Ch 10 그래프 (Graph)

$G=(V, E)$ V: set q vertices (정점)
E: set q edges (간선)

1. 용어들

$G'=(V', E')$ is a subgraph (부분 그래프)

$\Leftrightarrow V' \subseteq V \wedge E' \subseteq E$

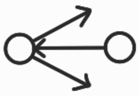
undirected (무향) 그래프

directed (유향, 방향) 그래프 digraph

weighted (가중치) 그래프

b는 a의 인접정점 (adjacant vertex)

deg(v) = 정점 v의 degree(차수) = v의 인접 정점수


in-degree
out-degree

$p=\langle V_0, V_1, V_2, \dots, V_k \rangle$: path (경로) p from V_0 to V_k

$V_0 \overset{p}{\rightsquigarrow} V_k \Leftrightarrow (V_{i-1}, V_i) \in E,\ i = 1 \sim k$

: cycle (순환) if $V_0 = V_k$

: simple path (단순경로)

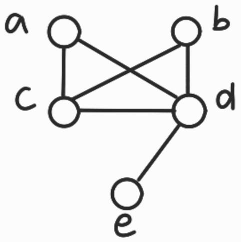
= cycle이 없는 경로 (=acyclic)

2. Adjacency matrix (인접 행렬) ← 그래프의 표현 방법

$G=(V, E)$ $|V|=n$

$A_G = [a_{ij}]$ $n \times n$

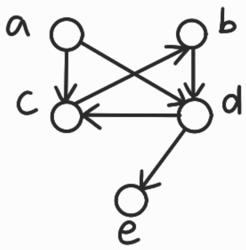
$$a_{ij} = \begin{cases} 1 & \text{if } (V_i, V_j) \in E \\ 0 & \text{if } (V_i, V_j) \notin E \end{cases}$$



→

A	a	b	c	d	e
a	0	0	1	1	0
b	0	0	1	1	0
c	1	1	0	1	0
d	1	1	1	0	1
e	0	0	0	1	0

symmetric



→

A	a	b	c	d	e
a	0	0	1	1	0
b	0	0	0	1	0
c	0	1	0	0	0
d	1	1	1	0	1
e	0	0	0	1	0

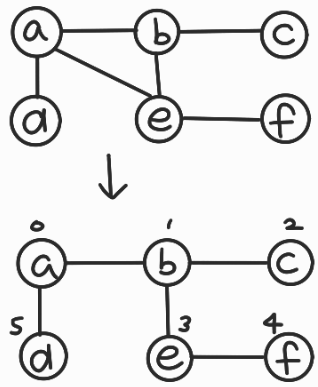
not symm.

3. 그래프 탐색 (Graph Traversal)   중복X 누락X

① 깊이 우선 탐색 (Depth-First Search : DFS)

: 연결을 따라서 가다가, 막히면 마지막 갈림길로 되돌아와 계속한다.
미로 찾기와 동일 → Back Tracking → stack 사용

```
DFS (v)
{   visit v (& mark v as visited)
    for (each u adj to v)
    {     if (u is not visited)
               DFS (u)
    }
}
```

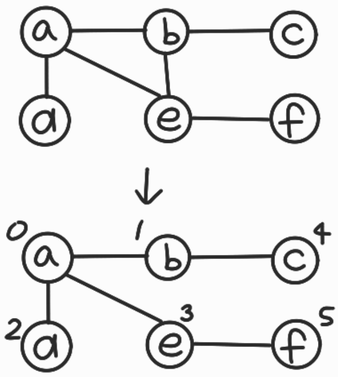


DFS 탐색트리

② 너비우선탐색 (Breath-First Search)

: 각 정점의 인접정점들을 차례로 방문 → Queue
가까운 거리부터

```
BFS (v)
{   visit V (& mark V as visited)
    Q에 V의 모든 미방문 인접 정점을 추가
    if (Q = ∅) end
    u = removed from Q
    BFS (u)
}
```



BFS 검색트리

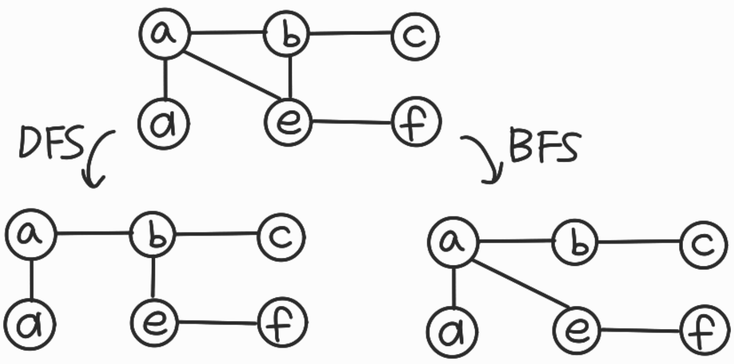
4. 신장 트리 (Spanning Tree)

$G = (V, E)$: connected

$T = (V, E')$: G의 신장 트리 (spanning tree)

⇔ $E' \subseteq E$인 트리 ← ① connected (연결) ② acyclic (비순환)

ex) G를 BFS나 DFS로 탐색한 결과도 G의 신장트리이다.



5. 최소 신장트리 (Minimum Spanning Tree : MST)

$G = (V, E)$: connected, weighted graph

$|V| = n$, $|E| = m$ $\quad n-1 \le m \le \frac{n(n-1)}{2}$

$M = (V, E')$: G의 MST

⇔ G의 신장 트리 중 edge weight 합이 최소가 되는 신장트리

① Kruskal's algorithm — 선택할 때마다 그 순간 가장 좋다고 생각되는 것 선택

간선 기반

: greedy algorithm 으로서, cycle을 만들지 않는 최소 weight의 edge들을 차례로 추가하여 MST 구성

(greedy choice)

* Union-Find 자료구조 ← cycle 확인용 자료구조

1. Sort E by weight
2. $E' = \emptyset$
3. while($|E'| < n-1$)

{ 1. Remove the smallest $e \in E$

2. if (e does not make any cycle in $(V, E'+e)$)

$E' \leftarrow E'+e$

}

⇒ (V, E') is MST ← $O(m \lg m)$

② Prim's algorithm

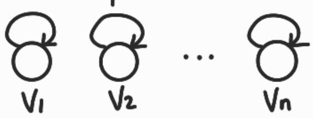
정점 기반 : 또 하나의 greedy algo 으로서, 정점을 추가할 때 최소 weight의 edge를 선택한다.

1. $V' = \{a\} \subseteq V$
   $E' = \emptyset$
2. while $(|E'| < n-1)$
   { $C = \{(a,b) \mid a \in V' \wedge b \in V - V'\}$
   Take $e = (a,b)$ ← smallest edge in C
   $V' = V' + b$
   $E' = E' + e$
   }

⇒ (V, E') is MST ← $O(n^2)$

㊉ G가 sparse → kruskal 유리 $(m \lg m)$
행렬의 값이 대부분 0, 간선↓
dense → Prim 유리 (n^2)
행렬의 값이 대부분 1, 간선↑

㊉ Union-Find data structure (Disjoint set)
given n seperated (disjoint) set

1. 초기화:  : self-rooted n trees

2. Find(x) // ret the root of x & path comparison (경로 압축)
원소 x가 속해 있는 집합의 root 노드 반환

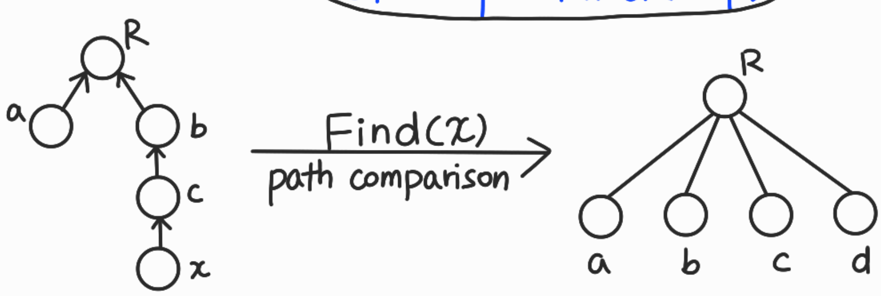
```
{ start = x
  while (x ≠ x.p) x = x.p      // going-up to the root
  Root = x, x = start
  while (x ≠ Root) { y = x.p; x.p = Root, x = y }
  ret Root                     // path compression
}
```

$O(\lg N)$

||

```
if (x.p = x) ret x
ret x.p = Find(x.p)
```

재귀적



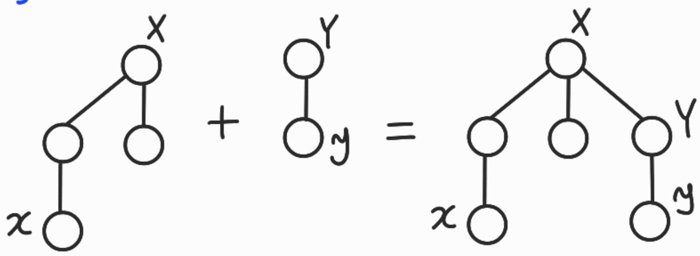
③ Union(x, y)   원소 x와 y가 속해있는 집합을 입력으로 받아 2개 집합의 합집합을 만든다

```
{ X = Find(x), Y = Find(y)
  if (X ≠ Y) Y.P = X      // 단 작은 쪽을 큰 쪽에 연결
}
```

$O(\lg N)$

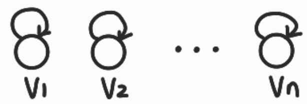
노드 번호	①	②
초기 : 부모 노드 번호	1	2
Union 후 : "	1	1

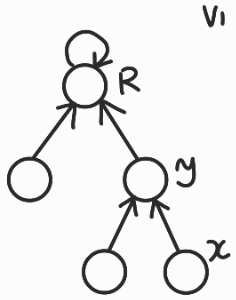
(더 작은 쪽으로 합침)



④ 구현: int parent[n], height[n]

초기화: parent[i] = i, height[i] = 0 (or 1)
$i = 1 \sim n$





parent[R] = R : root
parent[y] = R
parent[x] = y

⑤ 시간 분석

$$\text{Find}() = \begin{cases} O(\lg n) & \text{no compression} \\ O(\alpha(n)) & \text{with} \ '' \end{cases}$$

Ackerman 함수 : $\underbrace{\alpha(2^{65536}) = 5}_{\text{거의 상수}}$

Union()

6. 최단 경로 (shortest path : SP)

$G = (V, E)$: weight, $w_i > 0$

$w(a, b)$ = weight of edge (a, b)

표기

$p = \langle v_0, v_1, \cdots, v_k \rangle$: path p from v_0 to v_k : $v_0 \overset{p}{\leadsto} v_k$

$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$ = weight of p

$u \overset{sp}{\leadsto} v$: the shortest path from u to v is a path $u \overset{p}{\leadsto} v$ with minimum $w(p)$

SP problems:

- SPSP (Single Pair SP)
- SSSP (Single Source SP) - Dijkstra
- SDSP (Single Destination SP)
- APSP (All Pairs SP) - Floyd

출발 정점을 하나 정해놓고 이로부터 다른 모든 정점까지의 최단거리를 구하는 방식

그래프에 존재하는 모든 정점 사이의 최단경로 구하기

필요한 자료구조:

$G = (V, E)$ $\quad |V| = m \quad V = \{V_0, V_1, \dots, V_{n-1}\}$

1) $W[n][n]$: weight matrix of G

$$W_{ij} = \begin{cases} W(V_i, V_j) & \text{if } (V_i, V_j) \in E \\ \infty & \text{if } (V_i, V_j) \notin E \\ 0 & i = j \end{cases}$$

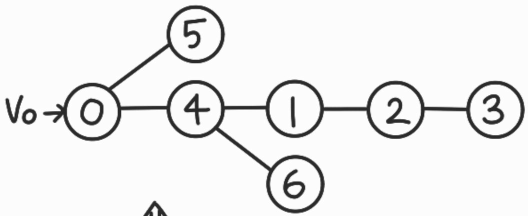
2) $D[n]$: 출발점 V_0로부터 각 정점까지의 최단거리 (distance)

$= V_0$-th row of $W[V_0][\,]$: 초기치

$(D[j] = W[V_0][j],\ j = 0 \sim n-1)$

3) $path[n]$: 출발점 V_0가 root인 최단경로트리 (SP tree)로서 각 정점의 부모 node를 저장

ex)



$\Updownarrow$

path =

0	1	2	3	4	5	6
0	4	1	2	0	0	4

① Dijkstra SSSP : $O(n^2)$

: single source V_0로부터 각 정점까지의 최단거리를 구한다 : $D[n]$

실제 최단경로는 V_0를 root로 하는 최단경로트리 (SP tree)를 구성하여 해결한다.

$G = (V, E) \quad |V| = n \quad V = \{V_0, V_1, \cdots, V_{n-1}\}$

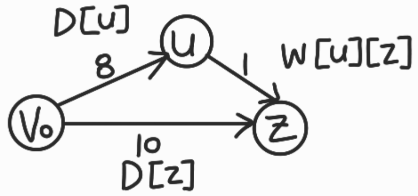
$W[n][n]$, $D[n]$ $path[n]$

algorithm

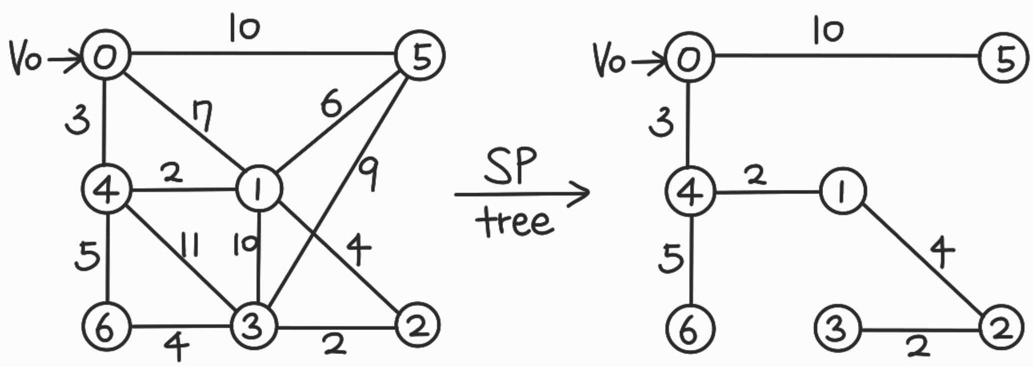
1. $S = \{V_0\}$ // single source 출발 노드 설정
2. $D[n] = V_0$-th row of W 출발 노드를 기준으로 각 노드의 최소 비용 저장
3. while ($|S| < n$)

$O(n^2)$

```
{   u = min D[u]   // D[u]가 최소가 u∉S : O(n)
    S = S + u
    for (each z∈S, adj to u)
    { D[z] = min(D[z], D[u] + W[u][z]) }
}
```



ex)



방문하지 않은 노드 중에서 가장 비용이 적은 노드

D	~~0~~ (V_0)	~~1~~	~~2~~	3	~~4~~	5	~~6~~	S
초기	0	7	∞	∞	3	10	∞	{0} ← 4 : 1, 3, 6
+4		5		14			8	{0, 4} ← 1 : 2, 3, 5
+1			9	〃		〃		{0, 4, 1} ← 6 : 3
+6				12				{0, 4, 1, 6} ← 2 : 3
+2				11				{0, 4, 1, 6, 2} ← 5 : 3
+5				〃				{0, 4, 1, 6, 2, 5} ← 3
final	0	5	9	11	3	10	8	{0, 4, 1, 6, 2, 5, 3}

* SP tree from D[n]

: single source V_0에 대한 각 정점으로의 최단 경로(SP)는 V_0-root 최단경로트리(SP tree)를 D로부터 구성 후 처리

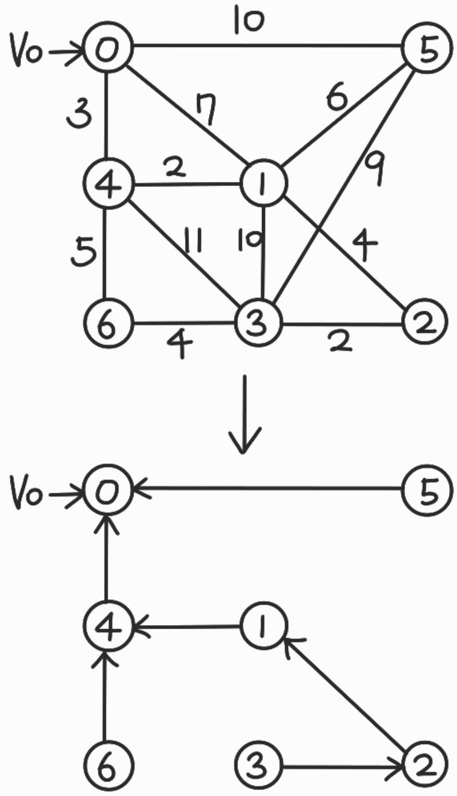
index	0	1	2	3	4	5	6
D	0	5	9	11	3	10	8

↓ sort index by D

	0	1	2	3	4	5	6
index	0	4	1	6	2	5	3
D	0	3	5	8	9	10	11

↓ SP tree

	0	1	2	3	4	5	6
path	-	4	1	2	0	0	4



SP tree(backward)

```
for (j = n-1 ~ 0)
{   J = index[j]   // 부모를 찾는 J
    for (i = j-1 ~ 0)
    {   I = index[i]   // J의 부모 후보 I
        if (D[J] - D[I] ≥ W[I][J])
            path[J] = I
    }
}
```

SP from V_0 to k (backward)

```
while (1)
{   print k
    if (k = V0) break
    k = path[k]
}
```

② Floyd APSP : $O(n^3)$

: Dijkstra를 n번 반복하는 것보다 훨씬 간단

$G = (V, E) \quad |V| = n, \; V = \{V_0, V_1, \cdots, V_{n-1}\}$

$W[n][n]$: weight matrix of G

$A[n][n]$: distance matrix

$A[i]$: i-th row of A

= shortest distance from V_i

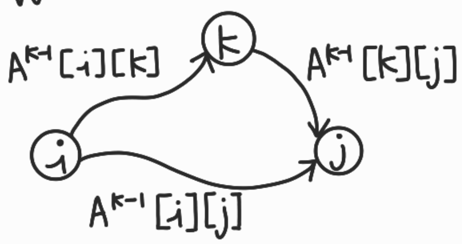
= $D[n]$ for single source V_i

Def)

$\boxed{A^k[i][j]}$ = i에서 출발하여, 0~k의 중간 점들만을 거쳐서 j로 가는 최단 경로의 거리

$A^{n-1}[i][j]$ = 정점제한 없이 $i \overset{SP}{\rightsquigarrow} j$ 의 거리 (final)

$A^{-1} = W$



$A^k[i][j] = \min(A^{k-1}[i][j], \; A^{k-1}[i][k] + A^{k-1}[k][j])$