제 11 강의 . 그래프의 응용

학습 목차

1. 스패닝 트리
2. 최소 스패닝 트리
3. 최단경로 문제
4. 이행성 폐포 문제

학습 가이드

그래프에 관한 문제를 그래프 자료구조를 이용하여 해결하여 본다. 다음은 그래프 자료구조를 이용하여 해결할 수 있는 문제들이다.

첫째로 경기도의 10개도시의 송유관을 건설한다고 할 때 어떻게 하면 중복시키지 않고(사이클이 없이) 모든 도시를 공급지와 연결을 시킬 것인가 하는 문제는 그래프에서 **스패닝 트리** 문제가 된다.

둘째 도로 건설 문제에서 도로를 연결하되 최소비용의 도로 건설 비용으로 전체 도시를 연결할 것인지(사이클이 없이) 하는 문제는 **최소 스패닝 트리** 문제가 된다.

세째 서울시의 지하철에서 종로1가에서 강남역까지 가는 방법 중 지하철을 바꿔 타더라도 가장 빨리(시간), 가장 짧은 거리, 혹은 지하철 역 수를 적게 거쳐서 가는 방법은 **최단 경로 문제가** 된다.

넷째 영업 사원이 승용차로 지방의 10개 도시를 한번 씩 만 거쳐서 다시 서울로 돌아오되 기름 값이 가장 적게 드는 방법은 **세일즈맨 문제**가 된다.

이러한 문제들을 해결하는 방법들은 오랜 동안 효율성을 중점에 두고 개발이 되어왔다. 앞의 3가지 방법에 대하여 배워보도록 하자.

1. 스패닝 트리

스패닝 트리 : 스패닝트리(spanning tree)는 “신장 트리”라고도 한다..

(정의) 그래프 G의 스패닝트리 G’는 그래프 G의 부속 그래프로 다음을 만족한다.

- V(G’) = V(G)
- G’ is connected
- |E(G’)| = n – 1

(해설) 스패닝 트리는 원래 그래프에서 정점은 그대로이고 간선은 노드 수보다 1개 적으며 원래 그래프를 연결된 상태로 만드는 그래프를 말한다. 간선의 수가 정 점의 수보다 1개 적으므로 그래프에 사이클이 없게 된다.

다음과 같은 응용에 사용된다.

첫째로 경기도의 10개도시의 송유관을 건설한다고 할 때 어떻게 하면 중복시키지 않고(사이클이 없이) 9개의 송유관으로 모든 도시를 원유 공급지와 연결을 시킬 것인가 하는 문제는 그래프에서 스패닝 트리 문제가 된다.

둘째는 건물간의 컴퓨터 네트웍을 연결할 때 싸이클이 없이 전체 건물이 연결되도록 통신 선을 구성할 것인가가 스패닝 트리 문제가 된다.

스패닝 트리(spanning trees) 구하기

스패닝 트리를 구하려면 노드(정점)의 수가 n인 그래프에서 그래프의 모든 부분이 연결되도록 n-1개의 에지(간선)을 선택하면 된다. 이것을 그래프가 규모가 작고 그림으로 그려져 있을 때는 쉽게 택할 수 있지만 그래프가 크고 그래프가 인접 행열이나 인접리스트로 표현되어 있을 때는 프로그램으로 에지를 선택하여야 한다. 깊이우선탐색(dfs)이나 너비우선탐색(bfs) 프로그램을 이용하면 쉽게 해결된다. 이 두 그래프 탐색 알고리즘은 그래프를 탐색하면서 에지를 따라 방문하게 되고 방문하게 되는 에지는 다음의 두 가지 에지로 구분된다.

- 그래프 G가 연결되어 있을 때

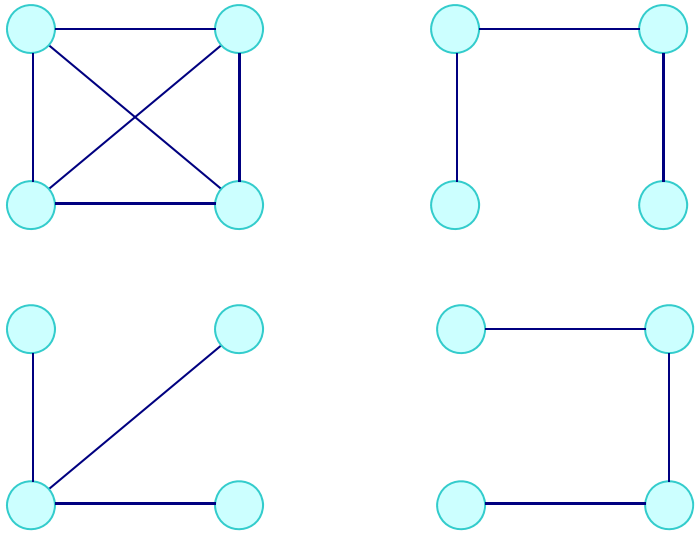
  트리 에지 T(tree edges) : 탐색 중에 방문하게 되는 에지 집합

  비트리 에지 N(nontree edges) : 나머지 에지들

  (이미 탐색 된 노드를 방문하여 확인하는 에지들)

- T에 포함된 에지들만 그래프에 그리면 G의 모든 노드들을 포함하는 트리 구조를 구성한다. 이렇게 만들어진 그래프는 그래프를 모두 연결하되 에지의 수를 최소로 하는 방법이 된다.

그래프(완전 그래프)의 예와 3개의 스패닝트리 예



스패닝 트리 종류

그래프 G의 스패닝 트리의 개수는 많다. 앞의 완전그래프의 예에서 가능한 스패닝 트리의 개수는 몇 개일까? 스패닝 트리를 구하는 방법에 따라 1개의 스패닝 트리를 구할 수 있지만, 앞에서 배운 dfs(), bfs() 알고리즘을 이용하여 스패닝 트리를 구할 수 있다.

깊이우선 스패닝트리(depth first spanning tree)

- dfs 알고리즘을 사용하여 방문 되어진 에지로 만들어진 스패닝트리

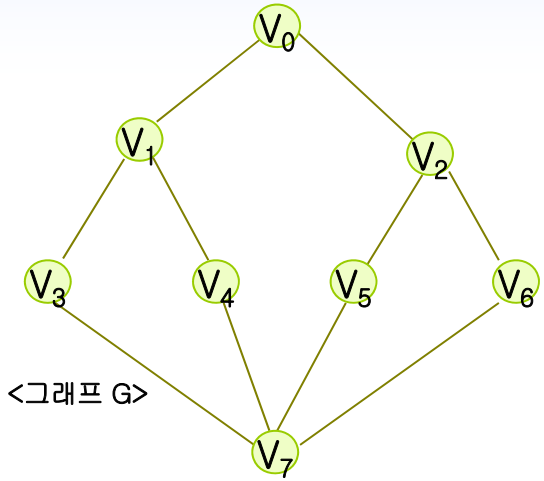
너비우선 스패닝트리(breadth first spanning tree)

- bfs 알고리즘을 사용하여 방문 되어진 에지로 만들어진 스패닝트리

참고 : 스패닝 트리의 성질

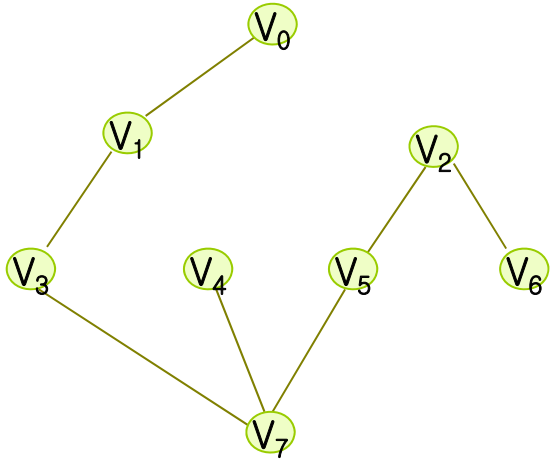
스패닝 트리에 새로운 에지(nontree edge), (v,w)를 더하면 사이클이 만들어진다.

그래프 G와 dfs(), bfs() 스패닝 트리 구하는 예 – 0번 노드에서 시작



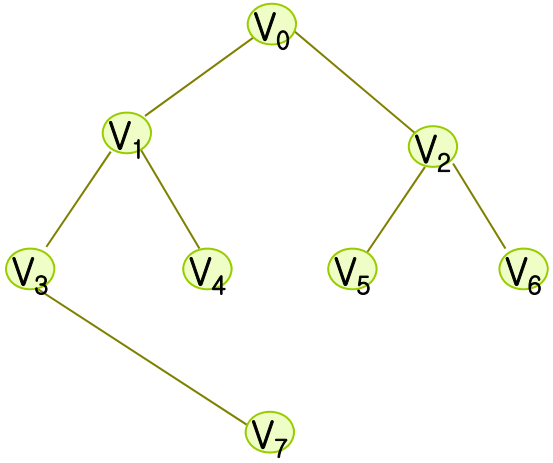
<그래프 G>

<그래프 G에 대한 스패닝트리(spanning trees)>



(a) 깊이우선 스패닝 트리

(dfs(0) spanning tree)



(b) 너비우선 스패닝 트리

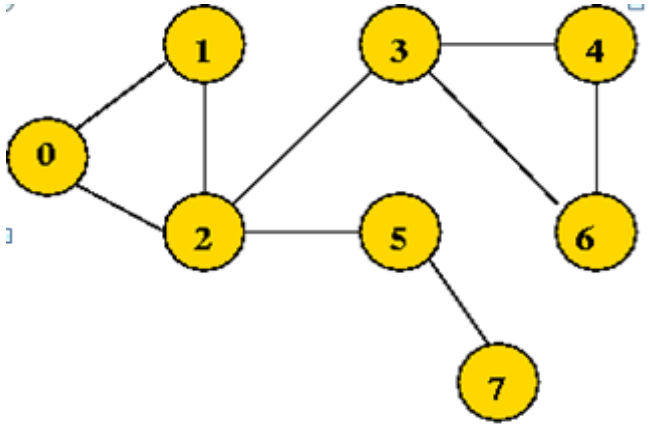
(bfs(0) spanning tree)

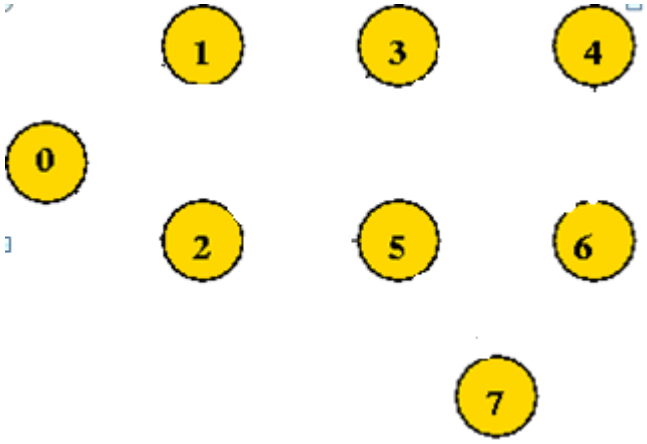
Q/A

다음 그래프의 스패닝 트리를 구하여라.

(1) 깊이우선 스패닝트리

(2) 너비우선 스패닝트리





2. 최소 스패닝 트리

스패닝 트리는 그래프의 모든 노드를 연결하고 에지의 개수를 최소인 n-1개 선택하는 문제이다. 만약 그래프의 각 에지에 비용(cost, 거리) 값이 주어진다면 n-1개의 에지를 택할 때 에지에 부여된 비용 값의 합을 최소로 하는 스패닝 트리를 구할 필요가 있다.

예를 들면 도로 건설 문제에서 도로를 연결하되 도로 건설 비용이 최소가 되도록 전체 도시를 연결할 것인지(사이클이 없이) 하는 문제는 최소 스패닝 트리 문제가 된다.

앞 절의 스패닝 트리에서 그래프의 에지들에 값이 주어질 때 스패닝 트리 중에서 에지에 주어진 값의 합을 최소로 하는 트리가 최소 스패닝 트리이다.

(용어) 최소 스패닝 트리(minimum cost spanning tree)

- 그래프에 대한 스패닝 트리 중 최소비용을 갖는 스패닝 트리
- 다음의 알고리즘으로 구한다(Kruskal's, Prim's, and Sollin's algorithms)

2. 최소 스패닝 트리

(1) Kruskal 알고리즘

(방법) 그래프에서 비용이 최소인 에지들을 한 개씩 노드만 그린 그래프에 더해 나간다. 비용이 최소인 에지만을 택해 나가기 때문에 택해진 n-1개의 에지의 비용 값의 합은 최소 값이 된다. 단 이때 에지를 더할 때 사이클을 만드는 에지는 스패닝 트리에 포함될 자격이 없으므로 버린다.

(알고리즘) 최소 스패닝 트리 구하는 Kruscal의 알고리즘

```
T = {};                          /* T에 구해진 에지를 더한다 */
while(T contains less than n-1 edges && E is not empty)
{
        choose a least cost edge (v,w) from E;   /*1*/
        delete (v,w) from E;
        if((v,w) does not create a cycle in T)    /*2*/
                add(v,w) to T;
        else  discard (v,w);
}
if(T contains fewer than n-1 edges)
        printf("no spanning tree₩n");
```

/*1*/ 번 문장 설명

choose a least cost edge (v,w) from E

- n개의 에지에서 최소인 에지를 구한다.

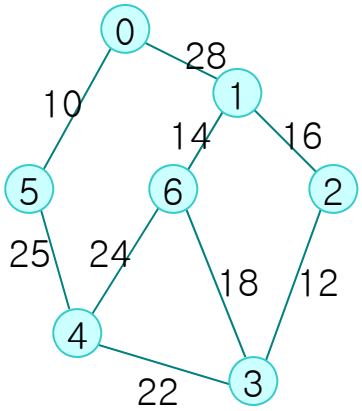
방법은 에지를 정렬하여 최소인 값을 하나씩 골라도되지만 효율적인 방법은 min heap(정렬 부분에서 나오게 됨)을 구성하고 한 개씩 골라내는 효율적인 방법이 있다. 이 방법은 heap의 구성시간(O(e)) + 매번 최소에지 탐색 시간(O(log2e))이 소요된다.

/* 2 */ 번 문장 설명

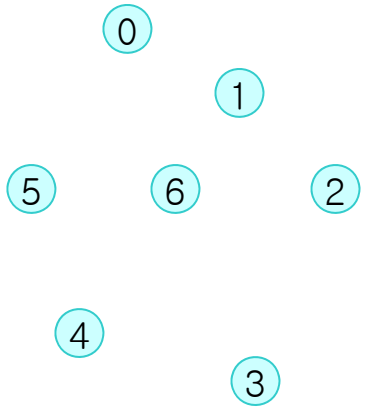
check that the new edge, (v,w), does not form a cycle in T

에지 (v,w)를 선택했을 때 사이클이 되는지 검사한다.

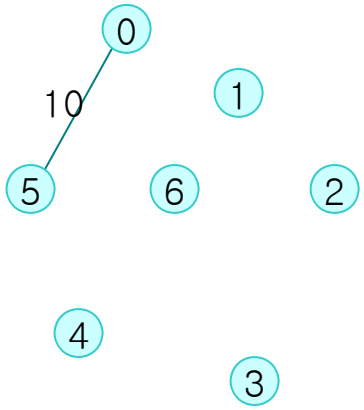
Kruskal 알고리즘 수행 과정



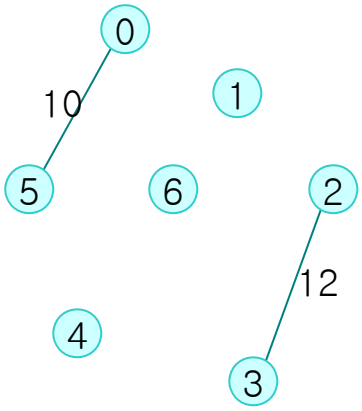
(a) 그래프 **G**



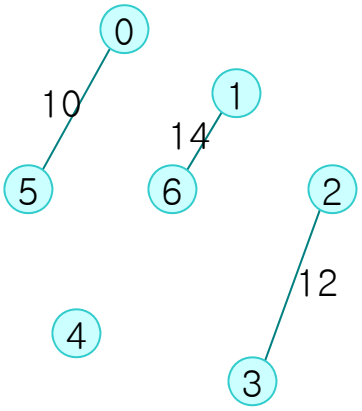
(b) 시작단계



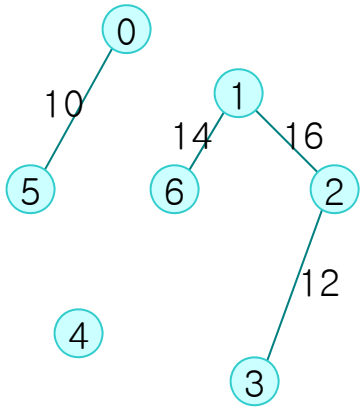
(c) 에지 **(0,5)** 추가
에지의 비용 값이 가장 적은 것을 선택



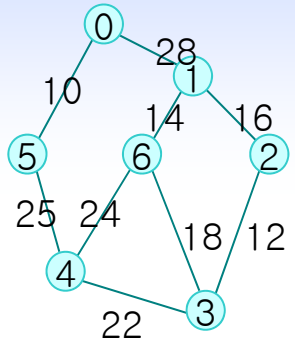
(d) 에지 **(2,3)** 추가



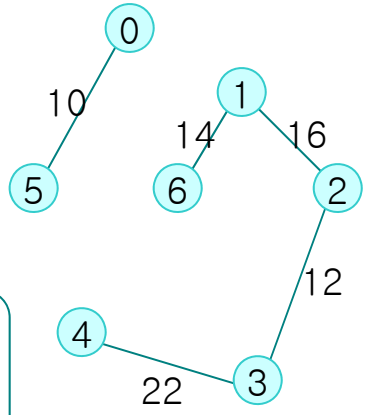
(e) 에지 **(1,6)** 추가



(f) 에지 **(1,2)** 추가

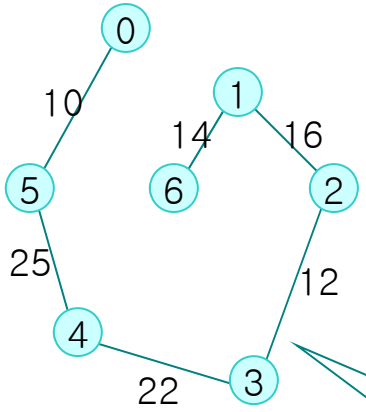


(a) 그래프 **G**



에지 (6,3)을 추가할 순서이나 사이클이 생기므로 버리고 다음 에지 (3,4) 추가

(g) 에지 **(3,4)** 추가



(h) 에지 **(4,5)** 추가 – 모두 연결된 상태

그래프에 더 이상 에지가 추가되면 그래프에 사이클이 생긴다

Kruskal's 알고리즘 수행 단계 요약

에지(edge)	비용(weight)	결과(result)	그림(figure)
-----	-----	시작l	(b)
(0,5)	10	선택	(c)
(2,3)	12	선택	(d)
(1,6)	14	선택	(e)
(1,2)	16	선택	(f)
(3,6)	18	버림	
(3,4)	22	선택	(g)
(4,6)	24	버림	
(4,5)	25	선택	(h)
(0,1)	28	더 이상 필요없음	

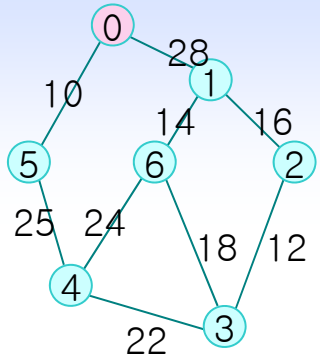
(2) Prim의 알고리즘

(방법) 최소 스패닝 트리를 구하는 두 번째 방법으로 Prim의 알고리즘이 있다. Prim의 알고리즘은 노드에서 시작하여 연결된 부속그래프를 만들어간다. 부속 그래프에서 새로운 노드를 선택할 때 연결된 에지 중 에지의 비용 값이 가장 작은 값이 노드를 선택하여 부속 그래프에 포함시킨다.

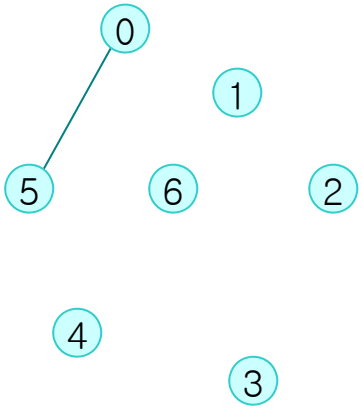
(알고리즘) 최소 스패닝 트리를 구하는 Prim의 알고리즘

```
T = { };            /* 초기에 빈 노드의 그래프 */
TV = {0};           /* 노드번호 0번에서 시작한다 */
while(T contains fewer than n-1 edges) {
        let (u,v) be a least cost edge
                such that u ∈ TV and v ∉ TV;
        if(there is no such edge) break;
        add v to TV;
        add (u,v) to T;
}
if(T contains fewer than n-1 edges)
        printf("no spanning tree₩n");
```

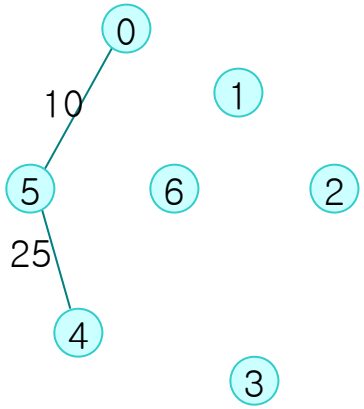
(2) Prim의 알고리즘 – 0번 노드에서 시작



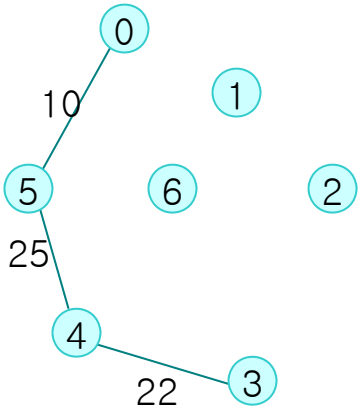
시작그래프 **G**



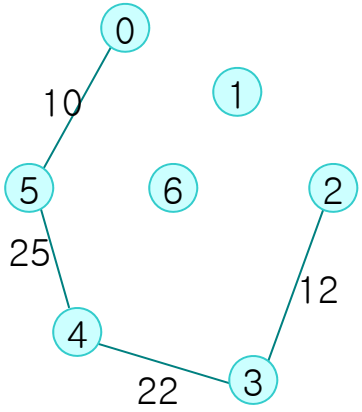
(a) **(0,5)** 포함 **TV={0,5}**



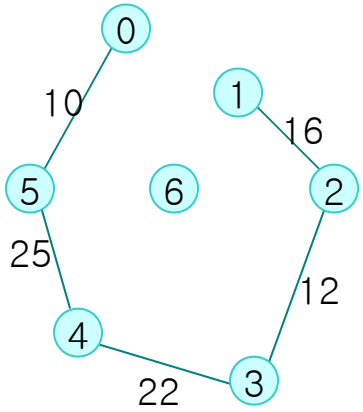
(b) **(5,4)** 포함 **TV={0,5,4}**



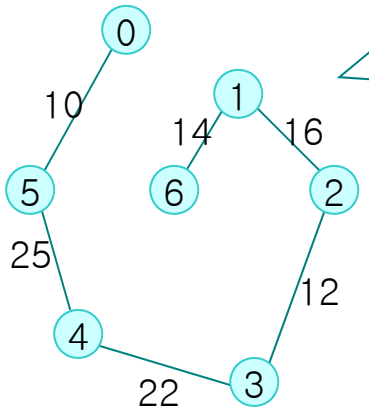
(c) **(4,3)** 포함 **TV={0,5,4,3}**



(d) **(3,2)** 포함 **TV={0,5,4,3,2}**



(e) **(2,1)** 포함 **TV={0,5,4,3,2,1}**



그래프에 더 이상 에지가 추가되면 그래프에 사이클이 생긴다

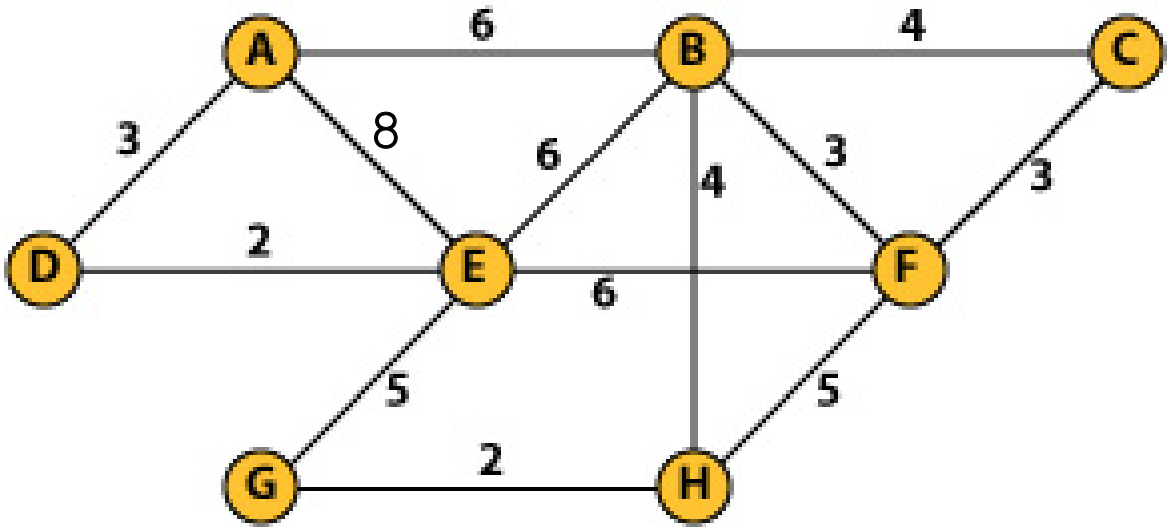
(f) **(1,6)** 포함 **TV={0,5,4,3,2,1,6}**

Q/A

다음 그래프의 스패닝 트리를 구하여라.

(1) 최소 스패닝트리(Kruscal algorithm)

(2) 최소 스패닝트리(Prim algorithm)



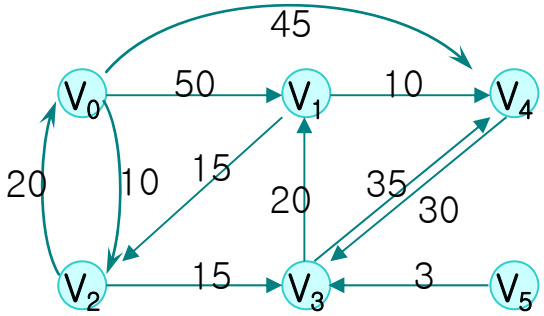
3. 최단경로(shortest path) 문제

최단경로 문제

서울시의 지하철에서 종로1가에서 강남역까지 가는 방법 중 지하철을 바꿔 타더라고 가장 빨리(시간), 가장 짧은 거리, 혹은 지하철역 수를 적게 거쳐서 가는 방법은 최단 거리 문제가 된다.

경로의 길이

경로상의 에지에 부여된 비용 값의 합으로 예제의 경우 경로상의 비용 값은 거리 혹은 시간 등이다. 그러나 전철역의 수 만 따진다면 전철역 간의 거리 값을 모두 1로 계산하면 된다.



(a) 비용값이 있는 그래프의 예

경로(path)	길이 합(length)
1) v0 v2	10
2) v0 v2 v3	25
3) v0 v2 v3 v1	45
4) v0 v4	45

(b) v0으로부터 각 정점까지 최단거리 경로와 거리값

3. 최단경로(shortest path) 문제

최단경로 문제를 위한 정의들

v_0 : 출발점(source vertex)

S : v0에서 최단 경로 문제를 구할 때 중간에 최단거리가 찾아진 노드의 집합
(set of vertices, including v0, whose shortest paths have been found)
w : 최단거리를 찾아야 할 노드의 집합(any vertex $\notin$ S)

distance[x] : 노드 v0으로 부터 S에 포함된 노드를 거쳐서 노드 x까지 의 거리 값 (the length of the shortest path starting from v0, going through vertices only in S, and ending in w)

최단경로 구하는 방법 설명

(1) 최단경로 문제는 한 지점 v_0 에서 다른 지점 w까지를 구하는 문제이지만 구하는 과정에서 출발점 v_0 에서 모든 도착점까지 한꺼번에 구하게 된다. 즉 모든 노드 w에 대하여 distance[w]를 구한다.

(2) 구하는 중간 과정에서 이미 최단 경로가 발견된 노드의 집합은 S라고 한다. 즉 S에 포함된 노드에 대해서는 최단 경로 계산이 끝나있다.

(3) 알고리즘이 시작될 때 각 노드에 대한 distance[w]는 출발 노드 x로 부터 바로 가는(다른 노드를 안 거치고) 거리 값이다. 구하는 중간 과정에서 distance[w]는 노드 w에 대한 출발점으로부터의 최단 거리 값이며 $w \in S$이면 distance[w]는 최종 값이고 그렇지 않으면 출발점 x에서 S를 거쳐서 가는 경로 중 최단 경로의 값이다.

(4) S에 포함될 새로운 노드를 택할 때는 $u \notin S$인 노드를 택하며 현재까지의 distance[u] 중 최소인 값을 택한다. 이 때 distance[ ], $w \notin S$의 값은 다음 값 중 작은 값을 택한다. 택한 후 distance[w] 값을 수정하는 데, 이 값은 현재 값과 최단 경로가 구해진 노드를 통과하여 w로 가는 길 중 짧은 길을 택하는 것이다.

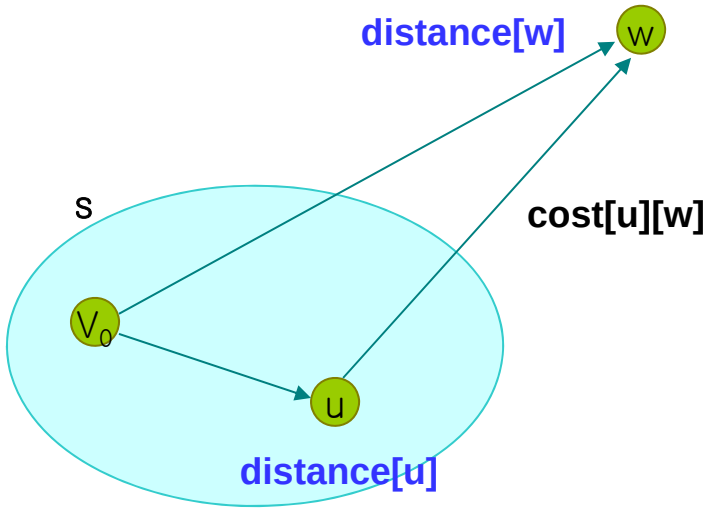
- 알고리즘 중간의 노드 w 선택 후 minimum 값의 계산

① S만 거쳐가는 거리(이미 계산되어있음) - distance[w],

② $u \in S$에 대하여 distance[u] + length(<u,w>)

최단경로 문제 – 새로운 노드에 대한 최단 경로의 계산

S에 포함될 새로운 노드를 택할 때는 u ∉ S인 노드를 택하며 현재까지의 distance[u] 중 최소인 값을 택한다. 이 때 distance[w], w ∉ S의 값은 다음 값 중 작은 값을 택한다. 이 값은 현재 값과 최단 경로가 구해진 노드 u를 통과하여 w로 가는 길 중 짧은 길을 택하는 것이다. 다른 짧은 경로는 없다.



min{*distance*[*w*], *distance*[*u*]+*cost*[*u*][*w*]}

```
/* 최단거리 문제에 대한 자료선언 */
#define MAX_VERTICES 6
int cost[][MAX_VERTICES] =
        {{   0,   50,   10, 1000,   45, 1000},
         {1000,    0,   15, 1000,   10, 1000},
         {  20, 1000,    0,   15, 1000, 1000},
         {1000,   20, 1000,    0,   35, 1000},
         {1000, 1000,   30, 1000,    0, 1000},
         {1000, 1000, 1000,    3, 1000,    0}};
int distance[MAX_VERTICES];
short int found[MAX_VERTICES];
int n = MAX_VERTICES;
```

found	f	f	f	f	f	f

distance	0	50	10	1000	45	1000

최단 경로(shortest path) 알고리즘 - Dijkstra's Algorithm

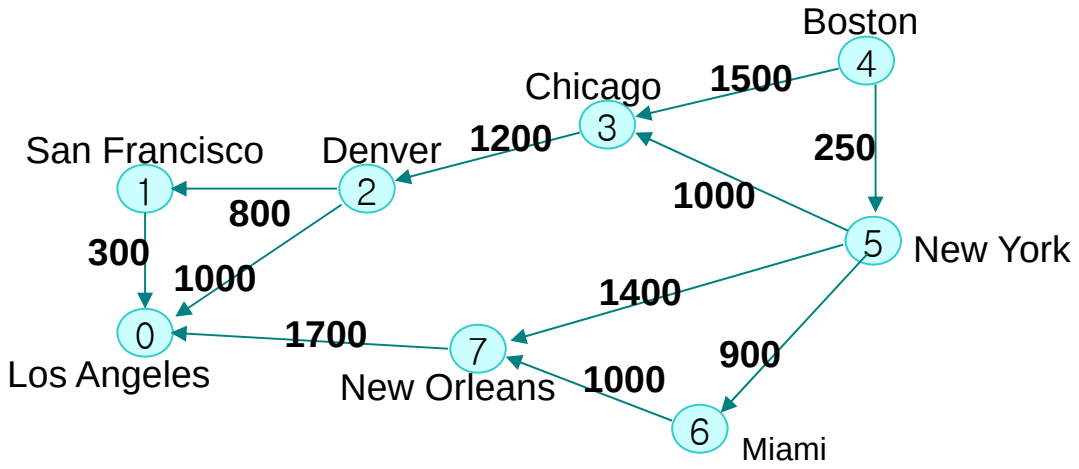
```
/* 최단 경로(shortest path) 알고리즘 */
void shortestpath(int v, int cost[][MAX_VERTICES], int distance[],
 int n, short int found[]) {
        int i, u, w;
        /* 초기화 작업 */
        for(i = 0; i < n; i++) {
                found[i] = FALSE;
                distance[i] = cost[v][i];
        }
        found[v] = TRUE;
        distance[v] = 0;
        for(i = 0; i < n-2; i++) {
                u = choose(distance, n, found); /*1*/
                found[u] = TRUE;
                for(w = 0; w < n; w++)
                        if(!found[w]) /*2*/
                          if(distance[u] + cost[u][w] < distance[w])
                             distance[w] = distance[u] +cost[u][w];
          }
   }
```

choose() 함수

```
/* choose() 함수 */
int choose(int distance[], int n, int found[]) {
int i, min, minpos;
        min = INT_MAX;
        minpos = -1;
        for(i = 0; i < n; i++) /*1*/
                if(distance[i] < min && !found[i]) {
                        min = distance[i];
                        minpos = i;
                }
        return minpos;
}
```

최단경로 문제 예

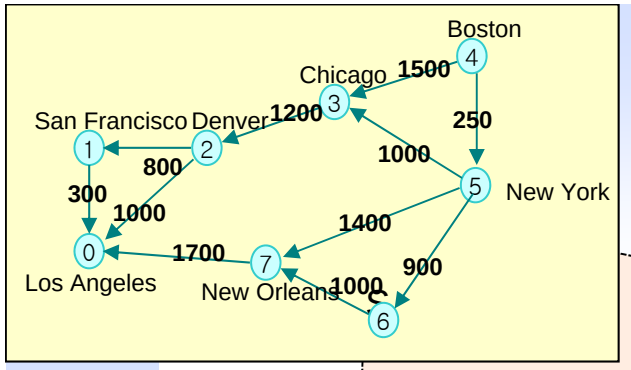
아래 지도에서 Boston 에서 각 도시까지 가는 최단 경로를 구하라.



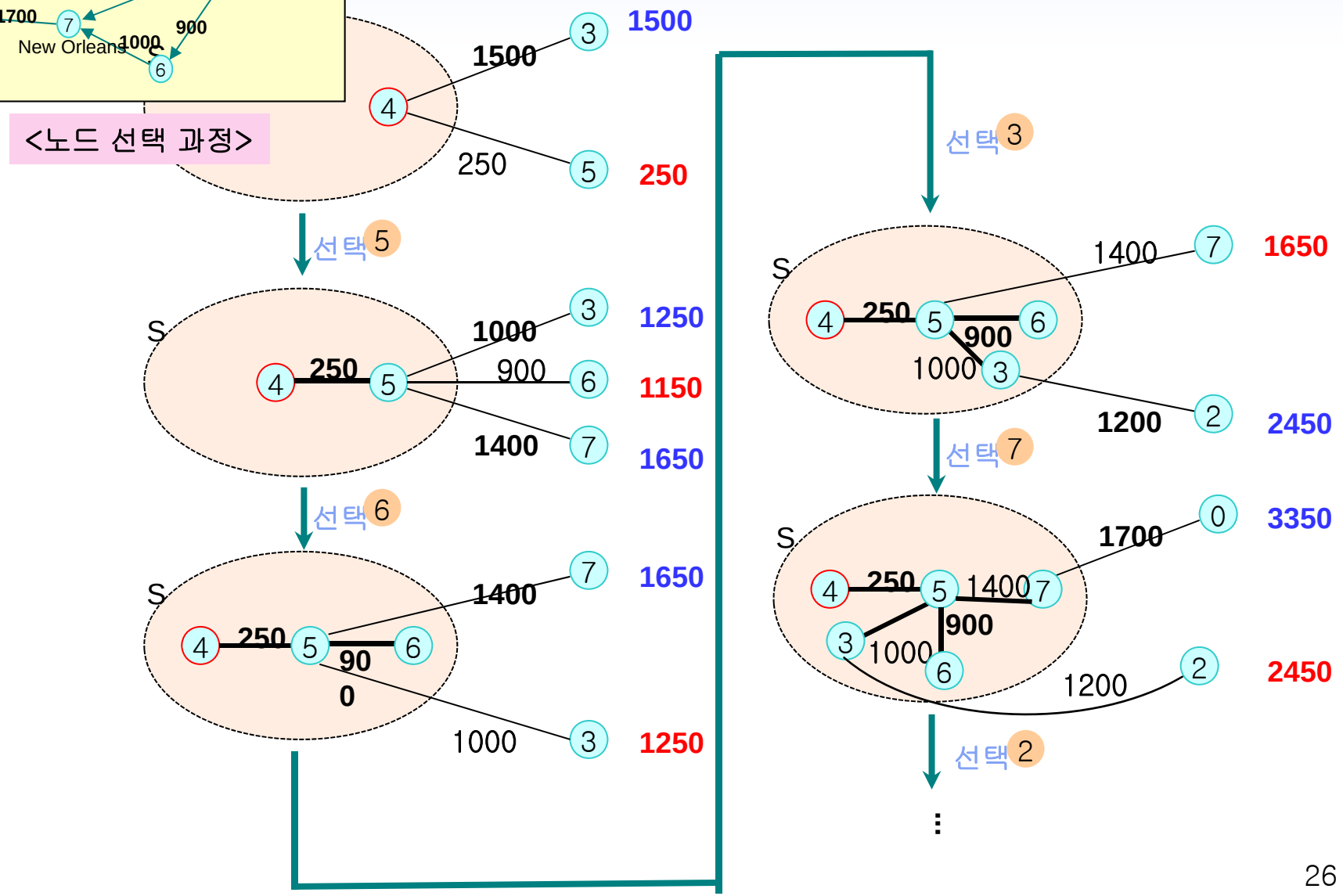
(a) 비행기 경로의 거리 값을 표시한 지도

	0	1	2	3	4	5	6	7
0	0							
1	300	0						
2	1000	800	0					
3			1200	0				
4				1500	0	250		
5				1000		0	900	1400
6							0	1000
7	1700							0

(b) 인접 행열로 표시한 거리 값

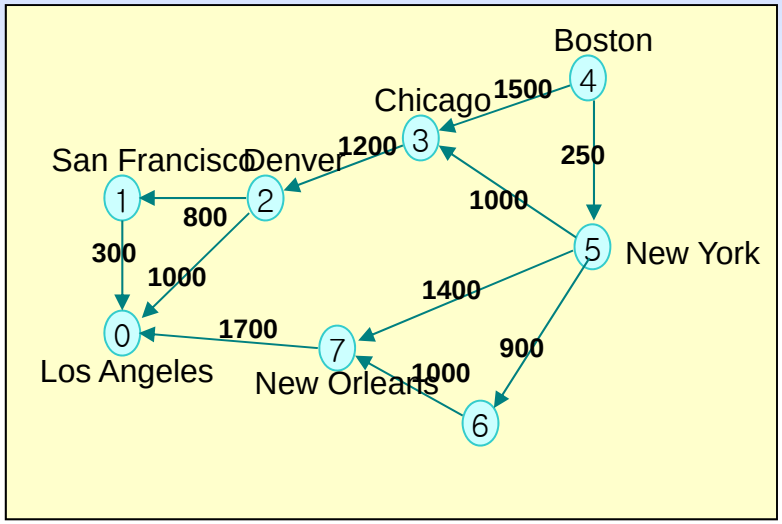


<노드 선택 과정>



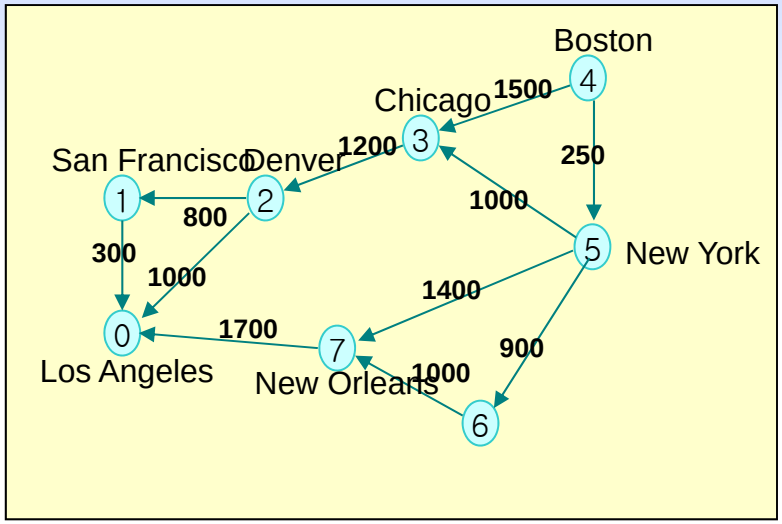
< C 자료구조 입문 >

3. 최단경로(shortest path) 문제



Iteration	S	Vertex selected	Distance							
			LA [0]	SF [1]	DEN [2]	CHI [3]	BOST [4]	NY [5]	MIA [6]	NO [7]
Initial	--	----	+∞	+∞	+∞	1500	0	250	+∞	+∞
1	{4}	5	+∞	+∞	+∞	1250	0	250	1150	1650
2	{4,5}	6	+∞	+∞	+∞	1250	0	250	1150	1650
3	{4,5,6}	3	+∞	+∞	2450	1250	0	250	1150	1650
4	{4,5,6,3}	7	3350	+∞	2450	1250	0	250	1150	1650
5	{4,5,6,3,7}	2	3350	3250	2450	1250	0	250	1150	1650
6	{4,5,6,3,7,2}	1	3350	3250	2450	1250	0	250	1150	1650
	{4,5,6,7,3,2,1}									

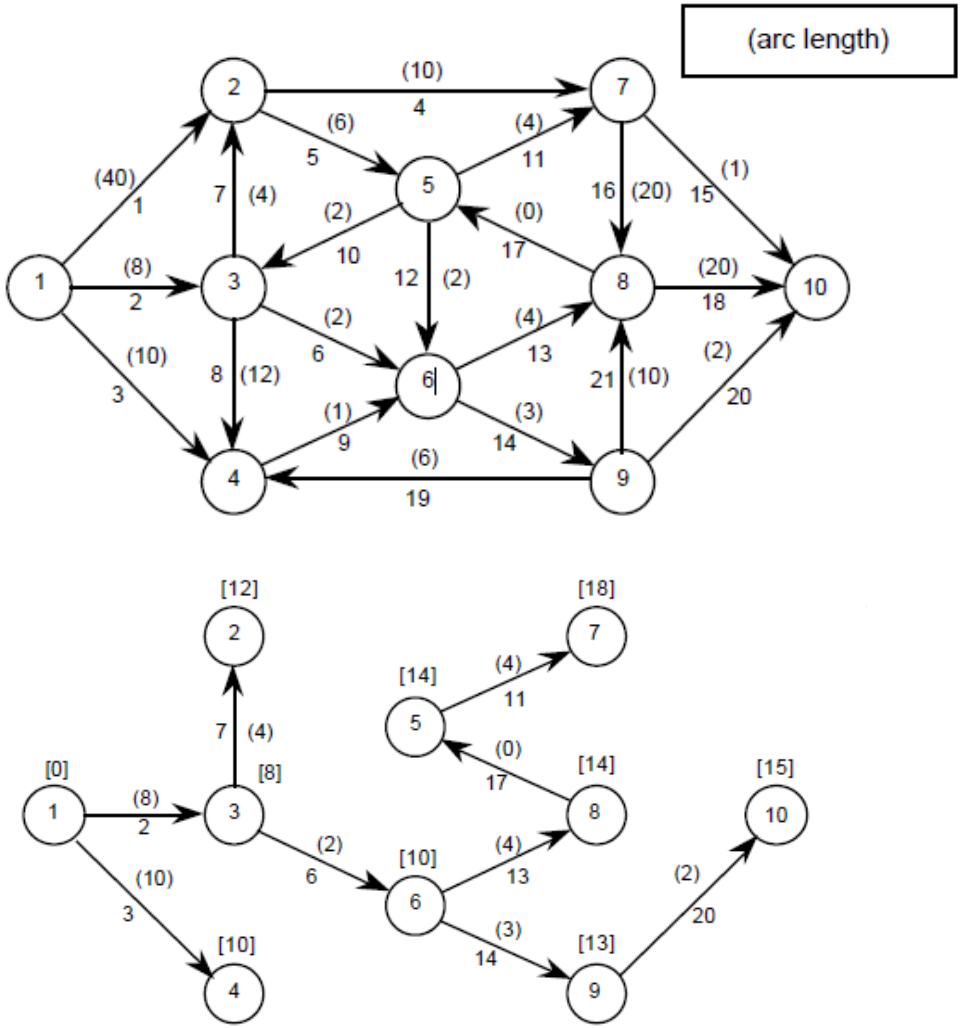
3. 최단경로(shortest path) 문제



Iteration	S	Vertex selected	Distance							
			LA [0]	SF [1]	DEN [2]	CHI [3]	BOST [4]	NY [5]	MIA [6]	NO [7]
Initial	--	----	+∞	+∞	+∞	1500	0	250	+∞	+∞
1	{ }									
2	{ }									
3	{ }									
4	{ }									
5	{ }									
6	{ }									
	{ }									

Q/A

다음 그래프의
최단 경로를 구하여라



4. 이행성 폐포(transitive closure) 문제

이행성 폐포 문제(transitive closure) :

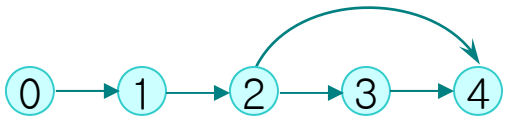
그래프의 노드 x 에서 경로 y로 가는 경로(path)가 있는지를 묻는 문제이다(최단거리 문제에서는 경로의 비용 값까지 구했다). 즉 그래프를 행열로 표시했을 때 x에서 y로 바로 가는 길(에지)이 있으면 “1”, 없으면 “0”을 저장한다. 경로는 이 에지뿐만 아니라 x에서 y로 가는 다른 경로가 있는지도 다 찾아야 한다. 이 문제를 좀 더 수학적으로 표시하면 다음과 같다.

정의) A+ : 그래프 G의 이행성 폐포 행열(transitive closure matrix)

- 노드 i에서 j로 가는 경로길이(> 0)가 있으면 A+[i][j] = 1
- 그렇지 않으면 A+[i][j] = 0
- 행열 A로 부터 A+를 구하는 방법은 행열을 곱하여 구할 수 있다.

정의) A* : 그래프 G의 재귀적 이행성폐포 행열(reflexive transitive closure matrix)

- 노드 i에서 j로 가는 경로길이(> 0)가 있으면 A*[i][j] = 1
- 그렇지 않으면 A*[i][j] = 0
- A+로 부터 A*[i][i] = 1 로 만들면 된다.



(a) 방향 그래프 **G**

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

(b) **G**의 인접 행열 **A**

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

(c) **G**의 $\mathbf{A}^{+}$

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

(d) **G**의 $\mathbf{A}^{*}$

그래프 G와 인접 행열 A, A^{+}, A^{*}

학습내용정리

스패닝 트리 문제는 예를 들면 경기도의 10개도시의 송유관을 건설한다고 할 때 어떻게 하면 중복시키지 않고(사이클이 없이) 모든 도시를 공급지와 연결을 시킬 것인가 하는 문제이다. 스패닝 트리를 구하는 방법은 많지만 dfs(), bfs() 그래프 탐색을 이용하여 방문하는 간선을 모으면 스패닝 트리가 된다. dfs() 방문으로 얻어진 트리를 깊이우선 스패닝 트리라고 하고 bfs() 방문으로 얻어진 스패닝 트리를 너비우선 스패닝 트리라고 한다.

최소 스패닝 트리는 예를 들면 도로 건설 문제에서 도로를 연결하되 최소비용의 도로 건설 비용으로 전체 도시를 연결할 것인지(사이클이 없이) 하는 문제이다. Kruscal과 Prim의 알고리즘을 이용하여 구할 수 있다.

최단경로 문제는 예를 들면 서울시의 지하철에서 종로1가에서 강남역까지 가는 방법 중 지하철을 바꿔 타더라도 가장 빨리(시간), 가장 짧은 거리, 혹은 지하철 역 수를 적게 거쳐서 가는 방법을 구하는 것이다. Dijkstra의 알고리즘으로 구할 수 있다.

이행성 폐포 문제를 그래프의 경로의 존재 여부를 찾아내는 문제이다.

그래프에 관한 이러한 알고리즘은 문제를 해결할 때 알고리즘이 얼마나 효율적으로 해결할 수 있는가를 보여준다.