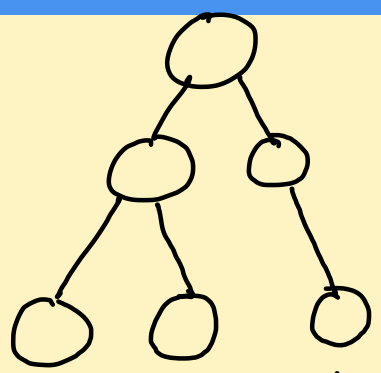


binary tree leaf ⇒ 자식이 없는 트리

balanced tree

Complete binary tree ⇒ 모든 노드들이 레벨 별로 왼쪽부터 채워져 있는 경우

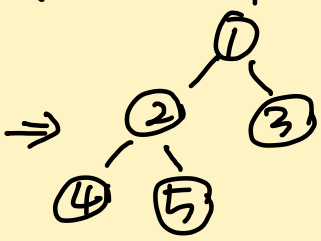
Full binary tree ⇒ 자식노드를 가지려면 2개를 가지던지 아예 가지지 않은 경우

Perfect binary tree ⇒ 모든 노드가 2개의 자식을 가지고 레벨도 정확하게 떨어지는 경우 ⇒ 피라미드 구조

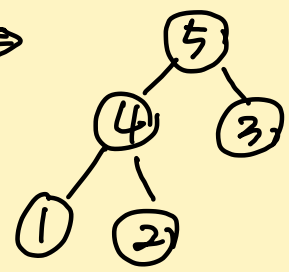
↳ $2^n - 1$

heap ~> 최댓값이나 최솟값을 찾아내기 위해 고안된 완전이진트리

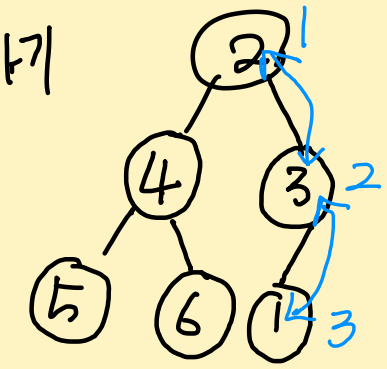
Min heap ⇒



Max heap ⇒

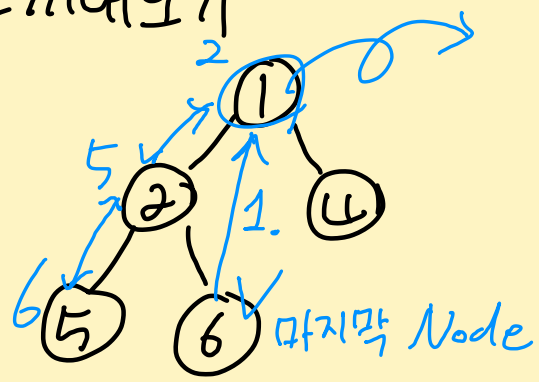


최소힙에 노드 삽입하기

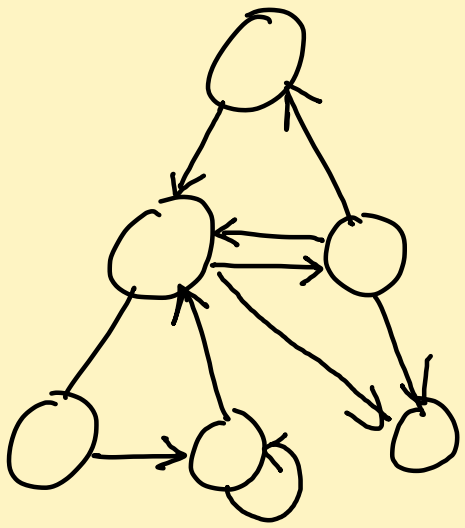


O(logn)의 시간 복잡도

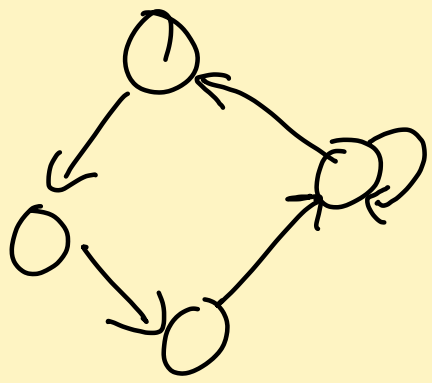
최소힙에서 노드꺼내오기



O(logn)의 시간 복잡도

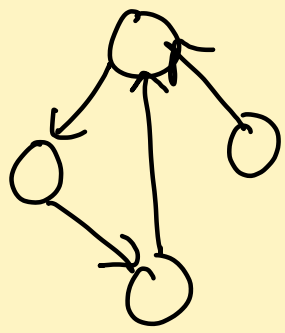


tree는 graph의 한 형태

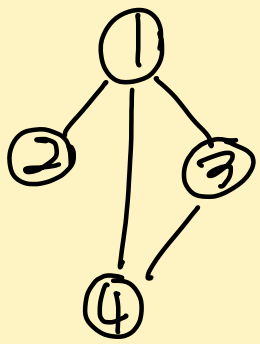


Self edge

Directed graph

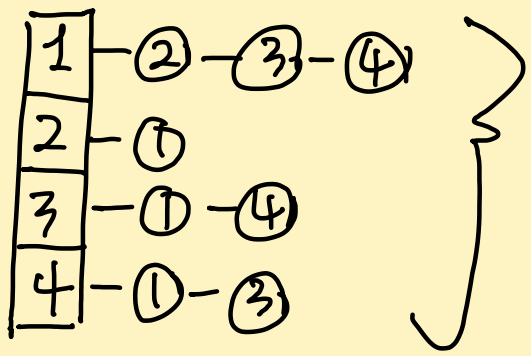


Adjacency matrix ⇒ 이차원 배열에 표현하는 방법



	1	2	3	4
1	0	1	1	1
2	1	0	0	0
3	1	0	0	1
4	1	0	1	0

Adjacency list



edge의 갯수를 m개라고 할 때

Node들의 갯수는 $2m$개가 된다.

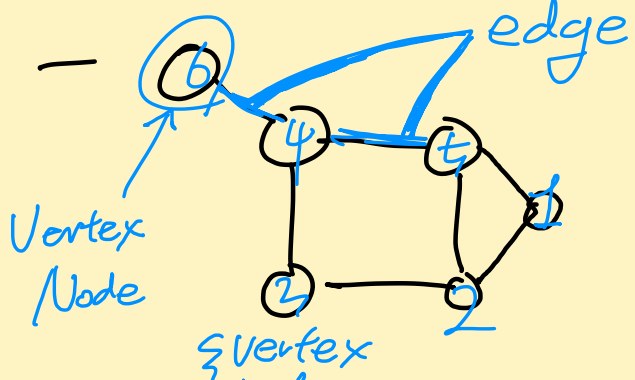
Graph는 수학적 개념이 사용됨.

자료구조

- 가장 복잡한 일반적인 자료구조
- $G=(V, E)$

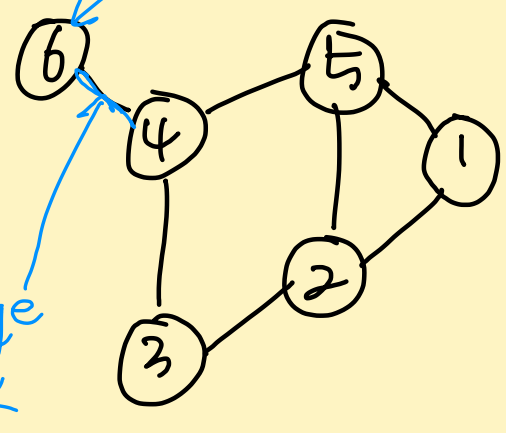
Vertex Set

Edge

- 

$V=\{1,2,3,4,5,6\}$

$E=\{(1,2),(2,5),(1,5),\cdots,(6,4)\}$



정점 degree : 4의 정점수 = 3 1의 정점수 = 2

G의 정점수 = 3

인접성

노드 4와 노드 5는 인접하다.

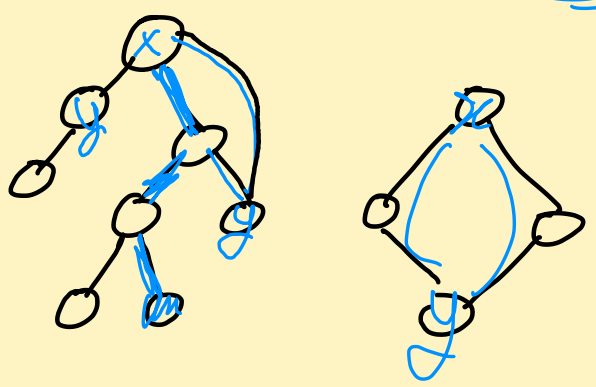
경로(path) 3 → 2 → 5

3 → 2 → 1 → 5

(3 → 2 → 1 → 2 → 5) X

사이클(cycle) 3 → 2 → 5 → 4 → 3

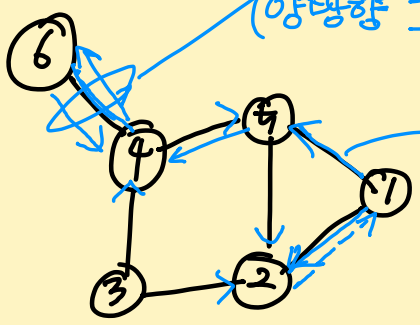
Quiz: cycle 없는 그래프 : 트리



graph 표현법 ⇒ 인접성 ↯ 1. 인접 행렬(adjacency Matrix) → 대각선으로 대칭
2. 인접 리스트(adjacency list) ⇝ 인접리스트
⇝ 연결 리스트

undirected graph (무방향 그래프)

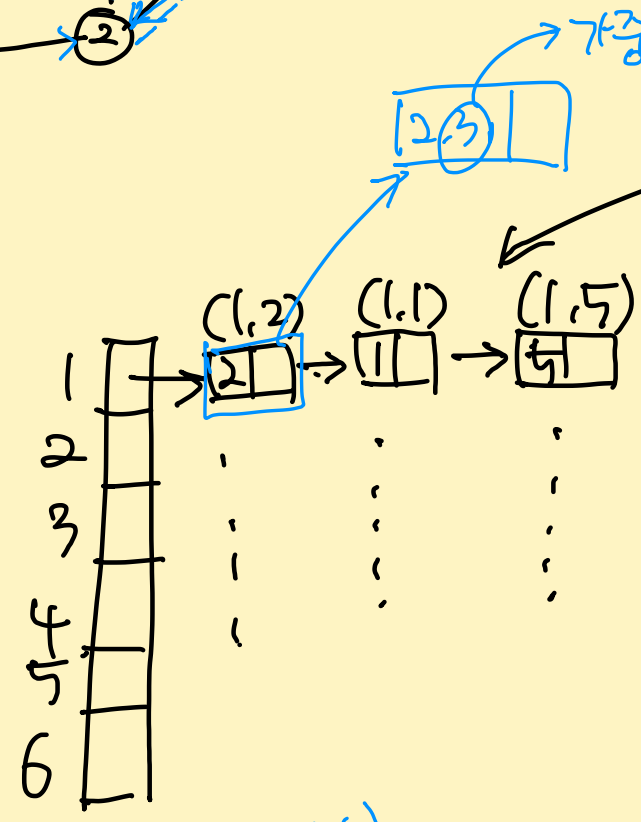
(양방향 그래프)



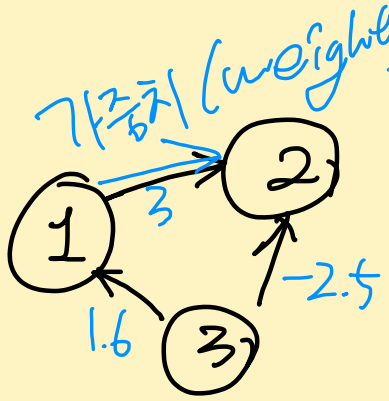
directed graph (방향 그래프)

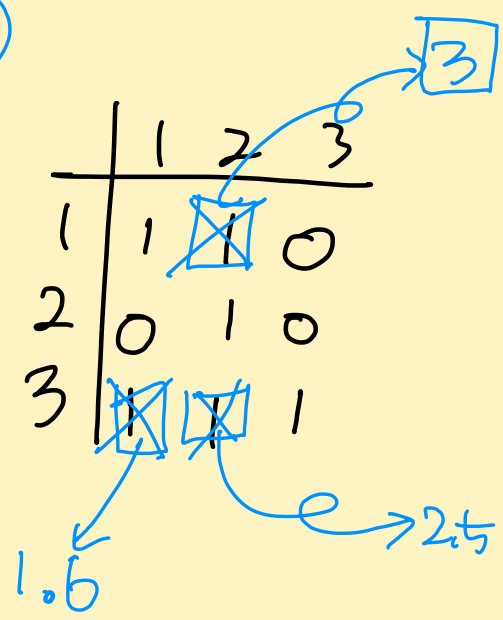
n^2 entry

가중치

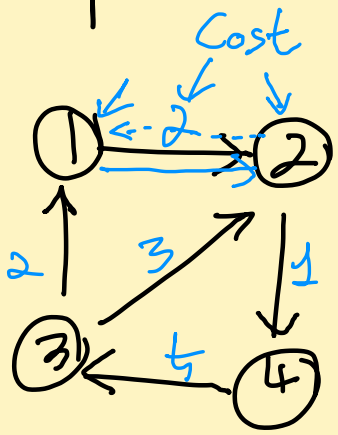


가중치 (weight)





Graph 자료구조와 기본연산



$G=(V,E)$

정점 노드 집합 — 에지 집합

$|V|=n, |E|=m$

1. 인접 행렬

$$G=\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$n \times n$

① memory: $O(n^2)$

② $(u,v) \in E$?: $G[u][v]==1$ (>0) $\Rightarrow O(1)$

③ u에 인접한 모든 노드 v에 대해

```
for v in range(1, n+1):
    do with G[u][v]
```

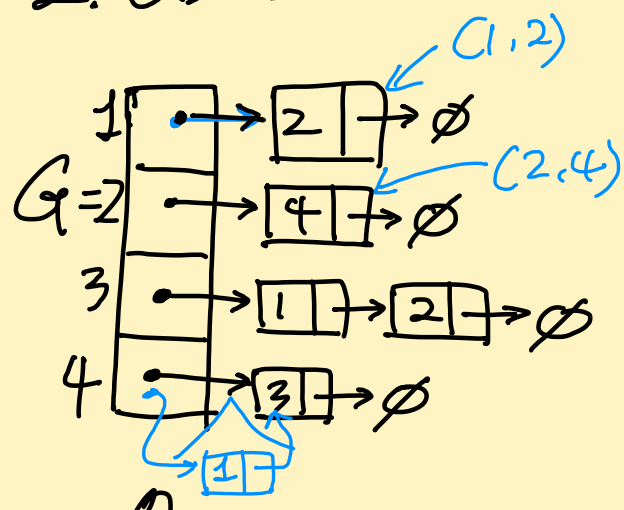
$O(n)$ — 인접한 노드가 많든지 적든지 동일

④ 새엣지 (u,v): $G[u][v]=1$ 삽입 $O(1)$

⑤ (u,v) 삭제: $G[u][v]=0$ $O(1)$

메모리 $O(n^2)$

2. 인접리스트



$O(n+m)$

G[u].search(v) — 인접리스트 $O(n)$ → 노드의 갯수만큼

for each edge in G[u]: — O(인접한 노드수), 인접

G[u].pushFront(v) — $O(1)$ — appendd(v)

x = G[u].search(v)

G[u].remove(x) — 삭제할 노드 $O(n)$

메모리 $O(n+m)$ → 엣지가 적으면 유리

Sparse ⇒ 엣지가 적음 ∴ 인접리스트 방식이 유리: 메모리 측면

dense ⇒ 엣지가 상당히 많음: 인접행렬과 메모리 차이가 크지 않음.