

2023 여름학기 동국대학교 SW역량강화캠프

13일차. 우선순위 큐

- Heap (우선순위 큐)

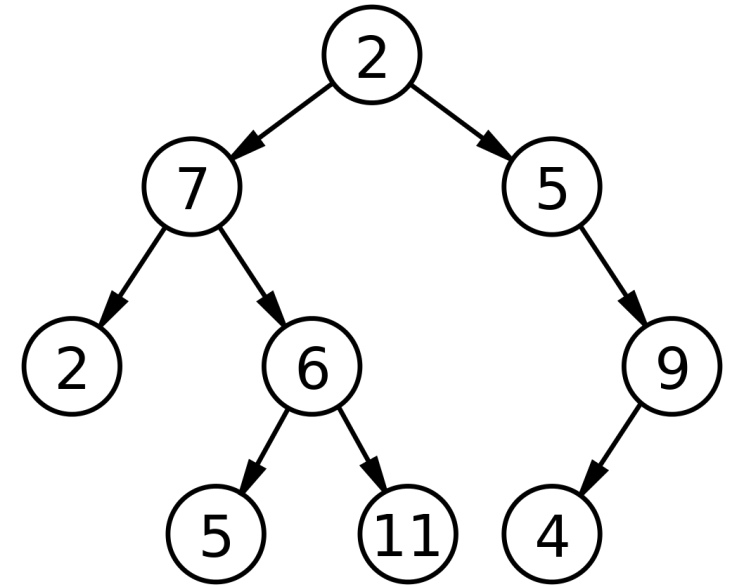
- ▶ 큐는 FIFO (First In First Out)

- ▶ 우선 순위 큐는 들어온 순서와 상관없이 우선순위에 따라 먼저 나가는 값 결정

- ▶ 삽입 $O(\lg N)$, 삭제 $O(\lg N)$

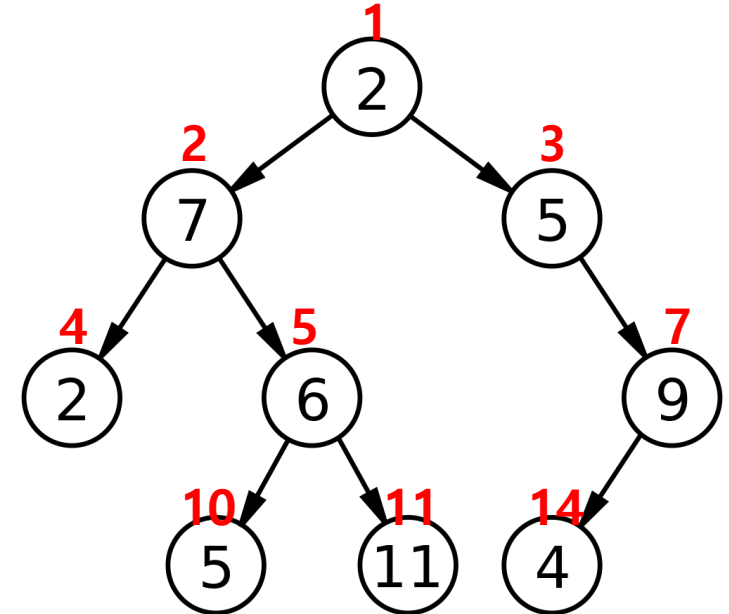
● 이진트리

- ▶ 각각의 노드가 최대 2개의 자식을 가질 수 있는 트리 자료구조
- ▶ 부모 노드, 자식 노드, 루트 노드, 리프 노드



● 배열로 표현하는 이진트리

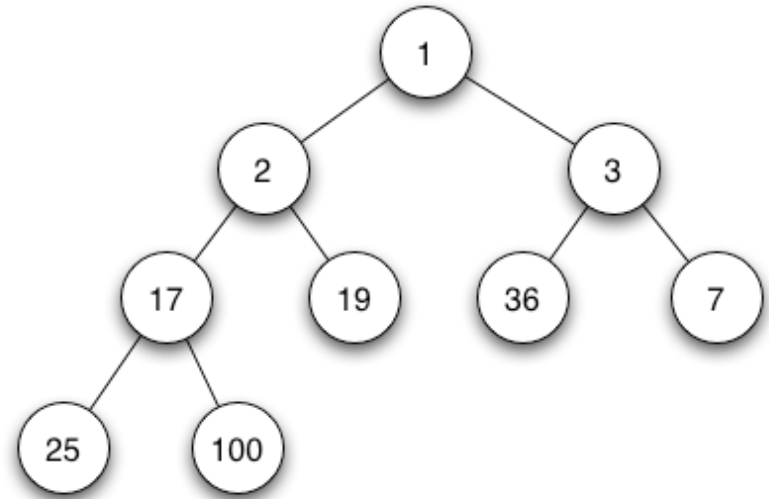
- ▶ 루트 노드의 번호는 1
- ▶ 왼쪽 자식 노드의 번호는 $2 * (\text{노드 번호})$
- ▶ 오른쪽 자식 노드의 번호는 $2 * (\text{노드 번호}) + 1$
- ▶ 부모 노드의 번호는 $(\text{노드 번호}) / 2$



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	7	5	2	6	-	9	-	-	5	11	-	-	4	-

● 힙

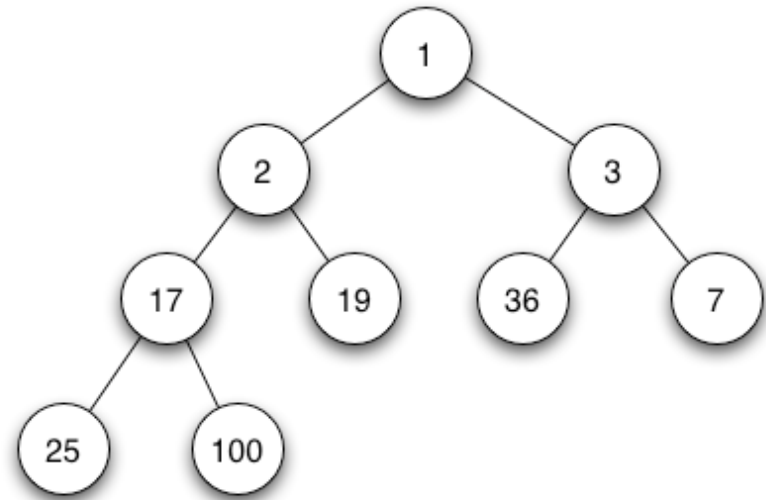
- ▶ 이진트리로 구현하는 우선순위 큐
- ▶ 배열로 표현하는 이진트리를 이용하여 앞에서부터 채워나간
- ▶ 부모 노드는 언제나 자식 노드보다 값이 작다(또는 크다)



● 힙

▶ 값의 추가

```
static void add(int x){  
    tree[++size] = x;  
  
    int idx=size;  
    while(idx > 1){  
        if(tree[idx] < tree[idx/2]) swap(idx, idx/2);  
        else break;  
  
        idx /= 2;  
    }  
}
```



● 힙

▶ 값의 제거

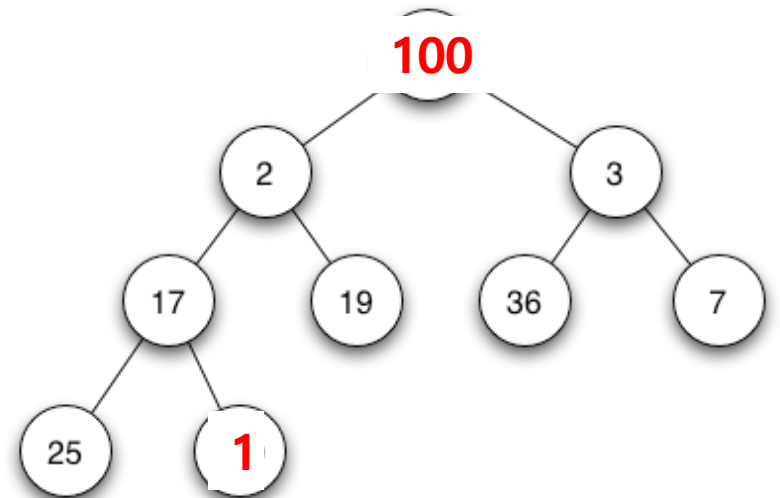
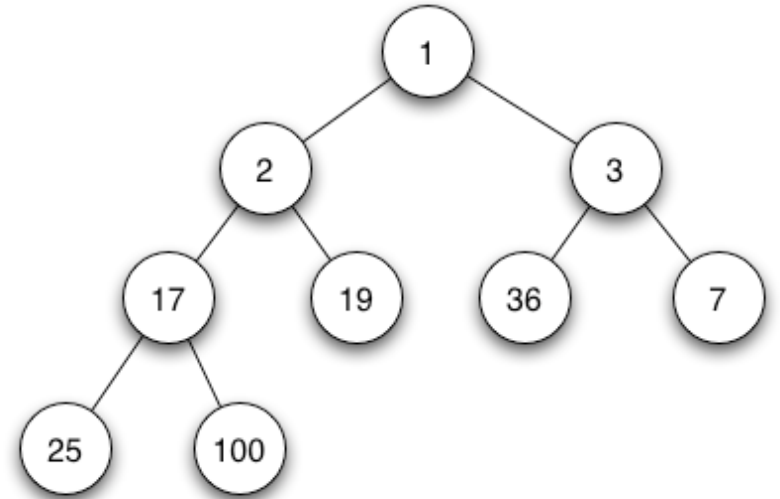
```
static int remove(){
    if(size == 0) return 0;
    int top = tree[1];

    tree[1] = tree[size];
    tree[size--] = 0;

    int idx = 1;
    while(idx*2 <= size){
        int l=idx*2, r=idx*2+1;

        if((r > size || tree[l] < tree[r]) && tree[idx] > tree[l]){
            swap(idx, l);
            idx = l;
        }
        else if(r <= size && tree[idx] > tree[r]){
            swap(idx, r);
            idx = r;
        }
        else break;
    }

    return top;
}
```



● 최소 힙 (4597)

| 문제

최소 힙을 구현해보자.

1. 힙에 자연수 x 를 넣는다.
2. 현재 가장 작은 값을 출력하고, 그 값을 최소 힙에서 제거한다.

처음에는 비어 있는 힙에서 시작한다.

우선순위 큐 핵심코드 (선언)

- ▶ 우선순위 큐 자료구조는 java.util.PriorityQueue 에 정의되어 있습니다.

```
import java.util.PriorityQueue;
```

- ▶ PriorityQueue 자료구조는 다음과 같이 선언할 수 있습니다.

```
PriorityQueue<Integer> pq = new PriorityQueue<>();  
//작은 수부터 나오는 우선순위 큐  
PriorityQueue<Integer> pq = new PriorityQueue<>(Collections.reverseOrder());  
//큰 수부터 나오는 우선순위 큐  
//참조형 변수로만 선언 가능합니다  
//PriorityQueue<Object> pq_name = new PriorityQueue<>();
```

```
from queue import PriorityQueue
```

```
pq = PriorityQueue()
```



우선순위 큐 핵심 코드 (메소드)

- ▶ Queue와 마찬가지로 size, isEmpty, offer, poll, peek 메소드를 사용할 수 있습니다.

```
int Size = pq.size(); // Size에 pq에 들어있는 원소의 수를 저장합니다. O(1)
boolean empty = pq.isEmpty(); // empty에 pq가 비어있으면 true, 아니면 false를 저장합니다. O(1)
pq.offer(x); // pq에 x를 집어넣습니다. O(lgN)
int y = pq.peek(); // pq에서 가장 우선순위가 높은 수를 y에 저장합니다. O(1)
int z = pq.poll(); // pq에서 가장 우선순위가 높은 수를 z에 저장하고 pq에서 제거합니다. O(lgN)
```

- ▶ contains 메소드를 이용하여 pq에 원하는 값이 들어있는지 확인할 수 있습니다.

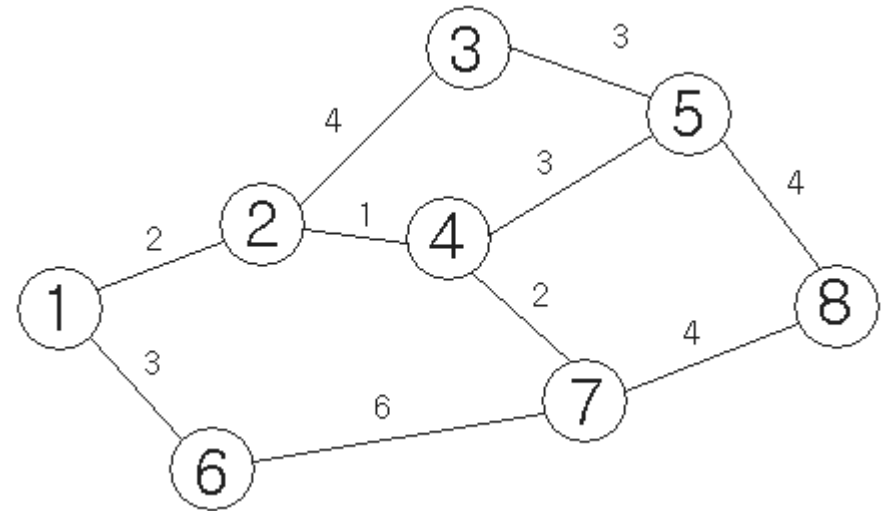
```
boolean containx = pq.contains(x); // pq에 값 x가 들어있는지 확인하여 그 결과를 저장합니다. O(N)
```

```
pq.put(x)
x = pq.get()
```



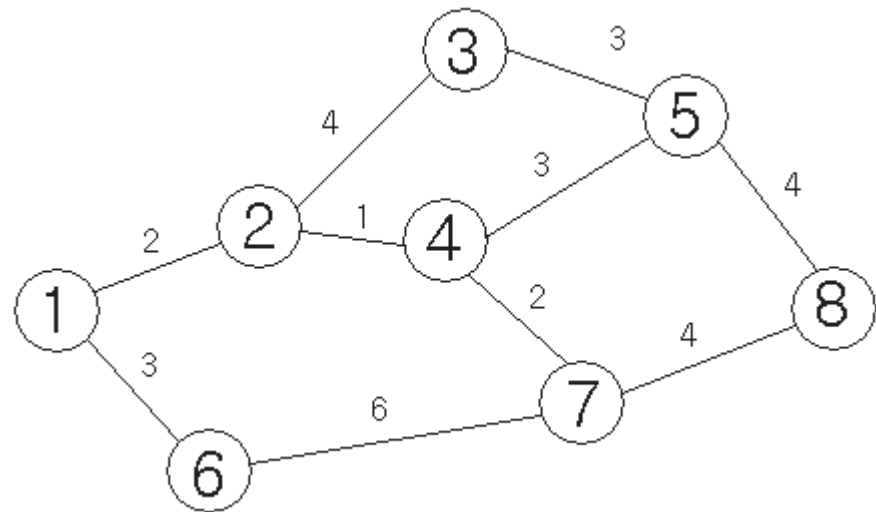
● 다익스트라

- ▶ 간선에 길이가 주어지는 그래프에서의 최단경로 구하기
- ▶ 간선의 길이가 모두 1이라면?
- ▶ 거리가 짧은 순서대로 진행하는 BFS



● 다익스트라

- ▶ $vst[]$ = 방문했다면 true, 아니라면 false(거리 확정 여부)
- ▶ $dist[]$ = 시작점으로부터의 거리
- ▶ pq에서 하나를 꺼내 점 X를 D만큼의 거리에 갈 수 있다
- ▶ $vst[X]$ 가 false라면 $vst[X]$ 를 true로 바꾸고 인접한 점 탐색



● 통행료 (4791)

| 문제

도원이가 다니는 백금고등학교의 1학년은 A반과 B반 2개의 반으로 이루어져 있다.

이번에 백금고등학교에서는 체육대회를 개최하는데, 많은 참여를 위해 1학년은 A반과 B반의 1대1 줄다리기를 진행하기로 하였다.

A반과 B반은 둘 다 n 명의 학생들로 이루어져 있는데, 각 반은 이 n 명의 학생을 1번부터 n 번까지 번호를 붙혀, 같은 번호의 학생끼리 1대1 줄다리기를 진행하게 된다.

A반에 다니고 있는 도원이는 인맥을 활용하여 B반의 1번 학생부터 n 번 학생까지 줄을 당기는 힘이 얼마인지 알아내었다..!

도원이는 A반 학생들의 줄을 당기는 힘이 얼마인지도 전부 알고 있기 때문에, 번호를 붙힐 때 B반의 학생들을 최대한 많이 이길 수 있도록 하려고 한다.

이 때 도원이가 B반 학생을 최대 몇 명 이기도록 A반 학생을 배치할 수 있을 지 출력해보자. (모든 학생의 당기는 힘은 전부 다르다!)

| 입력

첫째 줄에 A반과 B반의 학생의 수 n 이 주어진다 ($1 \leq n \leq 100,000$)

둘째 줄에 B반의 1번 학생부터 n 번 학생까지의 당기는 힘이 주어진다.

셋째 줄에 A반의 학생 n 명의 당기는 힘이 주어진다. (당기는 힘은 100만 이하의 자연수로 주어진다)

```
pq.add(new pqNode(1, 1));
while(!pq.isEmpty()){
    pqNode tmp = pq.poll();
    int v = tmp.getV();
    int dis = tmp.getDis();

    if(minDis[v] > 0) continue;
    minDis[v] = dis;

    for(Edge e: edge[v]){
        int u = e.getTo();
        int d = e.getDis();

        pq.add(new pqNode(u, dis+d));
    }
}

System.out.print(minDis[N]-1);
```

```
static PriorityQueue<pqNode> pq = new PriorityQueue<>();
```

```
class pqNode implements Comparable<pqNode>{
    private int v, dis;

    public pqNode(int v, int dis){
        this.v = v;
        this.dis = dis;
    }

    public int getV() { return this.v; }
    public int getDis() { return this.dis; }

    @Override
    public int compareTo(pqNode node){
        if(this.dis > node.getDis()) return 1;
        if(this.dis == node.getDis()) return 0;
        return -1;
    }
}
```