성신여자대학교

자료구조 실습

안양준

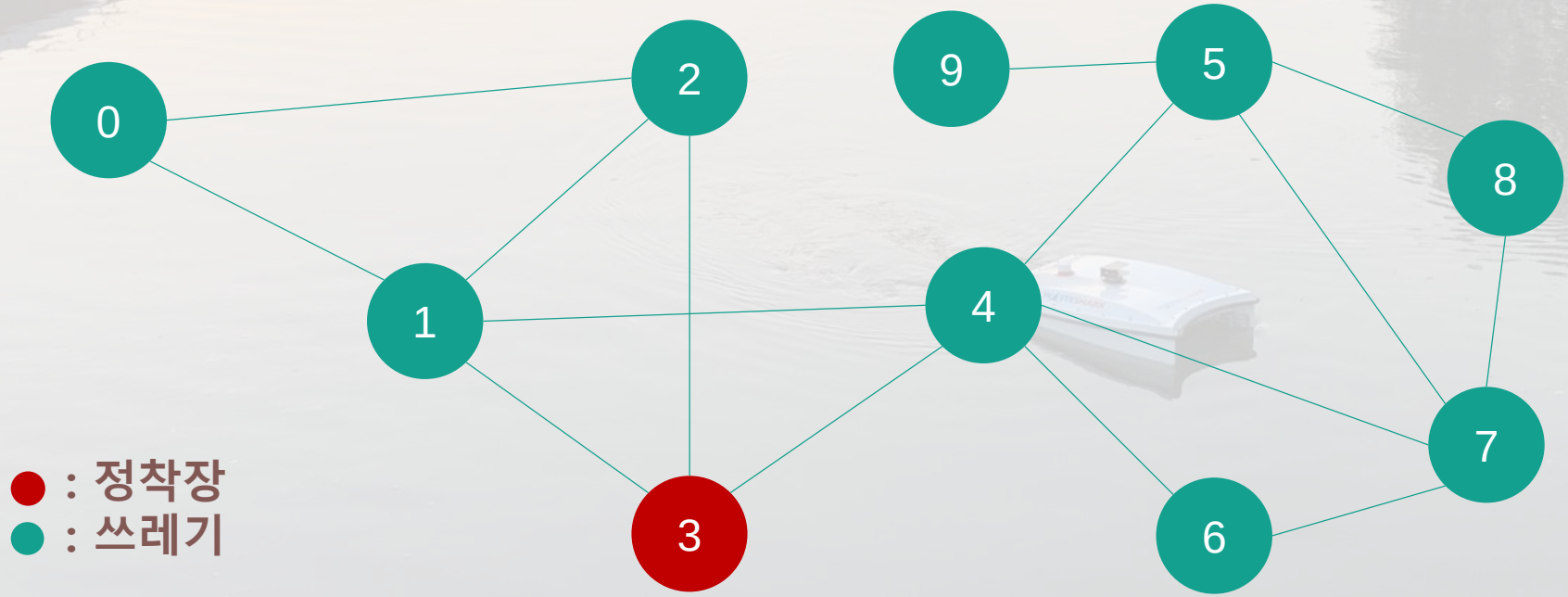
yangjunahn@sungshin.ac.kr

Overview

- **DFS 알고리즘**
- **그래프 표현**
  - 인접 행렬(adjacency matrix) / 인접 리스트(adjacency list)
  - 직접 구현(교재) / STL 이용
- **가중치 없는 그래프의 몇가지 알고리즘들**
- **가중치 그래프(Weighted Graph)**
  - 벨만포드(Bellman-Ford) 알고리즘
  - 다익스트라(Dijkstra) 알고리즘

문제 1

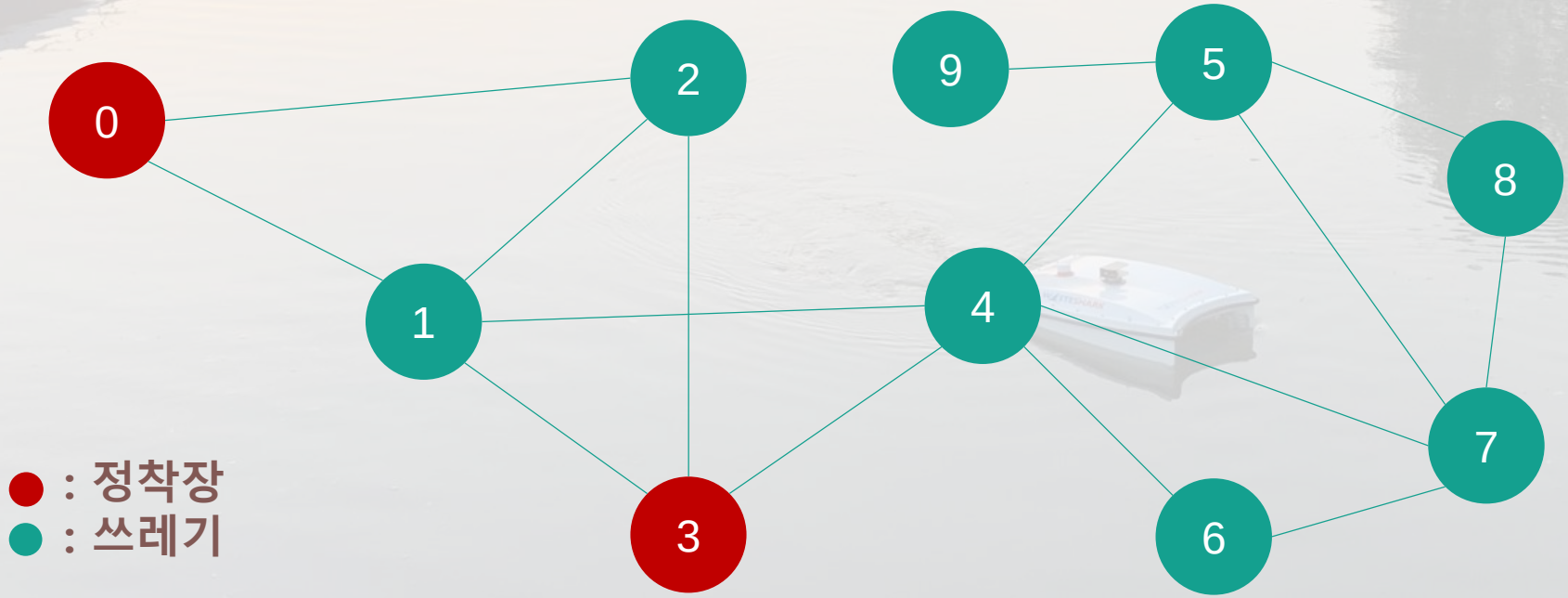
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 각 쓰레기가 정착장과 얼마나 떨어져 있는지 구하세요.



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

문제 2

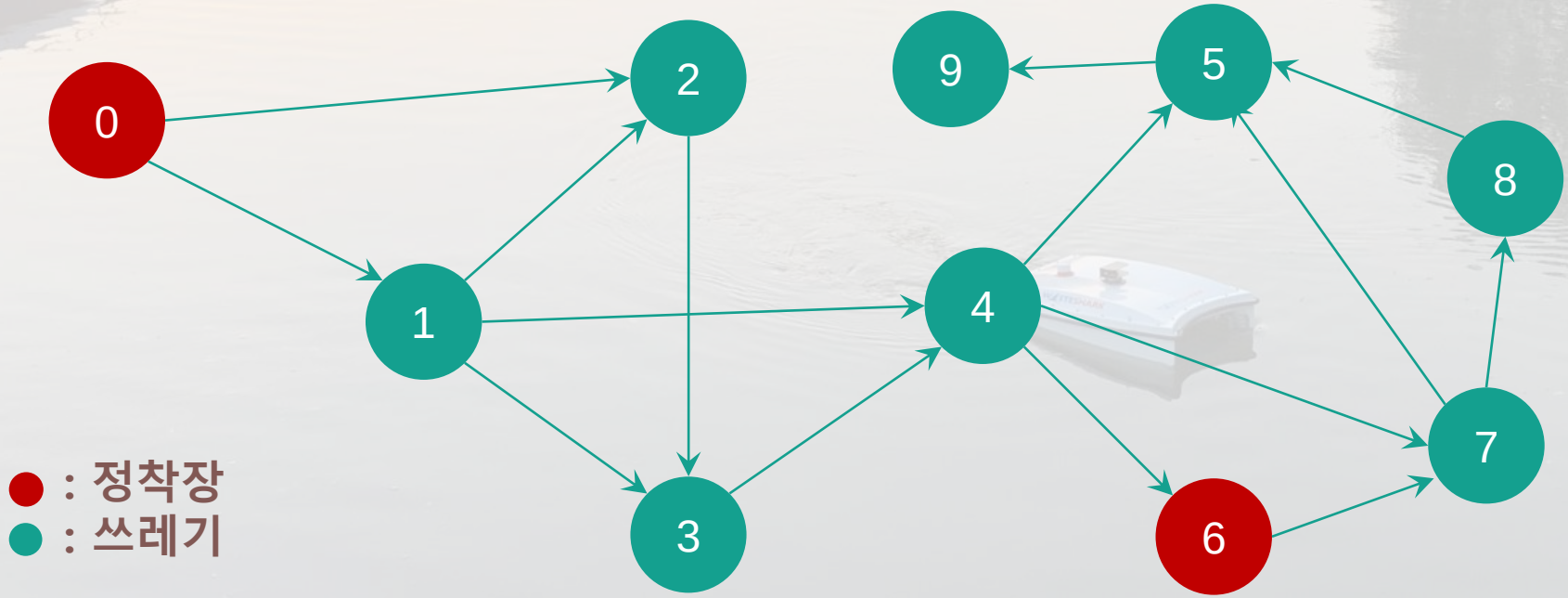
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 2개 이상 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 각 쓰레기가 가장 가까운 정착장과 얼마나 떨어져 있는지 구하세요.



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

문제 3

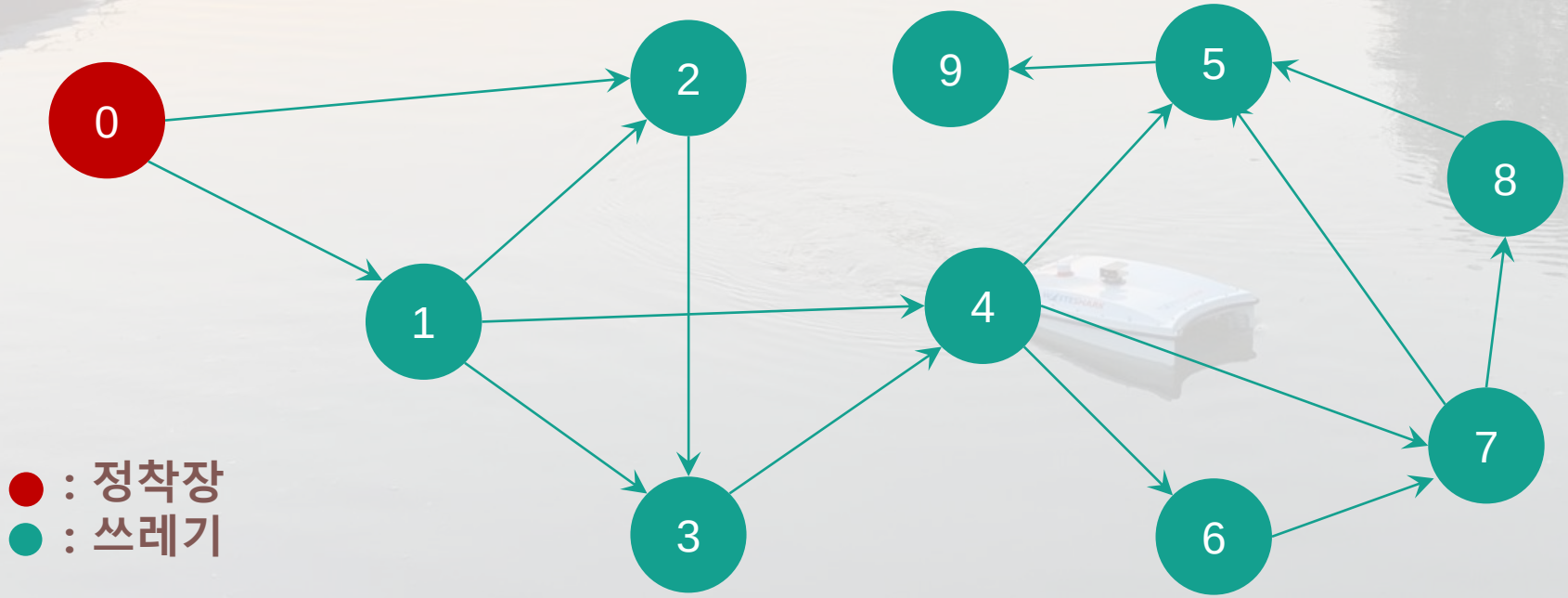
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 2개 이상 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 물살로 인해 역방향 운항은 불가합니다.
- 로봇은 한번의 충전으로 최소 몇 km를 가야 할까요?



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

문제 4

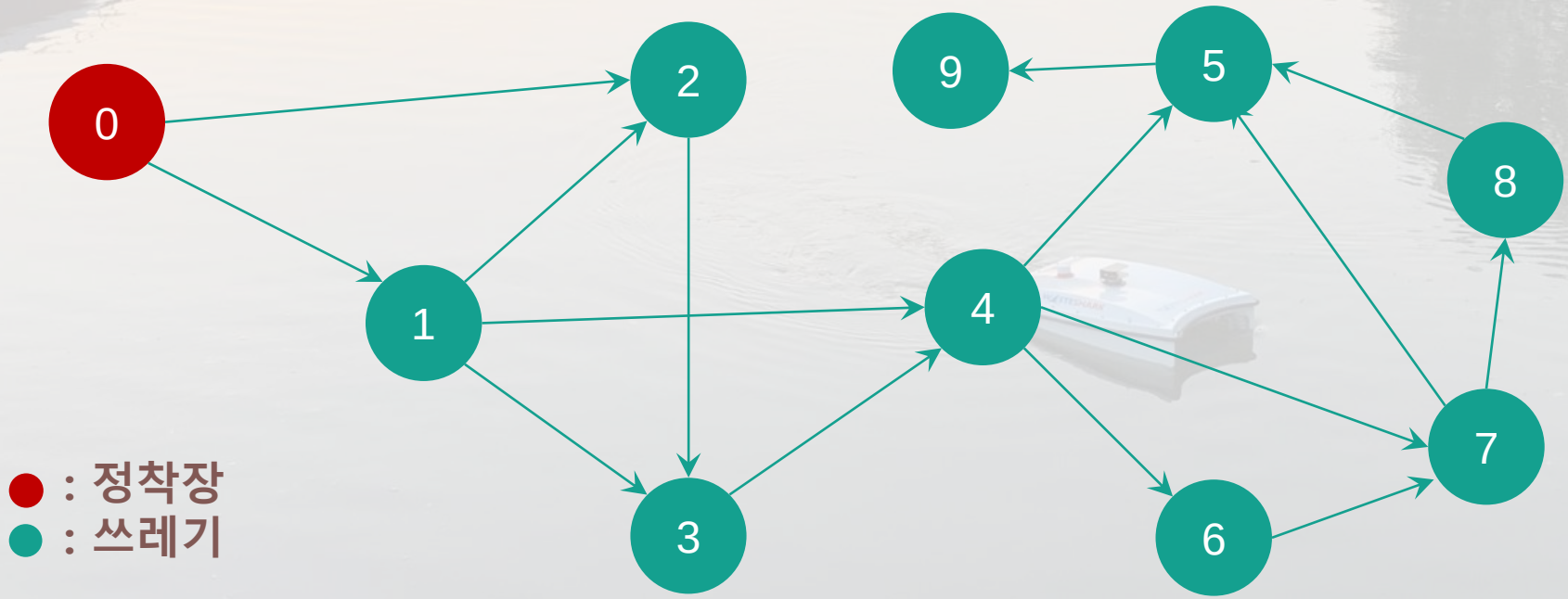
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 하나 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 물살로 인해 역방향 운항은 불가합니다.
- 9번 쓰레기를 수거하러 가는 도중 신호가 끊겼습니다. 수거 로봇이 이동하 있는 모든 경로를 나열하여, 수거 로봇의 위치를 특정하세요.



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

문제 5

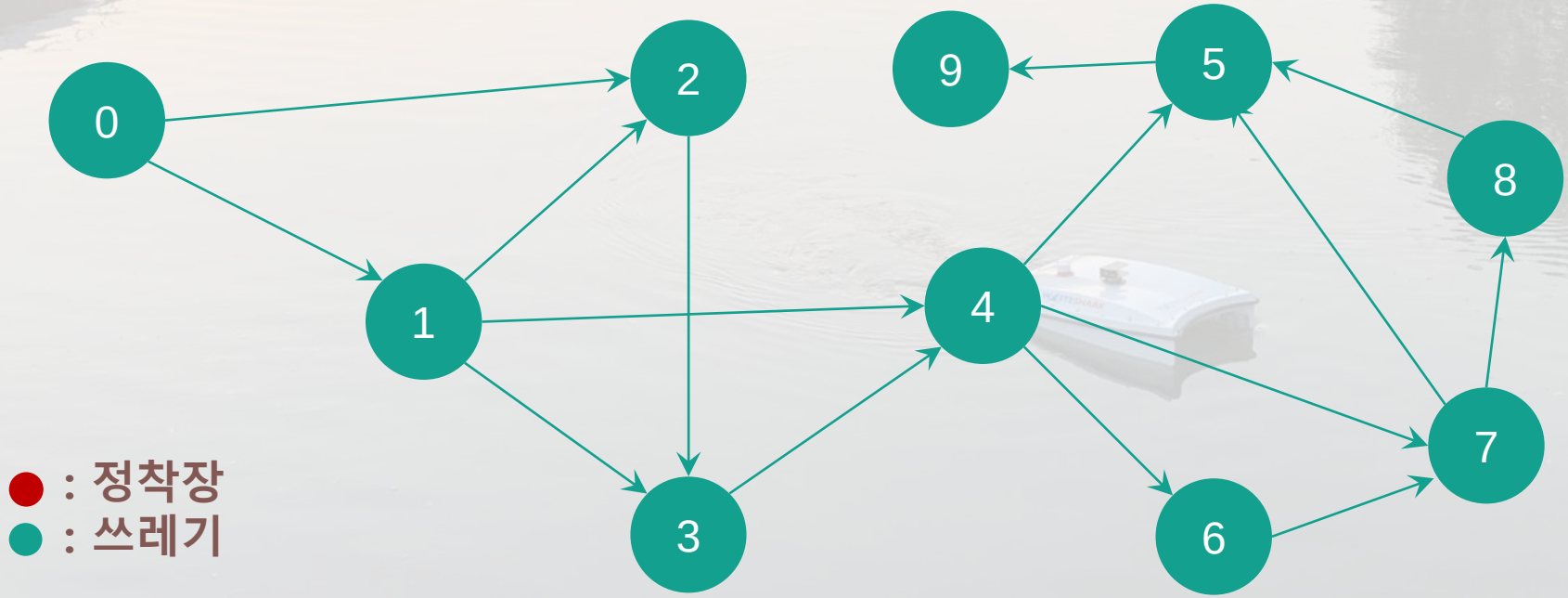
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 하나 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 물살로 인해 역방향 운항은 불가합니다.
- 로봇이 이동 중 같은 자리를 맴돌 가능성이 있는지 확인하세요.



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

추가 문제

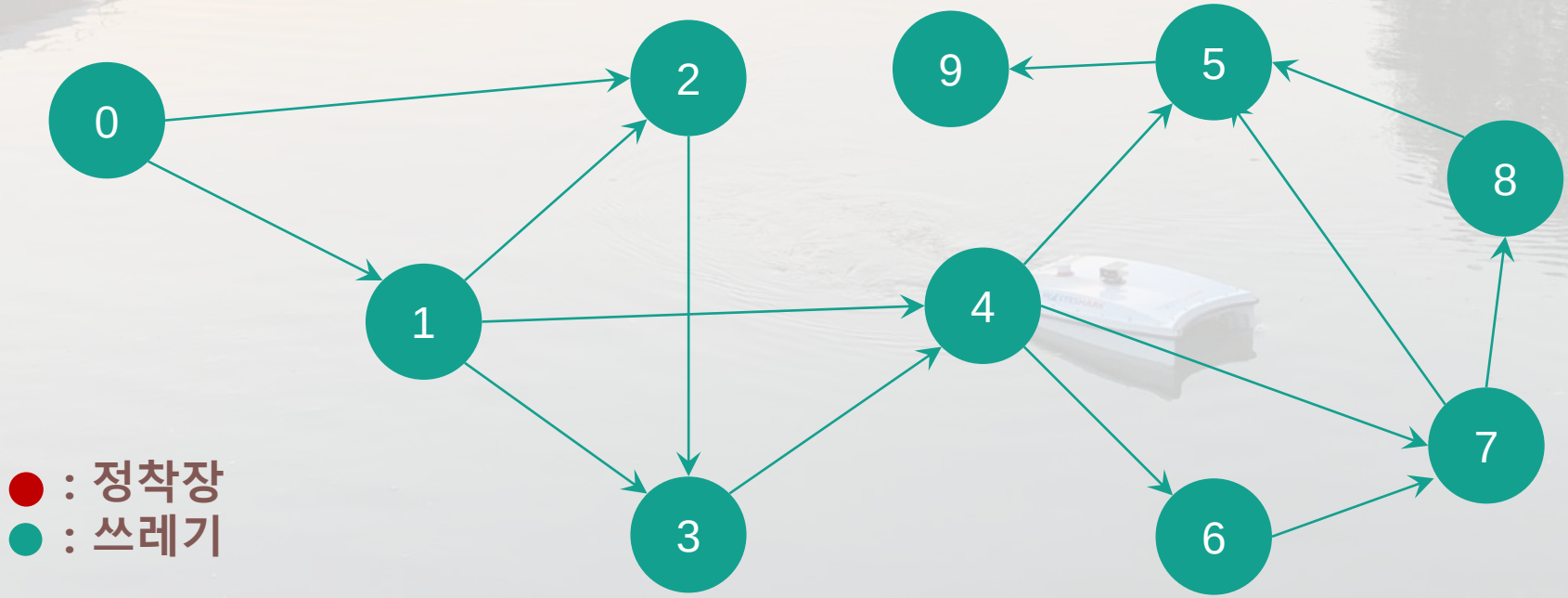
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 물살로 인해 역방향 운항은 불가합니다.
- 모든 쓰레기를 수거하고 정착장으로 돌아올 수 있는지 확인하세요.



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

추가 문제

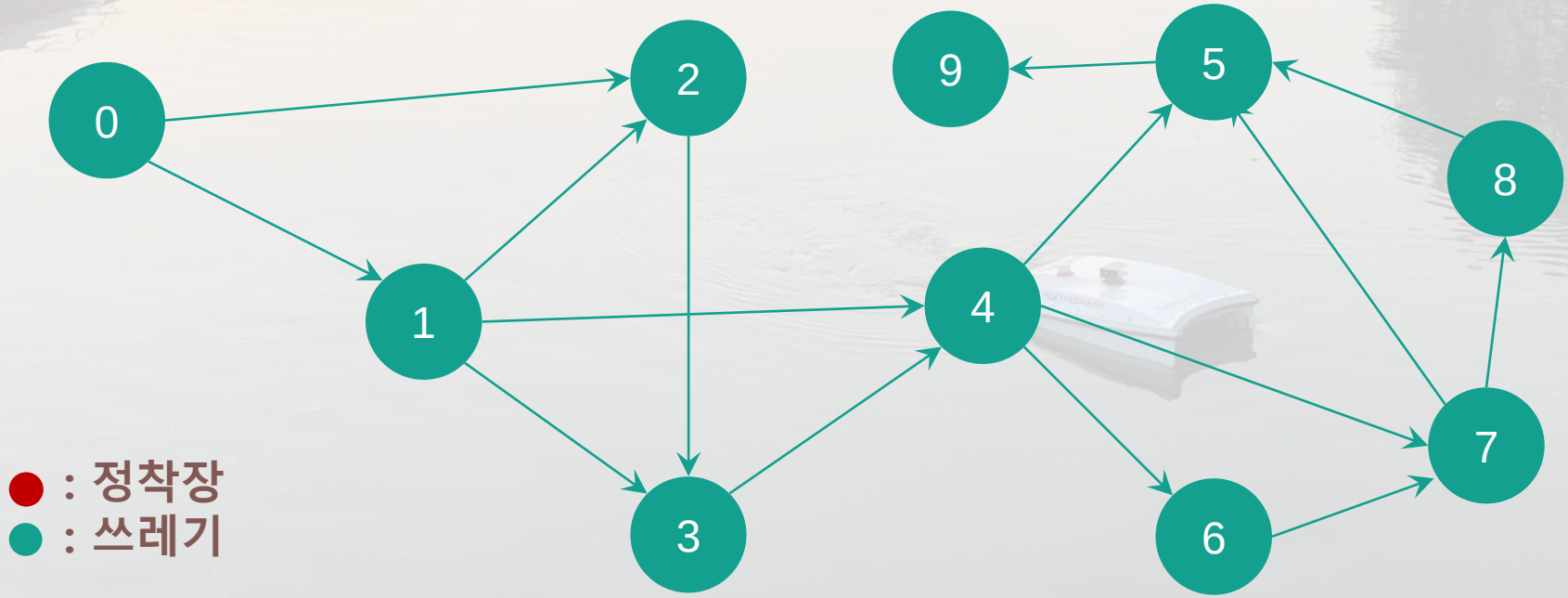
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 2개 이상 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 물살로 인해 역방향 운항은 불가합니다.
- 정착장을 설치해야 하는 위치는 어디일까요?



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

추가 문제

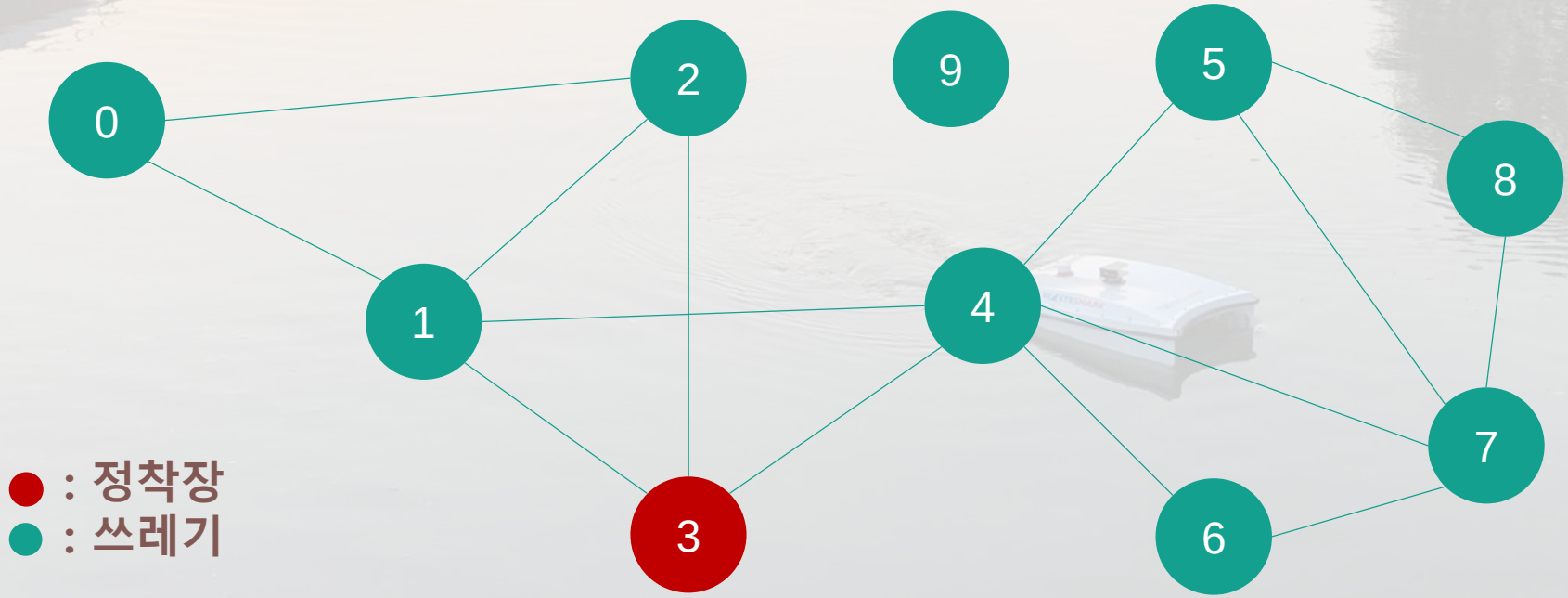
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장을 설치해야 합니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 물살로 인해 역방향 운항은 불가합니다.
- 최소 몇개의 정착장을 설치해야 하나요?



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

추가 문제

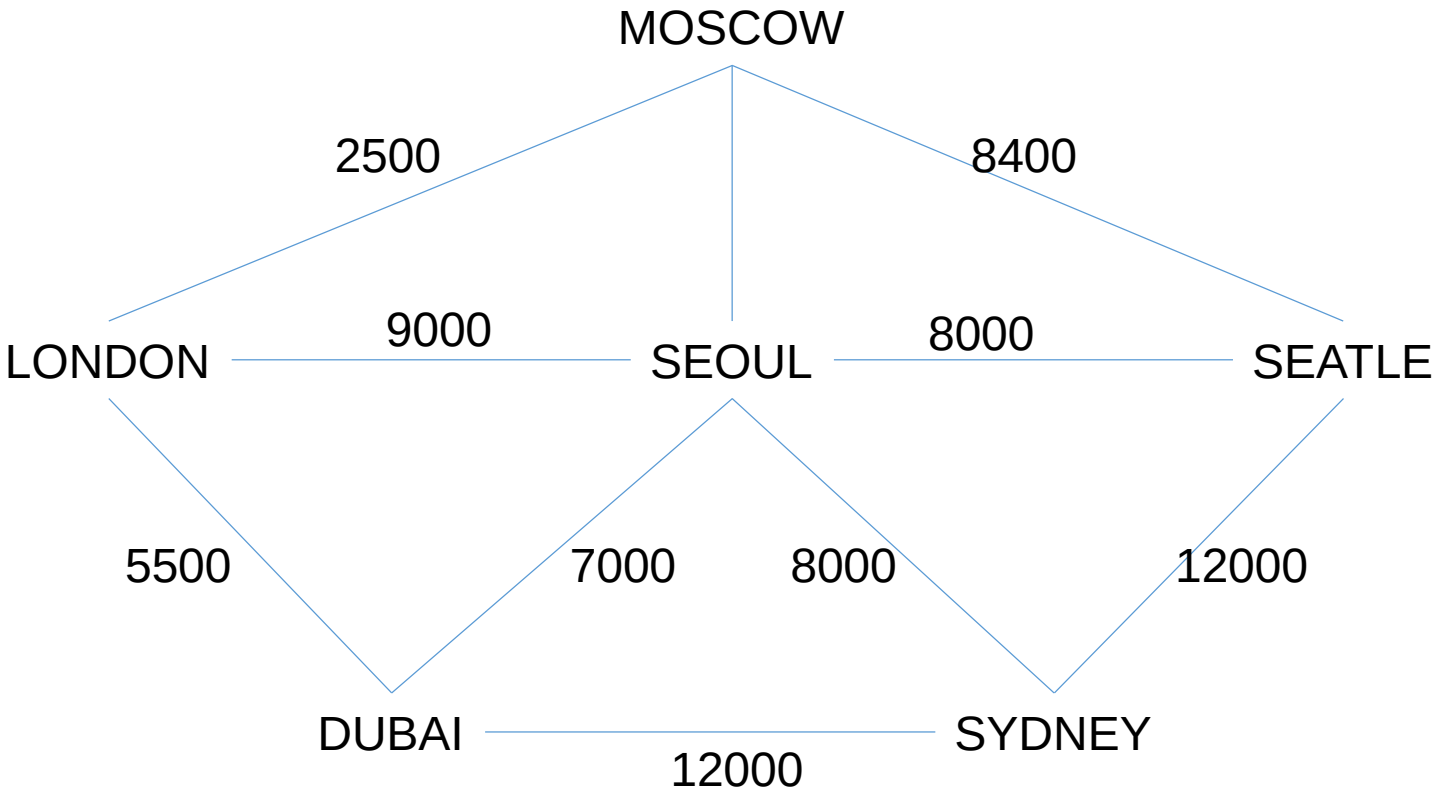
- 수거 로봇을 이용해 쓰레기를 수거합니다.
- 쓰레기 로봇이 충전할 수 있는 정착장이 있습니다.
- 쓰레기와 쓰레기, 정착장과 쓰레기 거리는 모두 1km로 동일합니다.
- 수거가 불가능한 쓰레기를 보고하세요.



AI융합학부 안양준 yangjunahn@sungshin.ac.kr

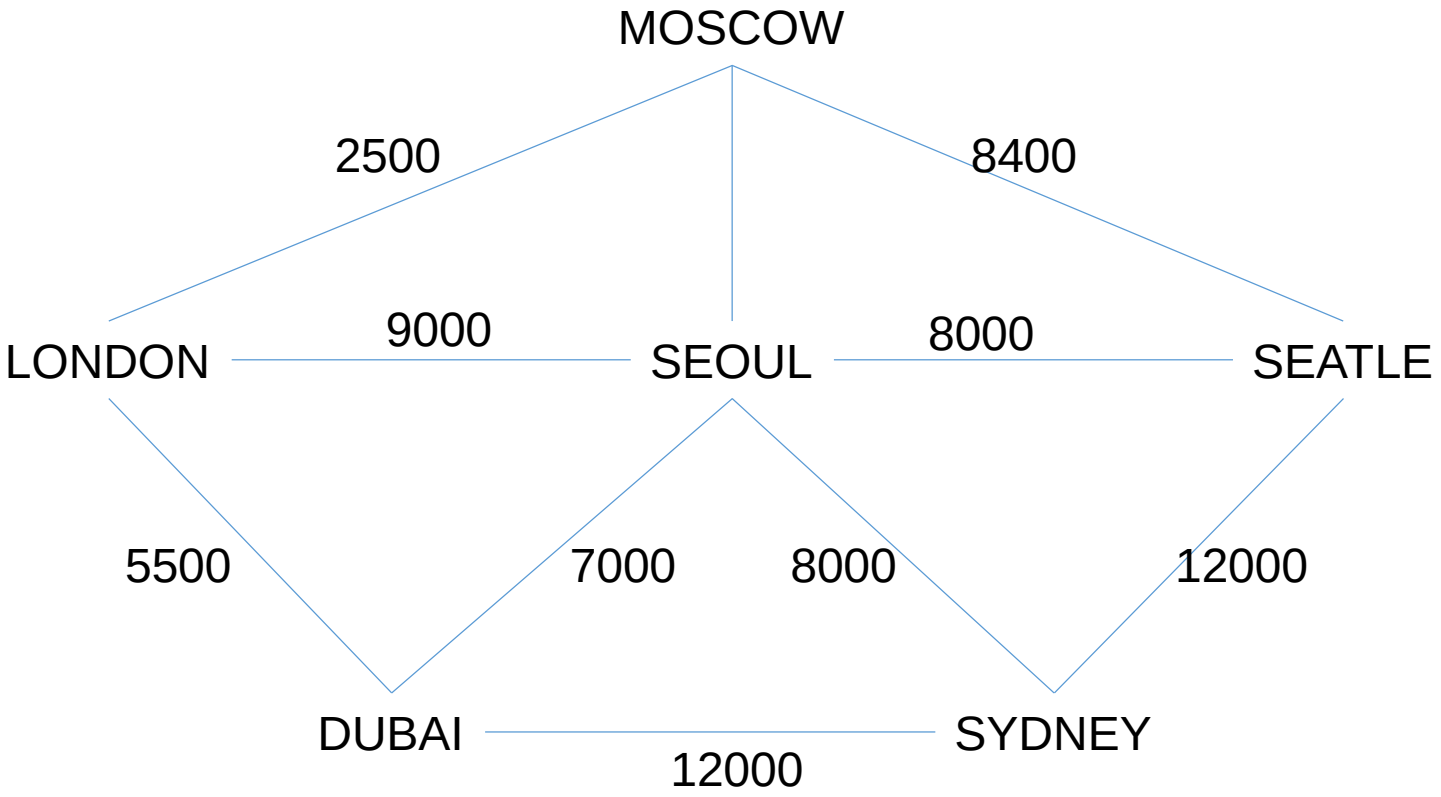
Example

- 아래 도시 네트워크를 그래프로 표현하세요.



HW10

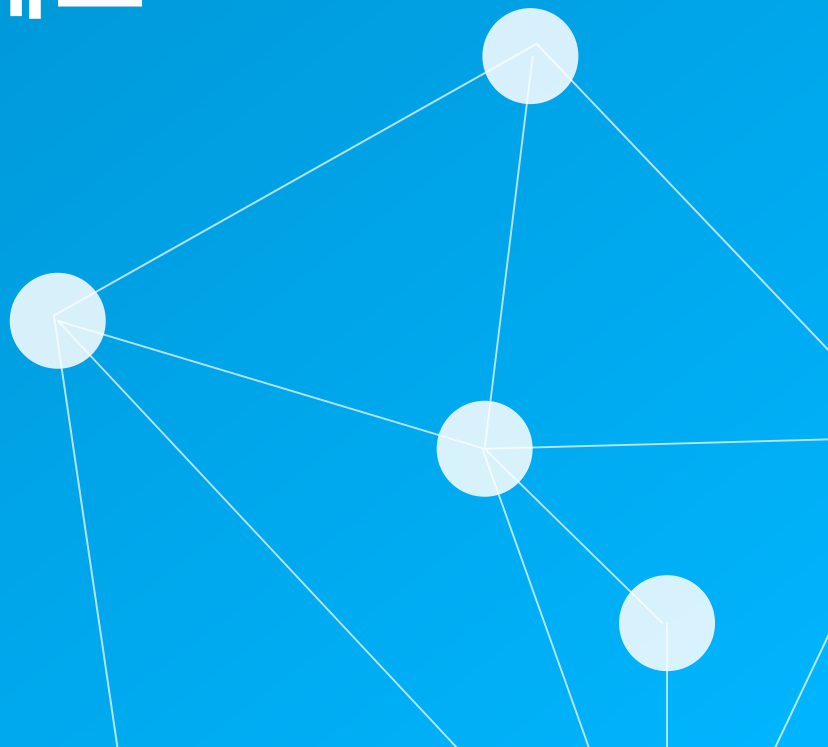
- STL을 적극 활용하여 오늘 배운 알고리즘을 구현하세요.



DATA
STRUCTURES
USING C++

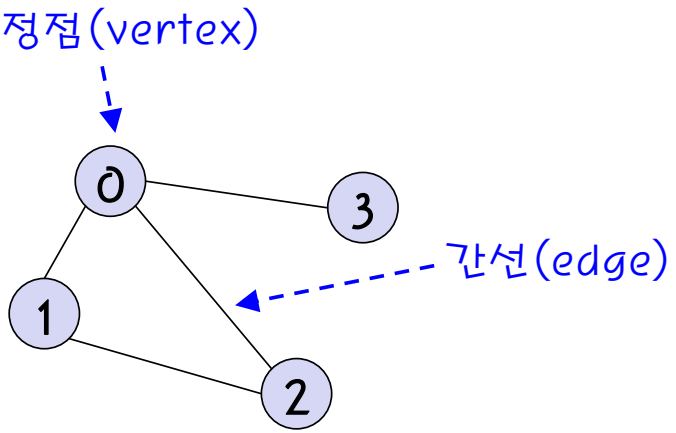
11 CHAPTER

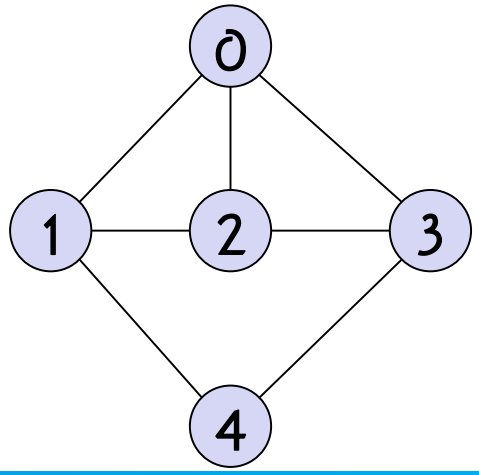
그래프

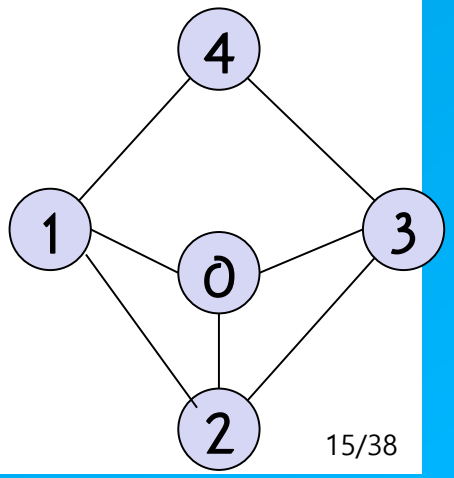


그래프 정의

- **그래프 G는 (V, E)로 표시**
  - **정점(vertices) 또는 노드(node)**
    - 여러 가지 특성을 가질 수 있는 객체 의미
    - V(G) : 그래프 G의 정점들의 집합
  - **간선(edge) 또는 링크(link)**
    - 정점들 간의 관계 의미
    - E(G) : 그래프 G의 간선들의 집합
  - 동일한 그래프?

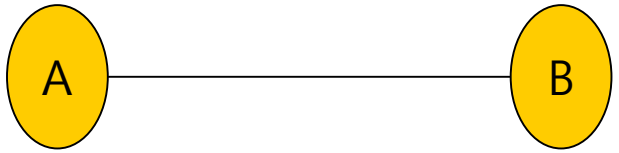




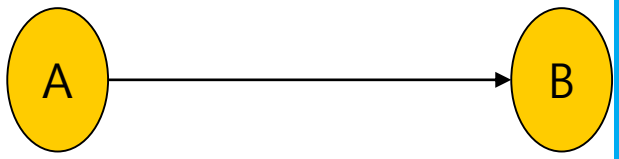


그래프 종류

- **간선의 종류에 따라**
  - **무방향 그래프(undirected graph)**
    - **간선을 통해서 양방향으로 갈수 있음**
    - (A, B)로 표현
    - (A, B) = (B, A)



  - **방향 그래프(directed graph)**
    - **간선을 통해서 한쪽 방향으로만 갈 수 있음**
    - 일방통행 길
    - <A, B> 로 표현
    - <A, B> ≠ <B, A>

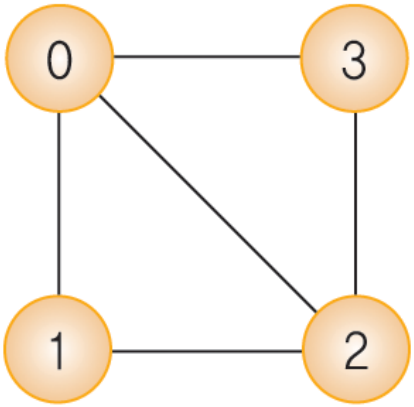


그래프 종류

- **가중치 그래프(weighted graph)**
  - 간선에 비용(cost)이나 가중치(weight)가 할당된 그래프
  - 가중치 그래프 예) 각 도시를 연결하는 도로망
    - 정점 : 각 도시를 의미
    - 간선 : 도시를 연결하는 도로 의미
    - **가중치 : 도로의 길이**

그래프 표현 예

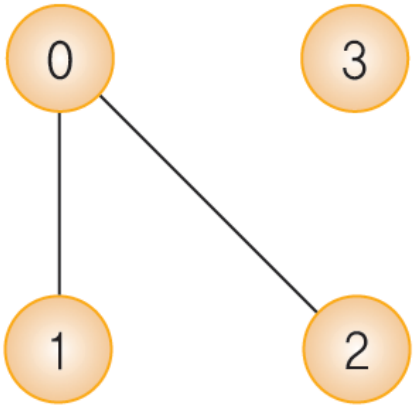
- 그래프 표현의 예



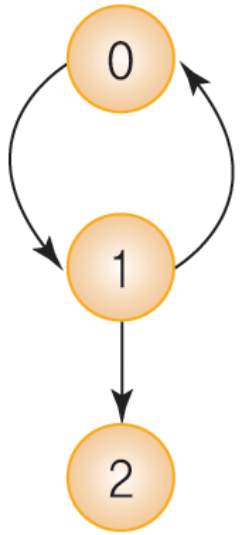
G1

V(G1)= {0, 1, 2, 3},
E(G1)= {(0, 1), (0, 2), (0, 3), (1, 2), (2, 3)}

V(G2)= {0, 1, 2, 3}
E(G2)= {(0, 1), (0, 2))}



G2



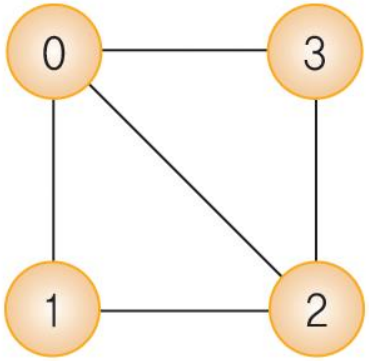
G3

V(G3)= {0, 1, 2}
E(G3)= {<0, 1>, <1, 0>, <1, 2>}

*directed graph의 경우

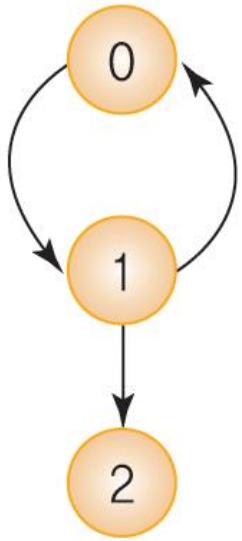
용어

- **인접 정점(adjacent vertex)**
  - **하나의 정점에서 간선에 의해 직접 연결된 정점**
  - G1에서 정점 0의 인접 정점
    - 정점 1, 정점 2, 정점 3



G1

- **차수(degree)**
  - **하나의 정점에 연결된 다른 정점의 수**
  - **무방향 그래프**
    - G1에서 정점 0의 차수: 3
  - **방향 그래프**
    - 진입차수 (in)
    - 진출차수 (out)
    - G2에서 정점 1의 진입차수 1, 진출차수 2



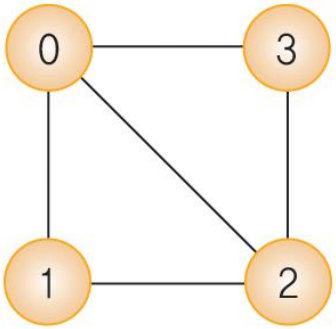
G3

용어

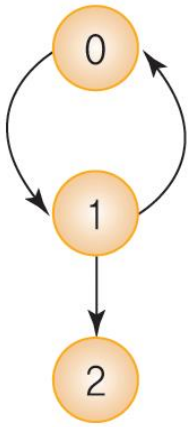
- **그래프의 경로(path)**
  - **무방향 그래프의 '정점 s'로부터 '정점 e'까지의 경로**
    - 정점의 나열 **s**, v1, v2, ..., vk, **e**
    - 반드시 간선 **(s, v1), (v1, v2), ... , (vk, e)** 존재해야 함
  - **방향 그래프의 '정점 s'로부터 '정점 e'까지의 경로**
    - 정점의 나열 **s**, v1, v2, ..., vk, **e**
    - 반드시 간선 **<s, v1>, <v1, v2>, ... ,<vk, e>** 존재해야 함
- **경로의 길이(length)**
  - 경로를 구성하는데 사용된 간선의 수

용어

- **단순 경로(simple path)**
  - 경로 중에서 반복되는 '간선'이 없는 경로
- **사이클(cycle)**
  - 시작 정점과 종료 정점이 동일한 **단순 경로**
- **예**
  - G1의1, 0, 2, 3은 단순경로
  - G1의 1, 0, 2, 0은 단순경로 아님
  - G1의 0, 1, 2, 0과 G3의 0, 1, 0은 사이클



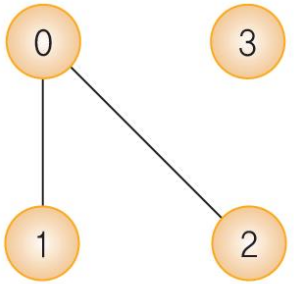
G1



G3

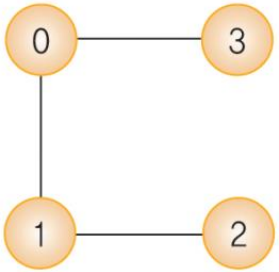
용어

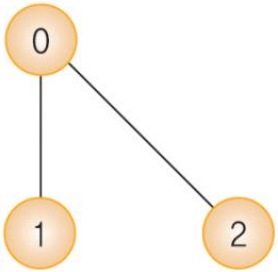
- **연결 그래프(connected graph)**
  - 모든 정점쌍에 대한 '경로' 존재
  - G2는 비연결 그래프임



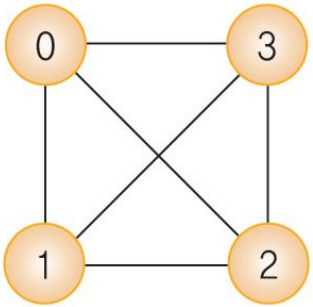
G2

- **트리(tree)**
  - 그래프의 특수한 형태로서 사이클을 가지지 않는 연결 그래프
  - 트리의 예





- **완전 그래프 (complete graph)**
  - 모든 정점이 직접 연결되어 있는 그래프

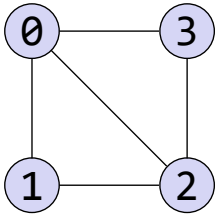


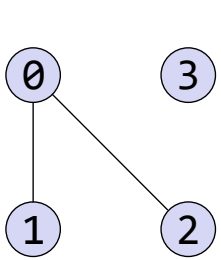
표현 방법

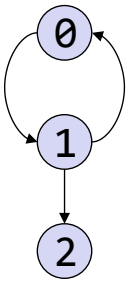
AI융합학부 안양준 yangjunahn@sungshin.ac.kr

그래프 표현 방법 1 : 인접 행렬

- **nxn 의 인접행렬 M을 이용**
  - if(간선 (i, j)가 그래프에 존재)
    - M[i][j] = 1
  - 그렇지 않으면
    - M[i][j] = 0
  - 대각선 성분은 모두 0
  - 무방향 그래프
    - 인접 행렬이 대칭







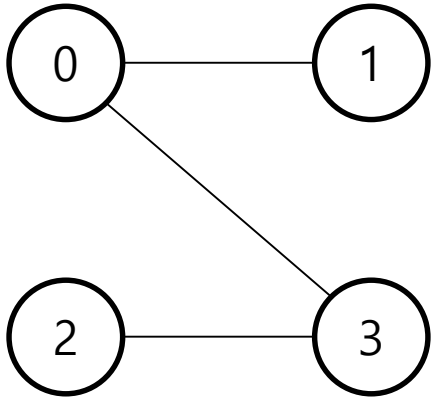
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

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

	0	1	2
0	0	1	0
1	1	0	1
2	0	0	0

예제

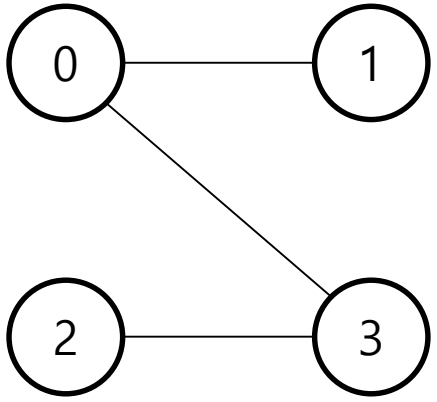
- **인접행렬을 M[i][j]을 이용하여 다음 그래프를 표현하세요.**
  - 다음 그래프는 if(간선 (i, j)가 그래프에 존재) M[i][j] = 1, 그렇지 않으면 M[i][j] = 0, 대각선 성분은 모두 0인 무방향 그래프입니다.



	0	1	2	3
0				
1				
2				
3				

예제 답

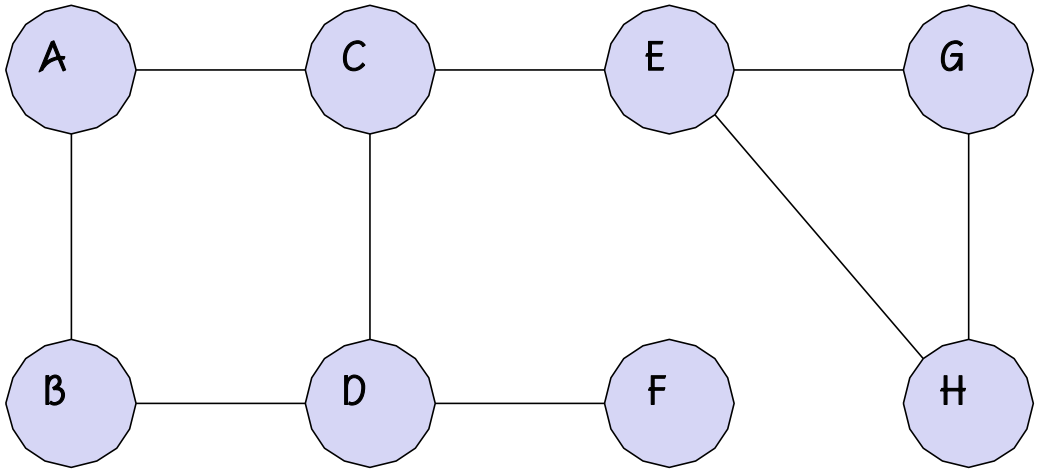
- **인접행렬을 M[i][j]을 이용하여 다음 그래프를 표현하세요.**
  - 다음 그래프는 if(간선 (i, j)가 그래프에 존재) M[i][j] = 1, 그렇지 않으면 M[i][j] = 0, 대각선 성분은 모두 0인 무방향 그래프입니다.



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

연습문제 1

- **인접행렬을 M[i][j]을 이용하여 다음 그래프를 표현하세요.**
  - 다음 그래프는 if(간선 (i, j)가 그래프에 존재) M[i][j] = 1, 그렇지 않으면 M[i][j] = 0, 대각선 성분은 모두 0인 무방향 그래프입니다.



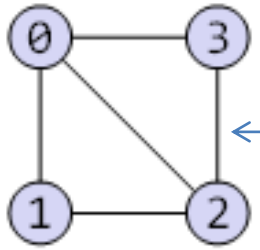
인접 행렬을 이용한 그래프 구현

- **클래스 다이어그램**
  - 변수 : 정점의 개수, 정점, 인접 링크

char vertices[]={'1','2','3','4',...}
adj[i][j] = 0 or 1

vertex
edge

AdjMatGraph
#size: *int*
#vertices: *char[]*
#adj: *int[][]*
+AdjMatGraph ()
+isEmpty(): *bool*
+isFull(): *bool*
+insertVertex(*char* v)
+deleteVertex(*int* v)
+insertEdge(*int* u, *int* v)
+deleteEdge(*int* u, *int* v)
+adjacent(*int* v): *int[]*
+display()



vertices[]

vertices[]에 저장하는 값

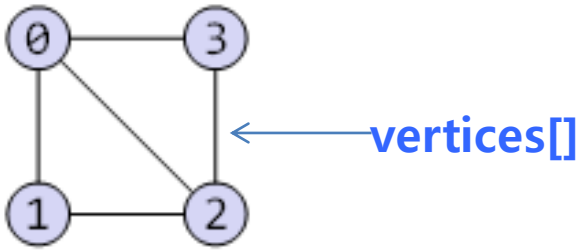
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

adj[][]

변수 vertices[] vs. adj[][]

예제 1

char vertices[]={'1','2','3','4',...}
adj[i][j] = 0 or 1



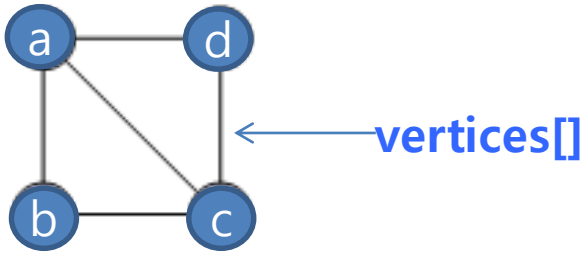
	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	1
3	1	0	1	0

vertices[]에 저장하는 값

adj[][]

예제 2

char vertices[]={'a','b','c','d',...}
adj[i][j] = 0 or 1



	a	b	c	d
a	0	1	1	1
b	1	0	1	0
c	1	1	0	1
d	1	0	1	0

vertices[]에 저장하는 값

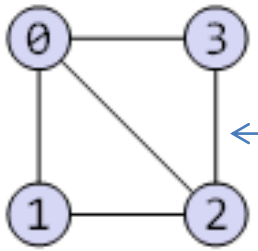
adj[][]

변수 vertices[] vs. adj[][]

예제 3

int vertices[]={0,1,2,3,...}
adj[i][j] = 0 or 1

vertices[]에 저장된 값의 사칙연산 등에 유용



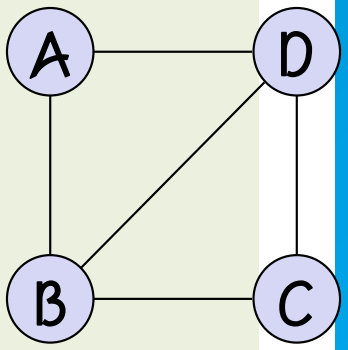
int vertices[]

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

vertices[]에 저장하는 값

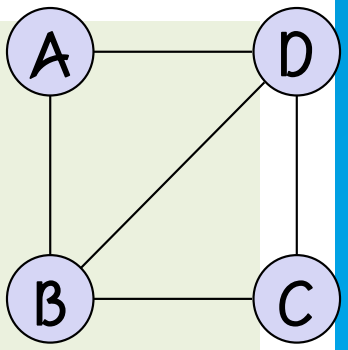
adj[][]

인접 행렬을 이용한 그래프 구현




```
#define MAX_VTXS    100

class AdjMatGraph {
protected:
     int    size;
     char   vertices[MAX_VTXS];
     int    adj[MAX_VTXS][MAX_VTXS];
public:
     AdjMatGraph( ) { reset(); }
     char getVertex(int i) { return vertices[i]; }
     int  getEdge(int i, int j) { return adj[i][j]; }
     void setEdge(int i, int j, int val) { adj[i][j] = val; }
     bool isEmpty()       { return size==0; }
     bool isFull()        { return size>=MAX_VTXS; }

     // 그래프 초기화 ==> 공백 상태의 그래프
     void reset() {
        size=0;
        for(int i=0 ; i<MAX_VTXS ; i++ )
        for(int j=0 ; j<MAX_VTXS ; j++ )
            setEdge(i,j,0);
    }
```

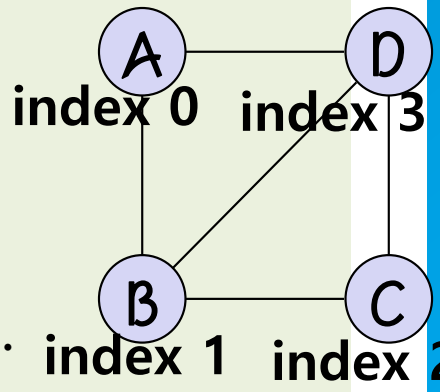
예) **char vertices[]={'A','B','C','D',...}**
adj[i][j] = 0 or 1




```
    // 정점 삽입
    void insertVertex( char name ) {
        if( !isFull() ) vertices[size++] = name;
        else printf("Error: 그래프 정점 개수 초과\n");
    }
    // 간선 삽입: 무방향 그래프의 경우임.
    void insertEdge( int u, int v ) {
        setEdge(u,v,1);
        setEdge(v,u,1);
    }
    // 그래프 정보 출력 (화면이나 파일에 출력)
    void display( FILE *fp = stdout) {
        fprintf(fp, "%d\n", size);                  // 정점의 개수 출력
        for( int i=0 ; i<size ; i++ ) {             // 각 행의 정보 출력
            fprintf(fp,"%c ", getVertex(i));        // 정점의 이름 출력
            for( int j=0 ; j<size ; j++ )           // 간선 정보 출력
                fprintf(fp, " %3d", getEdge(i,j));
            fprintf(fp,"\n");
        }
    }
};
```

char vertices[]={'A','B','C','D'}
adj[i][j] = 0 or 1

```
// class AdjMatGraph

void main()
{
    AdjMatGraph g;                  // 새로운 그래프 객체 생성

    for( int i=0 ; i<4 ; i++ )      // 정점 삽입: 'A' 'B', ...
        g.insertVertex('A'+i);      // size=4

    g.insertEdge(0,1);              // 간선 삽입
    g.insertEdge(0,3);
    g.insertEdge(1,2);
    g.insertEdge(1,3);
    g.insertEdge(2,3);

    printf("인접 행렬로 표현한 그래프\n");
    g.display();
}
```



Microsoft Visual Studio 디버그 콘솔

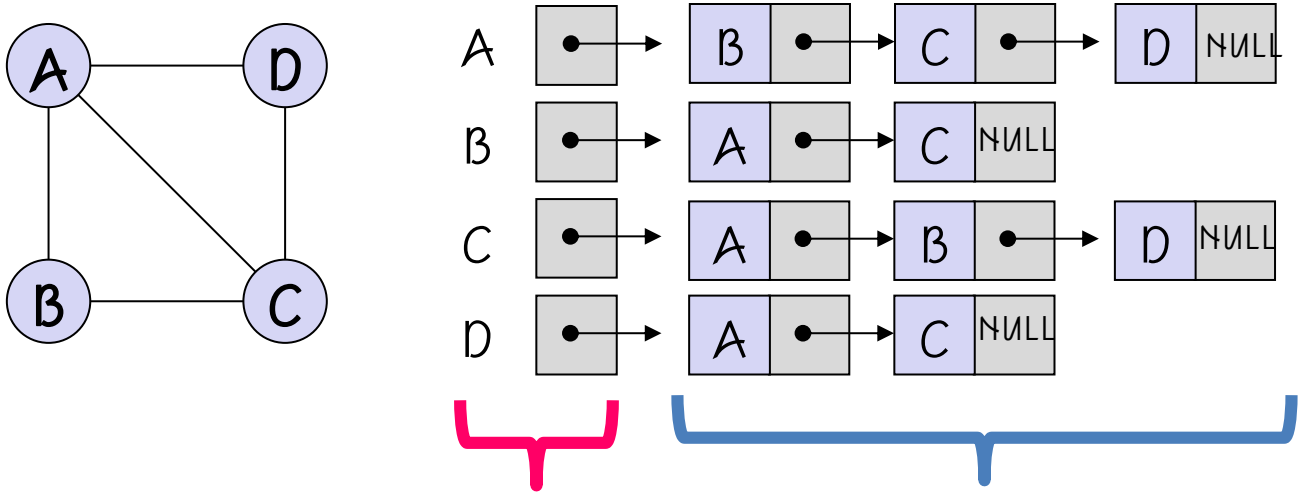
```
인접 행렬로 표현한 그래프
4
A 0 1 0 1
B 1 0 1 1
C 0 1 0 1
D 1 1 1 0
```

표현 방법 Linked List

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

그래프 표현 방법 2 : 인접 리스트

- **인접 리스트(adjacent list)**
  - 각 정점이 연결 리스트를 가짐
  - 인접한 정점들을 연결 리스트로 표현



포인터 변수

노드 타입을 가리키는 포인터 변수

Node* p

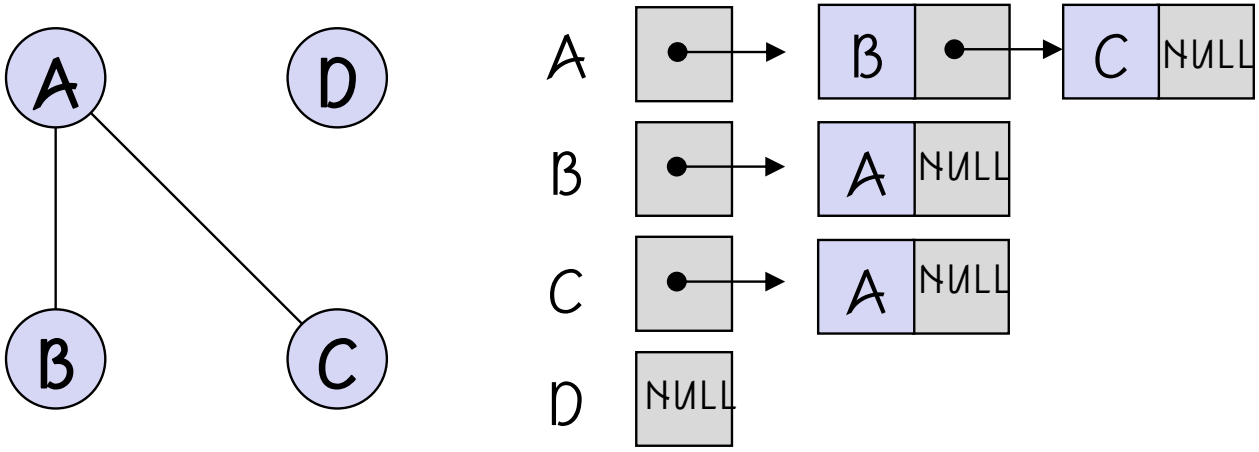
노드 연결 리스트

class Node

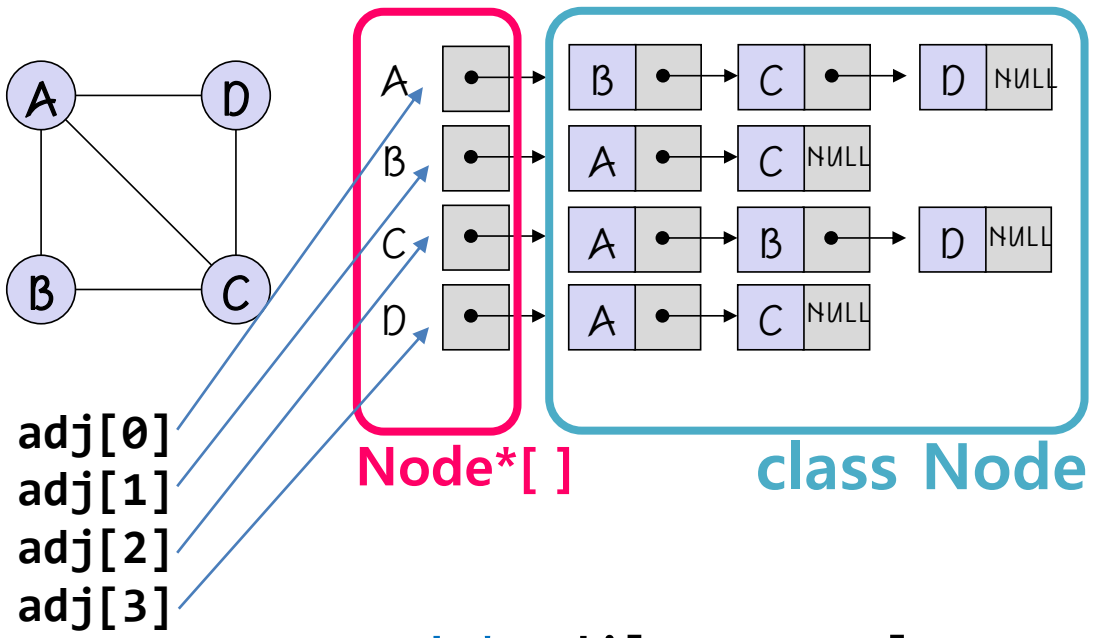
Node nd

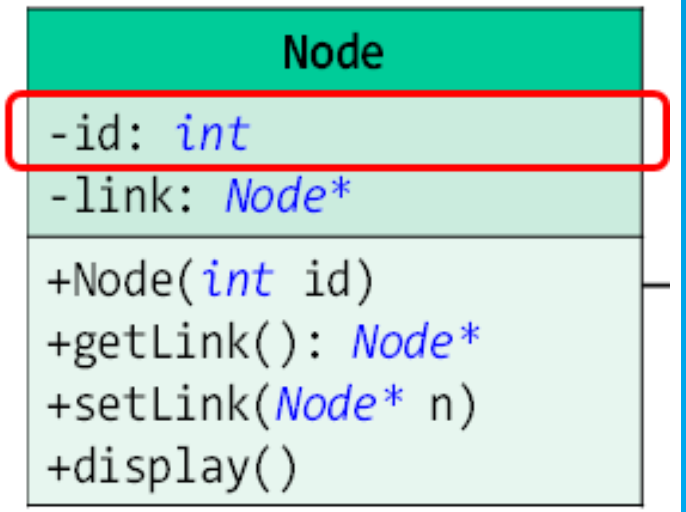
그래프 표현 방법 2 : 인접 리스트

- 인접 리스트(adjacent list) 예제



class Node






```
Node* adj[MAX_VTXS];
```

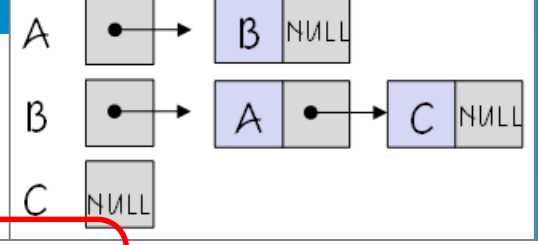
adj[i] = NULL; // 초기화

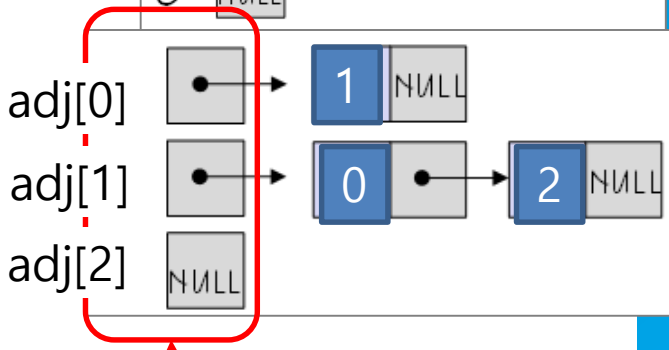
adj[0] = new Node(B);

class AdjListGraph

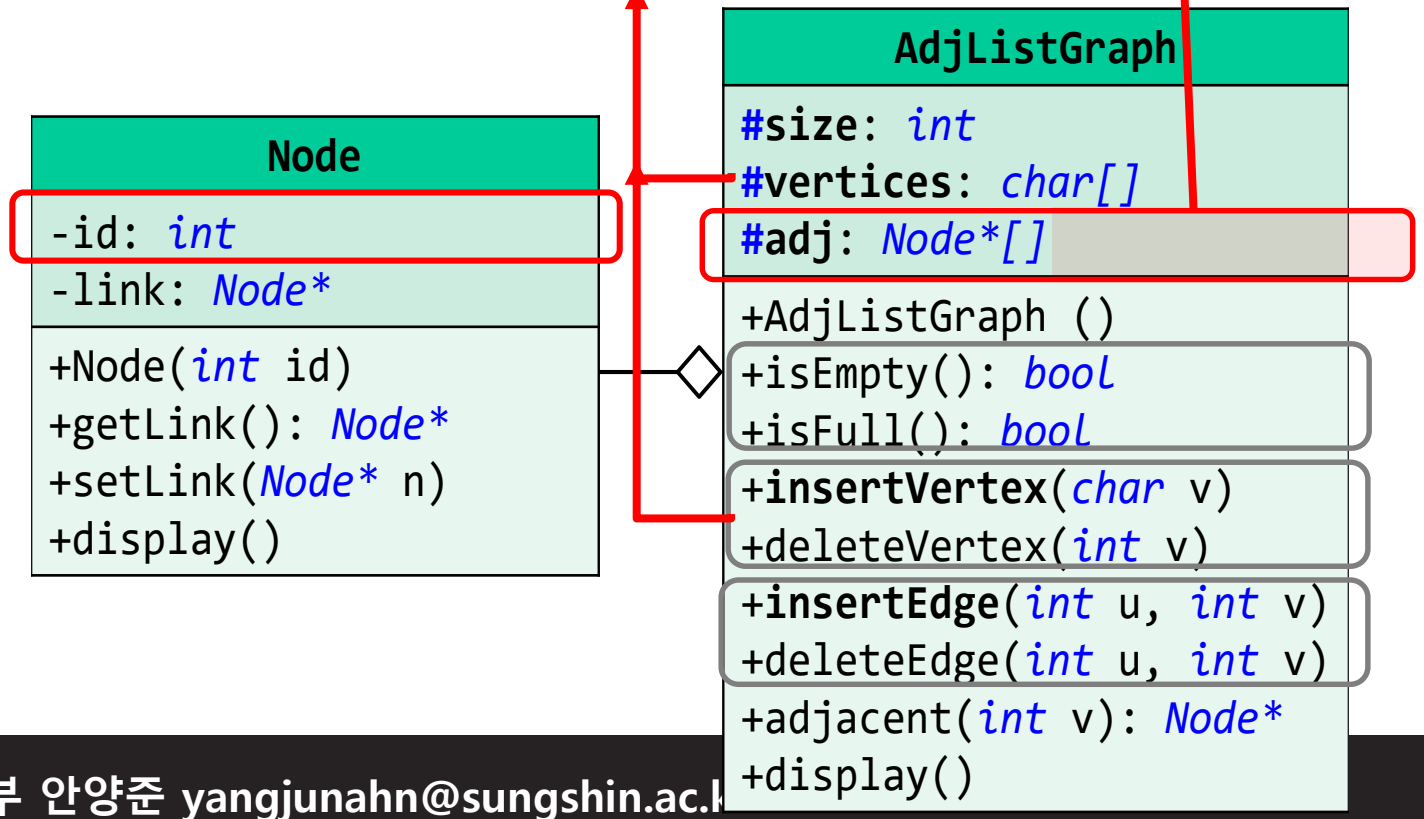
멤버 변수

int size // 그래프의 노드 개수
char vertices[] // 노드 정보
Node* adj[]



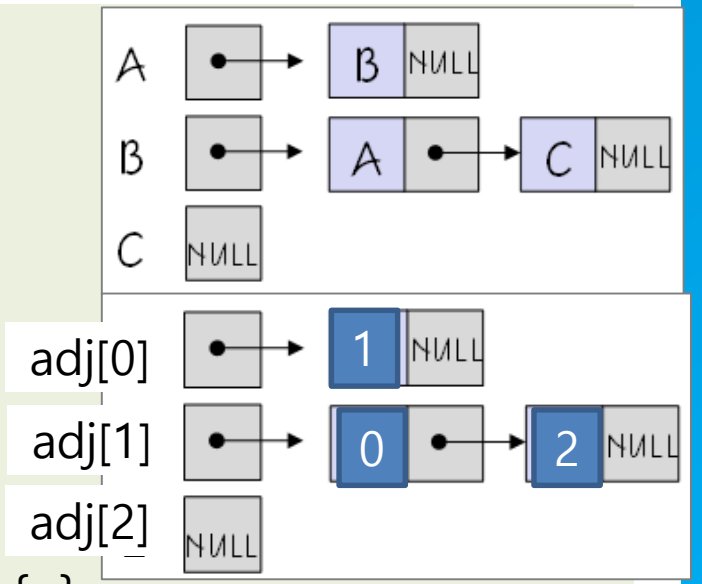


char vertices[]={'A','B','C','D'}



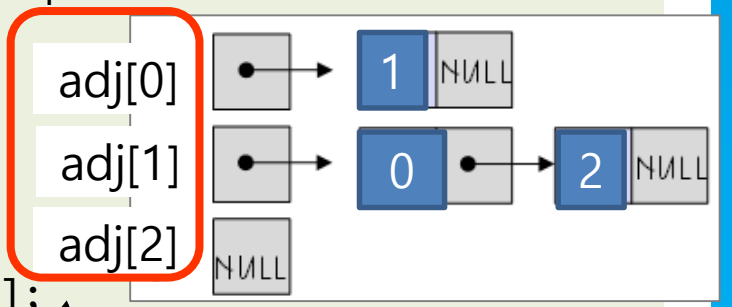
class AdjListGraph

```
// 인접 리스트를 이용한 그래프: 노드 클래스

#define MAX_VTXS 100


class Node {
protected:
    int id;       // 정점의 id
    Node* link;   // 다음 노드의 포인터
public:
    Node(int i, Node *l=NULL) : id(i), link(l) { }
    ~Node() {
        if( link != NULL ) delete link;
    }
    int   getId() { return id; }
    Node* getLink() { return link; }
    void  setLink(Node* l) { link = l; }
};
```

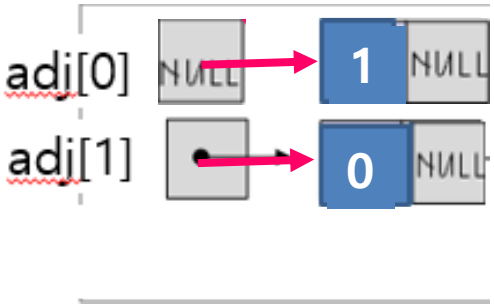


class AdjListGraph

```
// 연결 리스트를 위한 노드 그래프 클래스 포함
class AdjListGraph {
protected:
    int size;                  // 정점의 개수
    char vertices[MAX_VTXS]; // 정점 정보
    Node* adj[MAX_VTXS];       // 각 정점의 인접 리스트
public:
    AdjListGraph() : size(0) { }
    ~AdjListGraph(){ reset(); }
    void reset(void) {
        for( int i=0 ; i<size ; i++ )
            if( adj[i] != NULL ) delete adj[i];
        size = 0;
    }
    void insertVertex( char val ) {
        if( !isFull() ) {
            vertices[size] = val;
            adj[size++] = NULL;
        }
        else printf("Error: 그래프 정점 개수 초과\n");
    }
```



char vertices[]={'A','B','C','D'}



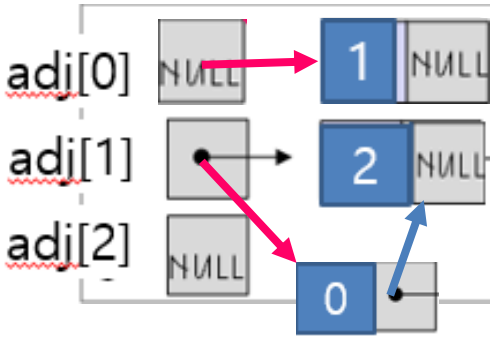
adj[0]=adj[1]=NULL

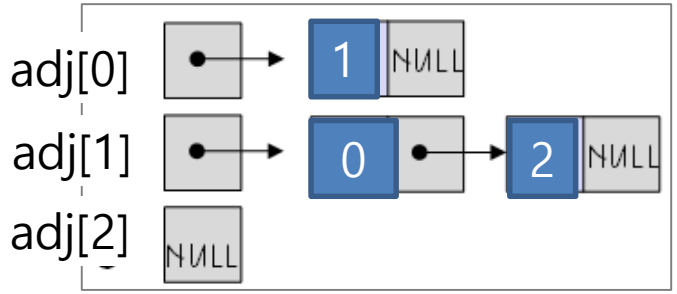
노드 1를 만들어서 adj[0]에 연결

노드 0를 만들어서 adj[1]에 연결

```
                                  0     1
void insertEdge( int u, int v) {     // 간선 삽입 연산
    adj[u] = new Node (v, adj[u]); // 인접 리스트에 추가
    adj[v] = new Node (u, adj[v]); // 방향 그래프 ==> 주석 처리함
}
void display( ) {
    printf("%d\n", size);                  // 정점의 개수 출력
    for( int i=0 ; i<size ; i++ ) {        // 각 행의 정보 출력
      printf("%c ", getVertex(i));           // 정점의 이름 출력
      for( Node *v=adj[i] ; v != NULL ; v=v->getLink() )
        printf(" %c", getVertex(v->getId()) );
      printf("\n");
    }
}
Node* adjacent(int v) { return adj[v]; }
```

int





노드 1를 만들어서 adj[0]에 연결

0 1

```
void insertEdge( int u, int v) {    // 간선 삽입 연산
    adj[u] = new Node (v, adj[u]); // 인접 리스트에 추가
    adj[v] = new Node (u, adj[v]); // 방향 그래프 ==> 주석 처리함
}
void display( ) {
    printf("%d\n", size);                  // 정점의 개수 출력
    for( int i=0 ; i<size ; i++ ) {        // 각 행의 정보 출력
      printf("%c ", getVertex(i));           // 정점의 이름 출력
      for( Node *v=adj[i]; v != NULL; v=v->getLink() )
        printf(" %c", getVertex(v->getId()) );
      printf("\n");
    }
}
Node* adjacent(int v) { return adj[v]; }
```

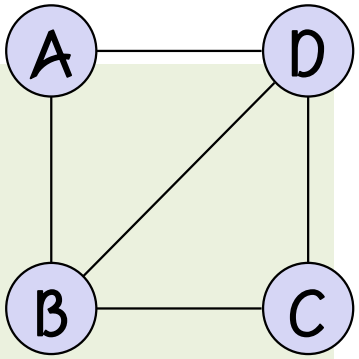
노드 0를 만들어서 adj[1]에 연결

int

class AdjListGraph

```
    bool isEmpty() { return size ==0; }
    bool isFull()  { return size >= MAX_VTXS; }
    char getVertex( int i ) {return vertices[i]; }

};  // end of class AdjListGraph
```

그래프 입출력 테스트




```
// class Node
// class AdjListGraph

void main()
{
    AdjListGraph g;                 // 새로운 그래프 객체 생성

    for( int i=0 ; i<4 ; i++ )
        g.insertVertex('A'+i);      // 정점 삽입: 'A' 'B', ...

    g.insertEdge(0,1); //A,B        // 간선 삽입
    g.insertEdge(0,3); //A,D
    g.insertEdge(1,2); //B,C
    g.insertEdge(1,3); //B,D
    g.insertEdge(2,3); //C,D
    printf("인접 리스트로 표현한 그래프\n");
    g.display();
}
```

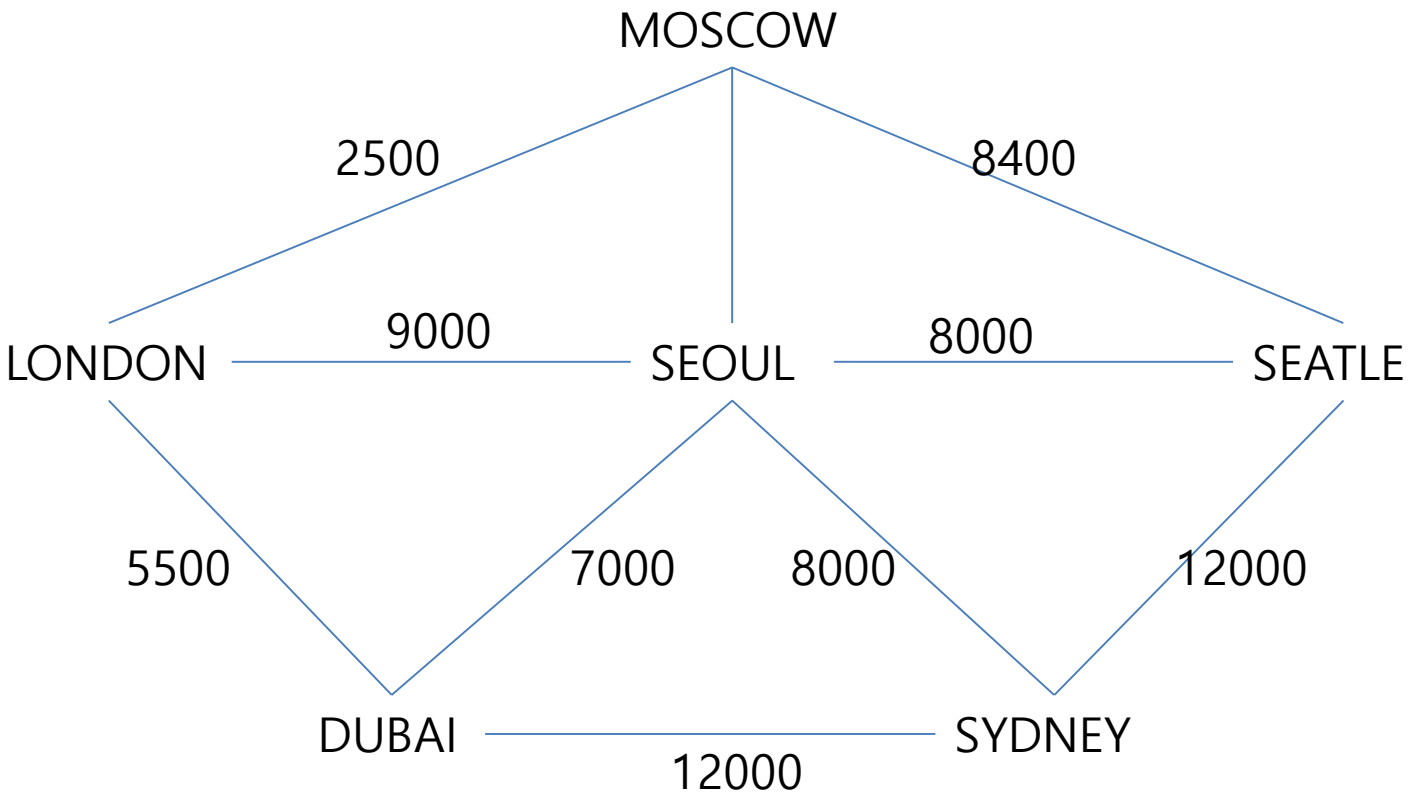
class Node
char vertices[]={'A','B','C','D'}

```
연결 리스트로 표현한 그래프
Number of vertex : 4
A D B
B D C A
C D B
D C B A
```

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

Example

- 아래 도시 네트워크를 그래프로 표현하세요.

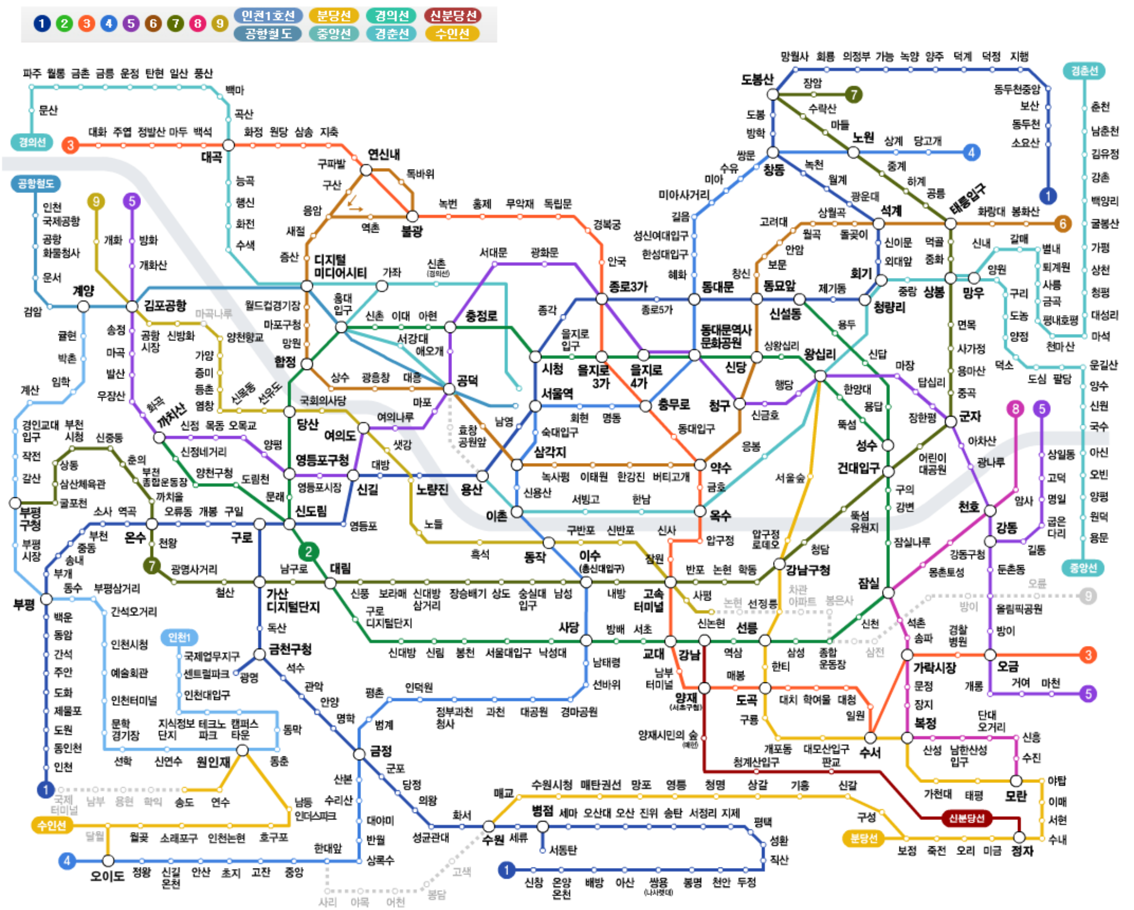


그래프 탐색

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

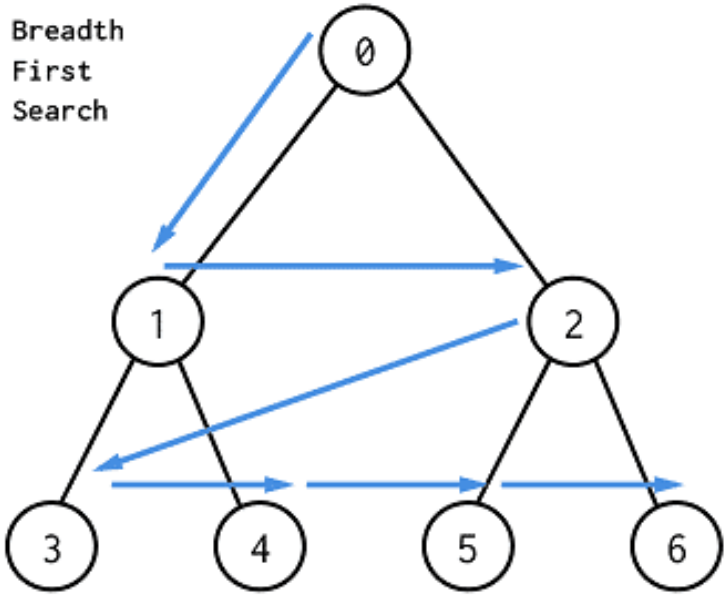
그래프 탐색

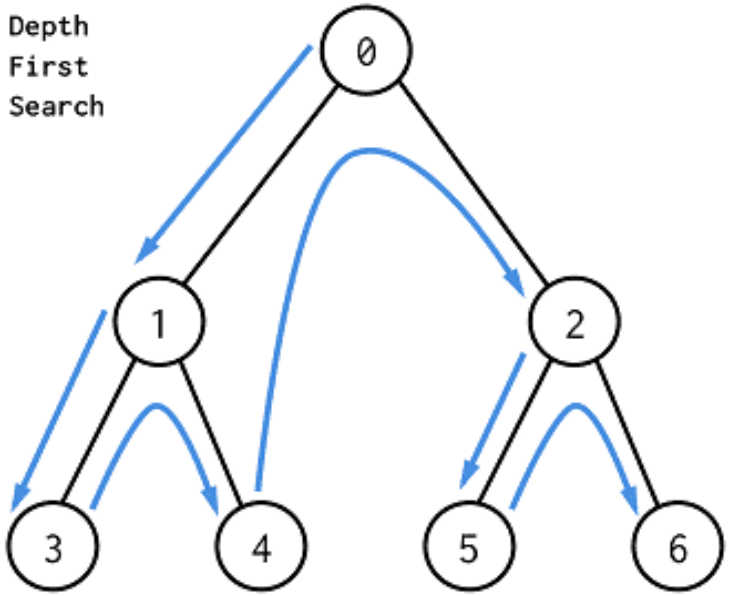
- 그래프의 가장 기본적인 연산
  - 시작 정점부터 차례대로 모든 정점들을 한 번씩 방문



탐색 방법

- **DFS vs. BFS**
  - 깊이 우선 탐색 (DFS)
  - 너비 우선 탐색 (BFS)



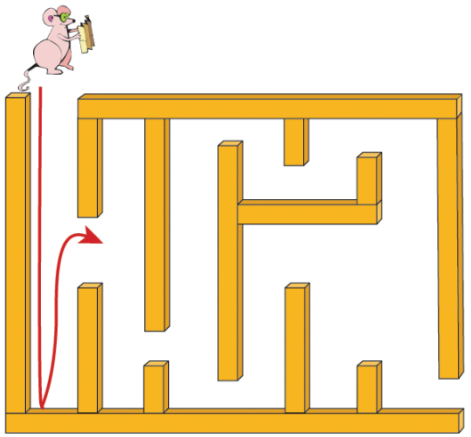


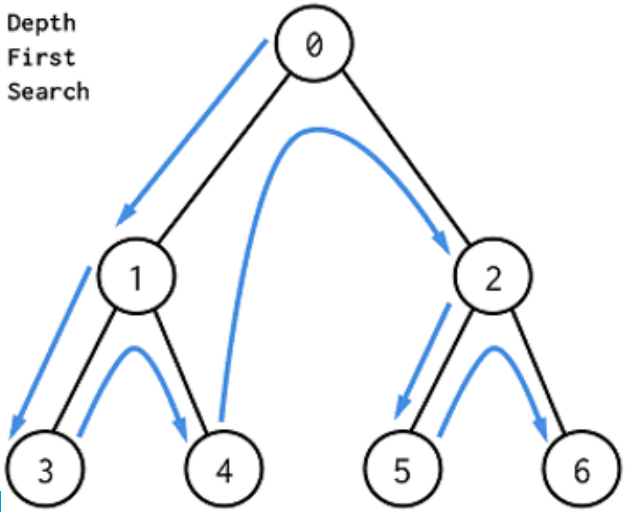
DFS

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

깊이 우선 탐색 (DFS)

- **DFS: depth-first search**
  - 한 방향으로 갈 수 있을 때까지 가다가 더 이상 갈 수 없게 되면 **가장 가까운 갈림길로 돌아와서** 이 곳으로부터 다른 방향으로 다시 탐색 진행
  - 되돌아가기 위해서는 **스택** 필요
    - **순환함수** 호출로 내부적으로 스택 이용

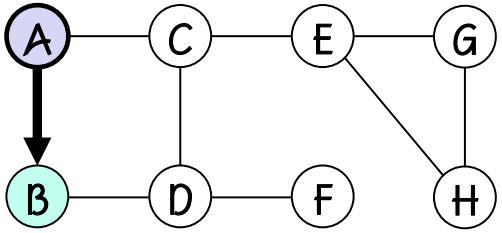




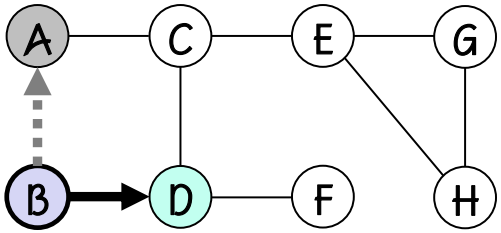

```
depthFirstSearch(v)
    v를 방문되었다고 표시;
    for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면)
            then depthFirstSearch(u)
```

DFS 알고리즘

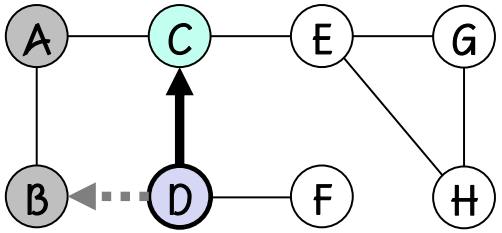
*경로가 여러 개일 때,
알파벳 순 또는 작은 노드 먼저 방문



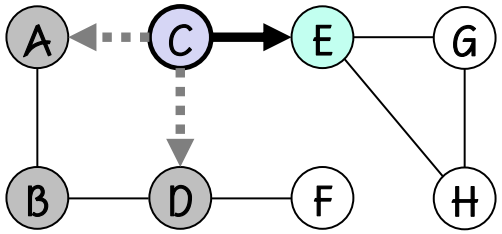
(a) A에서 시작: A->B



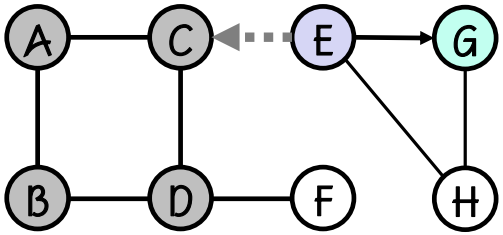
(b) B->D (A는 방문했음)



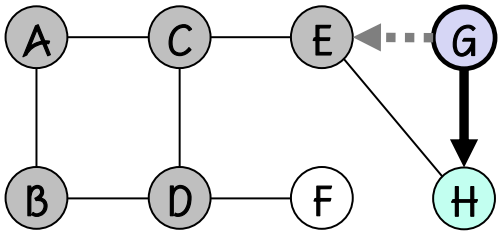
(c) D->C (B는 방문했음)



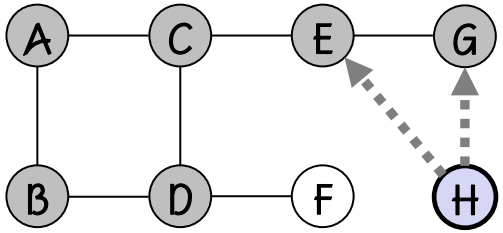
(d) C->E (A,D는 방문했음)



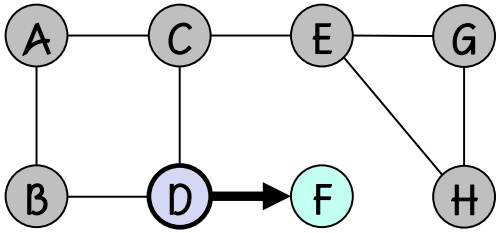
(e) E->G (C는 방문했음)



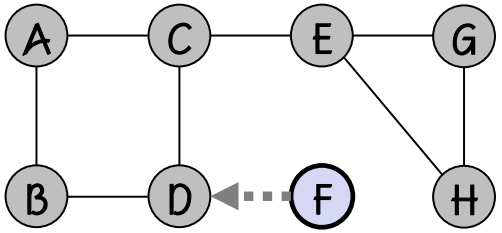
(f) G->H (E는 방문했음)



(g) H에서는 모두 방문 했음.
G,E,C,D순으로 되돌아 감.
D에서는 가지 않은 F가 있음.



(h) D->F



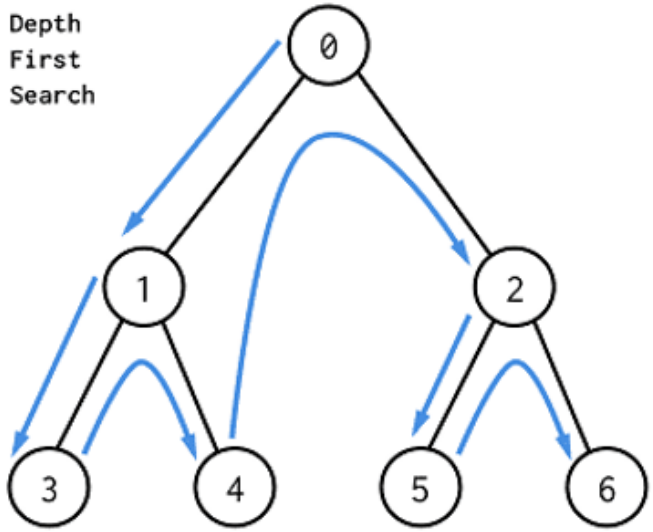
(i) F에서도 모두 방문 했음.
D,B,A순으로 되돌아 감.
탐색 완료.
방문 순서: ABDCEGHF

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

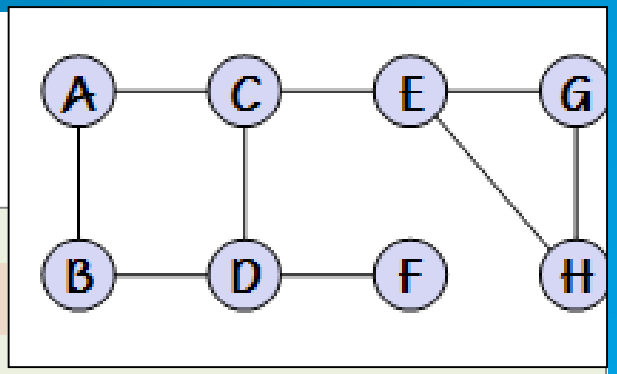
DFS algorithm

- **DFS: depth-first search**

```
depthFirstSearch(v)
    v를 방문되었다고 표시;
    for all u ∈ (v에 인접한 정점) do
        if (u가 아직 방문되지 않았으면)
            then depthFirstSearch(u)
```



DFS 구현 (인접 행렬)

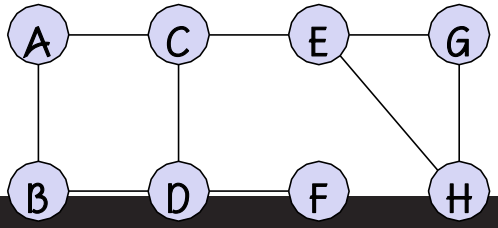



```
// 탐색 기능이 추가된 인접행렬 기반 그래프 클래스
class SrchAMGraph : public AdjMatGraph
{
    bool visited[MAX_VTXS];        // 정점의 방문 정보
public:
    void resetVisited() {          // 모든 정점을 방문하지 않았다고 초기화
        for( int i=0 ; i<size ; i++ )
            visited[i] = false;
    }
    bool isLinked(int u, int v) { return getEdge(u,v) != 0; }

    // 깊이 우선 탐색 함수
    void DFS( int v) {
        visited[v] = true;                  // 현재 정점을 방문함
        printf("%c ", getVertex(v));        // 정점의 이름 출력

        for( int w=0 ; w<size ; w++)
            if( isLinked(v,w) && visited[w]==false )
                DFS( w );                   // 연결 + 방문X => 순환호출
    }
};
```

DFS 구현 (인접 행렬)

```
// 깊이 우선 탐색 테스트 프로그램
// class SrchAMGraph
void main()
{
    SrchAMGraph g;                  // DFS 탐색기능이 있는 그래프 객체 생성
    g.load( "graph.txt" );          // 파일 "graph.txt"로부터 g를 설정함
    printf("그래프(graph.txt)\n");
    g.display();                    // 인접행렬을 화면에 출력

    printf("DFS ==> ");
    g.resetVisited();               // 모든 정점을 방문하지 않은 것으로 초기화
    g.DFS( 0 );                     // 0번째 정점(A)에서 깊이 우선 탐색 시작
    printf("\n");
}
```

AdjMatGraph 클래스



graph.txt

```
8
A 0 1 1 0 0 0 0 0
B 1 0 0 1 0 0 0 0
C 1 0 0 1 1 0 0 0
D 0 1 1 0 0 1 0 0
E 0 0 1 0 0 0 1 1
F 0 0 0 1 0 0 0 0
G 0 0 0 0 1 0 0 1
H 0 0 0 0 1 0 1 0
```

Microsoft Visual Studio 디버그 콘솔

```
그래프(grap.txt)
8
A 0 1 1 0 0 0 0 0
B 1 0 0 1 0 0 0 0
C 1 0 0 1 1 0 0 0
D 0 1 1 0 0 1 0 0
E 0 0 1 0 0 0 1 1
F 0 0 0 1 0 0 0 0
G 0 0 0 0 1 0 0 1
H 0 0 0 0 1 0 1 0
DFS ==> ABDCEGHF
```

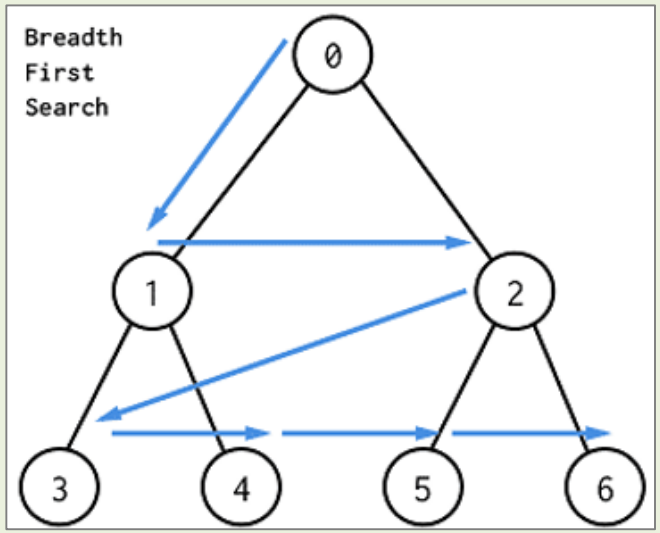
BFS

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

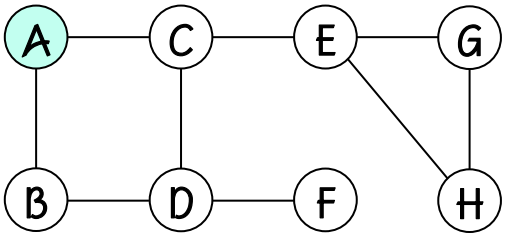
BFS

- **BFS: breadth-first search**
  - 시작 정점으로부터 가까운 정점을 먼저 방문하고
    멀리 떨어져 있는 정점을 나중에 방문하는 순회 방법
  - **Queue를 사용하여 구현**

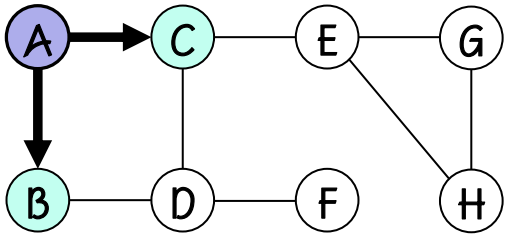
```
breadthFirstSearch(v)
    v를 방문되었다고 표시;
    큐 Q에 정점 v를 삽입;
    while (not is_empty(Q)) do
        Q에서 정점 w를 삭제하고 꺼낸다;
        for all u ∈ (w에 인접한 정점) do
            if (u가 아직 방문되지 않았으면)
                then    u를 큐에 삽입;
                        u를 방문되었다고 표시;
```



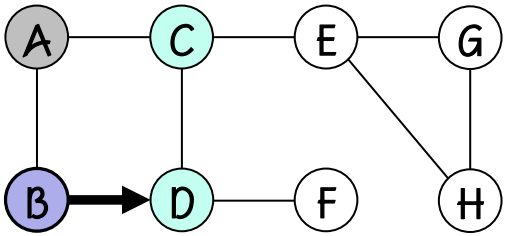
BFS 알고리즘



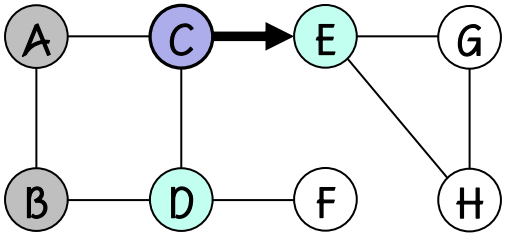
(a) A에서 시작.
큐 내용: A



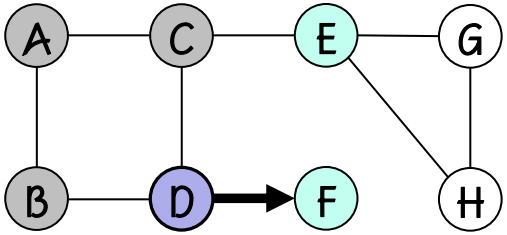
(b) A->B,C
큐 내용: BC



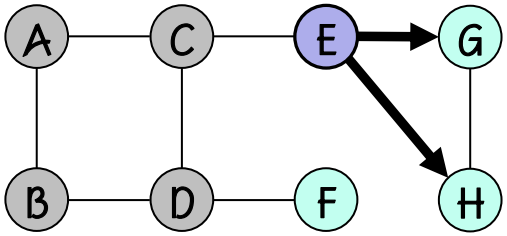
(c) B->D
큐 내용: CD



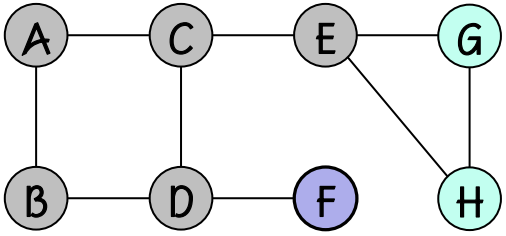
(d) C->E
큐 내용: DE



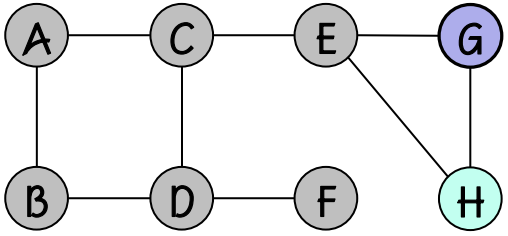
(e) D->F
큐 내용: EF



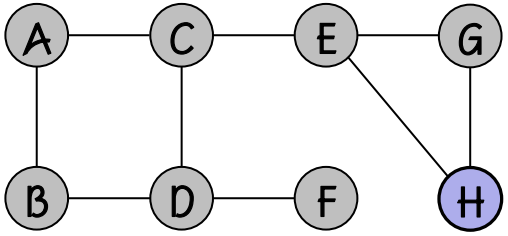
(f) E->G,H
큐 내용: FGH



(g) F에서는 모두 방문 했음.
큐 내용: GH



(h) G에서는 모두 방문 했음.
큐 내용: H

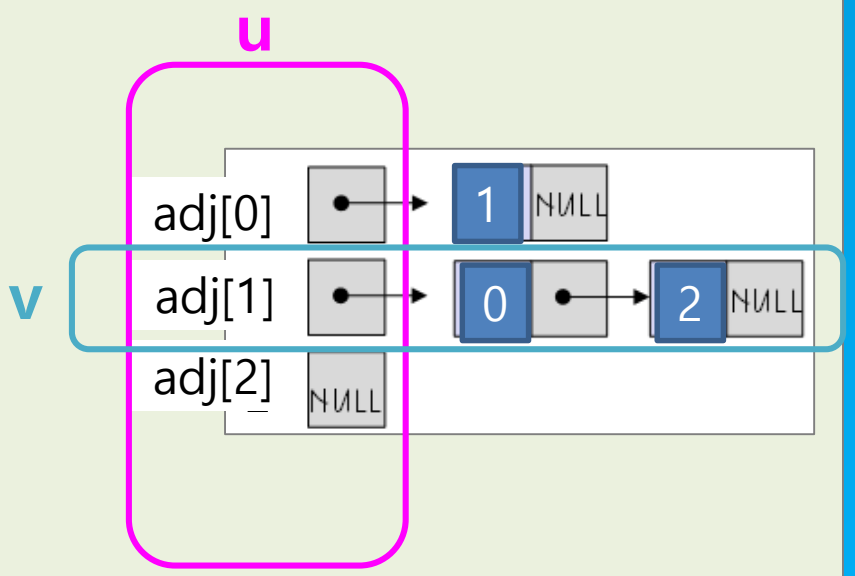


(i) H에서도 모두 방문 했음.
큐 공백상태 -> 탐색 완료.
방문 순서: ABCDEFGH

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

BFS 구현 (인접 리스트)

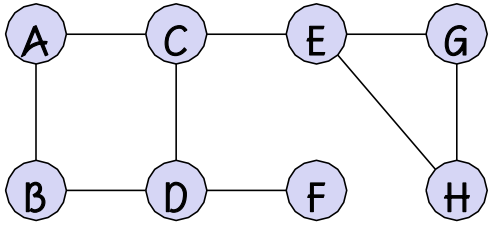
```
// 탐색 기능이 추가된 인접 행렬 기반 그래프 클래스
// class AjdListGraph
// class CircularQueue

class SrchALGraph : public AdjListGraph
{
   bool visited[MAX_VTXS];
public:
   void resetVisited()
   {
            for (int i=0; i<size; i++)
               visited[i] = false;
   }

   bool isLinked(int u, int v)
   {
            for ( Node *p = adj[u]; p!=NULL; p=p->getLink())
               if (p->getId() == v) return true;
            return false;
   }
```



BFS 구현 (인접 리스트)

```
// 탐색 기능이 추가된 인접 리스트 기반 그래프 클래스
// class AjdListGraph
// class CircularQueue
void BFS(int v) {
        visited[v] = true; // 현재 정점을 방문함
        printf("%c ", getVertex(v)); // 정점의 이름 출력

        CircularQueue que;
        que.enqueue(v); // 시작 정점을 큐에 저장
        while (!que.isEmpty()) {
            int v = que.dequeue();
            for (Node* w = adj[v]; w != NULL; w = w->getLink() )
            {
                int id = w->getId();
                if (!visited[id]) {
                    visited[id] = true;
                    printf("%c ", getVertex(id));
                    que.enqueue(id);
                }
            }
        }
    }
```

main.cpp

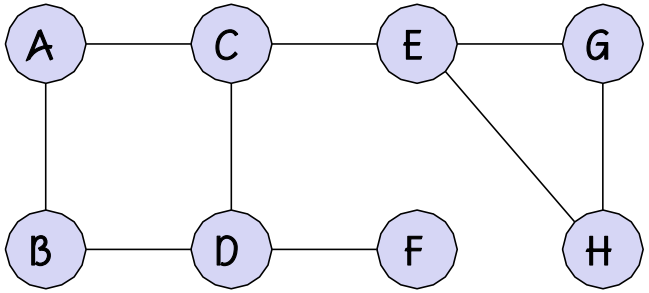


graph.txt

```
8
A  0 1 1 0 0 0 0 0
B  1 0 0 1 0 0 0 0
C  1 0 0 1 1 0 0 0
D  0 1 1 0 0 1 0 0
E  0 0 1 0 0 0 1 1
F  0 0 0 1 0 0 0 0
G  0 0 0 0 1 0 0 1
H  0 0 0 0 1 0 1 0
```

```
// class SrchALGraph

void main()
{
        SrchALGraph g;

        g.load( "graph.txt" );   // 그래프 데이터 읽기
        printf("그래프(graph.txt)₩n");
        g.display();

        printf("BFS ==> ");
        g.resetVisited();
        g.BFS( 0 );  // vertex A
        printf("₩n");
}
```

test case #1



데이터 파일 *graph.txt*

```
8
A    0  1  1  0  0  0  0  0
B    1  0  0  1  0  0  0  0
C    1  0  0  1  1  0  0  0
D    0  1  1  0  0  1  0  0
E    0  0  1  0  0  0  1  1
F    0  0  0  1  0  0  0  0
G    0  0  0  0  1  0  0  1
H    0  0  0  0  1  0  1  0
```

실행 결과

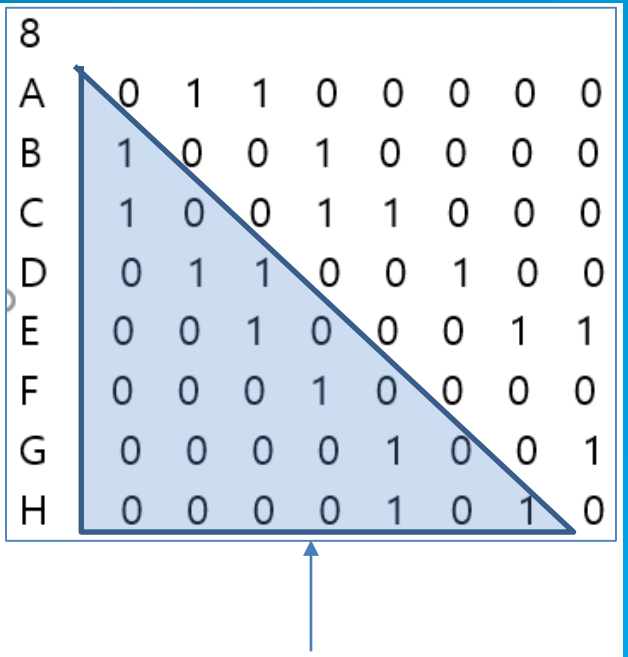
```
그래프(graph.txt)
Number of vertex : 8
A C B
B D A
C E D A
D F C B
E H G C
F D
G H E
H G E
BFS ==> A C B E D H G F
```

데이터 파일 읽기

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

Loading Data File

```
// class AdjMatGraph
// class AdjListGraph의 멤버 함수로 추가
 void load(char* filename) {
        ifstream fp(filename);
        if (fp.is_open())
        {
            int n, val;
            fp >> n;
            for (int i = 0; i < n; i++) {    // number of vertices
                char str[80];
                fp >> str;                   // vertex
                insertVertex(str[0]);
                for (int j = 0; j < n; j++) {
                    fp >> val;               // edge
                    if (i>j && val != 0) insertEdge(i, j);
                }
            }
        }
        else cout << "File can not be found !" << endl;
    }
```

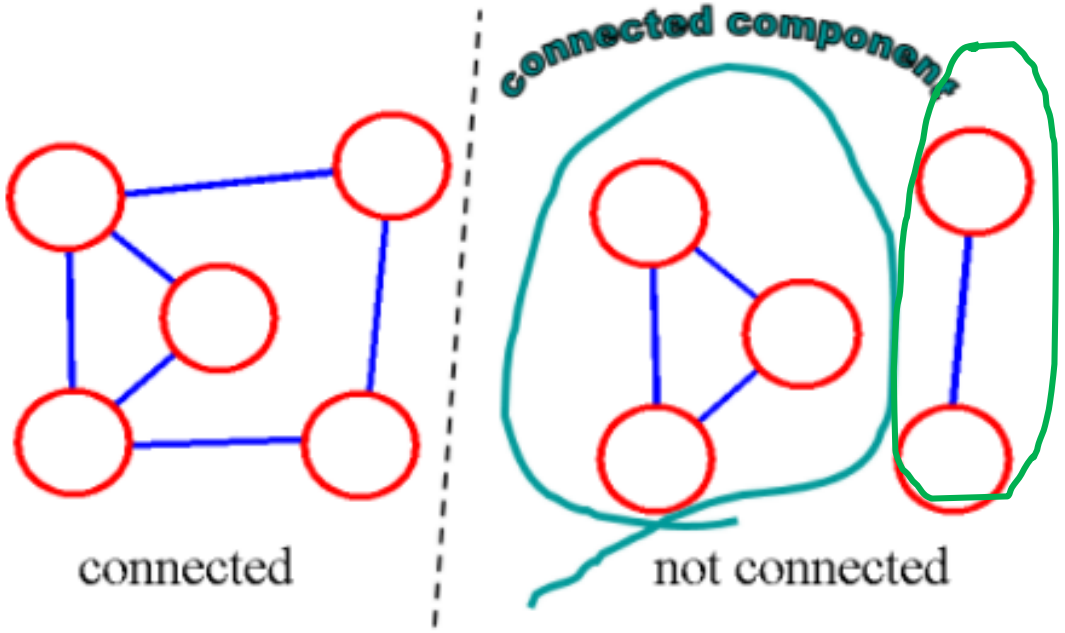


graph.txt

응용

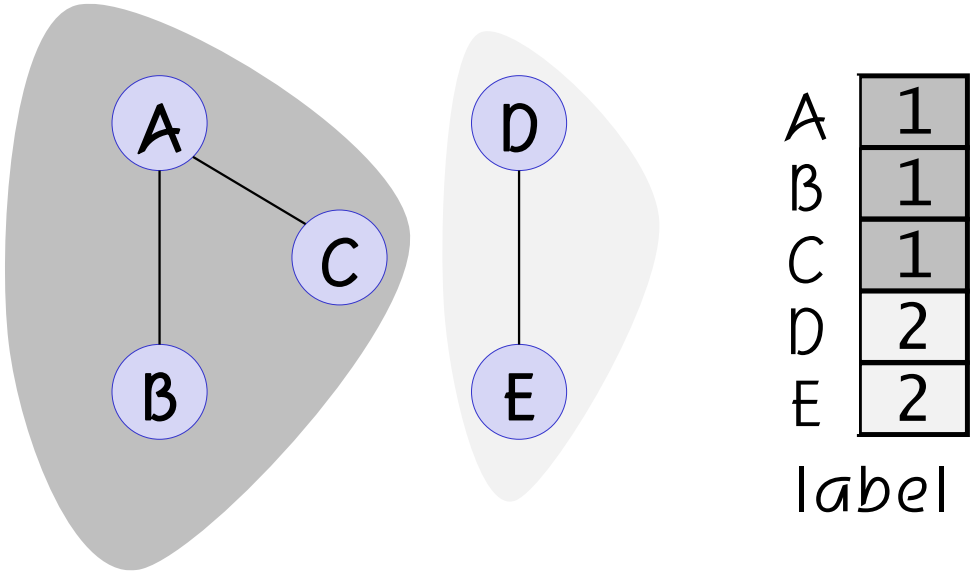
Connected Component

- **Connected Graph : any two vertices are connected by a path**
- **Connected Component : maximal connected subgraph of a graph**
- **application: network, communication**



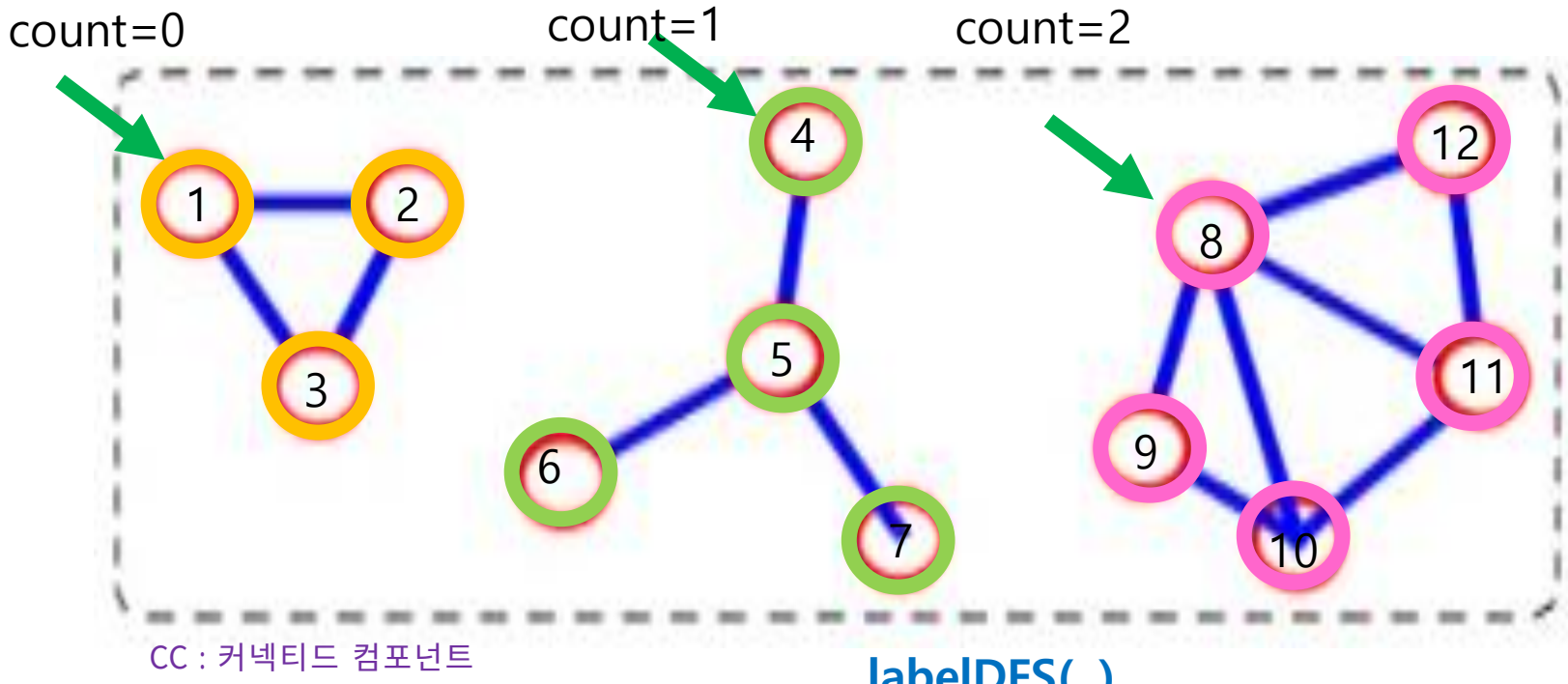
Connected Component Algorithm

- **최대로 연결된 부분 그래프들을 구함 → need searching**
  - DFS 또는 BFS 반복 이용
  - Labeling : 각 connected component 마다 레이블링, 1,2,3… a,b,c,….
    - **visited[v]=TRUE; 를 visited[v]=count; 로 교체**



Connected Component Algorithm

- **DFS 응용문제**



CC : 커넥티드 컴포넌트

CC알고리즘을 정리해 보면

(1) 특정 Node에서 DFS 진행 : 연결된 모든 노드 한번씩 방문

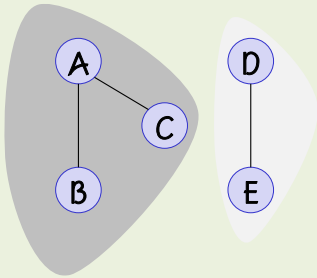
(2) count 증가

(3) 방문 안한 노드를 찾아서 이 노드에서 DFS 진행 ,

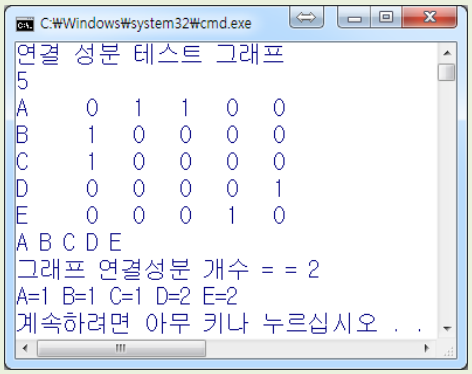
labelDFS(..)

findConnectedComponent(..)

```
class ConnectedComponentGraph : public SrchAMGraph {  // 클래스 상속
    int label[MAX_VTXS];          // CC 색상 (동일 CC에 동일 색상)
public:
    void labelDFS( int v, int color) { // 색상 레이블링
        visited[v] = true;        // 노드 v 방문 여부 기록
        label[v] = color;         // 노드 v의 색상
        for( int w=0 ; w<size ; w++)  // 모든 노드에 대해서
            if( isLinked(v,w) && visited[w]==false )
                labelDFS( w, color ); // 연결되어있고, 방문안한 노드 DFS
    }
    void findConnectedComponent( ) {
        int count = 0;                // 연결성분 개수 초기화
        for(int i=0; i<size ; i++)  // 모든 노드에 대해서
            if( visited[i]==false)  // 방문안한 노드이면
                labelDFS(i, ++count); // 이 노드에서 DFS시작
                                      // CC개수 증가
        printf("그래프 연결성분 개수 = = %d\n", count);
        for( int i=0 ; i<size ; i++ )// 모든 노드에 대해서
            printf( "%c=%d ", getVertex(i), label[i] ); // vertex=label 출력
        printf( "\n" );                                 // A=1 B=1 C=1 D=2 E=2
    }
};
```

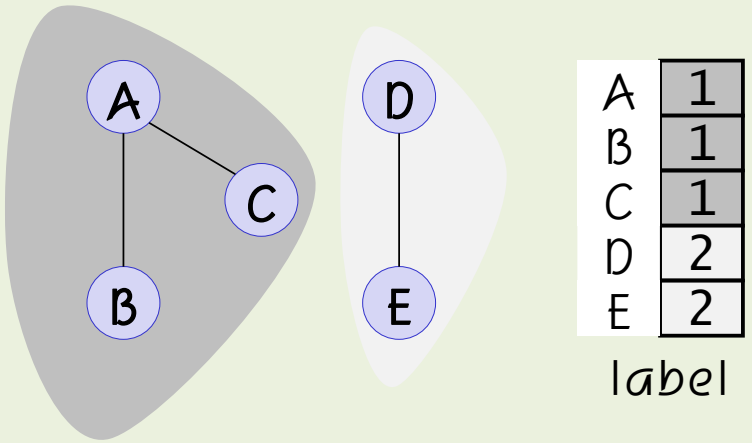


	label
A	1
B	1
C	1
D	2
E	2



```
// class ConnectedComponentGraph

void main()
{
  ConnectedComponentGraph cc;

  for (int i=0; i<5; i++)
        cc.insertVertex('A' + i);

  cc.insertEdge(1,0);
  cc.insertEdge(2,0);
  cc.insertEdge(3,4);

  printf("연결 성분 테스트 그래프 \n");
  cc.display();
  cc.resetVisited();
  cc.findConnectedComponent();
}
```

```
연결 성분 테스트 그래프
5
A    0   1   1   0   0
B    1   0   0   0   0
C    1   0   0   0   0
D    0   0   0   0   1
E    0   0   0   1   0
그래프 연결성분 개수 = = 2
A=1 B=1 C=1 D=2 E=2
```



Connected Component 응용문제: 안전영역

안전 영역 출처 분류

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	128 MB	33527	12721	8508	34.636%

문제

재난방재청에서는 많은 비가 내리는 장마철에 대비해서 다음과 같은 일을 계획하고 있다. 먼저 어떤 지역의 높이 정보를 파악한다. 그 다음에 그 지역에 많은 비가 내렸을 때 물에 잠기지 않는 안전한 영역이 최대로 몇 개가 만들어 지는 지를 조사하려고 한다. 이때, 문제를 간단하게 하기 위하여, 장마철에 내리는 비의 양에 따라 일정한 높이 이하의 모든 지점은 물에 잠긴다고 가정한다.

어떤 지역의 높이 정보는 행과 열의 크기가 각각 N인 2차원 배열 형태로 주어지며 배열의 각 원소는 해당 지점의 높이를 표시하는 자연수이다. 예를 들어, 다음은 N=5인 지역의 높이 정보이다.

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

Connected Component 응용문제: 안전영역

이제 위와 같은 지역에 많은 비가 내려서 높이가 4 이하인 모든 지점이 물에 잠겼다고 하자. 이 경우에 물에 잠기는 지점을 회색으로 표시하면 다음과 같다.

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

물에 잠기지 않는 안전한 영역이라 함은 물에 잠기지 않는 지점들이 위, 아래, 오른쪽 혹은 왼쪽으로 인접해 있으며 그 크기가 최대인 영역을 말한다. 위의 경우에서 물에 잠기지 않는 안전한 영역은 5개가 된다(꼭짓점으로만 붙어 있는 두 지점은 인접하지 않는다고 취급한다).

또한 위와 같은 지역에서 높이가 6이하인 지점을 모두 잠기게 만드는 많은 비가 내리면 물에 잠기지 않는 안전한 영역은 아래 그림에서와 같이 네 개가 됨을 확인할 수 있다.

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

Connected Component 응용문제: 안전영역

이와 같이 장마철에 내리는 비의 양에 따라서 물에 잠기지 않는 안전한 영역의 개수는 다르게 된다. 위의 예와 같은 지역에서 내리는 비의 양에 따른 모든 경우를 다 조사해 보면 물에 잠기지 않는 안전한 영역의 개수 중에서 최대인 경우는 5임을 알 수 있다.

어떤 지역의 높이 정보가 주어졌을 때, 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 계산하는 프로그램을 작성하시오.

입력

첫째 줄에는 어떤 지역을 나타내는 2차원 배열의 행과 열의 개수를 나타내는 수 N이 입력된다. N은 2 이상 100 이하의 정수이다. 둘째 줄부터 N개의 각 줄에는 2차원 배열의 첫 번째 행부터 N번째 행까지 순서대로 한 행씩 높이 정보가 입력된다. 각 줄에는 각 행의 첫 번째 열부터 N번째 열까지 N개의 높이 정보를 나타내는 자연수가 빈 칸을 사이에 두고 입력된다. 높이는 1이상 100 이하의 정수이다.

출력

첫째 줄에 장마철에 물에 잠기지 않는 안전한 영역의 최대 개수를 출력한다.

예제 입력 1 복사

```
5
6 8 2 6 2
3 2 3 4 6
6 7 3 3 2
7 2 5 3 6
8 9 5 2 7
```

예제 출력 1 복사

```
5
```

안전영역 문제 아이디어

- **높이가 1~100인 경우의 안전영역의 개수를 구해서 그 중에 최대값을 출력한다 !**
  - 높이는 1 100 사이의 정수

예제 입력 1

```
5
6 8 2 6 2
3 2 3 4 6
6 7 3 3 2
7 2 5 3 6
8 9 5 2 7
```

예제 출력 1

```
5
```

높이 = 4
(4이하 모두 물에 잠김)
안전영역 5개

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

높이 = 6
(6이하 모두 물에 잠김)
안전영역 4개

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

```
#include <iostream>
using namespace std;

int m[100][100];        // 행렬 입력
Int label[100][100];    // 레이블링
Int countNum[100];      // CC 개수 저장
int n;                  // 행렬 차원
```

bool isValid(int a, int b)

```
{
        return (0 <= a && a < n) && (0 <= b && b < n);
}
```

void dfs(int a, int b, int lb)

```
{
        int x, y;
        label[a][b] = lb;

        for (int i = 0; i < 4; i++) {
                switch (i)
                {
                case 0: x = a - 1; y = b; break;
                case 1: x = a + 1; y = b; break;
                case 2: x = a; y = b - 1; break;
                case 3: x = a; y = b + 1; break;
                }
                if (isValid(x, y) && label[x][y] == 1) dfs(x, y, lb);
        }
}
```

m

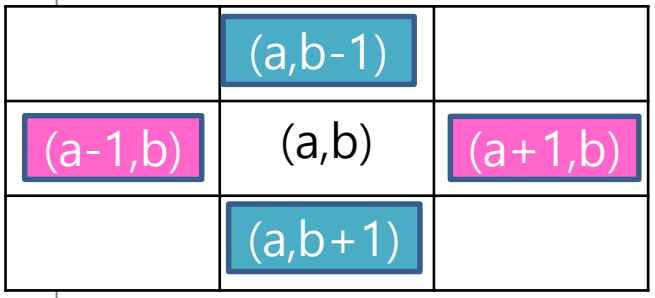
6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

label

m을 0과 1로 변환

예제 입력 1

```
5
6 8 2 6 2
3 2 3 4 6
6 7 3 3 2
7 2 5 3 6
8 9 5 2 7
```



```
bool isAllZero()  // label 배열의 값이 모두 0인지 체크 -> 안전영역 없으니 실행 중지
{
        for (int i = 0; i < n; i++)
          for (int j = 0; j < n; j++)
                if (label[i][j] != 0) return false;
        return true;
}

int getMax(int a[], int n)  // 배열 a의 최대값 구하기: CC 최대값
{
        int max = -1;
        for (int i = 0; i < n; i++)
                if (max < a[i]) max = a[i];
        return max;
}
```

안전영역 찾기 DFS 활용

- 행렬을 0 또는 1로 변환
- 1을 찾아서 DFS로 Connected Component를 찾는다.
- Connected Component의 레이블은 2부터 사용

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

높이 = 4
(4이하 모두 물에 잠김)
안전영역 5개

label

1	1	0	1	0
0	0	0	0	1
1	1	0	0	0
1	0	1	0	1
1	1	1	0	1

count =1

2	2	0	**1**	0
0	0	0	0	**1**
1	1	0	0	0
1	0	1	0	1
1	1	1	0	1

count =2

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

2	2	0	3	0
0	0	0	0	3
5	5	0	0	0
5	0	5	0	1
5	5	5	0	1

```
void safeArea(int n)
{
        for (int h = 0; h < 100; h++) {     // height h : 1~100
                int count = 0;                  // CC 개수 초기화
                for (int i = 0; i < n; i++) // label 배열을 0과 1로 표현
                for (int j = 0; j < n; j++)
                        if (m[i][j] > h + 1) label[i][j] = 1; // 높이 h보다 큰 부분만 1로
                        else label[i][j] = 0;

                if (isAllZero()) break;       // 높이 h보다 큰 부분이 없으면(all zero) stop

                for (int i = 0; i < n; i++) {  // test : 안전영역 출력
                for (int j = 0; j < n; j++)
                        cout << label[i][j];
                        cout << endl;
                }

                for (int i = 0; i < n; i++)   // Connected Component 알고리즘
                for (int j = 0; j < n; j++)   // 맵을 탐색하면서 label 1인 부분은 dfs로
                        if (label[i][j] == 1) {
                                count++;              // 새로운 CC를 찾을 때마다 증가
                                dfs(i, j, count + 1);  // 2부터 labeling
                        }
                countNum[h] = count;        // 각 높이 h에서의 CC 개수 저장
                cout << "높이: " << h << "  안전영역 갯수: " << count << endl;

        }
}
```

1	1	0	1	0
0	0	0	0	1
1	1	0	0	0
1	0	1	0	1
1	1	1	0	1

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

6	8	2	6	2
3	2	3	4	6
6	7	3	3	2
7	2	5	3	6
8	9	5	2	7

```
int main()
{
        // 입력
        cin >> n;
        for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
                cin >> m[i][j];

        // 안전영역 구하기
        safeArea(n);

        // 출력
        cout << getMax(countNum, 100) << endl;
}
```

예제 입력 1 복

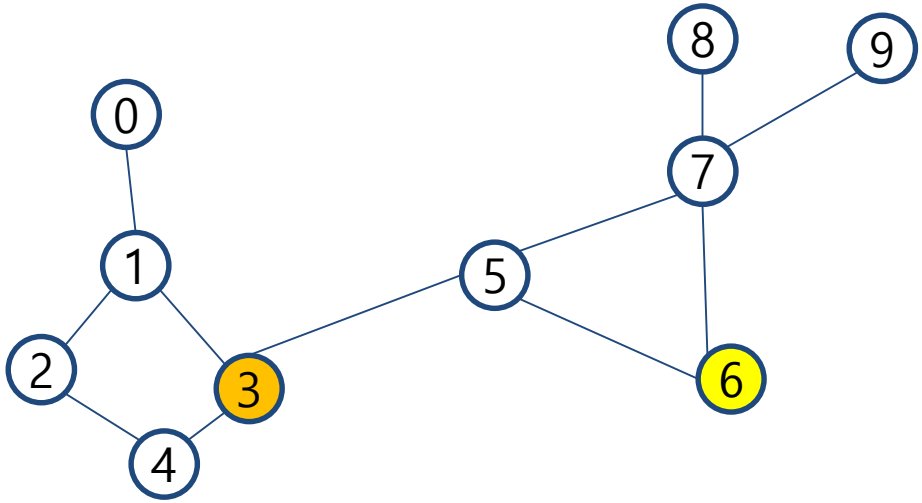
```
5
6 8 2 6 2
3 2 3 4 6
6 7 3 3 2
7 2 5 3 6
8 9 5 2 7
```

연습문제 2

- 연습문제 교재 p.461

(1) 교재에 있는 그래프에 대하여 정점 3에서 출발하여 깊이 우선 탐색을 한경우의 방문 순서는 ?

(2) 정점 6에서 출발하여 깊이 우선 탐색을 한 경우의 방문순서는 ?

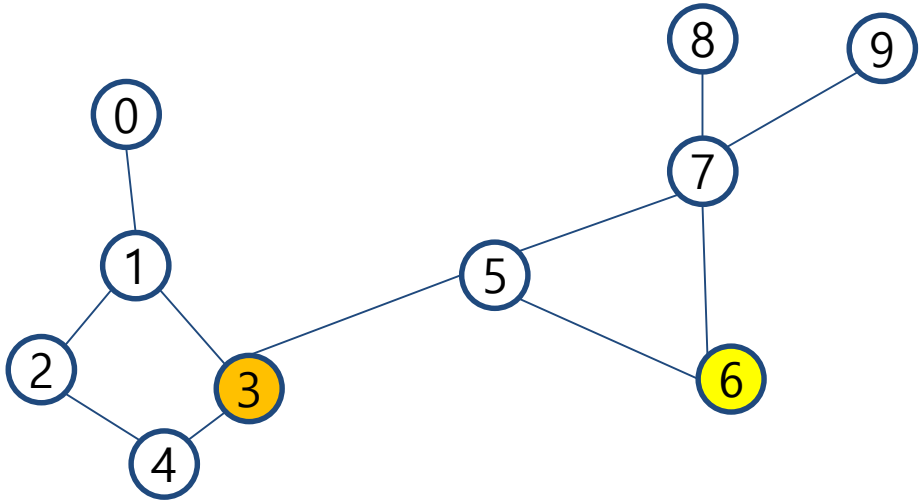


연습문제 3

- 연습문제 교재 p.461

(3) 교재에 있는 그래프에 대하여 정점 3에서 출발하여 너비 우선 탐색을 한경우의 방문 순서는 ?

(4) 정점 6에서 출발하여 너비 우선 탐색을 한 경우의 방문순서는 ?



Connected Component 개수 구하기

- **인접행렬과 깊이우선탐색을 이용하여 그래프 G의 connected component 개수를 구하세요.**
- **입력**
  - 그래프 G
- **출력**
  - Connected component 갯수

예제 입력 1

```
5
A 0 1 1 0 0
B 1 0 0 0 0
C 1 0 0 0 0
D 0 0 0 0 1
E 0 0 0 1 0
```

예제 출력 1

```
2
```

예제 입력 2

```
6
A 0 1 0 0 0 0
B 1 0 0 0 0 0
C 0 0 0 1 0 0
D 0 0 1 0 0 0
E 0 0 0 0 0 1
F 0 0 0 0 1 0
```

예제 출력 2

```
3
```

단지 번호 붙이기

유의사항 !!
안전영역 코딩 알고리즘에서 제시한 dfs(..) 함수를 반드시 사용하세요.

단지번호붙이기 출처 분류

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
1 초	128 MB	65808	26383	16799	38.421%

문제

<그림 1>과 같이 정사각형 모양의 지도가 있다. 1은 집이 있는 곳을, 0은 집이 없는 곳을 나타낸다. 철수는 이 지도를 가지고 연결된 집들의 모임인 단지를 정의하고, 단지에 번호를 붙이려 한다. 여기서 연결되었다는 것은 어떤 집이 좌우, 혹은 아래위로 다른 집이 있는 경우를 말한다. 대각선상에 집이 있는 경우는 연결된 것이 아니다. <그림 2>는 <그림 1>을 단지별로 번호를 붙인 것이다. 지도를 입력하여 단지수를 출력하고, 각 단지에 속하는 집의 수를 오름차순으로 정렬하여 출력하는 프로그램을 작성하시오.

0	1	1	0	1	0	0
0	1	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	0	1	1	1
0	1	0	0	0	0	0
0	1	1	1	1	1	0
0	1	1	1	0	0	0

<그림 1>

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

<그림 2>

단지 번호 붙이기

입력

첫 번째 줄에는 지도의 크기 N(정사각형이므로 가로와 세로의 크기는 같으며 5≤N≤25)이 입력되고, 그 다음 N줄에는 각각 N개의 자료(0혹은 1)가 입력된다.

출력

첫 번째 줄에는 총 단지수를 출력하시오. 그리고 각 단지내 집의 수를 오름차순으로 정렬하여 한 줄에 하나씩 출력하시오.
그리고, NxN 단지를 출력하는데 각 단지의 레이블을 각 단지내 집의 수로 표시하세요.

예제 입력 1 복사

```
7
0110100
0110101
1110101
0000111
0100000
0111110
0111000
```

예제 출력 1 복사

```
3
7
8
9
0 7 7 0 8 0 0
0 7 7 0 8 0 8
7 7 7 0 8 0 8
0 0 0 0 8 8 8
0 9 0 0 0 0 0
0 9 9 9 9 9 0
0 9 9 9 0 0 0
```

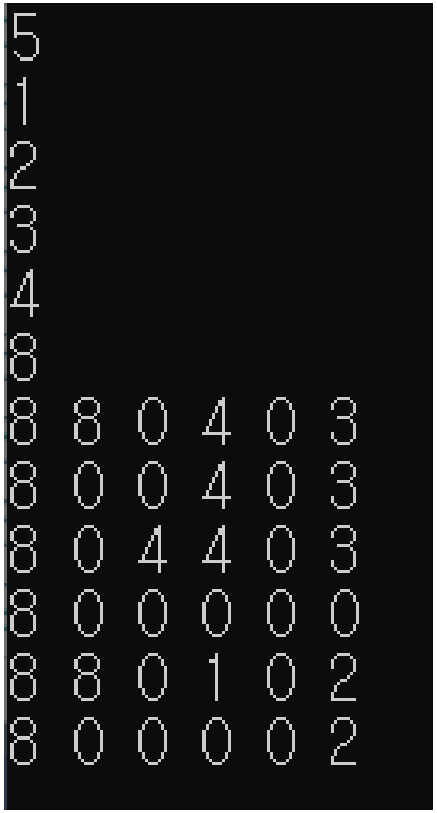
다음 슬라이드에
예제 입력 2 있어요~

단지 번호 붙이기

예제 입력 2

```
6
110101
100101
101101
100000
110101
100001
```

예제 출력 2



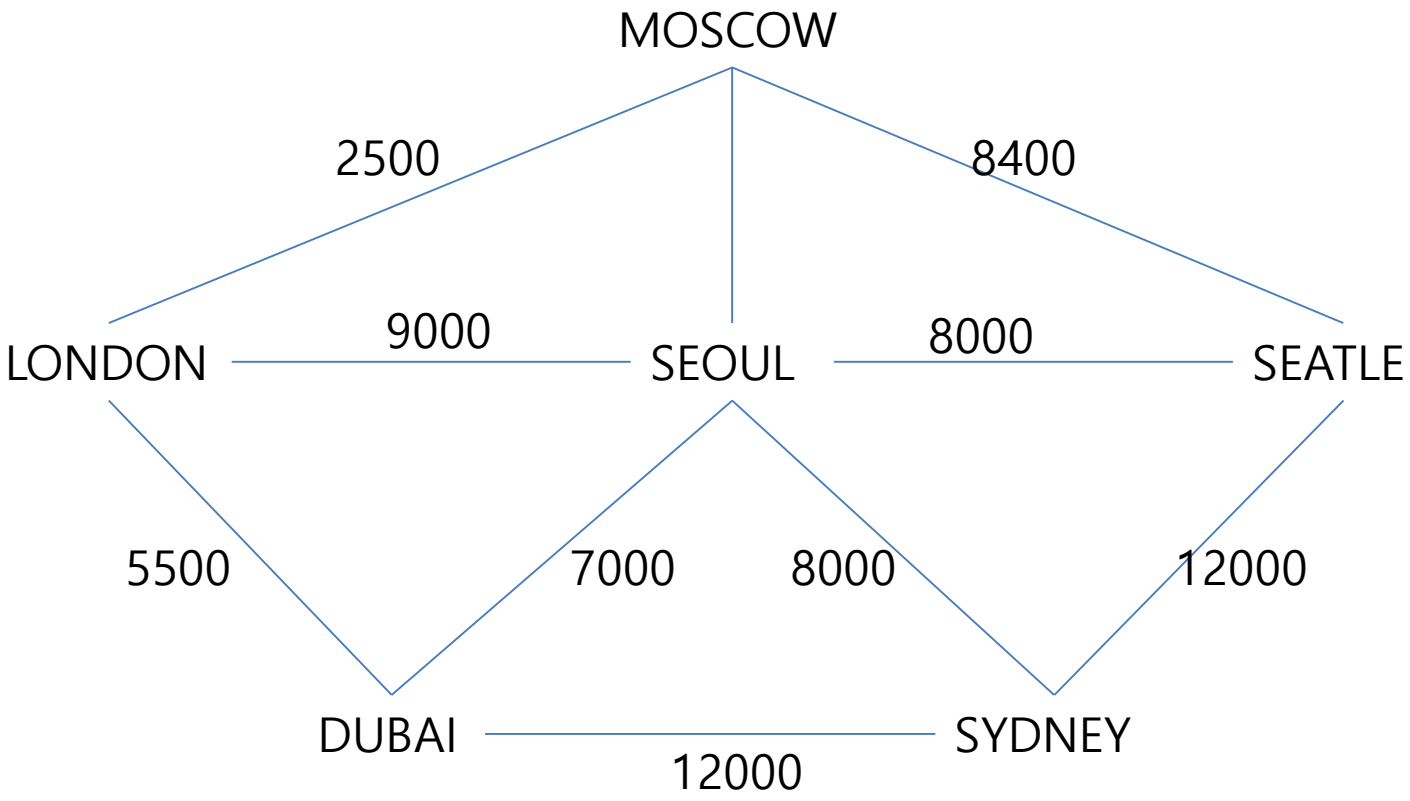
DFS+BFS 필수 문제

DFS+BFS 공부용. - njw1204

문제 번호	제목	정보	맞은 사람	제출	정답 비율
1260	DFS와 BFS	분류	20899	105742	32.620%
2178	미로 탐색	분류	18219	75532	36.767%
2606	바이러스	출처 분류	14621	46510	43.811%
2667	단지번호붙이기	출처 분류	16799	65808	38.421%
2644	촌수계산	출처 분류	5603	15998	45.343%
7569	토마토	출처 분류	7346	25694	39.650%
1697	숨바꼭질	출처 다국어 분류	15020	87381	24.799%
5014	스타트링크	출처 다국어 분류	4400	17737	34.483%
2468	안전 영역	출처 분류	8508	33527	34.636%
2573	빙산	출처 분류	4626	24446	26.504%
9205	맥주 마시면서 걸어가기	출처 다국어 분류	2510	9577	35.213%
14503	로봇 청소기	디버그 분류	8084	23658	51.711%

Example

- 아래 도시 네트워크를 STL을 이용해 그래프로 표현하세요.

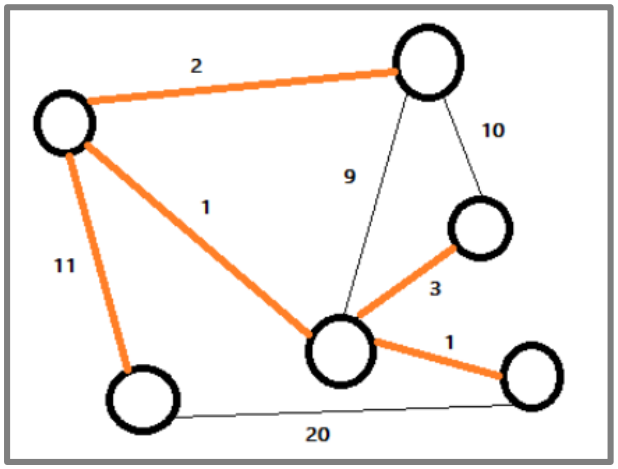


가중치 그래프

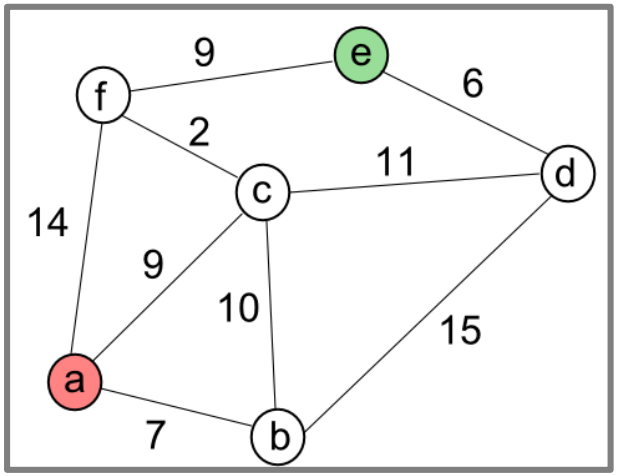
- **Weighted Graph**
  - 네트워크(network)라고도 함
  - 간선에 가중치가 할당된 그래프
    - 비용(cost)
    - 가중치(weight)
    - 길이(length)
  - 가중치 그래프 예
    - 정점 : 각 도시를 의미
    - 간선 : 도시를 연결하는 도로 의미
    - 가중치 : 도로의 길이

가중치 그래프

- **Minimum Spanning Tree**
  - **(MST)** is a subset of the edges of a connected, edge-weighted directed or undirected graph that **connects all the vertices together**, **without any cycles** and **with the minimum possible total edge weight**
  - **(예)** 도로 건설: 도시들을 모두 연결하면서 도로의 길이를 최소화



- **Shortest Path**
  - **(SP)** is a problem of **finding a path between two vertices** (or nodes) in a graph such that **the sum of the weights of its constituent edges is minimized.**

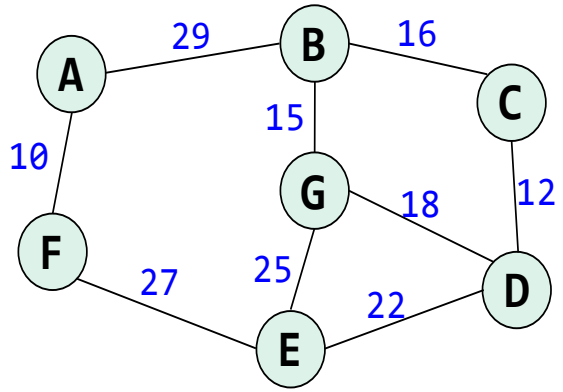


가중치 표현방법

- **인접 행렬을 이용한 그래프의 표현에서**
  - 방법 1: 가중치를 위한 추가적인 배열 사용
  - **방법 2: 인접 행렬 M[i][j]를 가중치를 위해 사용**
- **방법 2**
  - ➔ 추가적인 메모리 사용 없이 가중치 표현 가능
  - nxn 인접행렬 M을 이용한 그래프 표현에서 M[i][j] 가 간선 (i, j)의 가중치 값을 나타내도록 함
  - 인접 행렬 vs. 가중치 인접 행렬
    - 만약, 간선 (i,j)가 존재하지 않으면 → M[i][j] 가 매우 큰 값을 갖도록 함
    - 이 값은 가중치로 나타날 수 있는 범위 밖의 값 ➔ INF
  - 가중치는 반드시 양수이어야 함

표현 예

- 인접 행렬을 이용한 가중치 그래프의 표현 예



(a) 가중치 그래프

	A	B	C	D	E	F	G
A	0	29	∞	∞	∞	10	∞
B	29	0	16	∞	∞	∞	15
C	∞	16	0	12	∞	∞	∞
D	∞	∞	12	0	22	∞	18
E	∞	∞	∞	22	0	27	25
F	10	∞	∞	∞	27	0	∞
G	∞	15	∞	18	25	∞	0

(b) 인접 행렬을 이용한 표현

인접 행렬 가중치 그래프

- AdjMatGraph 클래스 상속을 이용해 WGraph 구현
- 간선 가중치의 최댓값으로 초기화

```
#define INF 9999                    // INF이면 간선없음

// 가중치 그래프를 표현하는 클래스
class WGraph : public AdjMatGraph {
public:
    void insertEdge( int u, int v, int weight ) {
        if( weight > INF ) weight = INF;
        setEdge(u, v, weight);
    }
    bool hasEdge(int i, int j) { return (getEdge(i,j)<INF); }
```

인접 행렬 가중치 그래프

파일의 형식은 다음과 같음. (graph.txt)

```
7
A  9999   29 9999 9999 9999   10 9999
B    29 9999   16 9999 9999 9999   15
C  9999   16 9999   12 9999 9999 9999
D  9999 9999   12 9999   22 9999   18
E  9999 9999 9999   22 9999   27   25
F    10 9999 9999 9999   27 9999 9999
G  9999   15 9999   18   25 9999 9999
```

```
void load(const char *filename) {
    FILE *fp = fopen(filename, "r");
    if( fp != NULL ) {
        int n, val;
        fscanf(fp, "%d", &n);                  // 정점의 전체 개수
        for(int i=0 ; i<n ; i++ ) {
            char str[80];
            int val;
            fscanf(fp, "%s", str);             // 정점의 이름
            insertVertex( str[0] );            // 정점 삽입
            for(int j=0 ; j<n ; j++){
                fscanf(fp, "%d", &val);        // 간선 정보
                insertEdge (i,j, val);         // 간선 삽입
            }
        }
    }
    fclose(fp);
}
};
```

MST

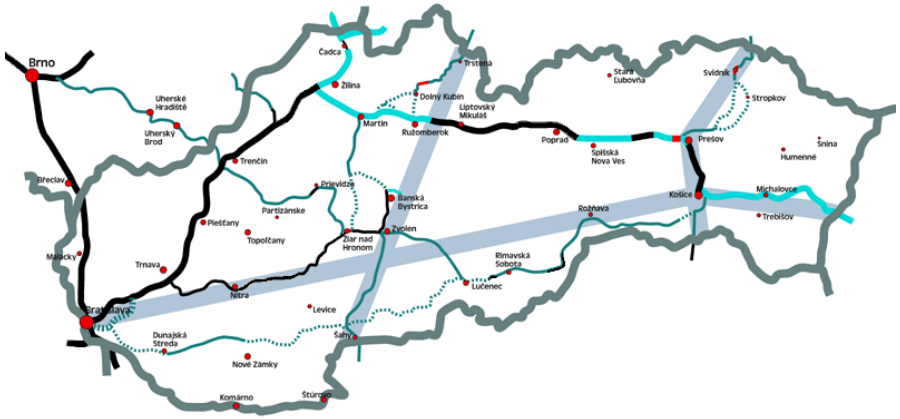
- **Kruskal's MST Algorithm**
- **Prim's MST Algorithm**

프림 알고리즘 부분이 이 자료에 없음!

- 한 정점에서 시작해 거기서 갈 수 있는 정점 중 가중치가 작은 쪽으로 간다.
- 다음 도착한 정점에서도 아직 방문하지 않은 곳 중 가중치가 작은 쪽으로 간다.
- 모든 정점을 방문할 때까지 반복한다.

최소비용 신장 트리 문제 (MST)

- **MST: Minimum Spanning Tree**
  - 네트워크에 있는 모든 정점들을 가장 적은 수의 간선과 비용으로 연결
- **MST의 응용**
  - 도로 건설: 도시들을 모두 연결하면서 도로의 길이를 최소화
  - 전기 회로: 단자를 모두 연결하면서 전선의 길이를 최소화
  - 배관: 배관을 모두 연결하면서 파이프의 총 길이를 최소화



최소비용 신장트리 조건

- **최소비용 신장트리 조건** ***summary***

  1. 간선의 가중치의 합이 최소이어야 한다.
  2. 반드시 (n-1)개의 간선만 사용해야 한다.
  3. 사이클이 포함되어서는 안 된다.

Kruskal 알고리즘

Prim 알고리즘

Kruskal의 MST 알고리즘

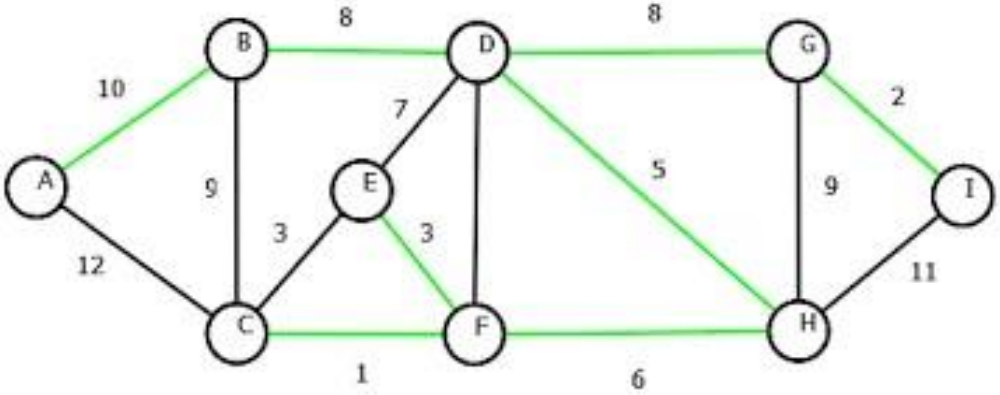
- **탐욕적인 방법(greedy method)**
  - 주요 알고리즘 설계 기법
  - **각 단계에서 최선의 답을 선택**하는 과정을 반복함으로써 최종적인 해답에 도달
  - 탐욕적인 방법은 항상 최적의 해답을 주는지 검증 필요
  - Kruskal MST 알고리즘은 최적의 해답임이 증명됨

Kruskal의 MST 알고리즘

- **알고리즘**
  - 각 단계에서 사이클을 이루지 않는 최소 비용 간선 선택

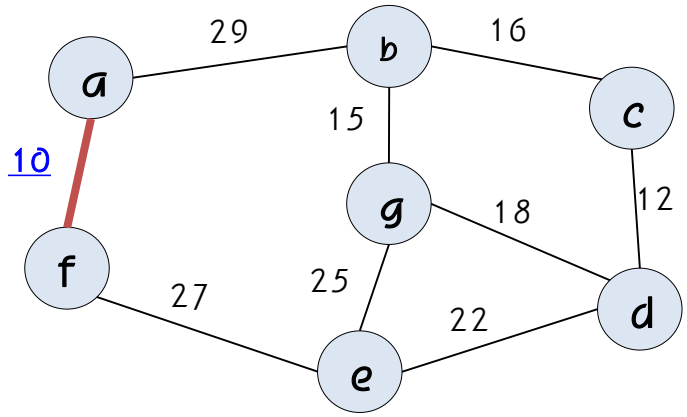
kruskal()

1. 그래프의 모든 간선을 가중치에 따라 **오름차순으로 정렬**한다.
2. **가장 가중치가 작은 간선** e를 뽑는다.
3. e를 신장 트리에 넣을 경우 **사이클이 생기면 삽입하지 않고** 2번으로 이동한다.
4. 사이클이 생기지 않으면 **최소 신장 트리에 삽입**한다.
5. **n-1개의 간선이 삽입될 때 까지** 2번으로 이동한다.

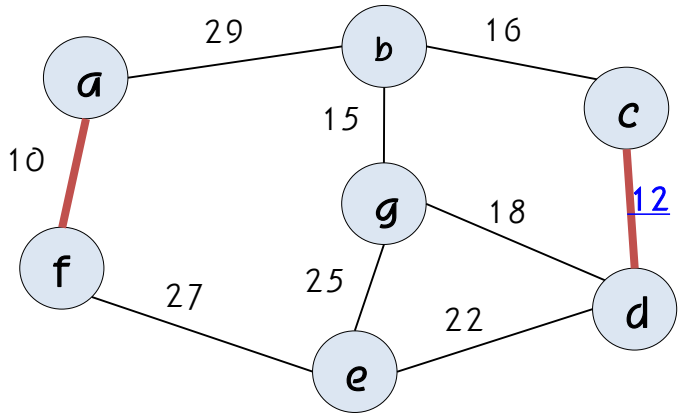


Kruskal의 MST 과정 (1)

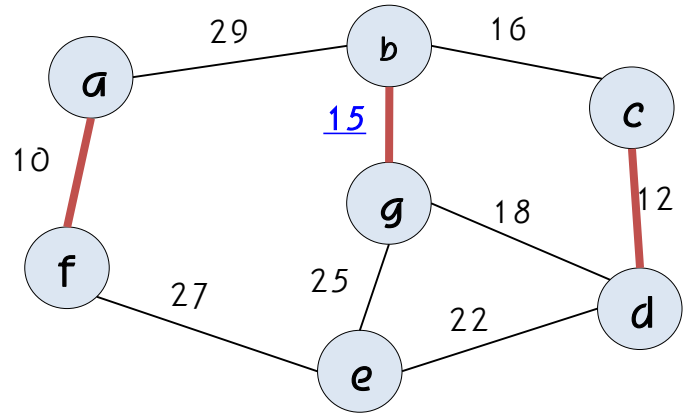
최소신장트리에 삽입
→ Red Edge로 표현



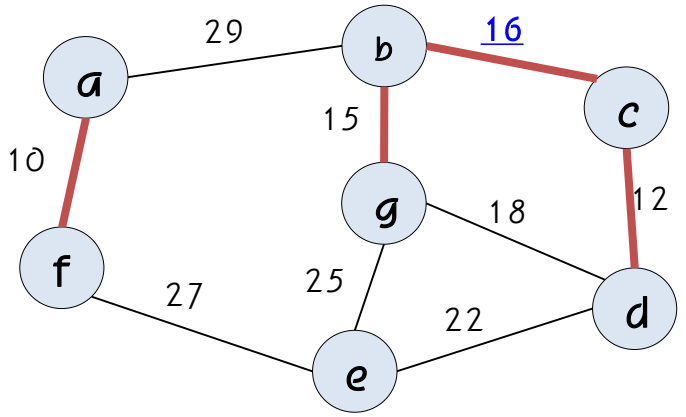
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



af	cd	bg	bc	dg	de	eg	ef	ab
10	**12**	15	16	18	22	25	27	29



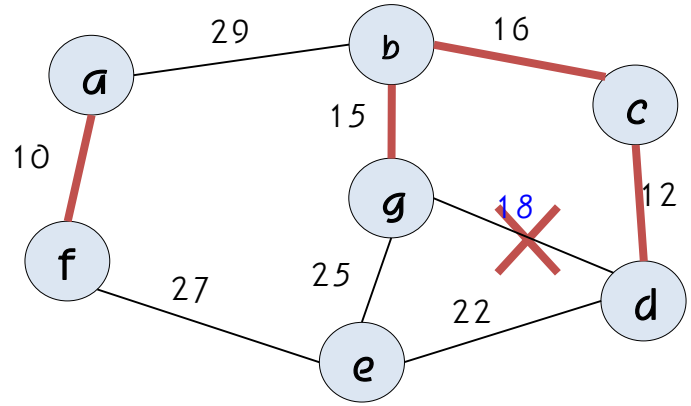
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	**15**	16	18	22	25	27	29

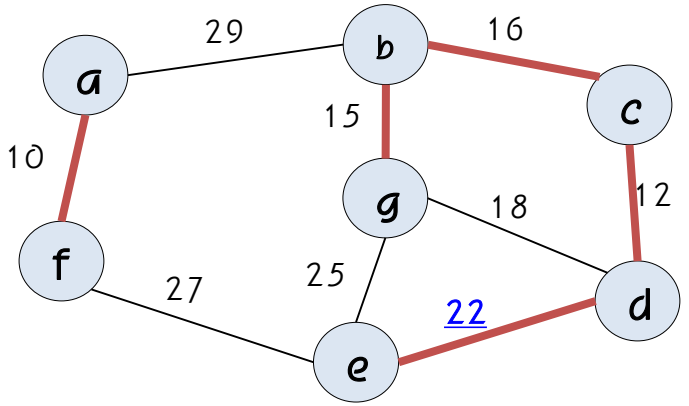


af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	**16**	18	22	25	27	29

Kruskal의 MST 과정 (2)

최소신장트리에 삽입
→ Red Edge로 표현

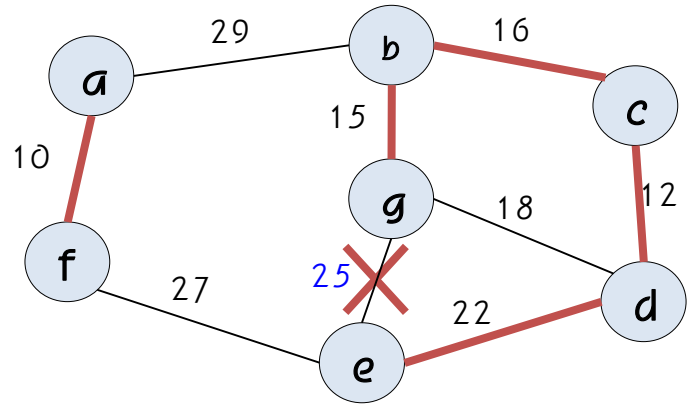


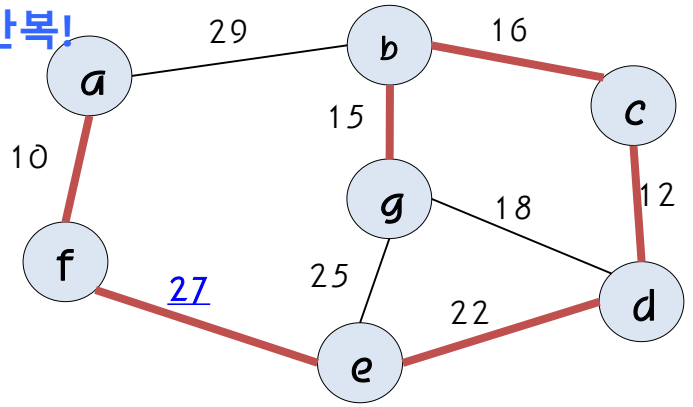


af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29

af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29

간선이 6개
될때까지 반복!

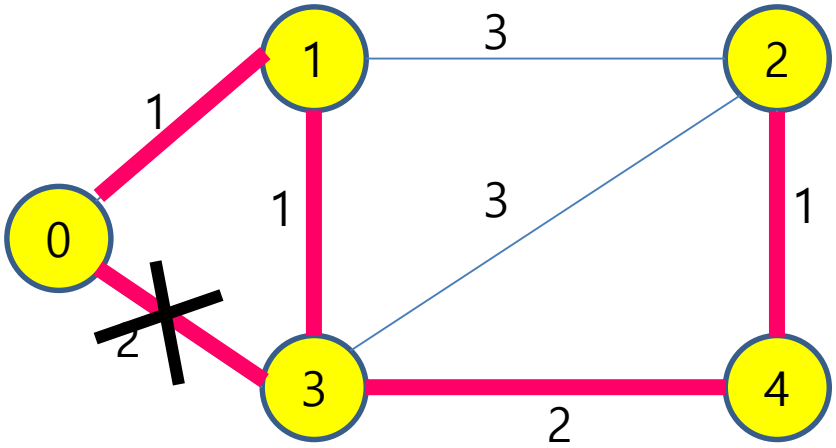




af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29

연습문제 4

- 아래 네트워크에 대하여 Kruskal의 MST 알고리즘을 이용해서 최소 비용 신장 트리가 구성되는 과정을 보여라. (최종 결과 트리에 (1)(2)(3)(4)....로 표시)

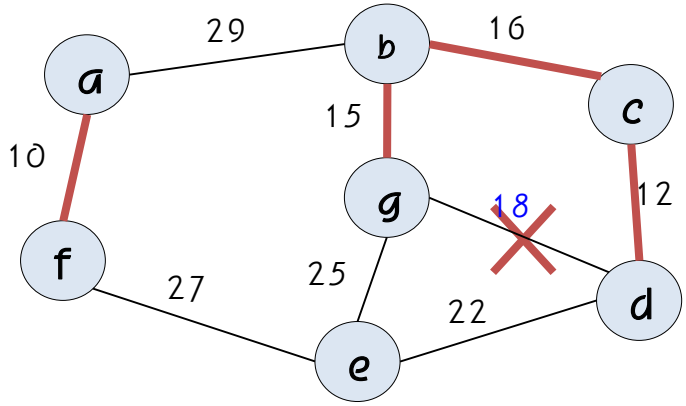


간선 4개가
선택될때까지 반복 !

MST

- **Kruskal's MST 'Union-Find algorithm'**

사이클 검사는 어떻게 할까요?

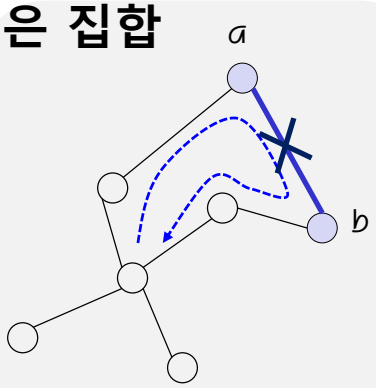


'집합' 개념 도입

연결되어 있는 정점들을 하나의 집합으로 구성

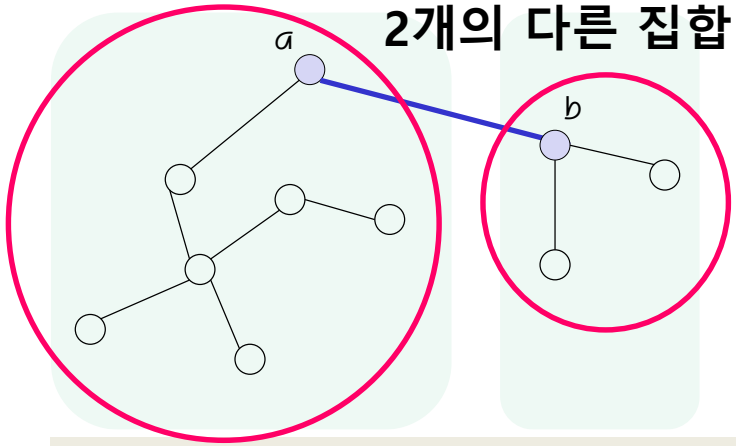
Edge a-b를 추가하고자 할 때, 사이클을 검사하는 방법

같은 집합



양끝 정점이 같은 집합에 속하는 경우
사이클 생김

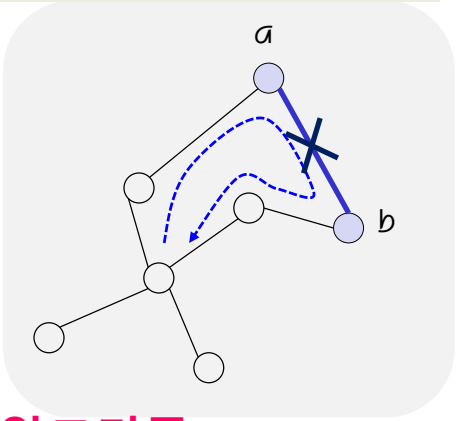
2개의 다른 집합



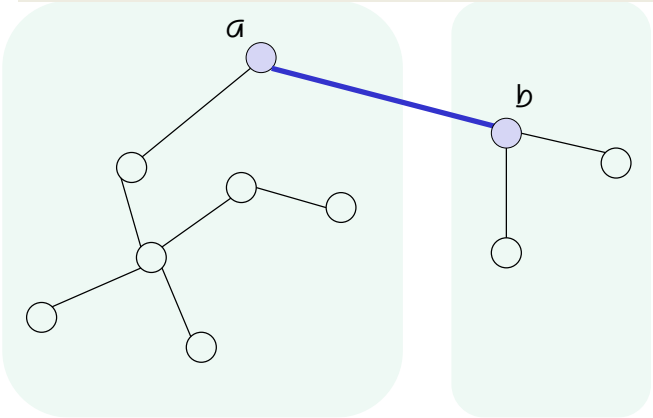
양끝 정점이 다른 집합에 속하는 경우
사이클 안생김

사이클의 검사→Find-Union 연산

양끝 정점이 **같은** 집합에 속하는 경우



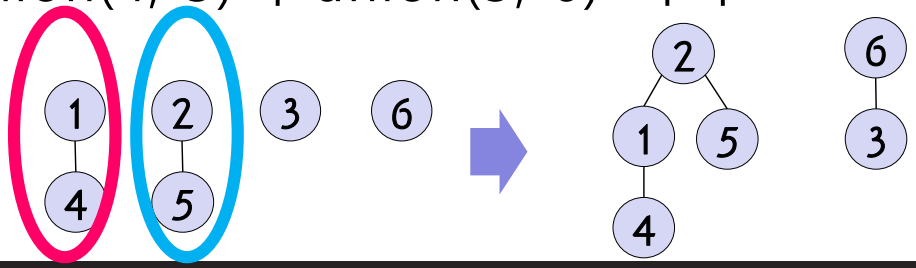
양끝 정점이 **다른** 집합에 속하는 경우



- **Union-Find 알고리즘**

  해당 원소가 속한 집합을 찾아서 루트끼리 연결

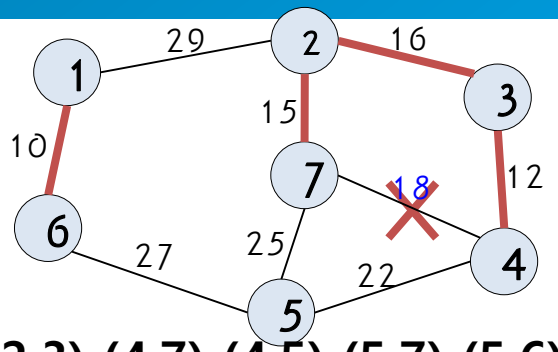
  - **Find**: 원소가 어떤 집합에 속하는지 알아냄
  - **Union**: 두 집합들의 합집합 만듦
  - 예: union(4, 5)와 union(3, 6) 처리



{1, 4} {5, 2} {3} {6} → {1, 4, 5, 2}, {3, 6}

Union 알고리즘
집합 {1,4}의 루트 1
집합 {2,5}의 루트 2
를 연결
Union(4,5)
=Union(1,2)
1의 parent를 2로!

정점 집합 클래스(Union-Find 연산) 구현

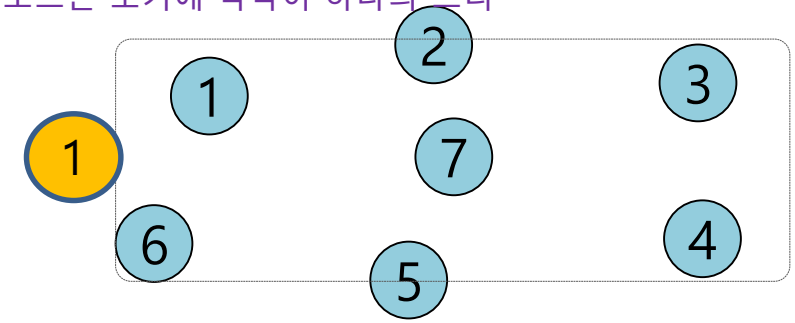



```
1  MST By Kruskal's Algorithm
     1    2    3    4    5    6    7
    -1   -1   -1   -1   -1   -1   -1

2  간선 추가 : A - F (비용:10)                                  (1,6) union
     1    2    3    4    5    6    7
     6   -1   -1   -1   -1   -1   -1

3  간선 추가 : C - D (비용:12)                                  (3,4) union
     1    2    3    4    5    6    7
     6   -1    4   -1   -1   -1   -1

4  간선 추가 : B - G (비용:15)                                  (2,7) union
     1    2    3    4    5    6    7
     6    7    4   -1   -1   -1   -1

5  간선 추가 : B - C (비용:16)                                  (2,3) union
     1    2    3    4    5    6    7                           루트끼리
     6    7    4   -1   -1   -1    4                           union

6  간선 추가 실패 : D - G (비용:18) (싸이클 생성)               (4,7) find
     1    2    3    4    5    6    7
     6    7    4   -1   -1   -1    4

7  간선 추가 : D - E (비용:22)                                  (4,5) union
     1    2    3    4    5    6    7
     6    7    4    5   -1   -1    4

8  간선 추가 실패 : E - G (비용:25) (싸이클 생성)               (5,7) find
     1    2    3    4    5    6    7
     6    7    4    5   -1   -1    4

9  간선 추가 : E - F (비용:27)                                  (5,6) union
     1    2    3    4    5    6    7
     6    7    4    5    6   -1    4
```

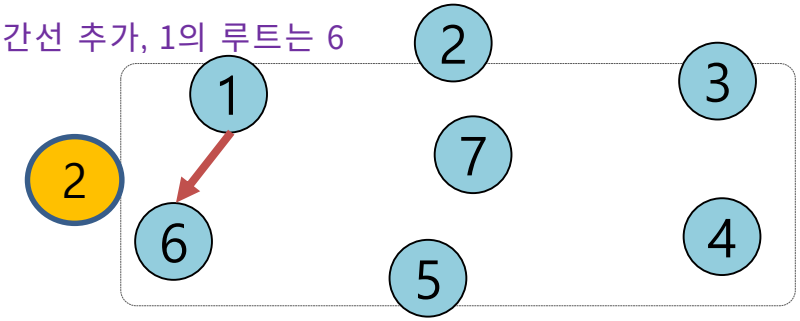
Sorting

(1,6) (3,4) (2,7) (2,3) (4,7) (4,5) (5,7) (5,6)

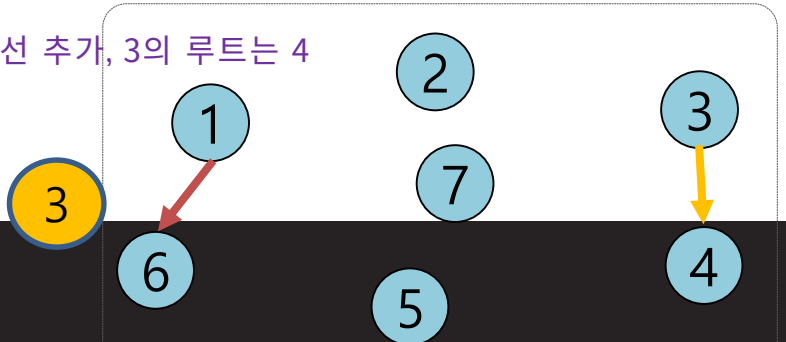
모든 노드는 초기에 각각이 하나의 트리



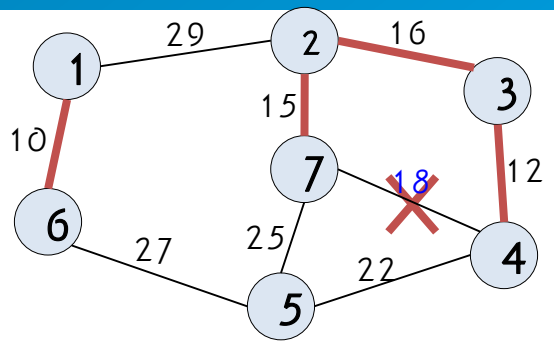
간선 추가, 1의 루트는 6



간선 추가, 3의 루트는 4



정점 집합 클래스(Union-Find 연산) 구현

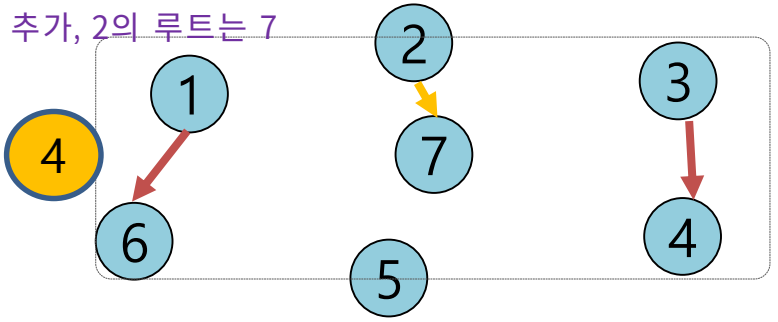



```
1  MST By Kruskal's Algorithm
     1    2    3    4    5    6    7
    -1   -1   -1   -1   -1   -1   -1

2  간선 추가 : A - F (비용:10)                                  (1,6) union
     1    2    3    4    5    6    7
     6   -1   -1   -1   -1   -1   -1

3  간선 추가 : C - D (비용:12)                                  (3,4) union
     1    2    3    4    5    6    7
     6   -1    4   -1   -1   -1   -1

4  간선 추가 : B - G (비용:15)                                  (2,7) union
     1    2    3    4    5    6    7
     6    7    4   -1   -1   -1   -1

5  간선 추가 : B - C (비용:16)                                  (2,3) union
     1    2    3    4    5    6    7                           루트끼리
     6    7    4   -1   -1   -1    4                           union

6  간선 추가 실패 : D - G (비용:18) (싸이클 생성)               (4,7) find
     1    2    3    4    5    6    7
     6    7    4   -1   -1   -1    4

7  간선 추가 : D - E (비용:22)                                  (4,5) union
     1    2    3    4    5    6    7
     6    7    4    5   -1   -1    4

8  간선 추가 실패 : E - G (비용:25) (싸이클 생성)               (5,7) find
     1    2    3    4    5    6    7
     6    7    4    5   -1   -1    4

9  간선 추가 : E - F (비용:27)                                  (5,6) union
     1    2    3    4    5    6    7
     6    7    4    5    6   -1    4
```

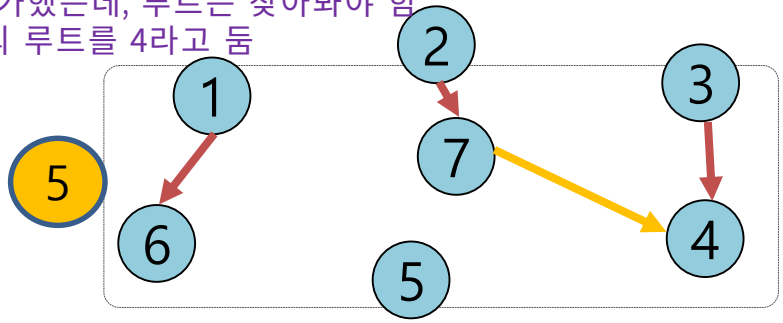
Sorting

(1,6) (3,4) (2,7) (2,3) (4,7) (4,5) (5,7) (5,6)

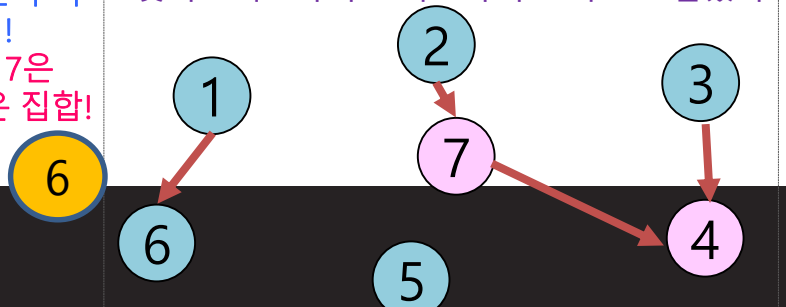
간선 추가, 2의 루트는 7



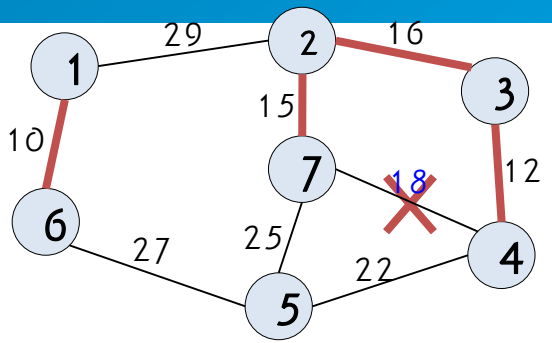
간선 추가했는데, 루트는 찾아봐야 함
일단 7의 루트를 4라고 둠



간선 추가 ← 찾아보니 7의 루트와 2의 루트가 4로 같았다
NO !
4와 7은
같은 집합!



정점 집합 클래스(Union-Find 연산) 구현



변수 int parent[..]

-1로 초기화

```
MST By Kruskal's Algorithm
  1   2   3   4   5   6   7
 -1  -1  -1  -1  -1  -1  -1
```

```
간선 추가 : A - F (비용:10)
  1   2   3   4   5   6   7
  6  -1  -1  -1  -1  -1  -1
```
(1,6) union

```
간선 추가 : C - D (비용:12)
  1   2   3   4   5   6   7
  6  -1   4  -1  -1  -1  -1
```
(3,4) union

```
간선 추가 : B - G (비용:15)
  1   2   3   4   5   6   7
  6   7   4  -1  -1  -1  -1
```
(2,7) union

```
간선 추가 : B - C (비용:16)
  1   2   3   4   5   6   7
  6   7   4  -1  -1  -1   4
```
(2,3) union
루트끼리 union

```
간선 추가 실패 : D - G (비용:18) (싸이클 생성)
  1   2   3   4   5   6   7
  6   7   4  -1  -1  -1   4
```
(4,7) find

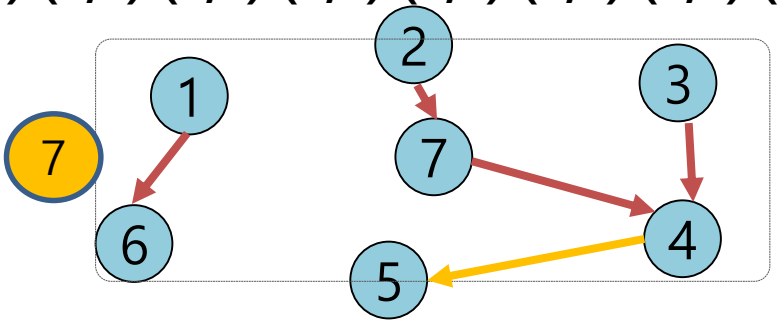
```
간선 추가 : D - E (비용:22)
  1   2   3   4   5   6   7
  6   7   4   5  -1  -1   4
```
(4,5) union

```
간선 추가 실패 : E - G (비용:25) (싸이클 생성)
  1   2   3   4   5   6   7
  6   7   4   5  -1  -1   4
```
(5,7) find

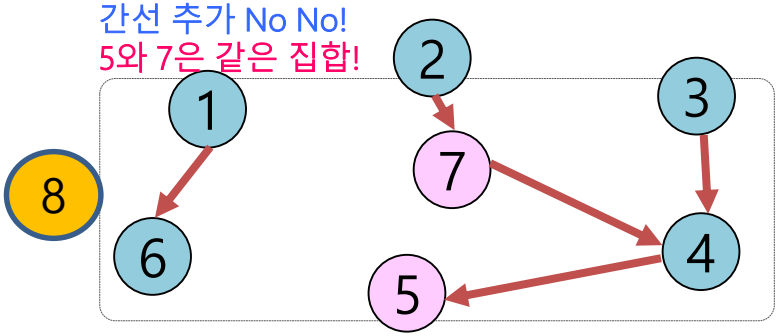
```
간선 추가 : E - F (비용:27)
  1   2   3   4   5   6   7
  6   7   4   5   6  -1   4
```
(5,6) union

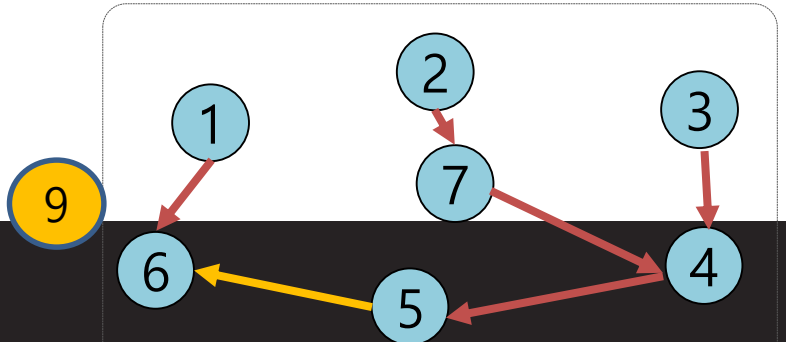
Sorting

(1,6) (3,4) (2,7) (2,3) (4,7) (4,5) (5,7) (5,6)



간선 추가 No No!
5와 7은 같은 집합!





정점집합 클래스(Union-Find 연산) 구현

- **변수**

```
부모 정점의 id:  int parent[MAX_VTXS];
집합 개수: int nSets;
```

- **함수**
- **FIND : v가 속한 집합을 찾아 root 반환**

```
int findSet( int v )
```

- **UNION : 2개의 집합 s1과 s2를 합침.**
- S1의 부모가 s2가 되도록 설정, s2가 부모가 되도록~

```
void unionSets(int s1, int s2)
```

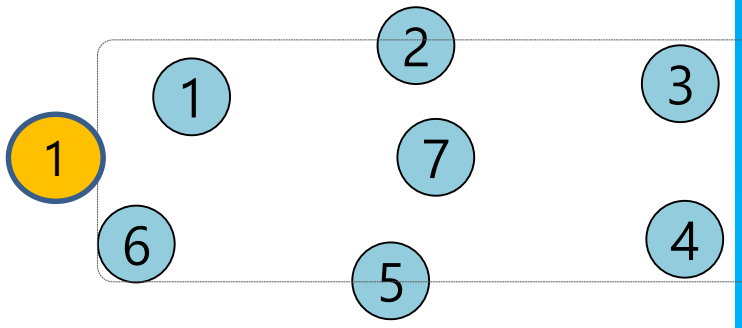
AI융합학부 안양준 yangjunahn@sungshin.ac.kr

정점집합 클래스(Union-Find 연산) 구현

```
// 정점 집합 클래스
class VertexSets {
    int         parent[MAX_VTXS]; // 부모 정점의 id
    int         nSets;            // 집합의 개수
public:
    VertexSets (int n) : nSets(n) {
        for( int i=0 ; i<nSets ;i++ )
            parent[i] = -1;      // 초기에 모든 정점이 고유의 집합에 속함
    }
bool isRoot( int i ) { return parent[i] < 0; }  // -1이면 root
```

	1	2	3	4	5	6	7
parents	-1	-1	-1	-1	-1	-1	-1

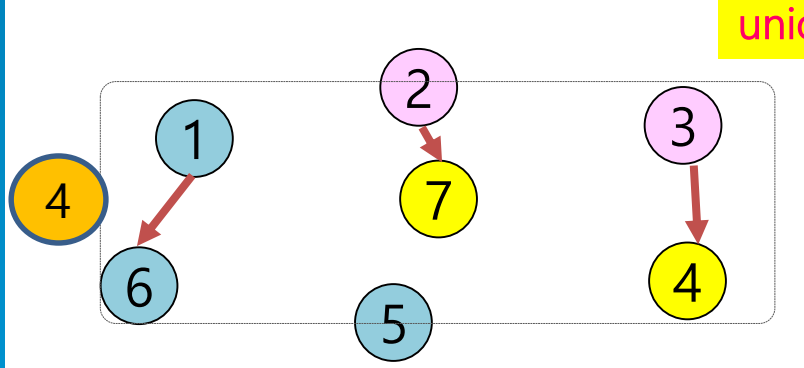
루트 배열 : 생성자 초기화

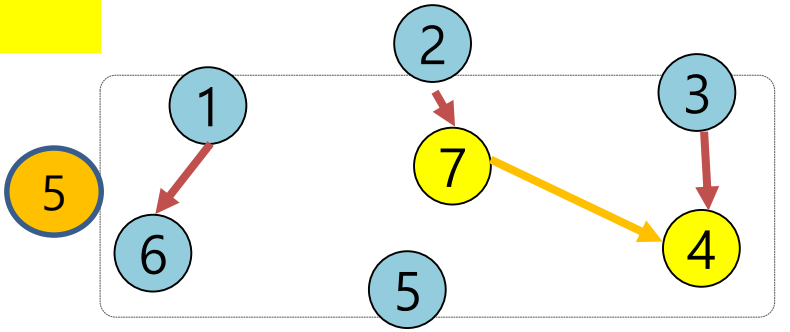


정점집합 클래스(Union-Find 연산) 구현

```
    int findSet( int v ) {                  // v가 속한 집합을 찾아 root 반환
        while (!isRoot(v)) v = parent[v]; // v가 속한 집합의 루트를 찾음
        return v;
    }
    void unionSets(int s1, int s2) { // 집합 s1을 집합 s2와 합침
        parent[s1] = s2;            // s1의 parent를 s2로 설정
        nSets--;                    // 2개의 집합을 합쳐서 집합 개수는 1 감소
    }
};
```

findSet(2) : return 7
findSet(3) : return 4
union(7,4)





간선 추가 : B - G (비용:15)

1	2	3	4	5	6	7
6	7	4	-1	-1	-1	-1

간선 추가 : B - C (비용:16)

1	2	3	4	5	6	7
6	7	4	-1	-1	-1	4

정점집합 클래스(Union-Find 연산) 구현

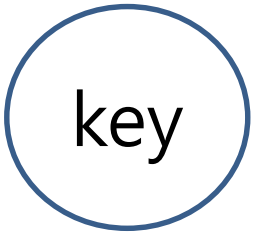
```
// 정점 집합 클래스
class VertexSets {
    int         parent[MAX_VTXS]; // 부모 정점의 id
    int         nSets;            // 집합의 개수
public:
    VertexSets (int n) : nSets(n) {
        for( int i=0 ; i<nSets ;i++ ) // 초기에 모든 정점은 root
            parent[i] = -1;      // 초기에 모든 정점이 고유의 집합에 속함
    }
    bool isRoot( int I ) { return parent[i] < 0; }

    int findSet( int v ) {       // v가 속한 집합을 찾아 root 반환
        while ( !isRoot(v) ) v = parent[v]; // v가 속한 집합의 root를 찾음
        return v;
    }
    void unionSets(int s1, int s2) {     // 집합 s1을 집합 s2에 합침
        parent[s1] = s2;          // s1의 parent를 s2로 설정
        nSets--;                  // 집합 2개를 합쳤으므로 집합 개수 1 감수
    }
};
```

MST

- Kruskal's MST 구현

기존 HeapNode 클래스

```
// 힙에 저장할 노드 클래스

class HeapNode
{
        int     key;
public:
        HeapNode(int key = 0) : key(key) { }
        ~HeapNode(void) { }

        void    setKey(int k) { key = k; }
        int     getKey() { return key; }
        void    display() { printf("\t%d", key); }
};
```



HeapNode에서 수정할 사항

```
// 힙에 저장할 노드 클래스

class HeapNode {
    int key;     // Key 값: 간선의 가중치
    int v1;      // 정점 1
    int v2;      // 정점 2
public:
    HeapNode(int k=0, int u=0, int v=0) : key(k), v1(u), v2(v) { }
    void setKey(int k) { key = k; }
    void setKey(int k, int u, int v ){
        key= k;  v1= u;  v2= v;
    }
    int getKey() { return key; }
    int getV1() { return v1; }
    int getV2() { return v2; }
    void display() { printf("\t(%2d-%2d) -- %d\n", v1, v2, key);}
};
```

수정

수정

수정

수정

수정

기존 MinHeap

```
// class MinHeap: 최소 힙 클래스 (프로그램 10.7)
...

    // 삽입 함수
    void insert( int key ) {
        if( isFull() ) return;
        int i = ++size;
        while( i!=1 && key < getParent(i).getKey()) {
            node[i] = getParent(i);
            i /= 2;
        }
        node[i].setKey( key );
    }

...
};
```

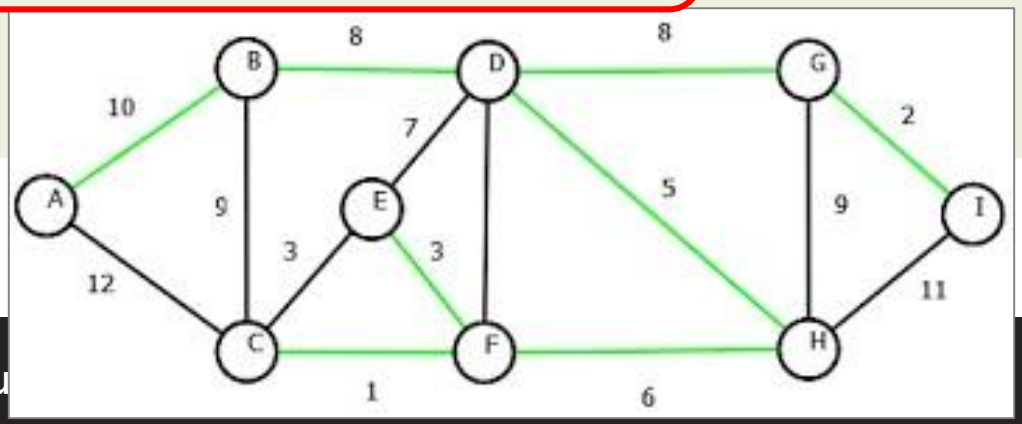
->Heap Sorting

MinHeap에서 수정할 사항

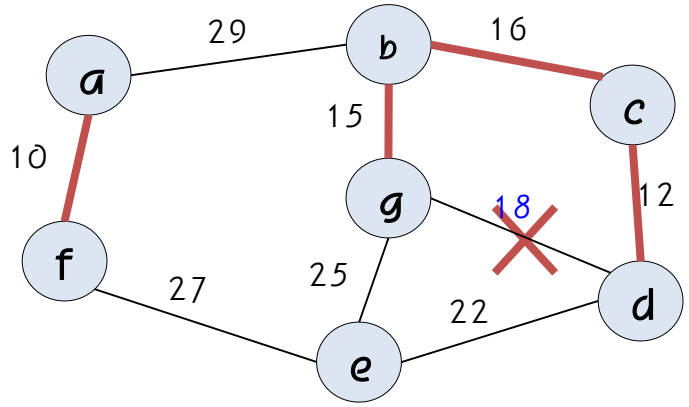
```
// class MinHeap: 최소 힙 클래스 (프로그램 10.7)
... // 코드 동일

    // 삽입 함수
    void insert( int key, int u, int v ) {
        if( isFull() ) return;
        int i = ++size;
        while( i!=1 && key < getParent(i).getKey()) {
            node[i] = getParent(i);
            i /= 2;
        }
        node[i].setKey( key, u, v );
    }

... // 코드 동일
};
```

->Heap Sorting

수정



Kruskal MST



kruskal()

1. 그래프의 모든 간선을 가중치에 따라 오름차순으로 정렬한다.
2. 가장 가중치가 작은 간선 e를 뽑는다.
3. e를 신장 트리에 넣을 경우 사이클이 생기면 삽입하지 않고 2번으로 이동한다.
4. 사이클이 생기지 않으면 최소 신장 트리에 삽입한다.
5. n-1개의 간선이 삽입될 때 까지 2번으로 이동한다.

MST 기능이 추가된 가중치 그래프

```
class WGraphMST : public WGraph {
public:
    void Kruskal( ) {      // kruskal의 최소 비용 신장 트리 프로그램
        MinHeap heap;
        for( int i=0 ; i<size-1 ; i++ )
        for( int j=i+1 ; j<size ; j++ )
            if( hasEdge(i,j) )
                heap.insert( getEdge(i,j), i, j ); // 모든 간선 삽입
        VertexSets set(size);                  // size개의 집합을 만듦
        int edgeAccepted=0;                    // 선택된 간선의 수
        while( edgeAccepted < size-1 ){  //
            HeapNode e = heap.remove(); //
            int uset=set.findSet(e.getV1()); // v1이 속한 집합의 루트 반환
            int vset=set.findSet(e.getV2()); // v2가 속한 집합의 루트 반환
            if( uset != vset ){                //
                printf( "간선 추가 : %c - %c (비용:%d)₩n",
                        getVertex(e.getV1()), getVertex(e.getV2()), e.getKey());
                set.unionSets(uset, vset);         // 두개의 집합을 합함.
                edgeAccepted++;
        }    };
    }
```

1.오름차순으로 정렬(Heap Sort)

정점 집합

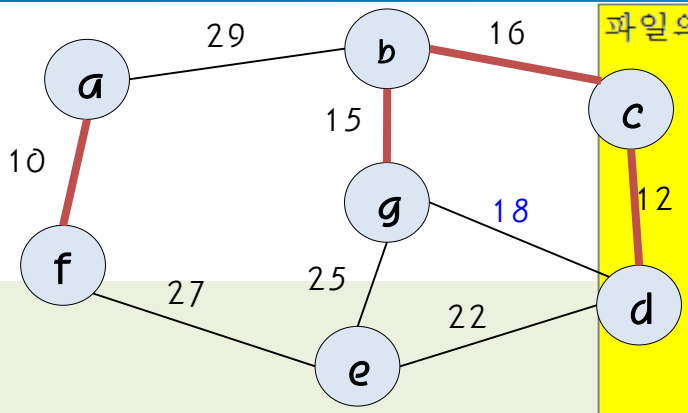
4.(n-1)개의 edge가 삽입될때까지

2.가장 작은 edge 선택

가장 작은 Edge를 삽입할 때 Cycle 체크하고 삽입

3.사이클 생기지 않으면 MST삽입

main()




```
// class WGraphMST

void main()
{
    WGraphMST g;
    g.load("graph.txt");

    printf("MST By Kruskal's Algorithm\n");
    g.Kruskal();
    system("pause");
}
```

```
과일의 형식은 다음과 같음. (graph.txt)
 7
A   9999   29 9999 9999 9999   10 9999
B     29 9999   16 9999 9999 9999   15
C   9999   16 9999   12 9999 9999 9999
D   9999 9999   12 9999   22 9999   18
E   9999 9999 9999   22 9999   27   25
F     10 9999 9999 9999   27 9999 9999
G   9999   15 9999   18   25 9999 9999
```

```
MST By Kruskal's Algorithm
  1   2   3   4   5   6   7
 -1  -1  -1  -1  -1  -1  -1

간선 추가 : A - F (비용:10)
  1   2   3   4   5   6   7
  6  -1  -1  -1  -1  -1  -1

간선 추가 : C - D (비용:12)
  1   2   3   4   5   6   7
  6  -1   4  -1  -1  -1  -1

간선 추가 : B - G (비용:15)
  1   2   3   4   5   6   7
  6   7   4  -1  -1  -1  -1

간선 추가 : B - C (비용:16)
  1   2   3   4   5   6   7
  6   7   4  -1  -1  -1   4

간선 추가 실패 : D - G (비용:18) (싸이클 생성)
  1   2   3   4   5   6   7
  6   7   4  -1  -1  -1   4

간선 추가 : D - E (비용:22)
  1   2   3   4   5   6   7
  6   7   4   5  -1  -1   4

간선 추가 실패 : E - G (비용:25) (싸이클 생성)
  1   2   3   4   5   6   7
  6   7   4   5  -1  -1   4

간선 추가 : E - F (비용:27)
  1   2   3   4   5   6   7
  6   7   4   5   6  -1   4
```

```
E:₩2020-2강의₩자료구조실습₩실습₩MST
입력 그래프: graph.txt
간선 추가 : A - E (비용:10)
간선 추가 : C - D (비용:12)
간선 추가 : B - G (비용:15)
간선 추가 : B - C (비용:16)
간선 추가 : D - E (비용:22)
간선 추가 : E - F (비용:27)
```

최소 스패닝 트리

https://www.acmicpc.net/problem/1197

시간 제한	메모리 제한	제출	정답	맞은 사람	정답 비율
2 초	128 MB	17541	7159	3817	37.717%

문제

그래프가 주어졌을 때, 그 그래프의 최소 스패닝 트리를 구하는 프로그램을 작성하시오.

최소 스패닝 트리는, 주어진 그래프의 모든 정점들을 연결하는 부분 그래프 중에서 그 가중치의 합이 최소인 트리를 말한다.

입력

첫째 줄에 정점의 개수 V(1 ≤ V ≤ 10,000)와 간선의 개수 E(1 ≤ E ≤ 100,000)가 주어진다. 다음 E개의 줄에는 각 간선에 대한 정보를 나타내는 세 정수 A, B, C가 주어진다. 이는 A번 정점과 B번 정점이 가중치 C인 간선으로 연결되어 있다는 의미이다. C는 음수일 수도 있으며, 절댓값이 1,000,000을 넘지 않는다.

최소 스패닝 트리의 가중치가 -2147483648보다 크거나 같고, 2147483647보다 작거나 같은 데이터만 입력으로 주어진다.

출력

첫째 줄에 최소 스패닝 트리의 가중치를 출력한다.

예제 입력 1 복사

```
3 3
1 2 1
2 3 2
1 3 3
```

예제 출력 1 복사

```
3
```

문제 번호	제목	정보	맞은 사람	제출	정답 비율
1197	최소 스패닝 트리	분류	5828	25321	39.288%
1922	네트워크 연결	분류	5683	17058	56.195%
1647	도시 분할 계획	분류	2140	6070	49.320%
2887	행성 터널	출처 다국어 분류	1583	5849	37.664%
17472	다리 만들기 2	분류	1568	8717	28.030%
4386	별자리 만들기	스페셜 저지 출처 다국어 분류	1239	2970	52.723%
6497	전력난	출처 다국어 분류	1180	4781	32.311%
1774	우주신과의 교감	출처 다국어 분류	954	5015	28.822%
1944	복제 로봇	분류	431	2656	24.899%
13418	학교 탐방하기	출처 분류	380	1475	34.862%
16398	행성 연결	출처 분류	375	1414	38.071%
14621	나만 안되는 연애	출처 분류	326	825	49.023%
10423	전기가 부족해	출처 분류	299	624	67.191%
4792	레드 블루 스패닝 트리	출처 다국어 분류	280	1482	28.513%
1626	두 번째로 작은 스패닝 트리	분류	274	4768	11.836%
1396	크루스칼의 공	분류	225	1232	30.738%
9373	복도 뚫기	스페셜 저지 출처 다국어 분류	219	1877	21.878%
13905	세부	출처 분류	212	840	31.361%

가중치 그래프

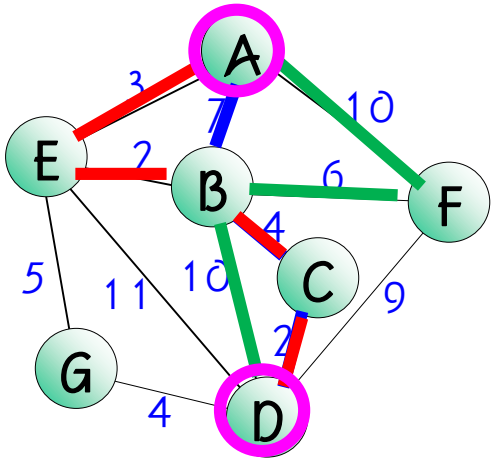
- **Minimum Spanning Tree**
- **Shortest Path**

최단경로 SHORTEST PATH

- **최단경로 문제**
  - **두 정점을 연결하는 여러 경로들 중에서 간선들의 가중치 합이 최소가 되는 경로를 찾는 문제**
  - 가중치는 비용, 거리, 시간 등

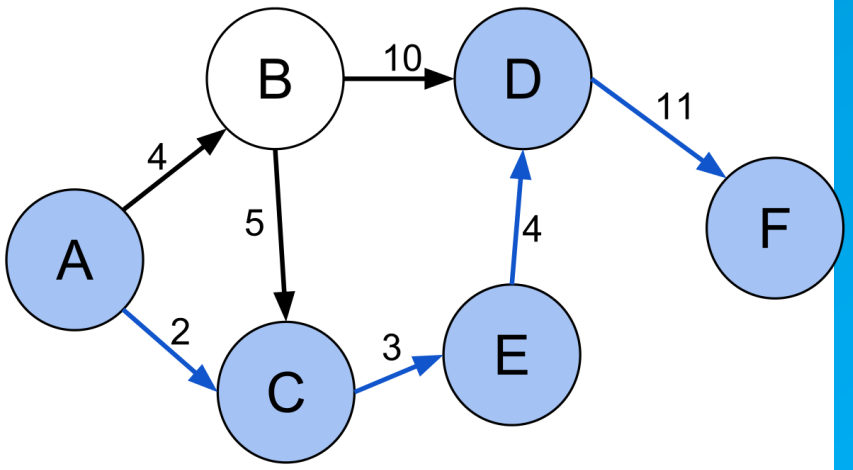
최단경로 SHORTEST PATH

- **A에서 D까지의 최단경로는 ?**
  - 경로1: (A,B,C,D): 비용 = 7 + 4 + 2 = 13
  - 경로2: (A,E,B,C,D): 비용 = 3 + 2 + 4 + 2 = 11
  - 경로3: (A,F,B,D): 비용 = 10 + 6 + 10 = 26



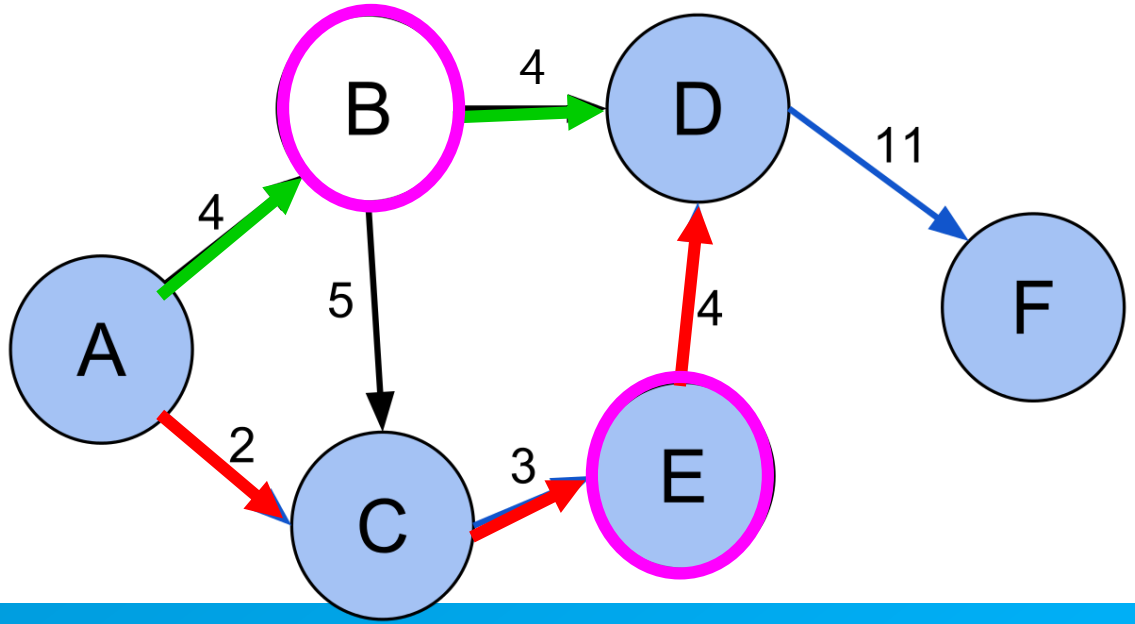
최단 경로 알고리즘

- **Single-Source Shortest Paths**
  - one vertex v to all paths
  - Dijkstra
- **All-Pairs Shortest Paths**
  - all to all
  - Floyd-Warshall



다익스트라 Dijkstra Algorithm

- **Shortest Path Problem**
  - 시작 정점 v에서 다른 모든 정점까지의 최단 경로를 찾는 알고리즘
- **idea**
  - 적은 가중치 먼저 선택
  - 이미 찾은 최단경로 A-C-E를 이용하여 최단경로 A-C-E-D를 찾는다

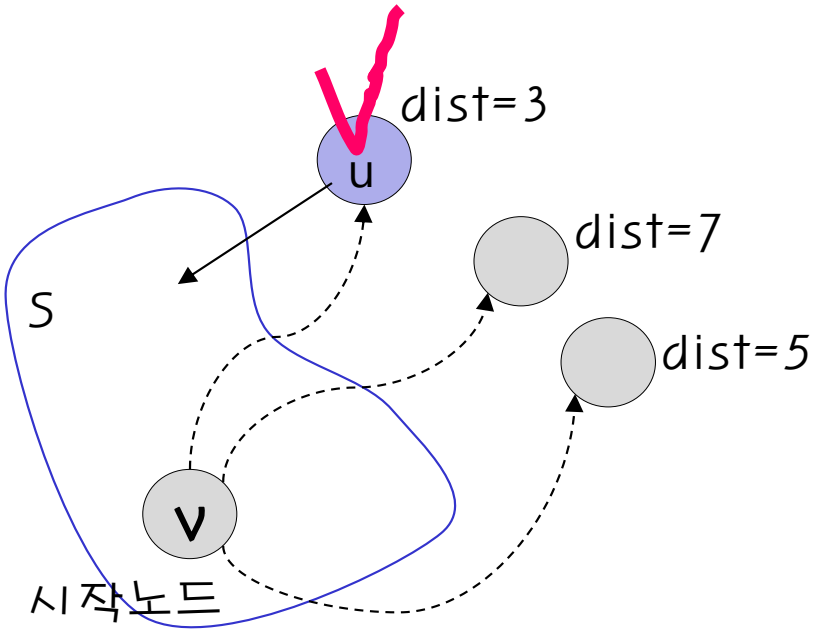


다익스트라 Dijkstra Algorithm

- **시작 정점 v에서 다른 모든 정점까지의 최단 경로를 찾는 알고리즘**
  - **시작 정점 v**
  - **집합 S** : v에서부터의 최단경로를 이미 찾은 정점들의 집합
  - **dist[u] 배열**: S에 있는 정점들 만을 거쳐서 가는 v-u 간 경로의 길이
  - 초기값
    - S={v}
    - dist[u] = v-u간에 간선이 있으면 weight(v,u) 그렇지 않으면 무한대
  - 아이디어
    - 매 단계에서 S안에 있지 않은 정점들 중에서 가장 dist 값이 작은 정점을 S에 추가
    - S에 포함되지 않은 남은 정점들의 dist 값 갱신

다익스트라 Dijkstra Algorithm

- S에 새로운 정점 u를 추가하고
- S에 있지 않은 다른 정점들 w까지의 기존 경로와 u를 거쳐가는 경로 중에 shortest path 조사
- S에 있지 않은 다른 정점들의 distance 값 갱신



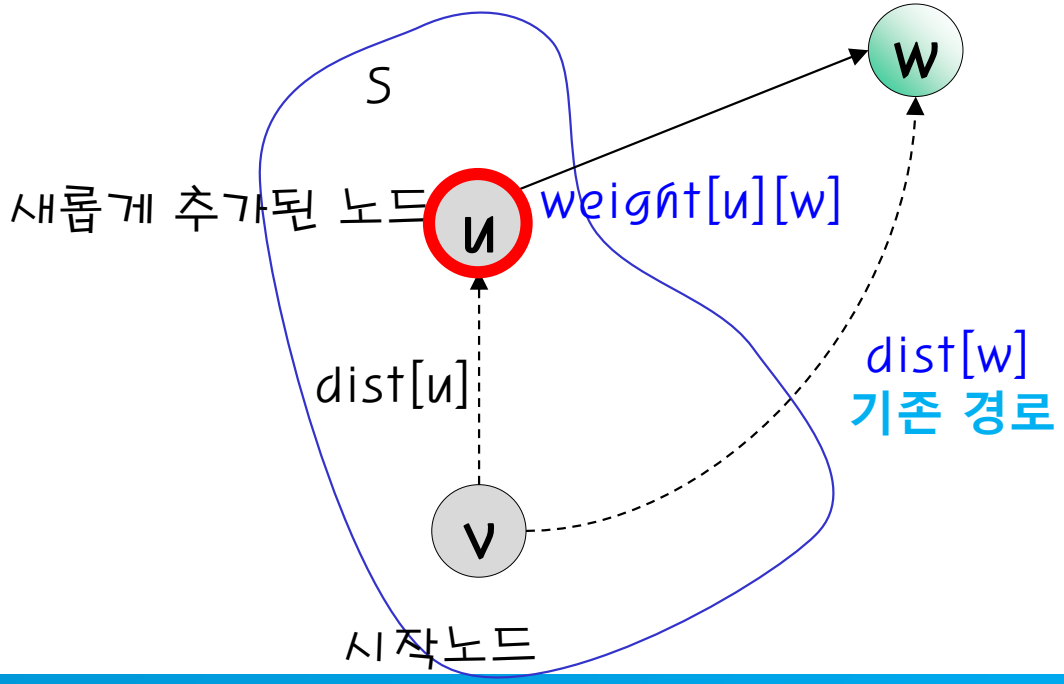
다익스트라 Dijkstra Algorithm

- S에 새로운 정점 u를 추가하고
- S에 있지 않은 다른 정점들 w까지의 기존 경로와 u를 거쳐가는 경로 중에 shortest path 조사
- S에 있지 않은 다른 정점들의 distance 값 갱신

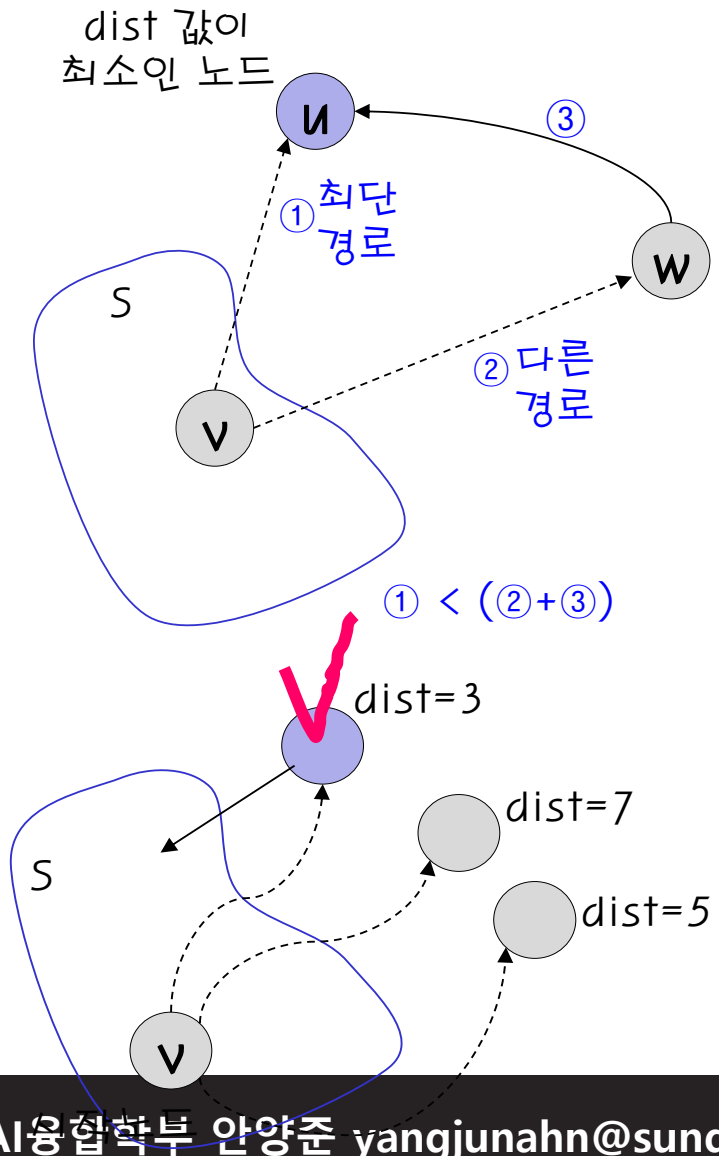
기존 경로

dist[w] = min(dist[w], dist[u]+weight[u][w])

노드 u를 거쳐가는 경로가 최단 경로인지 조사



다익스트라 알고리즘 증명



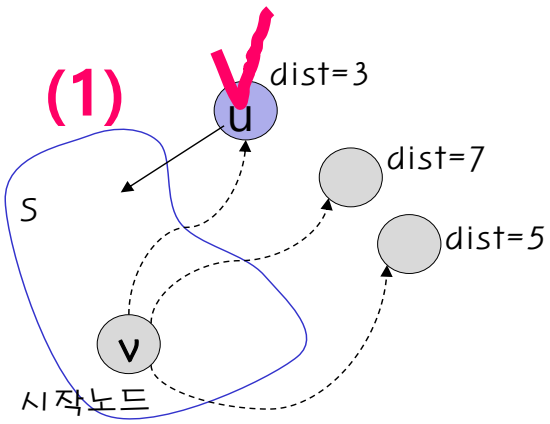
- S에 속하지 않은 정점 중에 distance 값이 가장 작은 정점 u 선택
- v에서 u까지의 최단경로는 ①
- 이때, S에 속하지 않는 w를 거쳐가는 더 짧은 경로가 있다고 가정하자.
- 매 단계에서 distance 값이 가장 작은 정점들을 추가하면 모든 정점까지의 최단거리를 구할 수 있다.
- 따라서, shortest path는 정점 u를 거쳐간다.
  shortest paths : S = S + {u}

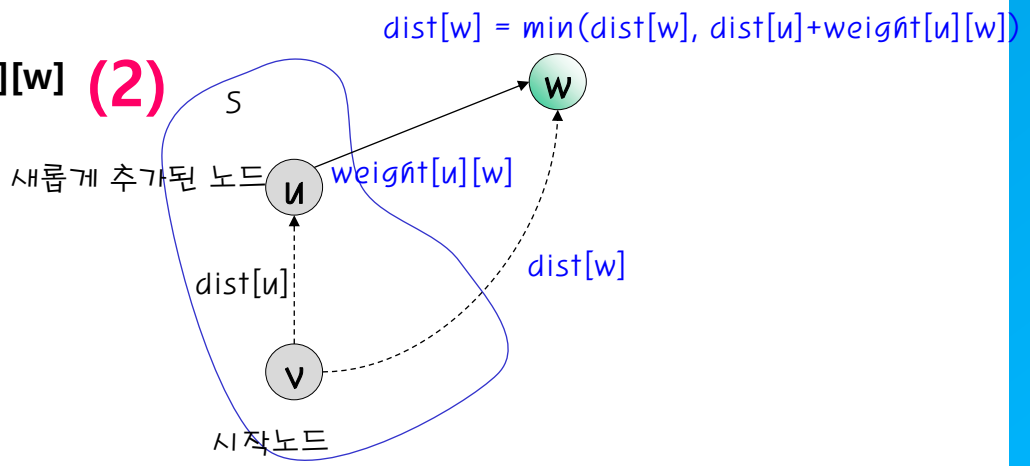
Algorithm:
Dijkstra's Shortest Path

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

pseudo code : Dijkstra Algorithm

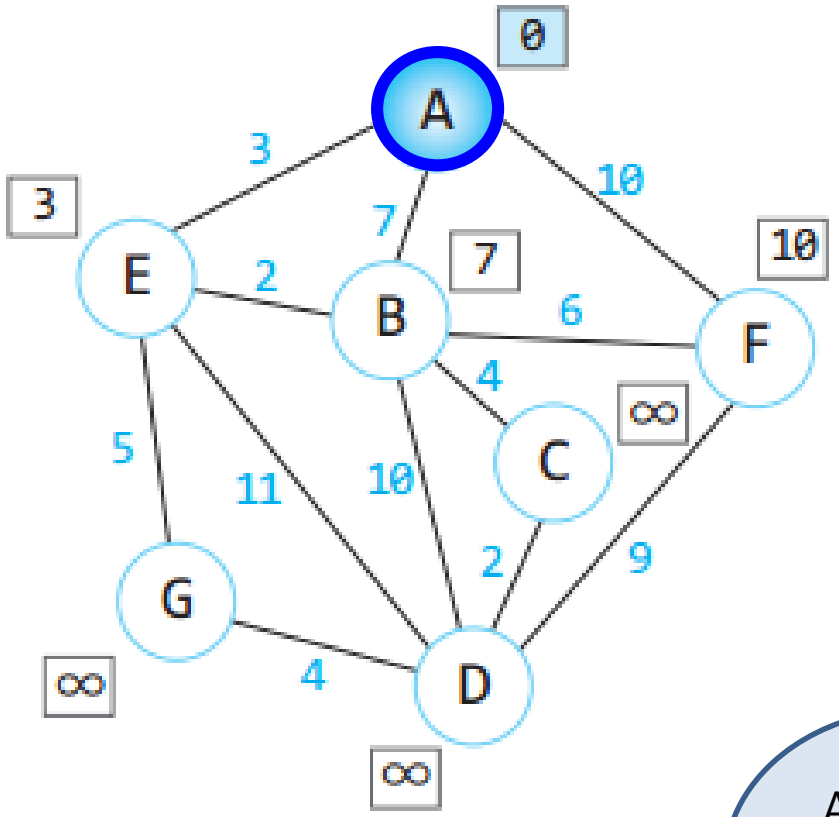
```
shortest_path_dijkstra(v)
S← {v}
for 각 정점 w in G do
    dist[w] = weight[v][w]
    while 모든 정점이 S에 포함되지 않으면 do
(1)  u ← 집합 S에 속하지 않는 정점 중에서 최소 dist를 갖는 정점 선택
(2)  S ← S union {u}
        for u에 인접하고 S에 있지 않은 모든 정점 w do
            if ( dist[u] + weight[u][w] < dist[w] )
                then dist[w] ← dist[u] + weight[u][w]
```





알고리즘 실행 과정: Step1

- starting from A



시작 정점
A를 집합
S에 추가

S={A}

dist(A)=w(A,A)=0
dist(B)=w(A,B)=7
dist(C)=w(A,C)=∞
dist(D)=w(A,D)=∞
dist(E)=w(A,E)=3
dist(F)=w(A,F)=10
dist(G)=w(A,G)=∞

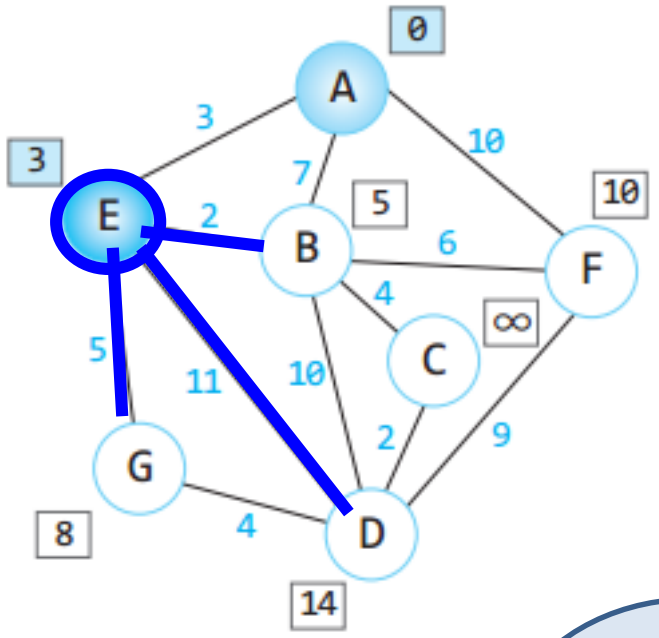
가장 가까운 곳으로 감
Next vertex ?
E

A에 인접
한 정점
B,E,F

알고리즘 실행 과정: Step2

● 노드 E 추가

정점 E를 집합 S에 추가



S={A, E} dist(E)=3

~~dist(A)=0~~

dist(B)=min(dist(B),dist(E)+w(E,B))=min(7,3+2)=5

dist(C)=min(dist(C),dist(E)+w(E,C))=min(∞,3+∞)=∞

dist(D)=min(dist(D),dist(E)+w(E,D))=min(∞,3+11)=14

~~dist(E)=w(A,E)=3~~

dist(F)=min(dist(F),dist(E)+w(E,F))=min(10,3+∞)=10

dist(G)=min(dist(G),dist(E)+w(E,G))=min(∞,3+5)=8

update !

E에 인접한 정점 B,D, E,G 갱신

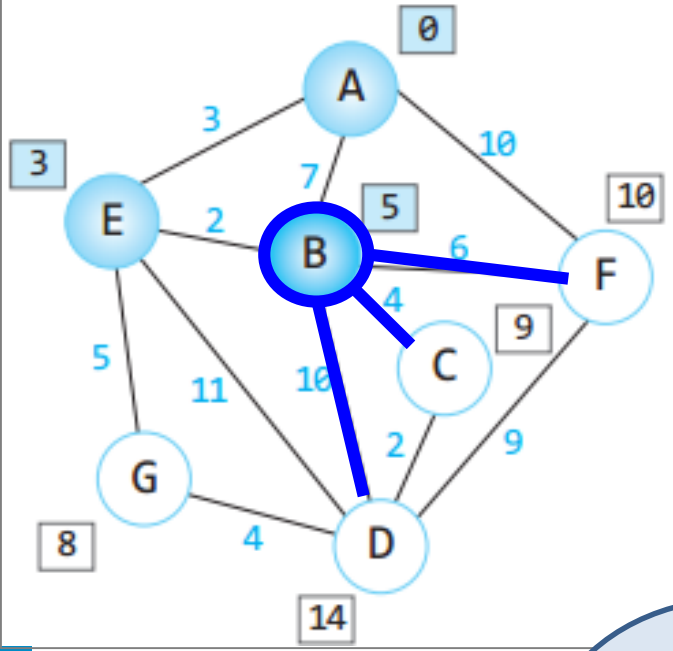
Next vertex ?

B

알고리즘 실행 과정: Step3

● 노드 B 추가

정점 B를 집합 S에 추가



S={A, E, B} dist(B)=5

~~dist(A)=0~~

~~dist(B)=5~~

dist(C)=min(dist(C),dist(B)+w(B,C))=min(∞,5+4)=9

dist(D)=min(dist(D),dist(B)+w(B,D))=min(14,5+10)=14

~~dist(E)=3~~

dist(F)=min(dist(F),dist(B)+w(B,F))=min(10,5+6)=10

dist(G)=min(dist(G),dist(B)+w(B,G))=min(8,5+∞)= 8

B에 인접한 정점 B,C,D,F 갱신

Next vertex ?

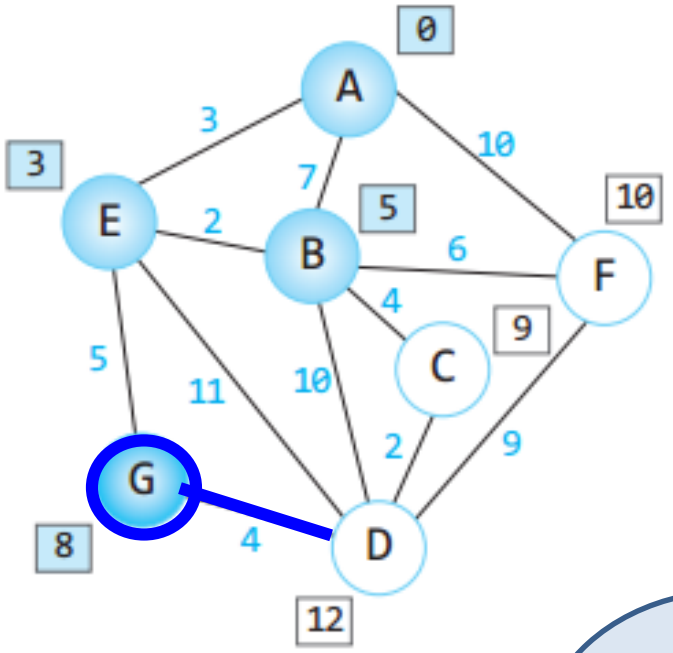
G

B에 인접한 A,E는 이미 집합 S에 포함.
이미 최단경로이므로 갱신 필요없음.

알고리즘 실행 과정: Step4

● 노드 G 추가

정점 G를 집합 S에 추가



S={A, E, B, G} dist(G)=8

~~dist(A)=0~~

~~dist(B)=5~~

dist(C)=min(dist(C),dist(G)+w(G,C))=min(9,8+∞)=9

dist(D)=min(dist(D),dist(G)+w(G,D))=min(14,8+4)=12

~~dist(E)=3~~

dist(F)=min(dist(F),dist(G)+w(G,F))=min(10,8+∞)=10

~~dist(G)=8~~

Next vertex ?

C

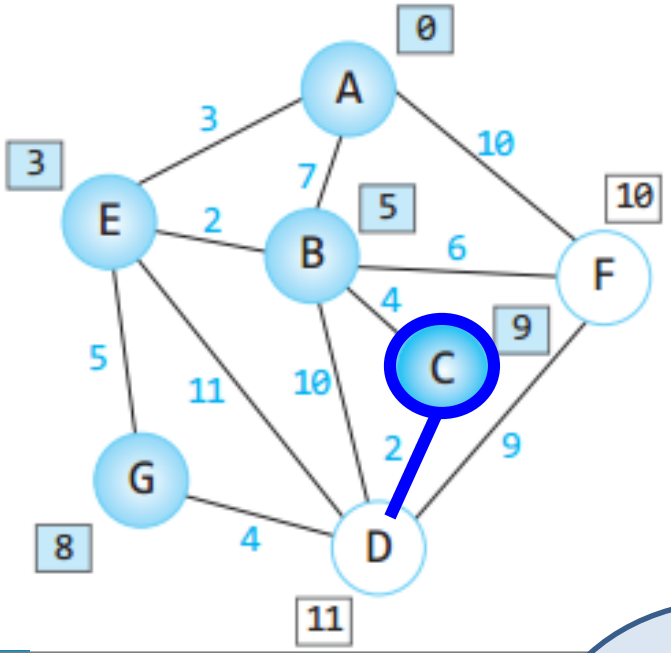
G에 인접한 정점 G,D 갱신

G에 인접한 E는 이미 집합 S에 포함.
이미 최단경로이므로 갱신 필요없음.

알고리즘 실행 과정: Step5

● 노드 C 추가

정점 C를 집합 S에 추가



S={A, E, B, G, C} dist(C)=9

~~dist(A)=0~~
~~dist(B)=5~~
~~dist(C)=9~~
dist(D)=min(dist(D),dist(C)+w(C,D))=min(12,9+2)=11
~~dist(E)=3~~
dist(F)=min(dist(F),dist(C)+w(C,F))=min(10,9+∞)=10
~~dist(G)=8~~

C에 인접한 정점 C,D 갱신

Next vertex ?

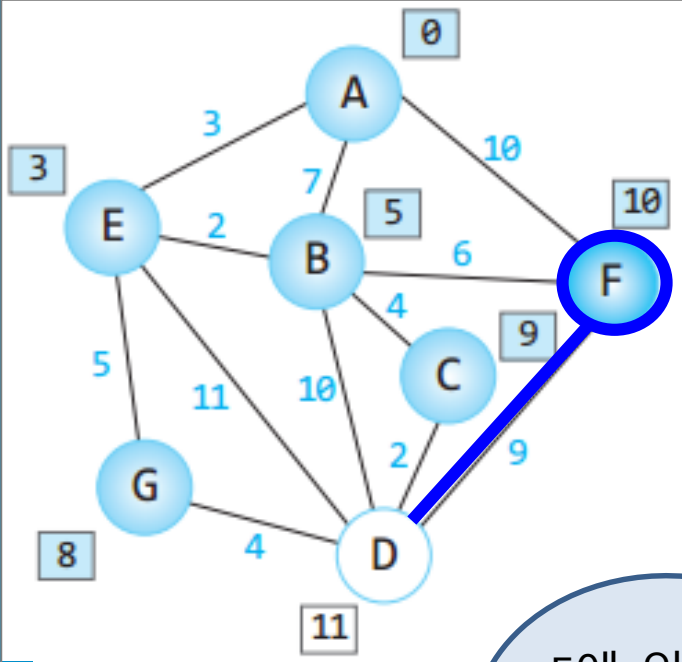
F

C에 인접한 B는 이미 집합 S에 포함.
이미 최단경로이므로 갱신 필요없음.

알고리즘 실행 과정: Step6

● 노드 F 추가

정점 F를 집합 S에 추가



S={A, E, B, G, C, F} dist(F)=10

~~dist(A)=0~~
~~dist(B)=5~~
~~dist(C)=9~~
dist(D)=min(dist(D),dist(F)+w(F,D))=min(11,10+9)=11
~~dist(E)=3~~
~~dist(F)=10~~
~~dist(G)=8~~

F에 인접한 정점 D,F 갱신

Next vertex ?

D

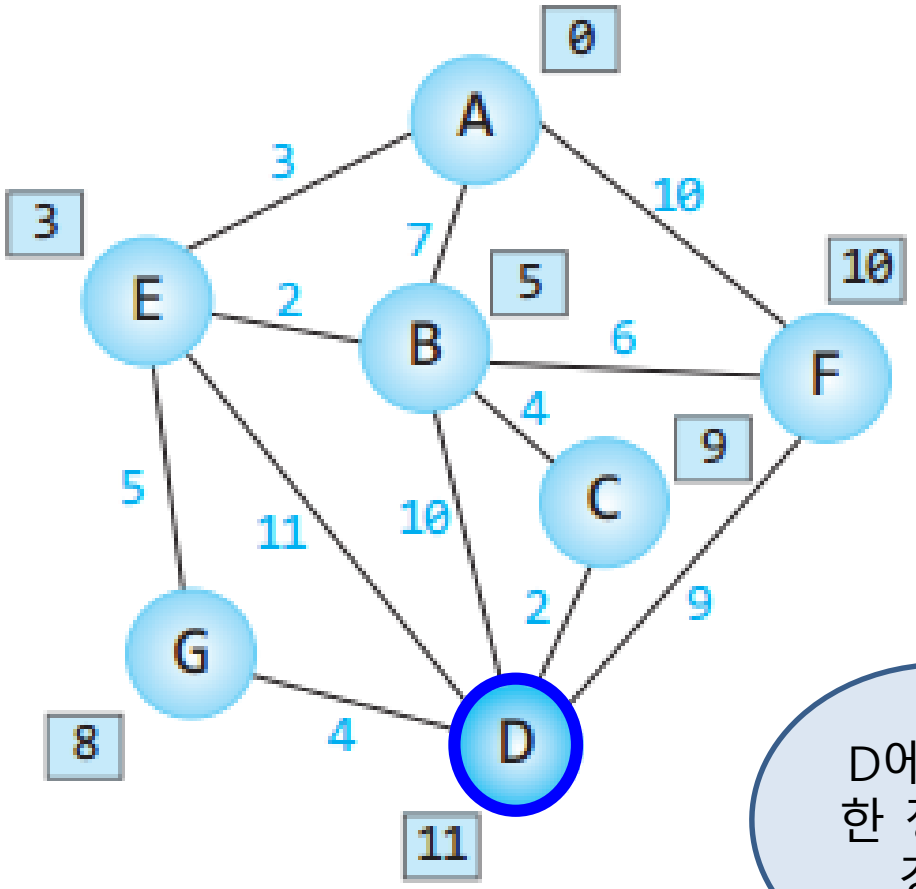
F에 인접한 A,B는 이미 집합 S에 포함.
이미 최단경로이므로 갱신 필요없음.

알고리즘 실행 과정: 7

● 노드 D 추가

정점 D를 집합 S에 추가

dist(D)=11



S={A, E, B, G, C,F,D}

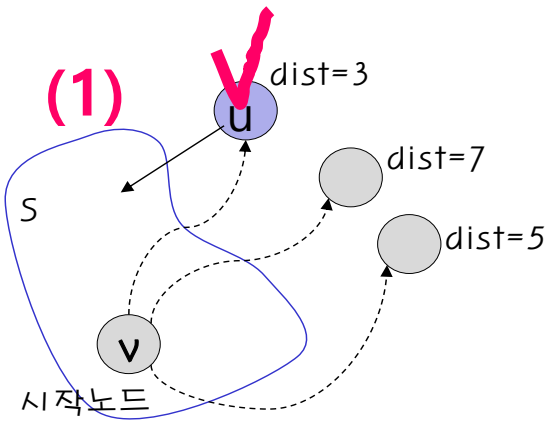
dist(A)=0
dist(B)=5
dist(C)=9
dist(D)=11
dist(E)=3
dist(F)=10
dist(G)=8

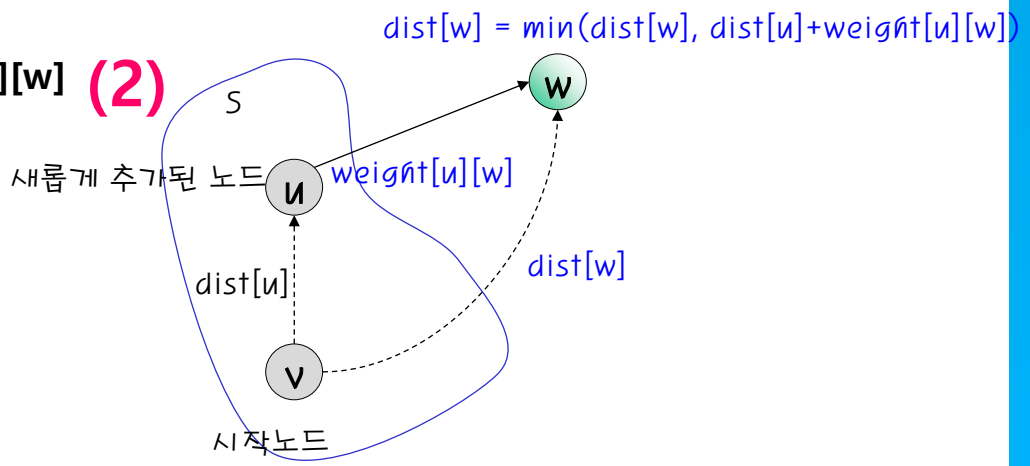
D에 인접한 정점 D 갱신

D에 인접한 B,C,E,F,G는 이미 집합 S에 포함.

pseudo code : Dijkstra Algorithm

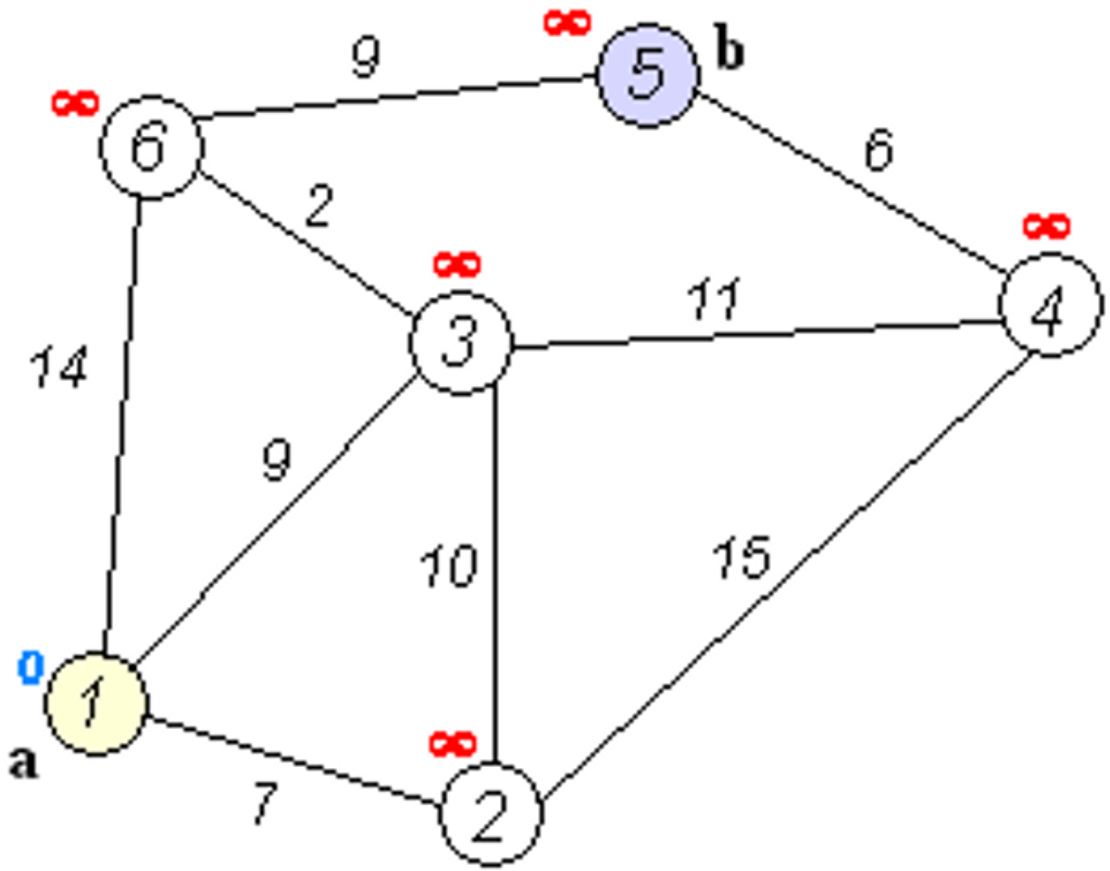
```
shortest_path_dijkstra(v)
S← {v}
for 각 정점 w in G do
    dist[w] = weight[v][w]
    while 모든 정점이 S에 포함되지 않으면 do
(1)  u ← 집합 S에 속하지 않는 정점 중에서 최소 dist를 갖는 정점 선택
(2)  S ← S union {u}
        for u에 인접하고 S에 있지 않은 모든 정점 w do
            if ( dist[u] + weight[u][w] < dist[w] )
                then dist[w] ← dist[u] + weight[u][w]
```





예제

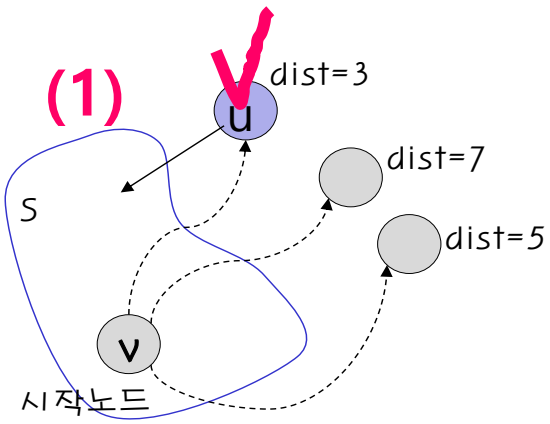
Dijkstra 알고리즘 각 단계에서의 집합 S와 dist(정점)의 값을 구하세요.

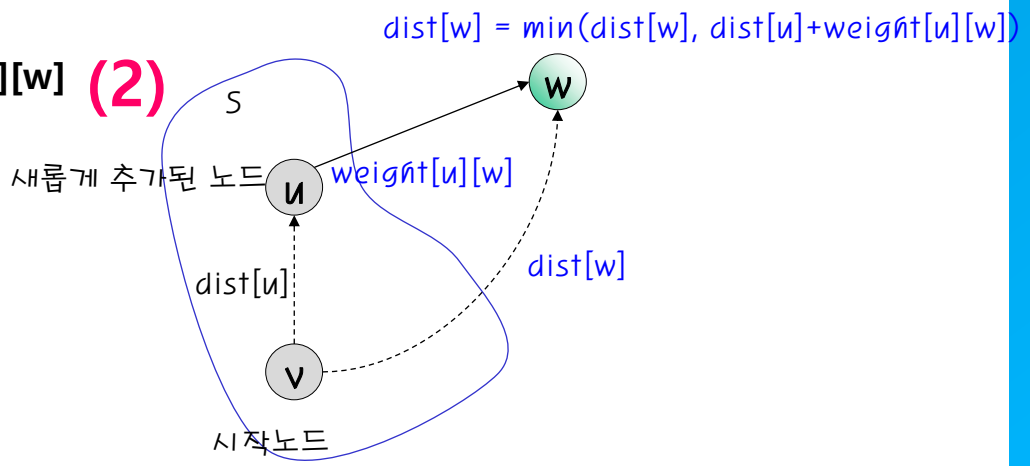


CODE: Dijkstra's Shortest Path

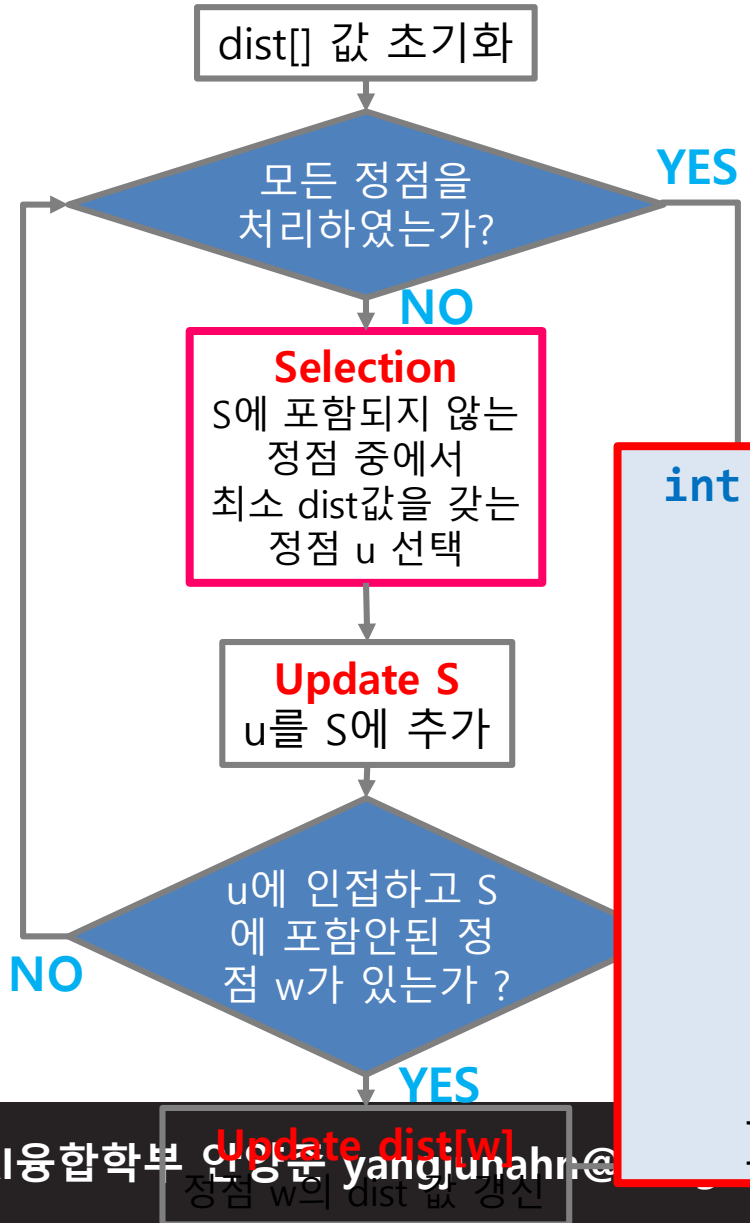
pseudo code : Dijkstra Algorithm

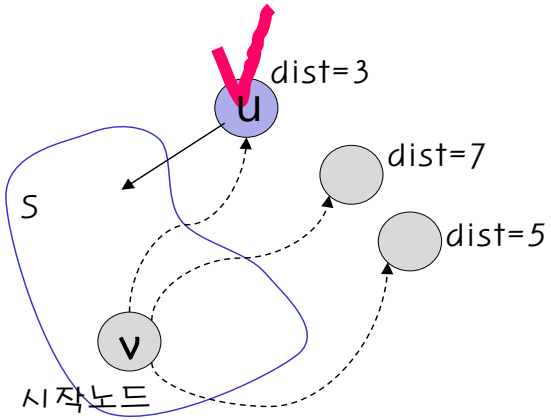
```
shortest_path_dijkstra(v)
S← {v}
for 각 정점 w in G do
    dist[w] = weight[v][w]
    while 모든 정점이 S에 포함되지 않으면 do
(1)  u ← 집합 S에 속하지 않는 정점 중에서 최소 dist를 갖는 정점 선택
(2)  S ← S union {u}
        for u에 인접하고 S에 있지 않은 모든 정점 w do
            if ( dist[u] + weight[u][w] < dist[w] )
                then dist[w] ← dist[u] + weight[u][w]
```





pseudo code : Dijkstra Algorithm

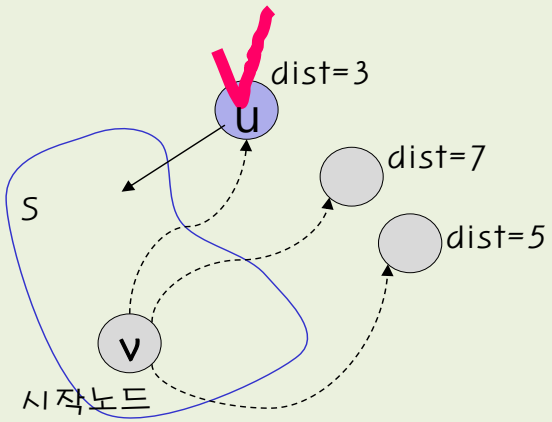





```
int chooseVertex() {// 가장 비용 적은 미방문 정점을 반환
        int min = INF; //최소값 구하기 위해 큰수로 초기화
        int minpos = -1;//정점 u 초기화
        //모든 정점에 대해서: size=정점의 개수
        //최소 dist 값을 찾는다. (S에 포함되지 않는 정점 중에서)
        for( int i=0 ; i<size ; i++ )
            if( dist[i]< min && !found[i] )
            {
                min = dist[i]; //최소값 갱신
                minpos = i; //최소값 갖는 정점 u 저장
            }
        return minpos; //최소값 정점 u 반환
    }
```

Dijkstra의 최단경로 코드

```
// Dijkstra알고리즘의 최단 경로 탐색 기능이 추가된 그래프
class WGraphDijkstra : public WGraph {
    int  dist[MAX_VTXS];     // 시작노드로부터의 최단경로 거리
    bool found[MAX_VTXS];    // 방문한 정점 표시 → 집합 S
public:
    int chooseVertex() {    // 가장 비용 적은 미방문 정점을 반환
        int min = INF;
        int minpos = -1;
        for( int i=0 ; i<size ; i++ )
            if( dist[i]< min && !found[i] ){
                min = dist[i];
                minpos = i;
            }
        return minpos;
    }
    void printDistance() { //모든 정점들의 dist[] 값 출력
        for( int i=0 ; i<size ; i++)
            printf("%5d", dist[i]);
        printf("\n");
    }
```



Dijkstra의 최단경로 코드

```
    // Dijkstra의 최단 경로 알고리즘: start 정점에서 시작함.
    void ShortestPath( int start ) {
        for( int i=0 ; i<size ; i++) {  //초기화: dist[], found[]
            dist[i] = getEdge(start,i); //인접행렬 값 반환(간선 가중치)
            found[i] = false;           //처음에 S집합은 비어있음.
        }

        found[start] = true;  // S에 포함
        dist[start] = 0;      // 최초 거리
        for( int i=0 ; i<size ; i++ ){
            printf("Step%2d:", i+1);
            printDistance();          // 모든 dist[] 배열값 출력
            int u = chooseVertex(); // S에 속하지 않은 비용 가장 작은 정점 반환
            found[u] = true;          // 집합 S에 포함
            for( int w=0 ; w<size ; w++) {
                if( found[w] == false )//S에 속하지 않는 노드 w의 dist값 갱신
                    if( dist[u] + getEdge(u,w) < dist[w] )
                        dist[w] = dist[u] + getEdge(u,w);
            }       // u를 거쳐가는 것이 최단 거리이면 dist[] 갱신
        }
    }
};
```

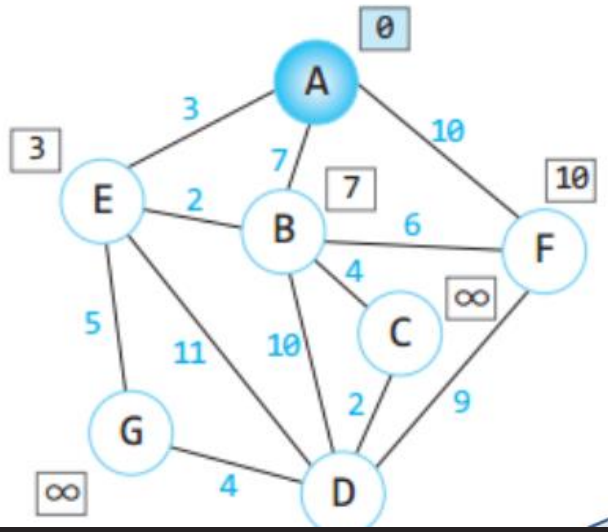
Selection

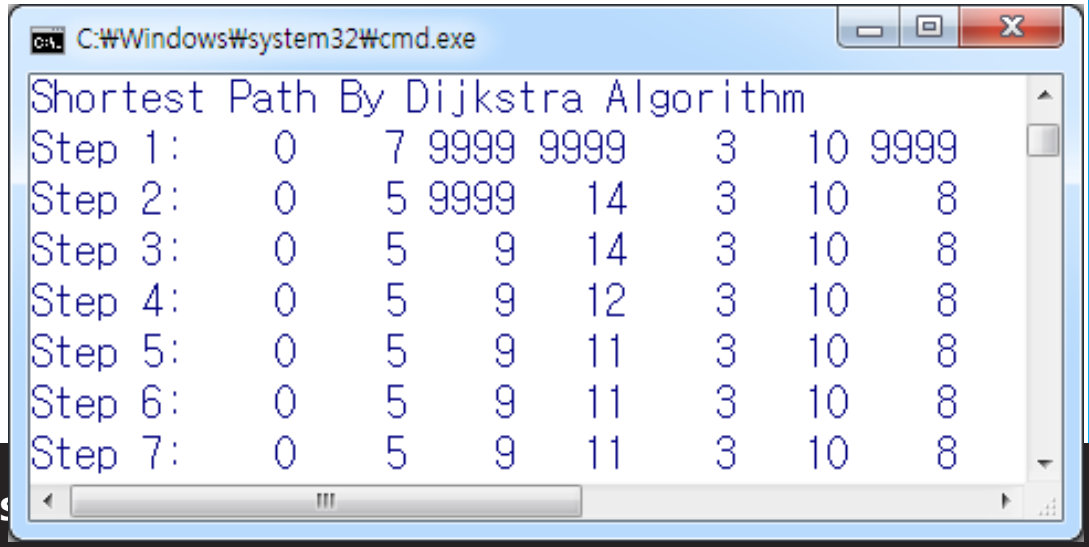
Update S

Update dist[w]

Dijkstra의 최단경로 테스트

```
// class WGraphDijkstra
void main()
{
        WGraphDijkstra g;

        g.load( "graph.txt" );
        printf("Dijkstra의 최단경로 탐색을 위한 그래프: graph_sp.txt\n");
        g.display();

        printf("Shortest Path By Dijkstra Algorithm\n");
        g.ShortestPath( 0 );
}
```

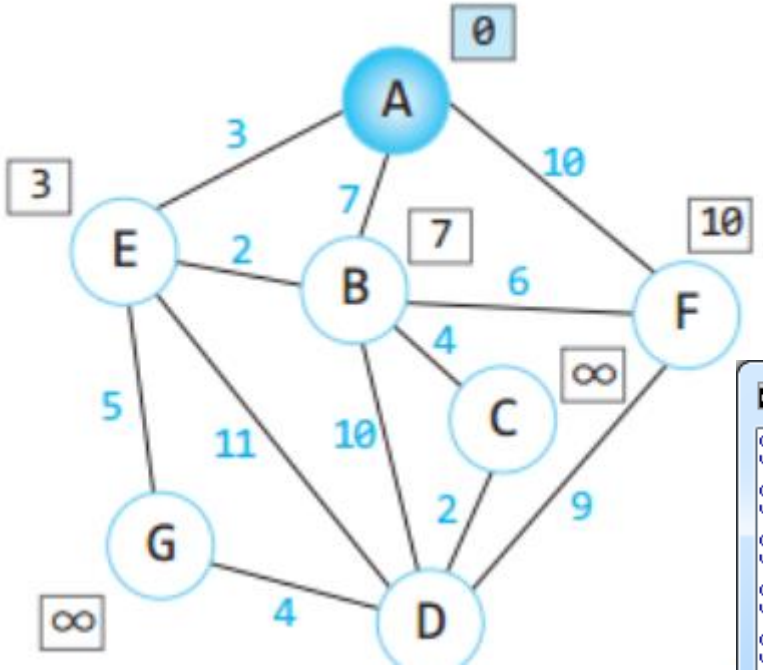




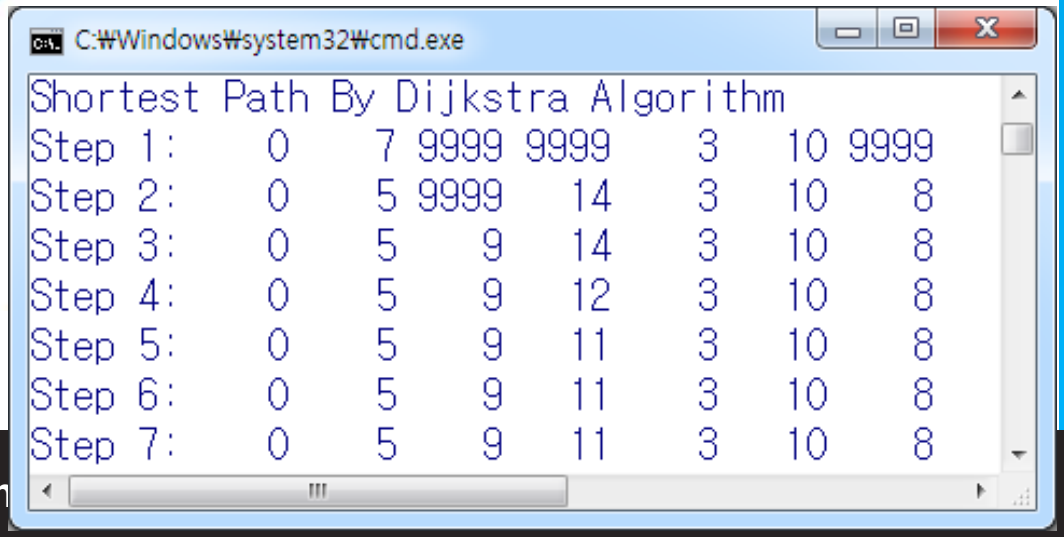
Self practice(1): test case 1

- **Step by Step 테스트 해보기**

Test case 1. 강의 교재



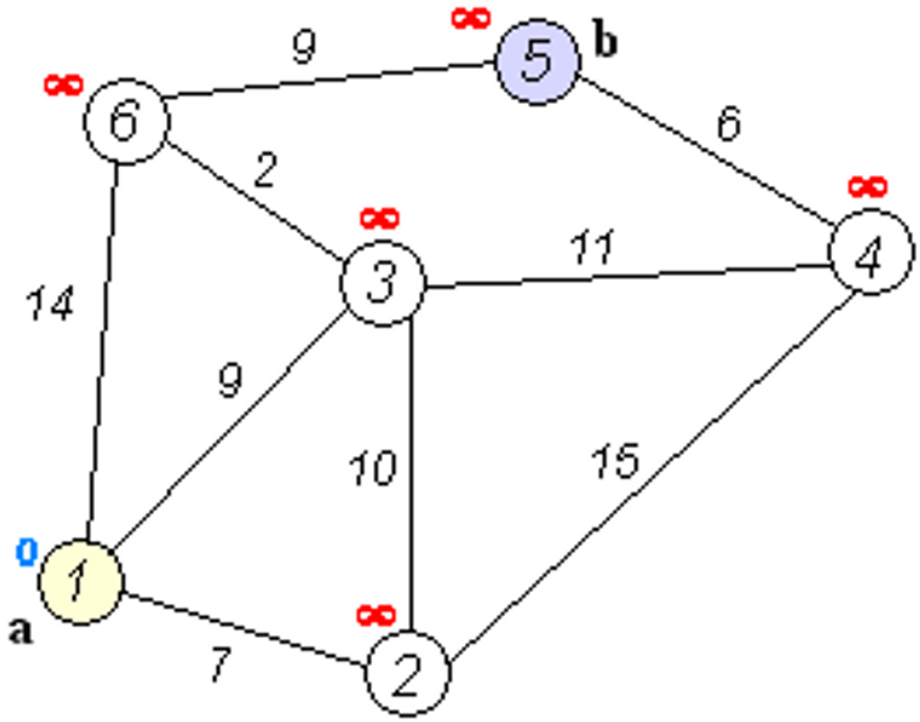
	A	B	C	D	E	F	G
A	0	7	∞	∞	3	10	∞
B	7	0	4	10	2	6	∞
C	∞	4	0	2	∞	∞	∞
D	∞	10	2	0	11	9	4
E	3	2	∞	11	0	∞	5
F	10	6	∞	9	∞	0	∞
G	∞	∞	∞	4	5	∞	0



Self practice(1): test case 2

- Step by Step 테스트 해보기

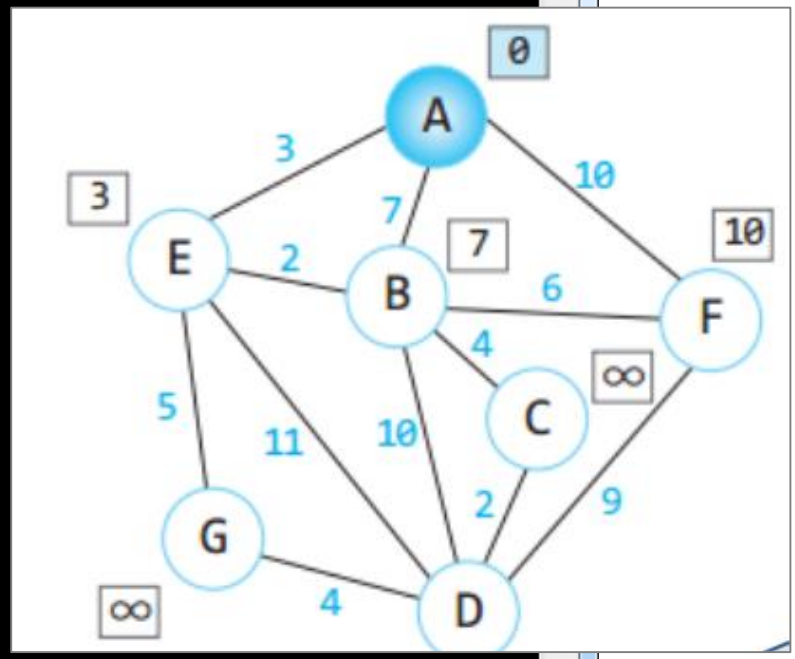
Test case 2. 예제1



Self practice (2)

Dijkstra's Shortest Path 알고리즘에서 각 정점까지의 shortest path를 출력하세요. (노란색 부분 참조)

```
C:₩변혜원₩2018-강의₩2018-2₩자료구조실습₩강의자료₩소스코드(CPP,PROJ)₩12장-가중치그래...
최단경로 탐색을 위한 그래프: graph_sp.txt
7
A      0    7 9999 9999    3   10 9999
B      7    0    4   10    2    6 9999
C   9999    4    0    2 9999 9999 9999
D   9999   10    2    0   11    9    4
E      3    2 9999   11    0 9999    5
F     10    6 9999    9 9999    0 9999
G   9999 9999 9999    4    5 9999    0
Shortest Path By Dijkstra Algorithm
Step 1:    0    7 9999 9999    3   10 9999
Step 2:    0    5 9999   14    3   10    8
Step 3:    0    5    9   14    3   10    8
Step 4:    0    5    9   12    3   10    8
Step 5:    0    5    9   11    3   10    8
Step 6:    0    5    9   11    3   10    8
Step 7:    0    5    9   11    3   10    8
[최단경로: A<-B] B-E-A
[최단경로: A<-C] C-B-E-A
[최단경로: A<-D] D-C-B-E-A
[최단경로: A<-E] E-A
[최단경로: A<-F] F-A
[최단경로: A<-G] G-E-A
계속하려면 아무 키나 누르십시오 . . .
```



연습문제 5

거리는 d, 최단거리는 δ

방문한 곳 S

아직 방문 X Q

S

Q

s

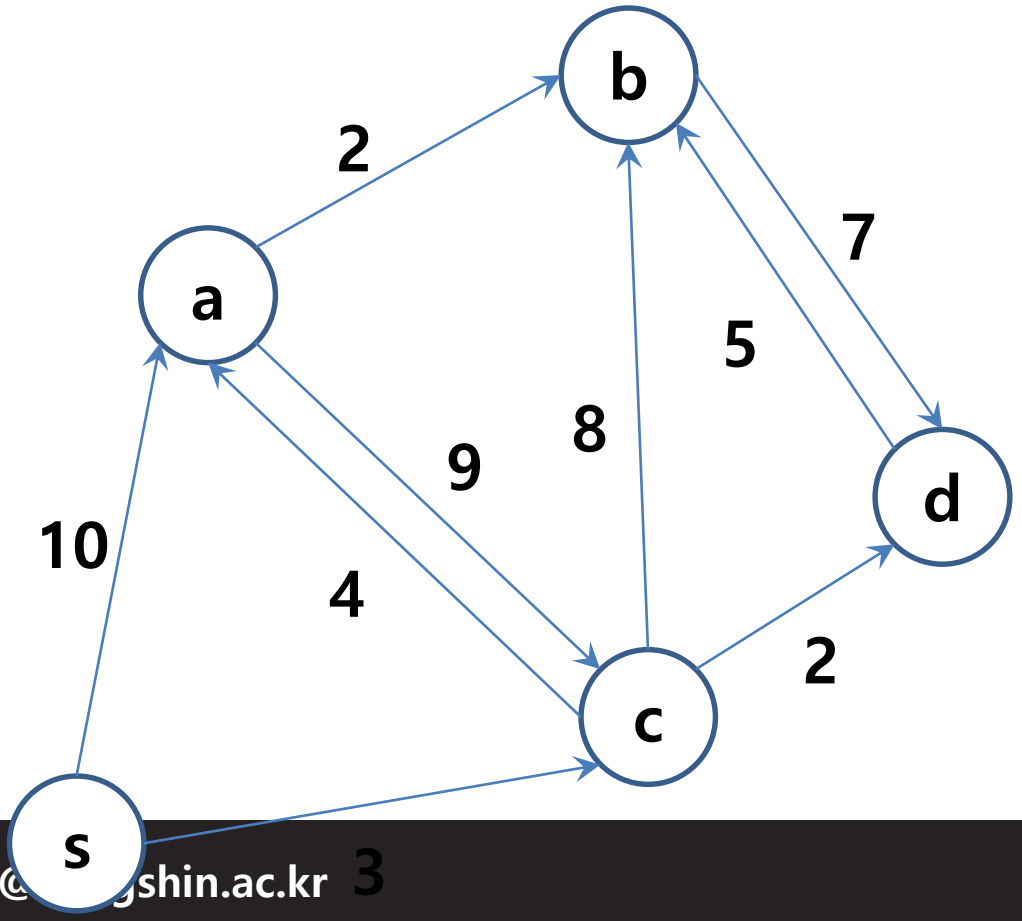
a

b

c

d

V	$\delta(s, v)$
s	
a	
b	
c	
d	



다익스트라 과정
- 거리 초기값은 전부 무한
- 가장 처음 나온 정점을 기준으로 간선이 연결된 정점의 거리 업데이트
- 다음 정점(아직 방문하지 않은 점 중 가장 가까운 곳) 기준으로 간선이 연결된 정점 거리 확인
- 그리고 아까 시작점에서 썼던 거리와 이번에 쓴 거리를 비교해 더 짧은 쪽으로 업데이트
- 이런 식으로 (n)번째 방문한 정점에서 근처 정점까지의 거리를 모두 확인하고 (n - 1)번째 방문한 정점에서 업데이트했던 거리와 크기를 비교해 더 짧은 쪽으로 업데이트

위의 저 과정을 한 번 실행하면 **시작점을 기준으로** 나머지 모든 정점까지의 최단거리를 알 수 있음

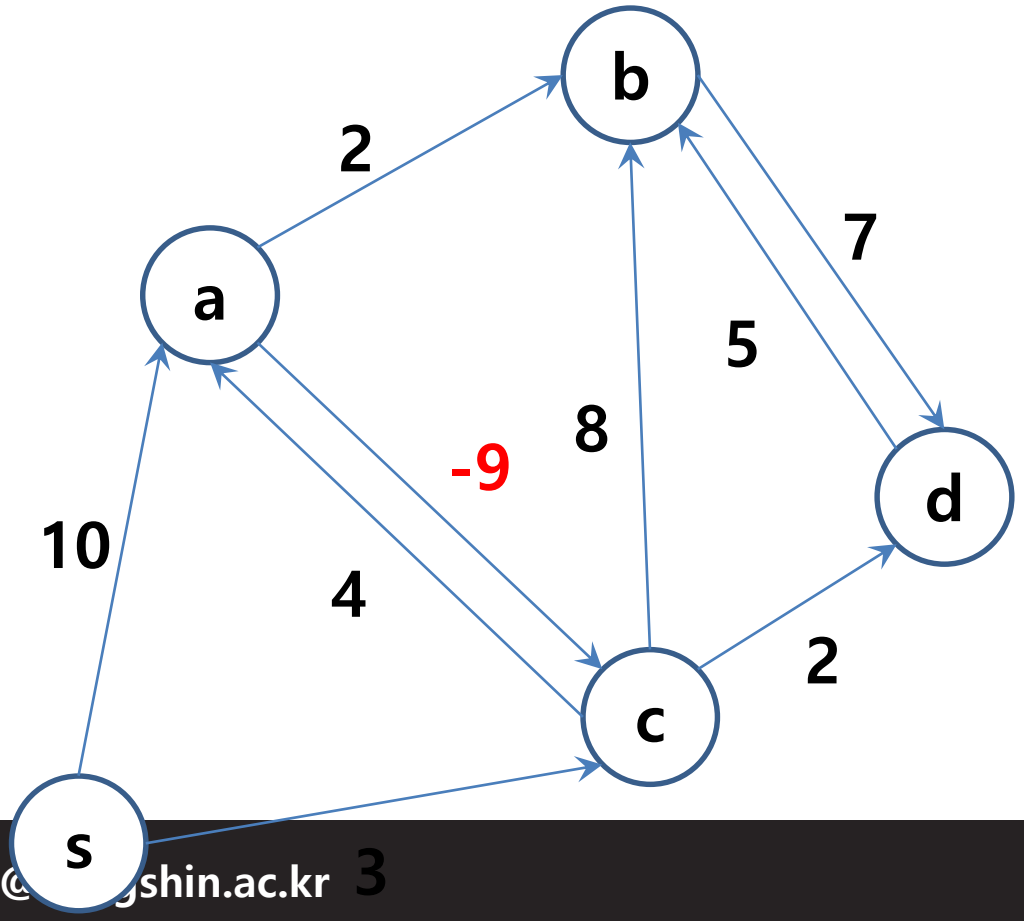
수업이 너무 정신없어서 간단하게 혼자 개념 정리

크루스칼 : 간선 가중치를 모두 정렬한 후, 사이클이 생기지 않게, 가중치가 작은 것부터 경로에 추가한다
프림 : 시작점을 찍고, 아직 방문하지 않은 곳을 향해, 갈 수 있는 한 최대한 가중치가 작은 길만 골라 다닌다
다익스트라 : 시작점을 찍고, 탐색을 시작한다. n번째 방문한 정점에서 갈 수 있는 정점들(간선이 연결됨)까지의 거리는 시작점~n 까지의 거리 + n~그 점까지의 거리이다. 이 거리와 기존에 조사해뒀던 거리(n-1번째 정점을 방문하고 기록한 거리들) 중 짧은 쪽을 기록한다.

연습문제 6

S	Q
	s
	a
	b
	c
	d

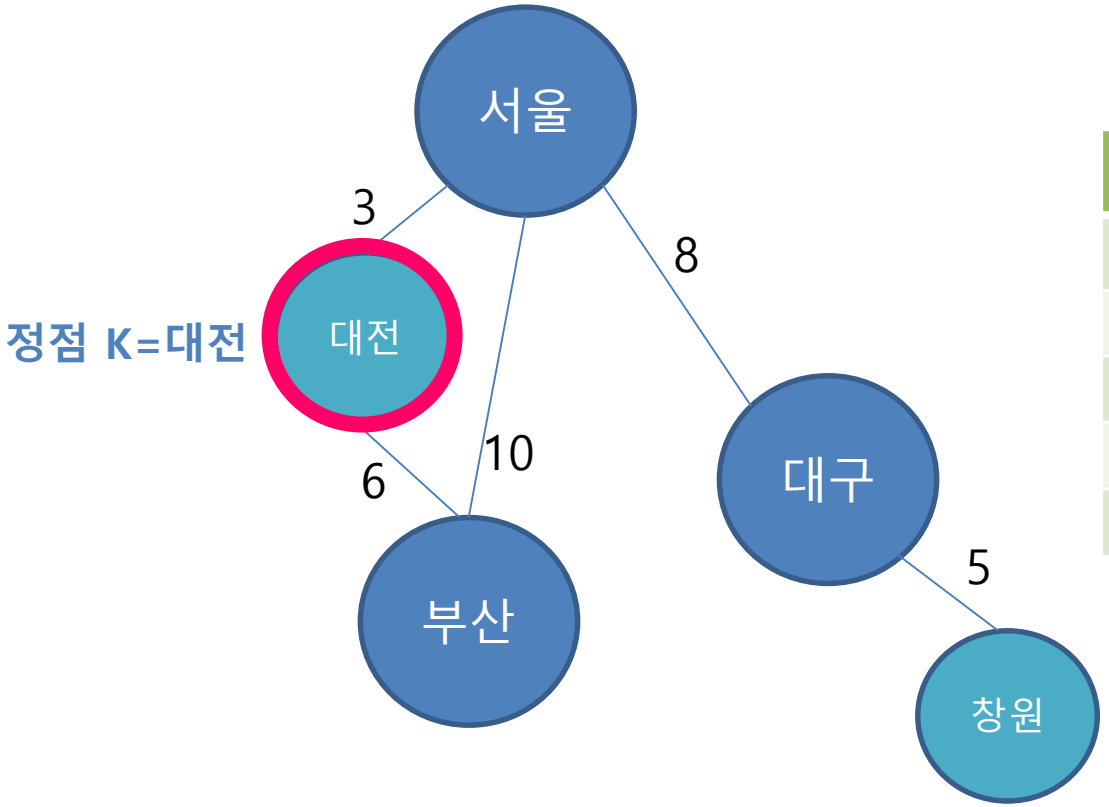
V	$\delta(s, v)$
s	
a	
b	
c	
d	



Dijkstra 알고리즘 시간 복잡도

- **그래프에 n개의 정점이 있다면**
  - for문 내부에 for문 → **$O(n^2)$**
  - Dijkstra의 최단경로 알고리즘은 **$O(n^2)$의 시간 복잡도**
- **Floyd Algorithm**
  - O(n3) 동일
  - 모든 정점 사이의 최단경로를 한꺼번에 찾아준다.

Floyd의 최단경로 알고리즘



	서울	대전	대구	부산	창원
서울	--	3	8	9	*
대전	3	--	*	6	*
대구	8	*	--	*	5
부산	9	6	*	--	*
창원	*	*	5	*	--

***모든 경로 초기화: edge 존재하면 weight 존재하지 않으면 *(무한대)

기본적인 아이디어:

(1) 모든 경로의 비용을 초기화한다.

(2) 정점 k를 거쳐가는 경로의 길이를 계산한다.

(3) 더 적다면 경로를 수정한다.

"경로에 정점 k를 추가하였을 때, 기존 경로의 길이보다 더 짧아질 수 있다."

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

Floyd의 최단경로 알고리즘

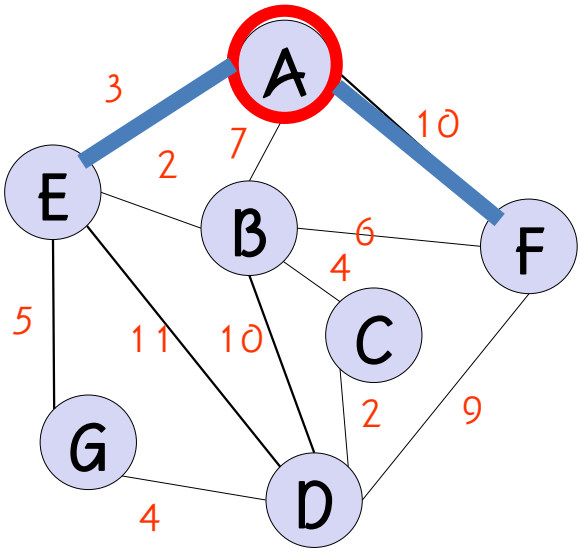
- 모든 정점 사이의 최단경로를 찾는 알고리즘
  - 2차원 배열 A를 이용하여 3중 반복을 하는 루프로 구성
    - 배열 A의 초기 값은 인접 행렬 weight
    - 인접 행렬 weight 구성
      - i==j 이면, weight[i][j]=0
      - 두 정점 i, j 사이에 간선이 존재하지 않으면, weight[i][j]=∞
      - i, j 사이에 간선이 존재하면, weight[i][j]는 간선 (i, j)의 가중치

floyd(G)

```
for k ← 0 to n - 1  // 정점 k를 거쳐가는 경로의 길이가 짧다면 경로 수정
    for i ← 0 to  n - 1 // 모든 정점 (i,j)의 SP 계산
       for j ← 0 to n - 1
             A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

Floyd의 최단경로 알고리즘

k=정점 A를 거쳐가는 경로



	A	B	C	D	E	F	G
A	0	7	INF	INF	3	10	INF
B	7	0	4	10	2	6	INF
C	INF	4	0	2	INF	INF	INF
D	INF	10	2	0	11	9	4
E	3	2	INF	11	0	13	5
F	10	6	INF	9	13	0	INF
G	INF	INF	INF	4	5	INF	0

무한대 **E-F** E-A-F (13)

기본적인 아이디어:

(1) 모든 정점간의 SP를 기본 비용으로 초기화하고

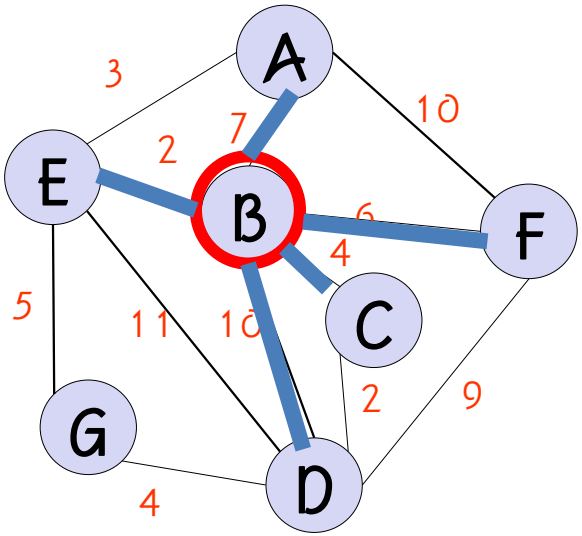
(2) 정점 k를 거쳐가는 경로의 비용이 더 작은지 체크하여

(3) 더 작다면 모든 정점간의 SP를 수정한다.

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

Floyd의 최단경로 알고리즘

k=정점 B를 거쳐가는 경로



	A	B	C	D	E	F	G
A	0	7	11	17	3	10	INF
B	7	0	4	10	2	6	INF
C	11	4	0	2	6	10	INF
D	17	10	2	0	11	9	4
E	3	2	6	11	0	8	5
F	10	6	10	9	8	0	INF
G	INF	INF	INF	4	5	INF	0

무한대

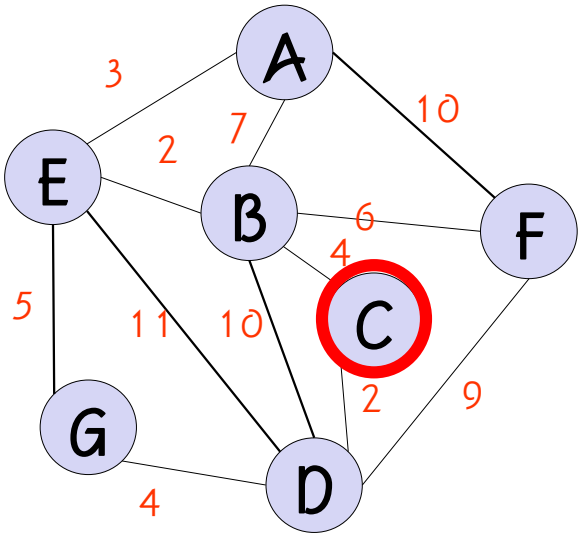
A-C	A-B-C (11)
A-D	A-B-D (17)
C-E	C-B-E (6)
C-F	C-B-F (10)
E-F	E-B-F (8)

E-A-F (13)

AI융합학부 안양준 yangjunahn@sungshin.ac.kr

Floyd의 최단경로 알고리즘

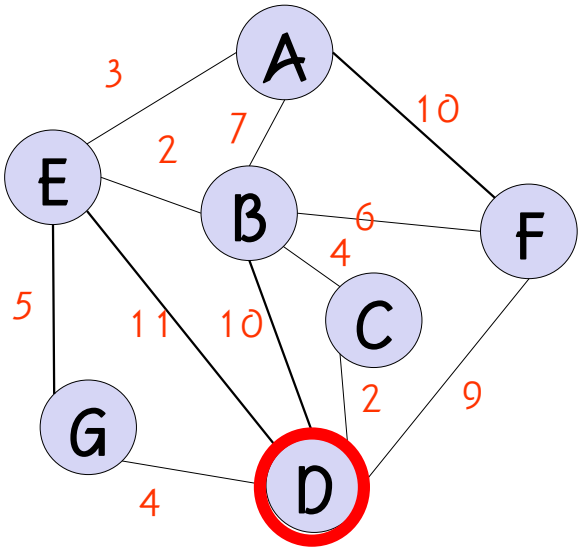
k=정점 C를 거쳐가는 경로



	A	B	C	D	E	F	G
A	0	7	11	13	3	10	INF
B	7	0	4	6	2	6	INF
C	11	4	0	2	6	10	INF
D	13	6	2	0	8	9	4
E	3	2	6	8	0	8	5
F	10	6	10	9	8	0	INF
G	INF	INF	INF	4	5	INF	0

Floyd의 최단경로 알고리즘

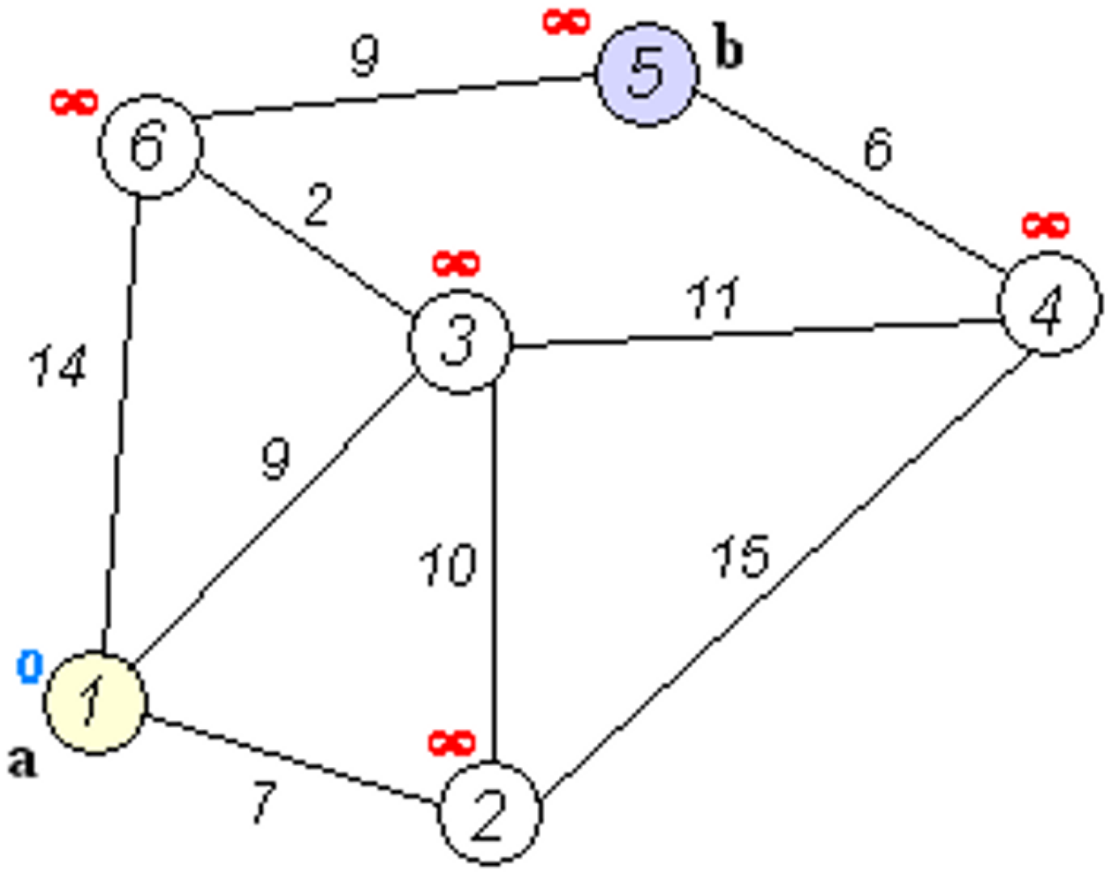
k=정점 D를 거쳐가는 경로



	A	B	C	D	E	F	G
A	0	7	11	13	3	10	17
B	7	0	4	6	2	6	10
C	11	4	0	2	6	10	6
D	13	6	2	0	8	9	4
E	3	2	6	8	0	8	5
F	10	6	10	9	8	0	13
G	17	10	6	4	5	13	0

예제

Floyd 알고리즘 각 단계에서의 6x6 행렬의 값을 구하세요.



Pseudo Code: Floyd Shortest Path

- 모든 정점 사이의 최단경로를 찾는 알고리즘

floyd(G)

```
for k ← 0 to n - 1  // 정점 k를 거쳐가는 경로의 길이가 짧다면 경로 수정
    for i ← 0 to  n - 1 // 모든 정점 (i,j)의 SP 계산
       for j ← 0 to n - 1
             A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

Floyd의 최단경로 프로그램

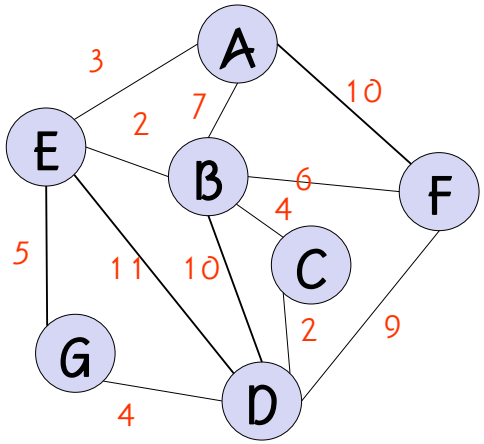
```
// Floyd 알고리즘의 최단 경로 탐색 기능이 추가된 그래프
class WGraphFloyd : public WGraph
{
    int A[MAX_VTXS][MAX_VTXS]; // 최단경로 거리
public:
    void ShortestPathFloyd( ) {
        for( int i=0 ; i<size ; i++ ) // 기본 길이로 초기화
        for( int j=0 ; j<size ; j++ ) A[i][j] = getEdge(i,j);

        for(int k=0 ; k<size ; k++ ){ // 정점 k를 거치는 경우
            for( int i=0; i<size ; i++ ) // 모든 (i,j) 경로 수정
            for( int j=0; j<size ; j++ )
                if (A[i][k]+A[k][j] < A[i][j])
                    A[i][j] = A[i][k] + A[k][j];
            printA( ); // 각 단계에서의 최단경로 거리 출력 : k번
        }
    }
```

Floyd의 최단경로 프로그램

```
    void printA() {
        printf("====================================\n");
        for( int i=0; i<size; i++){ // 모든 (i,j) 경로 거리 출력
            for(int j=0; j<size; j++) {
                if( A[i][j] == INF ) printf(" INF ");
                else printf("%4d ", A[i][j]);
            }
            printf("\n");
        }
    }
};
```

Floyd의 최단경로 프로그램

```
// class WGraphFloyd

void main()
{
    WGraphFloyd f;

    f.load("graph.txt");
    printf("Dijkstra의 최단경로 탐색을 위한 그래프: graph_sp.txt₩n");
    f.display();

    printf("Shortest Path By Dijkstra Algorithm₩n");
    f.ShortestPathFloyd();
}
```



Floyd 알고리즘 실행 결과

```
최단거리(wgraph_sp.txt)
7
A     0    7    -    -    3   10    -
B     7    0    4   10    2    6    -
C     -    4    0    2    -    -    -
D     -   10    2    0   11    9    4
E     3    2    -   11    0   13    5
F    10    6    -    9   13    0    -
G     -    -    -    4    5    -    0
Shortest Path By Floyd Algorithm
====================================
    0    7  INF  INF    3   10  INF
    7    0    4   10    2    6  INF
  INF    4    0    2  INF  INF  INF
  INF   10    2    0   11    9    4
    3    2  INF   11    0   13    5
   10    6  INF    9   13    0  INF
  INF  INF  INF    4    5  INF    0
====================================
    0    7   11   17    3   10  INF
    7    0    4   10    2    6  INF
   11    4    0    2    6   10  INF
   17   10    2    0   11    9    4
    3    2    6   11    0    8    5
   10    6   10    9    8    0  INF
  INF  INF  INF    4    5  INF    0
====================================
    0    7   11   13    3   10  INF
    7    0    4    6    2    6  INF
   11    4    0    2    6   10  INF
   13    6    2    0    8    9    4
    3    2    6    8    0    8    5
   10    6   10    9    8    0  INF
  INF  INF  INF    4    5  INF    0
```

```
====================================
    0    7   11   13    3   10   17
    7    0    4    6    2    6   10
   11    4    0    2    6   10    6
   13    6    2    0    8    9    4
    3    2    6    8    0    8    5
   10    6   10    9    8    0   13
   17   10    6    4    5   13    0
====================================
    0    5    9   11    3   10    8
    5    0    4    6    2    6    7
    9    4    0    2    6   10    6
   11    6    2    0    8    9    4
    3    2    6    8    0    8    5
   10    6   10    9    8    0   13
    8    7    6    4    5   13    0
====================================
    0    5    9   11    3   10    8
    5    0    4    6    2    6    7
    9    4    0    2    6   10    6
   11    6    2    0    8    9    4
    3    2    6    8    0    8    5
   10    6   10    9    8    0   13
    8    7    6    4    5   13    0
====================================
    0    5    9   11    3   10    8
    5    0    4    6    2    6    7
    9    4    0    2    6   10    6
   11    6    2    0    8    9    4
    3    2    6    8    0    8    5
   10    6   10    9    8    0   13
    8    7    6    4    5   13    0
계속하려면 아무 키나 누르십시오 . . .
```