ICPC Sinchon

08. Tree, Graph

08. 트리, 그래프

2023 Summer Algorithm Camp

목차

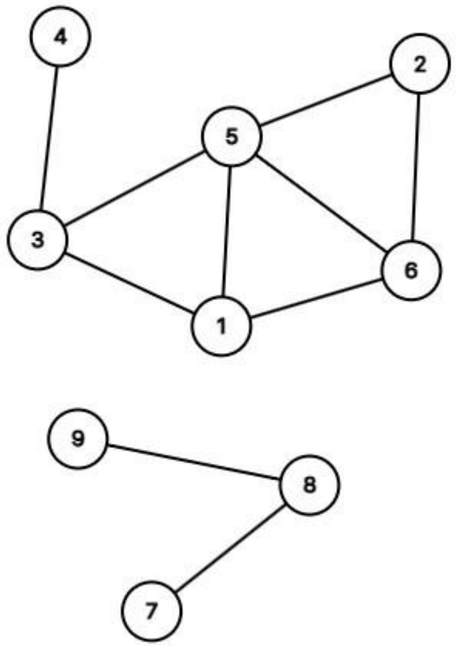
2023 Summer Algorithm Camp

그래프 (Graph)

- **정점**Vertex, Node과 **간선**Edge, 모서리으로 표현하는 자료구조
  - 쉽게 표현하면, 정점은 “점”이고 간선은 “점을 잇는 선”임.

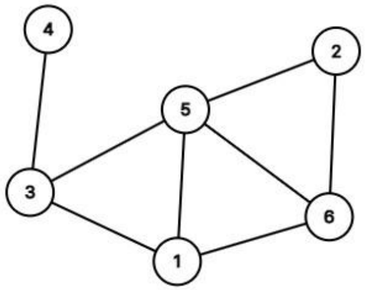
- 정점 집합을 V, 간선 집합을 E라고 하면, 그래프는 **G(V, E)**라고 나타낸다.

- 대표적인 비선형 자료구조
  - 하나의 정점은 여러 개의 정점과 연결될 수 있다.

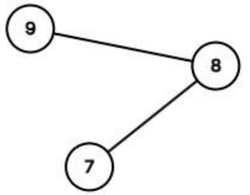


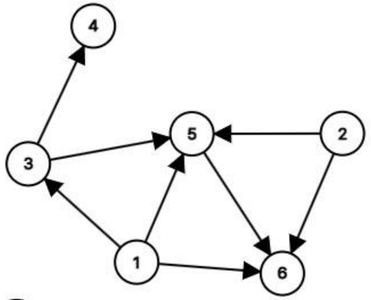
용어 - 간선

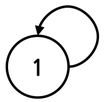
- 간선은 **꼬리 정점**tail이 u이고 **머리 정점**head이 v일 때, (u, v)와 같이 표기한다.
  - 정점 u와 v는 **인접**adjacent한다고 한다.
  - 간선 (u, v)는 정점 u, v에 **부속**incident되어 있다고 한다.

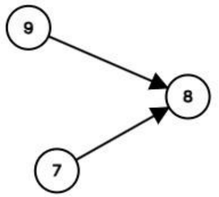


- 방향이 있으면 방향단방향, 유향
  - 방향이 있는 간선 (u, v)는 u에서 v로만 움직일 수 있고, v에서 u로 움직일 수 없다.
  - **저자에 따라 방향 있는 간선을 <u, v>로 표현하는 경우도 있다.**
- 방향이 없으면 무방향무향
  - 방향이 없는 간선 (u, v)는, u에서 v로 움직이거나, v에서 u로도 움직일 수 있다.









- 같은 정점으로 다시 돌아오는 간선 (u, u)는 **루프**loop라고 한다.

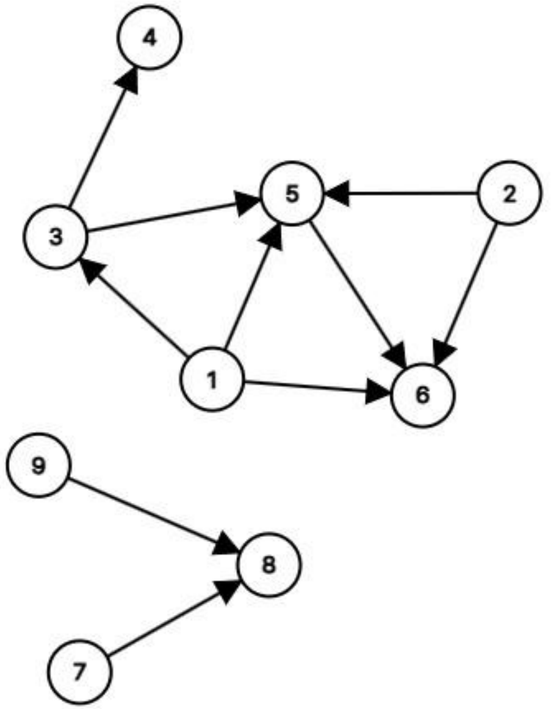
2023 Summer Algorithm Camp

용어 - 차수

- 정점의 **차수**degree는 해당 정점에 부속된 간선의 수로 정의한다.
  - 정점 v의 차수는 보통 $\deg v$ 라고 나타냄.
  - **루프**는 같은 정점의 차수에 **2**만큼 기여한다.

  - 간단한 성질handshaking lemma: $\sum_{v \in V} \deg v = 2|E|$
  - 모든 정점의 차수의 합은 **간선의 수 * 2**와 같다.

- 정점의 **진입** 차수in-degree는 정점을 **머리**로 하는 간선의 수로 정의한다.
- 정점의 **진출** 차수out-degree는 정점을 **꼬리**로 하는 간선의 수로 정의한다.



노드	1	2	3	4	5	6	7	8	9
진입 차수	0	0	1	1	3	3	0	2	0
진출 차수	3	2	2	0	1	0	1	0	1

용어 - 보행, 트레일, 경로

- **보행**walk은 정점을 연결하는 간선들을 순서대로 나열한 수열이다.
- **트레일**trail은 모든 간선이 서로 다른 보행이다.
- **경로**path는 모든 정점이 서로 다른 트레일이다.
  - 모든 정점이 다르므로, 사용되는 모든 간선도 다르다.

  - 경로가 “같은 정점”을 여러 번 포함하여도 된다고 정의하는 저자도 있다.
  - 이런 경우, “서로 다른 정점”을 방문하는 경로임을 강조하기 위해서 **단순 경로**Simple Path라는 용어를 사용하기도 한다.

- **길이**length는 사용한 간선의 수이다.
  - 가중치가 할당된 그래프에서는 간선의 가중치의 합으로 정의될 수도 있다.

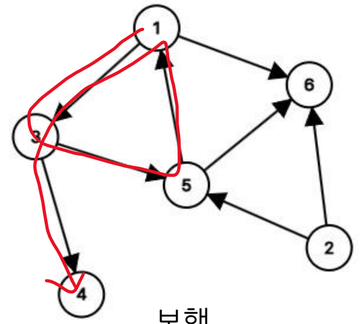
- **사이클**cycle은 처음과 끝 정점이 같으면서 비어 있지 않은 트레일이다.

- **보행, 트레일, 경로는 문제나 저자에 따라 다르게 정의하는 경우가 있어 꼭 유의해야 한다.**

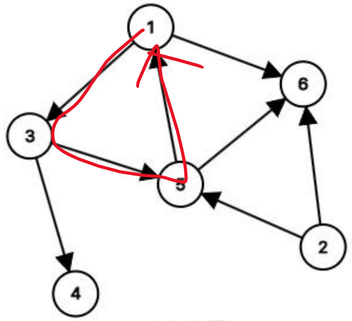
용어 - 보행, 트레일, 경로

- **보행**walk은 정점을 연결하는 간선들을 순서대로 나열한 수열이다.
- **트레일**trail은 모든 간선이 서로 다른 보행이다.
- **경로**path는 모든 정점이 서로 다른 트레일이다.

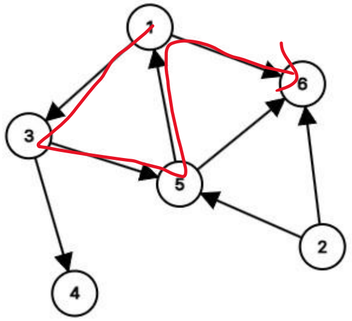
- **사이클**cycle은 처음과 끝 정점이 같으면서 비어 있지 않은 트레일이다.



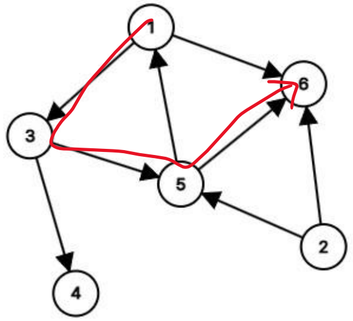
보행



사이클



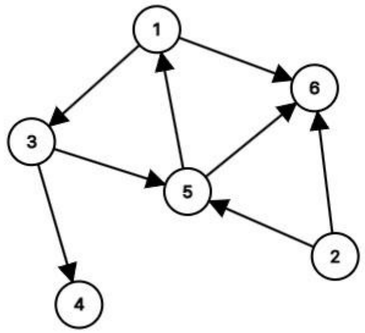
트레일

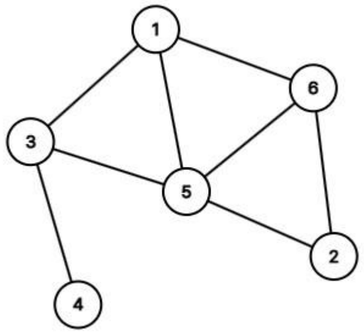


경로

용어 - 방향/무방향 그래프

- 간선에 방향이 있는지, 없는지에 따라 그래프를 부르는 이름이 다르다.
  - **방향**단방향, 유향 그래프**directed** graph
  - **무방향**무향 그래프**undirected** graph



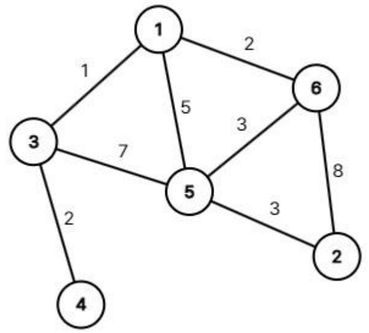


2023 Summer Algorithm Camp

용어 - 가중 그래프

- 간선에 **가중치**weight를 할당할 수 있다.
- 가중치가 있는 간선으로 이루어진 그래프를 **가중 그래프**weighted graph라고 한다.

- ex) 각 도로를 이용하는 데 걸리는 **시간**을 간선의 **가중치**로 하여 그래프를 모델링할 수 있다.
  - 어떤 지점에서 어떤 지점으로 이동하는 데 걸리는 최단 시간 구하기 (길 찾기 등..)
  - 최단 경로를 구하는 알고리즘(다익스트라)은 10회차에서 다룹니다.

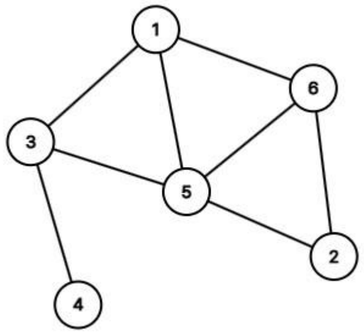


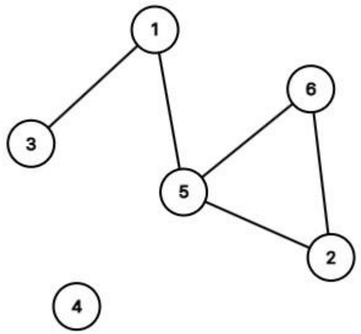
2023 Summer Algorithm Camp

용어 - 부분 그래프

- 정점 집합 V, 간선 집합 E로 구성된 그래프 G(V, E)가 있다.
- 정점과 간선 **일부를 제외**하여혹은 **일부만 포함**하여 새로운 그래프 G'(V', E')를 만들 수 있다.
  - 구체적으로 V' ⊆ V, E' ⊆ E를 만족하면 됨.
  - 이때, E'에 속한 간선의 양 끝 두 정점도 당연히 V'에 포함되어야 함.

- 이때, 그래프 G'는 G의 **부분 그래프**subgraph라고 하고, G' ⊆ G라고 표기할 수 있다.

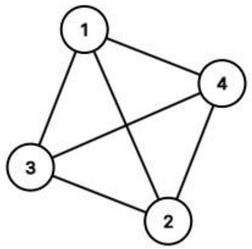




용어 - 완전 그래프

- 정점 집합 V, 간선 집합 E로 구성된 그래프 G(V, E)가 있다.
- 모든 u, v ∈ V (u ≠ v) 에 대하여 u와 v를 연결하는 간선이 존재하면 $(u, v) \in E$ **완전 그래프**라고 한다.

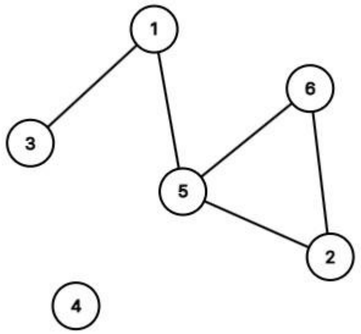
- 완전 그래프는 중복 간선이 없는 그래프에서 **간선의 수가 최대**이다.
  - 정점의 수가 n개인 완전 그래프를 K_n이라고 한다.
  - K_n의 간선 수는 **n * (n - 1) / 2**이다.

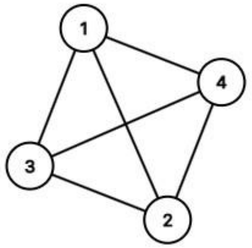


용어 - 연결 그래프

- 모든 정점이 연결된 그래프를 **연결 그래프**connected graph라고 한다.
  - 정점 u와 정점 v가 연결되어 있다는 것은 u와 v 사이의 경로가 존재한다는 것을 의미한다.

- 그렇지 않은 경우에는 disconnected graph라고 한다.





질문?

2023 Summer Algorithm Camp

트리 (Tree)

- 일반적으로 무방향 그래프를 고려함.

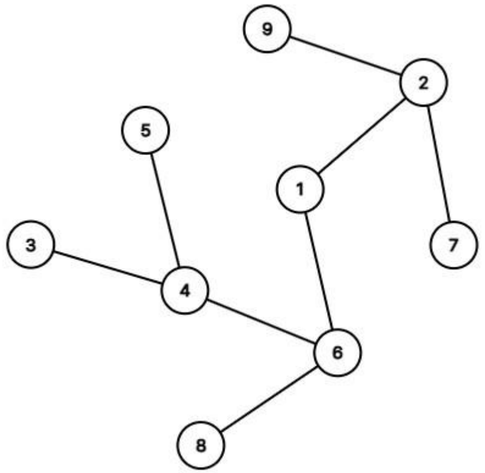
- 트리Tree는 **서로 다른 두 정점 사이의 경로가 정확히 하나**인 그래프이다. 다음은 동치인 조건들이다. (단, 정점의 수는 n)
  - 연결되어 있으면서 사이클이 없는 그래프
  - 사이클이 없는 그래프에 어떤 간선을 추가하더라도 사이클이 형성되는 그래프
  - 연결된 그래프에 어떤 간선을 제거하더라도 항상 연결이 끊기는 그래프disconnected graph

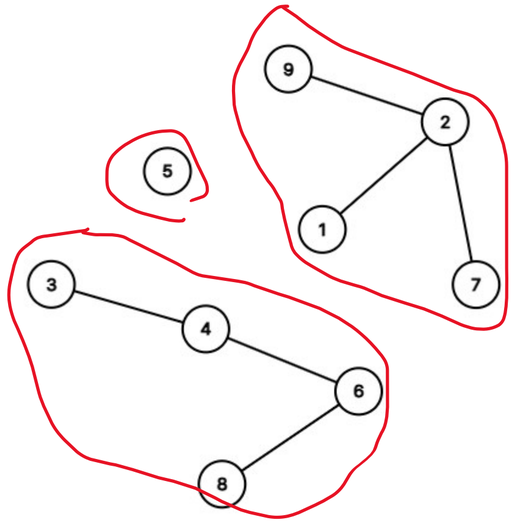
  - 연결되어 있고, n - 1개의 간선을 가지는 그래프
  - 사이클이 없으면서, n - 1개의 간선을 가지는 그래프

- 숲Forest은 **서로 다른 두 정점 사이의 경로가 최대 하나**인 그래프이다. 다음은 동치인 조건들이다.
  - 사이클이 없는 그래프
  - 그래프의 컴포넌트들 각각이 모두 트리인 그래프

트리

- 트리Tree는 **서로 다른 두 정점 사이의 경로가 정확히 하나**인 그래프이다.
- 숲Forest은 **서로 다른 두 정점 사이의 경로가 최대 하나**인 그래프이다.

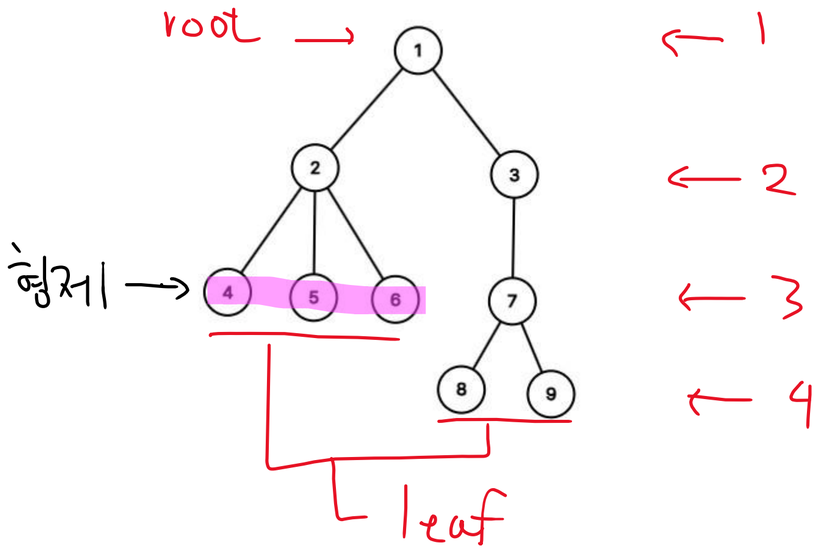




용어 - 트리

- 트리는 **계층형 관계**를 표현할 수 있다.
  - 부모 - 자식 관계
  - 루트 노드는 부모가 없는 노드
  - 루트 노드를 제외한 모든 노드의 부모의 수는 1개
  - 모든 노드의 자식의 수는 0개 이상

- **루트 노드**root node : 부모가 없는 노드, 트리에 1개만 있음
- **리프 노드**leaf node : 자식이 없는 노드
- **형제 노드**sibling node : 같은 부모를 가지는 노드
- **노드의 깊이**depth, level : 루트 노드로부터의 거리
  - 루트 노드의 깊이를 1로 정의한다면, “루트 노드로부터의 거리 + 1”
  - 0-based인지, 1-based인지에 따라서 다름
- **트리의 높이**height : 가장 깊은 노드의 깊이



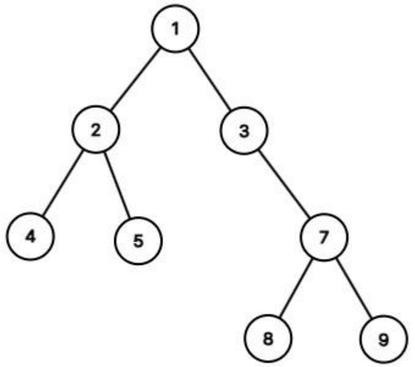
용어 - 이진 트리

- 모든 정점의 자식 수가 최대 n개 이하인 트리를 n진 트리n-ary tree라고 한다.

- n = 2이면 **이진 트리**binary tree라고 한다.
  - 자식의 수가 최대 2개이므로 왼쪽left, 오른쪽right 자식을 정의할 수 있다.

- **완전 이진 트리**Complete Binary Tree
  - 트리의 마지막 레벨을 제외한 나머지 레벨에서는 정점이 완전히 채워져 있어야 한다.
  - 트리의 마지막 레벨에 있는 노드는 왼쪽에서부터 오른쪽으로 순서대로 채워져 있어야 한다.

  - 완전 이진 트리는 여러 좋은 성질을 가지고 있다.
  - 이런 성질을 활용한 자료구조들의 STL 사용법을 살펴볼 예정이다.



이진 트리

240

질문?

그래프의 표현 방법

- 노드의 수가 n개이면 정점은 1, 2, …, n번 정점까지 있다.
  - 정점은 단순 숫자(int)로 나타낸다.
- 간선의 정보만 잘 저장하면 된다. 크게 두 가지 방법이 있다.

- 인접 행렬Adjacency Matrix : N * N 크기를 갖는 2차원 배열로 관리한다.
- 인접 리스트Adjacency List : 각 정점에 대하여 인접한 정점의 목록만 저장한다.

그래프의 표현 방법 - 인접 행렬

- $1 \leq i \leq N$, $1 \leq j \leq N$인 i, j에 대하여
  - 간선 (i, j)가 존재하면 E[i][j] = 1
  - 간선 (i, j)가 존재하지 않으면 E[i][j] = 0
- 로 정의한다.

- 장점
  - 정점 두 개가 주어지면, 두 정점을 잇는 간선이 존재하는지 O(1)에 판별할 수 있음.
  - 그래프가 빽빽dense할 때 유리함.

- 단점
  - 간선의 수와 관계 없이 항상 $O(N^2)$의 공간 복잡도가 필요함.
  - 어떤 정점 하나와 인접한 정점의 목록을 보는 데 항상 O(N)의 시간이 걸림. 모든 정점에 대해 이 작업을 반복하면 $O(N^2)$의 시간이 걸림.

2023 Summer Algorithm Camp

그래프의 표현 방법 - 인접 행렬

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n; cin >> n;
    vector<vector<int>> E(n + 1, vector<int>(n + 1));
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        E[u][v] = E[v][u] = 1;
    }
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) cout << E[i][j] << " ";
        cout << "\n";
    }
}
```

input
```
5
2 3
1 4
4 5
3 1
```

output
```
0 0 1 1 0
0 0 1 0 0
1 1 0 0 0
1 0 0 0 1
0 0 0 1 0
```

그래프의 표현 방법 - 인접 리스트

- $1 \leq i \leq N$, $1 \leq j \leq N$인 i, j에 대하여
  - E[i] = {j | 간선 (i, j)가 존재하면}
- 로 정의한다.

- 즉, 특정 정점과 인접한 정점의 목록만 저장함.
  - 각 정점마다 인접한 정점의 수가 다르므로 std::vector과 같은 동적 배열을 사용하여 구현해야 함.

- 장점
  - 간선의 수와 비례하여 자료를 저장하므로, 간선의 수가 적으면 인접 행렬보다 적은 공간이 필요함.
  - 각 정점에 대하여 인접한 정점을 보는 작업이 쉬움. 모든 정점에 대하여 이 작업을 반복하면 O(|E|)의 시간만 필요로 함.

- 단점
  - 정점 두 개가 주어졌을 때, 두 정점이 인접하는지 판별하는 것이 어려움.

2023 Summer Algorithm Camp

그래프의 표현 방법 - 인접 리스트

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    int n; cin >> n;
    vector<vector<int>> E(n + 1);
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v;
        E[u].push_back(v);
        E[v].push_back(u);
    }
    for (int i = 1; i <= n; i++) {
        cout << i << ": ";
        for (int j : E[i]) cout << j << " ";
        cout << "\n";
    }
}
```

```
input
5
2 3
1 4
4 5
3 1


output
1: 4 3
2: 3
3: 2 1
4: 1 5
5: 4
```

질문?

2023 Summer Algorithm Camp

STL - priority_queue

- STL에는 우선순위 큐priority_queue가 구현되어 있다.
  - 우선순위가 높은가장 작거나 가장 크거나 등 원소가 먼저 나오는 자료구조이다.
  - 보통은 heap이라고 불리는 자료구조로 구현한다. (관심 있는 분들은 따로 찾아보셔도 좋다. 그렇게 어려운 내용은 아니다.)

- 할 수 있는 연산
  - push(x) : x를 삽입한다.
  - pop() : 우선순위가 가장 높은 원소를 제거한다.

- 시간 복잡도
  - push : O(logN)
  - pop : O(logN)

STL - priority_queue

- 기본적으로 STL의 priority_queue는 큰 값이 먼저 나온다.
  - 보통 줄여서 pq라고 선언함.

- pq = {1, 2, 3, 4, 5}가 들어 있을 때, pop 되는 순서는
  - 5, 4, 3, 2, 1 순서

2023 Summer Algorithm Camp

STL - priority_queue

```
#include <queue>
```

std::priority_queue<T>

std::priority_queue<T, vector<T>, comp>

- comp는 비교 함수, 기본 비교 함수는 less<>
- 비교 함수 순으로 원소를 정렬했을 때 가장 큰(맨 뒤) 원소우선순위가 가장 높은 원소부터 pop됨

- void push(T x);
- void pop();
- T top();

- bool empty();
- unsigned int size();

2023 Summer Algorithm Camp

STL - priority_queue

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    priority_queue<int> pq;
    for (int i = 1; i <= 3; i++) pq.push(-i);
    for (int i = 1; i <= 3; i++) pq.push(i);
    while (!pq.empty()) cout << pq.top() << " ", pq.pop(); // 3 2 1 -1 -2 -3
}
```

2023 Summer Algorithm Camp

STL - priority_queue

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    priority_queue<int, vector<int>, greater<>> pq;
    for (int i = 1; i <= 3; i++) pq.push(-i);
    for (int i = 1; i <= 3; i++) pq.push(i);
    while (!pq.empty()) cout << pq.top() << " ", pq.pop(); // -3 -2 -1 1 2 3
}
```

질문?

STL - set, map

- 균형 이진 트리balanced binary tree, bbst로 구현된 자료구조로 std::set과 std::map이 있다.

- 원소의 삽입, 삭제, 탐색을 모두 O(logN)에 지원하는 자료구조이다.
  - set은 원소의 **중복 없이** 정렬된 순서로 저장하는 자료구조
  - map은 중복되지 않는 키에 대하여 (키, 값)의 튜플로 구성되는 자료구조이다.

STL - set

```
#include <set>
```

`std::set<T>`

`std::set<T, comp>`

- comp는 비교 함수
- range-based loop은 비교 함수 순으로 순회함

- `insert(T a);`
- 삭제
  - `erase(T a); // 값으로 삭제`
  - `erase(std::set<T>::iterator pos); // 위치(iterator)로 삭제`
- 탐색
  - `std::set<T>::iterator lower_bound(T a); // a 이상인 원소의 위치`
  - `std::set<T>::iterator upper_bound(T a); // a 초과인 원소의 위치`

2023 Summer Algorithm Camp

STL - set

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    set<int> S;
    S.insert(1); S.insert(5); S.insert(7); S.insert(7); // 중복 원소 7
    for (int i : S) cout << i << " "; cout << "\n"; // 1 5 7
    cout << *S.lower_bound(2) << " " << *S.upper_bound(5) << "\n"; // 5 7
    S.erase(5);
    for (int i : S) cout << i << " "; cout << "\n"; // 1 7
    auto it = S.lower_bound(0);
    cout << *next(it) << " " << distance(S.begin(), next(it)); // 7 1
}
```

240

STL - map

```
#include <map>
```

`std::map<K, V>`

`std::map<K, V, comp>`

- K에 대한 비교 함수(기본은 less)만 정의되어 있으면 어떤 자료형이든 가능함
- comp는 비교 함수, K에 대한 비교 함수 기준으로 정렬됨
- range-based loop은 비교 함수 순으로 순회함, structure binding을 사용할 수 있음

- 삽입
  - `mp[K] = V; // 배열처럼 접근`
  - `mp.insert({K k, V v}); // 이미 같은 키가 있으면 덮어씌움`
- 삭제
  - `erase(T k); // 키로 삭제`
  - `erase(std::map<K, V>::iterator pos); // 위치(iterator)로 삭제`

STL - map

- 탐색
  - mp[K]; // 배열처럼 접근, 키가 없으면 (K, 0)을 map에 넣고 0을 반환함.
  - mp.count(K); // K가 map에 있으면 1, 없으면 0을 반환함.
  - std::map<K, V>::iterator lower_bound(T k); // 키가 k 이상인 원소의 위치
  - std::map<K, V>::iterator upper_bound(T k); // 키가 k 초과인 원소의 위치

- 순회
  - for (const auto &kv : mp) // kv.first, kv.second
  - for (const auto &[k, v] : mp)

2023 Summer Algorithm Camp

STL - map

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    map<int, int> mp;
    mp[0] = 1;
    mp.insert({8, 2});
    for (const auto &[k, v] : mp) cout << k << " " << v << ", "; // 0 1, 8 2,
    cout << "\n" << mp.size() << " " << mp.count(1) << "\n"; // 2 0
    if (!mp[1]) cout << mp.size() << " " << mp.count(1) << "\n"; // 3 1
    mp.erase(8);
    for (const auto &[k, v] : mp) cout << k << " " << v << ", "; // 0 1, 1 0,
}
```

240

질문?

문제

- **필수 문제**
- BOJ 1927 (최소 힙)
- BOJ 11279 (최대 힙)
- BOJ 1822 (차집합)
- BOJ 25192 (인사성 밝은 곰곰이)
- BOJ 4358 (생태학)

- **연습/심화 문제**
- BOJ 11286 (절댓값 힙)
- BOJ 23057 (도전 숫자왕)
- BOJ 17430 (가로등)
- BOJ 1655 (가운데를 말해요)