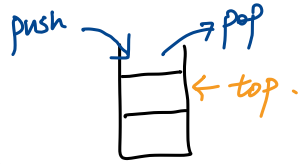


# 1. Data Structure.

## 1.1. Stack.




- LIFO (Last In First Out)
- $\text{pop}()$  : 스택에서 가장 위에 있는 원소 반환 및 제거.
- $\text{push}(\text{item})$  : 스택의 맨 위에  $\text{item}$ 을 삽입.

ex) 재귀 알고리즘, undo, 역순 문자열 만들기, 수식의 괄호 검사.

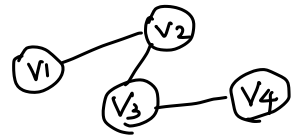
ex - VPS, Valid Parenthesis String

## 1.2 Queue

- FIFO (First In First Out)
- 
- $\text{pop}()$  : 큐의 맨 앞 (처음) 원소 반환 및 제거
  - $\text{push}(\text{item})$  : 큐의 맨 뒤에  $\text{item}$ 을 삽입.

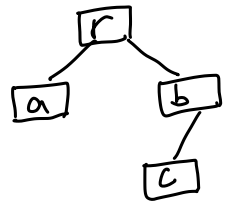
ex) BFS, Cache 구현, 프로세스 관리

## 1.3 Graph



- 정점과 간선으로 구성됨. (각개체간의 관계를 표현할 수 있는 자료구조).
- 그래프는 네트워크 모델이다.
- 방향 / 무방향, 순환 (Cyclic) / 비순환 (Acyclic)

## 1.4. Tree



· 노드 (정점) 들로 이루어진 자료구조. (그래프의 한 종류)

① 1개 이상의 루트 노드

② 루트노드는 0개 이상의 자식노드, 자식노드도 동일.

· Cycle이 없는 하나의 연결그래프 (Connected Graph)

No loop  
No circuit.

또는 DAG (Directed Acyclic Graph) 의 한 종류.  
방향성이 있다!

· 트리는 계층구분이다. (최소 연결 트리).

· 노드가  $N$ 개인 트리는 항상  $(N-1)$ 개의 간선을 가진다.

· 순회는 Pre-, In-, Post-order.

ex) 이진트리, 이진 탐색 트리, 균형트리 (AVL트리, Red-Black 트리),  
이진 힙 (최대힙, 최소힙) 등.

## 1.5 Heap

Priority Queue: 우선순위가 높은 순으로 삭제.

· '완전 이진 트리'의 일종으로 우선순위 큐를 위하여 만들어진 자료구조.

→ Heap으로 구현 (삽입/삭제:  $O(\log N)$ )

· 여러개의 값들 중에서 최댓값/최소값을 빠르게 찾는데 활용됨.

· 부모노드의 키 값  $\geq$  자식노드의 키 값. (또는 반대)

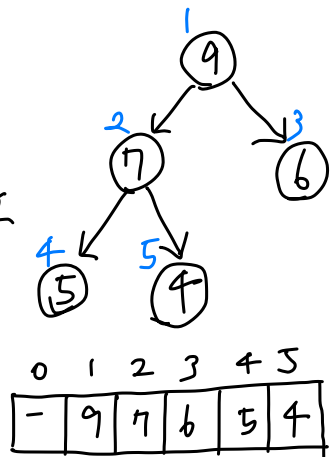
· 중복된 값 허용 (BST에서는 허용하지 않음)

- Heap에서 부모 - 자식 노드의 관계.

- 왼쪽 자식의 인덱스 = 부모의 인덱스  $\times 2$

- 오른쪽 자식의 인덱스 = (부모의 인덱스  $\times 2$ ) + 1

- 부모의 인덱스 = 자식의 인덱스 / 2.



## 1.6. Hash Table.

- Key-Value 형태의 자료구조.

- C++ STL 기준 `map<key, value>`

- `insert (make_pair (key, value))` - 삽입

- `erase (key)` / `clear()` - 삭제

- `find (key)` - iterator 반환 / `count(key)` - 개수 반환 - 범위

## 2. Algorithm.

### 2.1. 기본 정렬 알고리즘.

Sorting	Worst	Avg	Best	장점	단점
Bubble	$N^2$	$N^2$	$N^2$	단순함	비효율적.
Selection	$N^2$	$N^2$	$N^2$	비교횟수 > 교환횟수	비효율적
Insertion	$N^2$	$N^2$	$N$	Best $O(N)$	Worst $O(N^2)$
Quick	$N^2$	$N \log N$	$N \log N$	가장 많이 위한 분할, 배열	Pivot에 따라 차이남.
Merge.	$N \log N$	$N \log N$	$N \log N$	가장 많이 위한 분할, 배열	'추가적인 메모리 사용'
Heap	$N \log N$	$N \log N$	$N \log N$	항상 $O(N \log N)$	퀵보다 느림 / Not Stable

- Bubble : 인접한 값을 비교 및 교환.
- Selection : 오름차순일 경우, 앞에서부터 가장 작은 값을 찾아서 교환 및 반복.
- Insertion : 정렬되어 있는 경우  $O(N)$ , 현재 위치에서 그 이하 배열들을 비교하여 자신이 들어갈 위치를 찾아 해당 위치에 삽입.
- Quick : pivot를 기준으로 작은 값들은 왼쪽, 큰 값들은 오른쪽에 옮기는 분할과 동시에 정렬을 함. 배열 크기  $N \times$  분할 깊이  $\log N$ .
- Merge : 분할정복, 배열의 크기가 1보다 작거나 같을 때까지 분할 진행.  
합병과정  $O(N) \times$  분할과정  $O(\log N)$

## 2.2 MST (Minimum Spanning Tree)

그래프의 모든 정점을 연결하는 트리, 최소 연결된 그래프.

! ST 중에서도 사용된 간선들의 가중치의 합이 최소인 트리

· 간선의 가중치의 합이 최소 /  $N-1$ 개의 간선 / No Cycle.

### ① Kruskal MST 지역적인 최적해 가짐.

· Greedy Method을 이용하여 네트워크 (가중치를 간선에 할당한 그래프)의 모든 정점을 최소 비용으로 연결하는 최적 해법을 구하는 것.

① 그래프의 간선들을 가중치의 오름차순으로 정렬

② 정렬된 간선 리스트에서 순서대로 사이클을 형성하지 않은 간선을 선택함.

· 가장 낮은 가중치부터 선택

· 사이클을 형성하는 간선 제외.

'Union-Find' 알고리즘을 통해

사이클 형성 여부 판단.

③ 해당 간선을 현재의 MST의 집합에 추가함.

간선의 수

· 시간복잡도 :  $O(E \log E)$  → 간선이 적은 Sparse Graph에 적합.

### ② Prim MST

· 시작 정점에서부터 출발하여 ST 집합을 단계적으로 확장해나가는 방법.

① 시작 단계에서는 시작 정점만이 MST 집합에 포함됨.

② 앞 단계에서 만들어진 MST 집합에 인접한 정점들 중에서 최소 간선으로 연결된 정점을 선택하여 트리를 확장함.

㉔ 위의 과정을 트리가  $N$ -개의 간선을 가질 때까지 반복함.

시간복잡도 :  $O(N^2) \rightarrow$  간선이 많은 Dense Graph에 적함.

3. Operating System.