL

- `std::priority_queue<int>`
  - 預設是最大值
  - 最小值用 `std::priority_queue<int, std::vector<int>, std::greater<int>>`
- 也可以直接用 `std::set` / `std::multiset` 等內建二元搜尋樹找最大值 / 最小值
- CPP reference
  - std::priority_queue
  - std::set
  - std::multiset

Practice

- https://acm.cs.nthu.edu.tw/contest/3035/
- std::priority_queue：https://ideone.com/9RoLwM
- std::multiset：https://ideone.com/CoQjM6