DATA
STRUCTURES
USING C

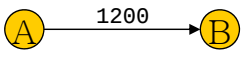
- 가중치 그래프의 개념을 이해한다.
- 가중치 그래프를 표현하는 방법을 이해한다.
- 최소 비용 신장 트리 알고리즘을 이해한다.
- 최단 경로 알고리즘을 이해한다.
- 가중치 그래프를 이용한 문제해결 능력을 배양한다.

CHAPTER 12

가중치 그래프

12.1 가중치 그래프란?

- 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

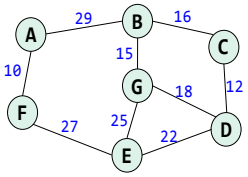


12.2 가중치 표현방법

- 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 의 인접행렬 M을 이용한 그래프 표현에서 M[i][j] 가 간선 (i,j)의 가중치 값을 나타내도록 함
  - 문제점?
    - 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

가중치 그래프 표현 예

- 인접 행렬을 이용한 가중치 그래프의 표현 예

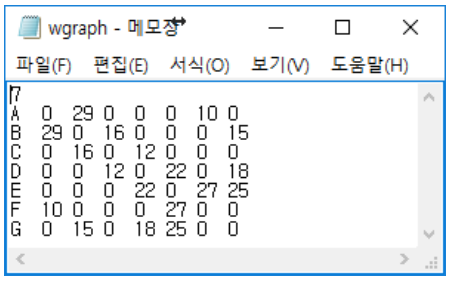


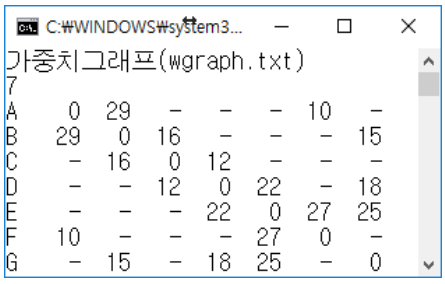
(a) 가중치 그래프

	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

- 가중치 그래프의 파일 형식 예와 화면 출력





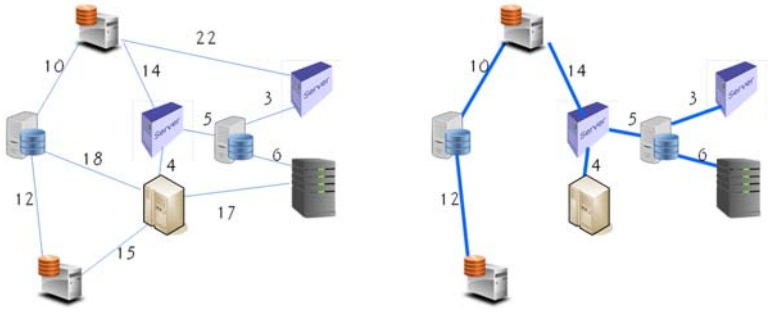
12.3 최소비용 신장트리 문제

- 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
  - 통신: 전화 케이블 망을 구성하면서 전화선의 길이가 최소화
  - 배관: 배관을 모두 연결하면서 파이프의 총 길이를 최소화

최소비용 신장트리 조건

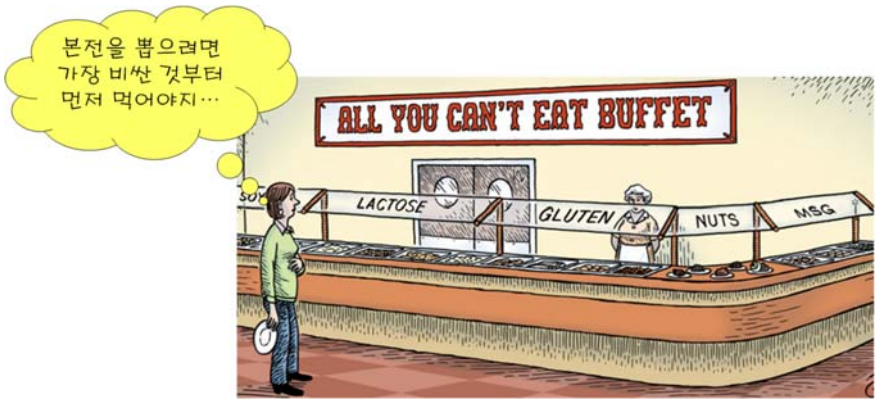
- 간선의 가중치의 합이 최소
- 반드시 (n-1)개의 간선만 사용
- 사이클이 포함되면 안됨



- Kruskal 알고리즘
- Prim 알고리즘

Kruskal의 MST

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

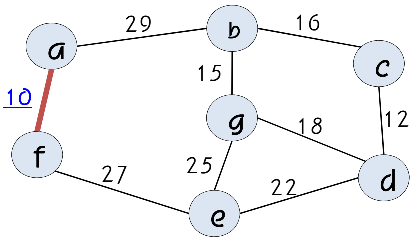


Kruskal의 MST

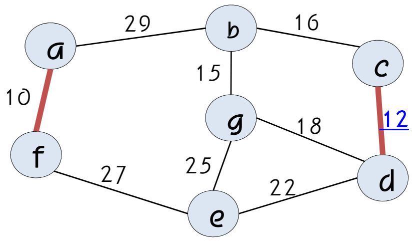
- 알고리즘
  - 각 단계에서 사이클을 이루지 않는 최소 비용 간선 선택

kruskal()

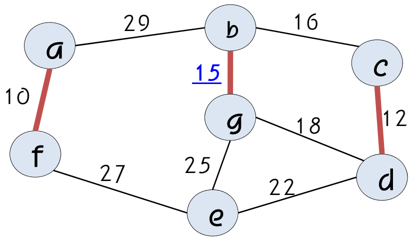
1. 그래프의 모든 간선을 가중치에 따라 오름차순으로 정렬한다.
2. 가장 가중치가 작은 간선 e를 뽑는다.
3. e를 넣어 신장트리에 사이클이 생기면 넣지 않고 2번으로 이동한다.
4. 사이클이 생기지 않으면 최소 신장 트리에 삽입한다.
5. n-1개의 간선이 삽입될 때 까지 2번으로 이동한다.



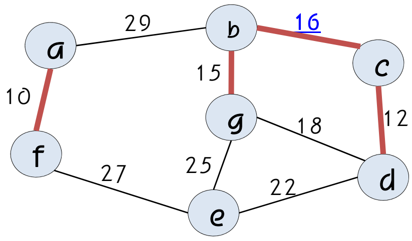
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



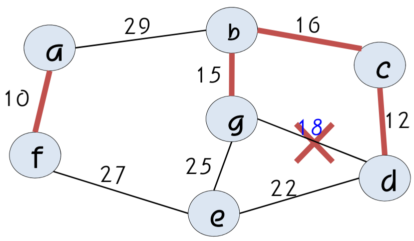
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



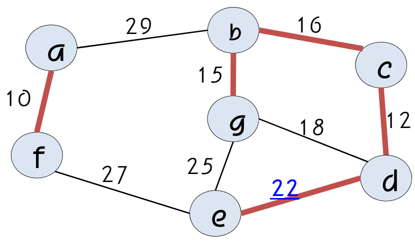
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



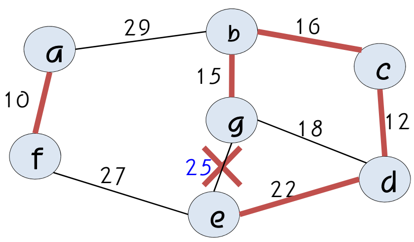
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



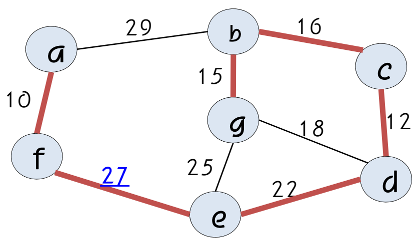
af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29



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

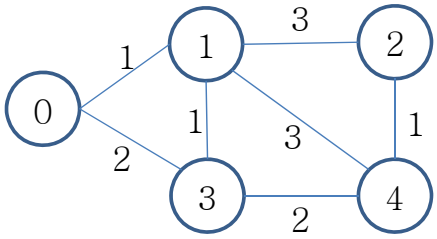


af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29

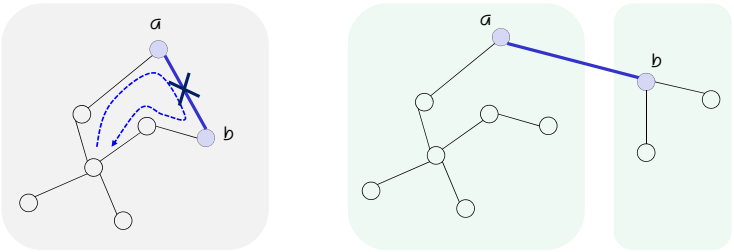


af	cd	bg	bc	dg	de	eg	ef	ab
10	12	15	16	18	22	25	27	29

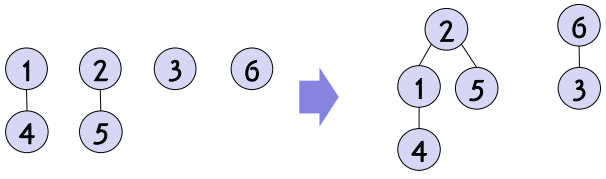
- 아래의 네트워크에 대하여 Kruskal의 MST 알고리즘을 이용해서 최소 비용 신장 트리가 구성되는 과정을 보여라.



사이클의 검사→Union-Find



- Union-Find 알고리즘
  - Find: 원소가 어떤 집합에 속하는지 알아냄. 속한 집합을 반환
  - 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-Find 연산

```
int parent[MAX_VTXS];
int set_size;



void init_set ( int nSets)
{
  int i;
  set_size = nSets;
  for( i=0 ; i<nSets ;i++ )
    parent[i] = -1;
}
```

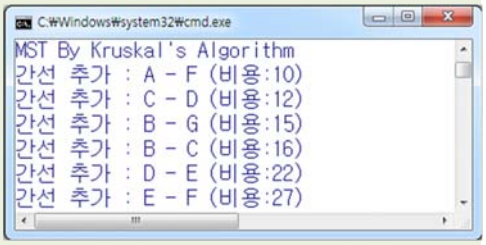
```
int find_set( int id)
{
  while ( parent[id] >= 0 )
    id = parent[id];
  return id;
}

void union_set(int s1, int s2)
{
  parent[s1] = s2;
  set_size--;
}
```


최소 가중치 간선 뽑기 → 최소힙

```
typedef struct HeapNode {       // 힙에 저장할 항목의 자료형
   int   key;                   // 간선의 가중치
   int   v1;                    // 한쪽 정점의 인덱스
   int   v2;                    // 다른 쪽 정점의 인덱스
} HNode;
#define Key(n) (n.key)          // 힙 노드 n의 키 값: 간선의 가중치
```

```
void Kruskal( )
{
        int i, j, edgeAccepted=0, uset, vset;
        HNode e;
        init_heap();
        init_set( vsize);
        for( i=0 ; i<vsize-1 ; i++ )
        for( j=i+1 ; j<vsize ; j++ )
                if (adj[i][j] < INF) {
                        e.key = adj[i][j];
                        e.v1 = i;
                        e.v2 = j;
                        insert_heap(e);
                }
        while( edgeAccepted < vsize-1 ){
                e = delete_heap( );
                uset = find_set( e.v1 );
                vset = find_set( e.v2 );
                if( uset != vset ){
                        printf( "간선 추가 : %c - %c (비용:%d)\n",
                                vdata[e.v1], vdata[e.v2], e.key);
                        union_set(uset, vset);
                        edgeAccepted++;
                }
        }
}
```



Kruskal의 MST 알고리즘 복잡도

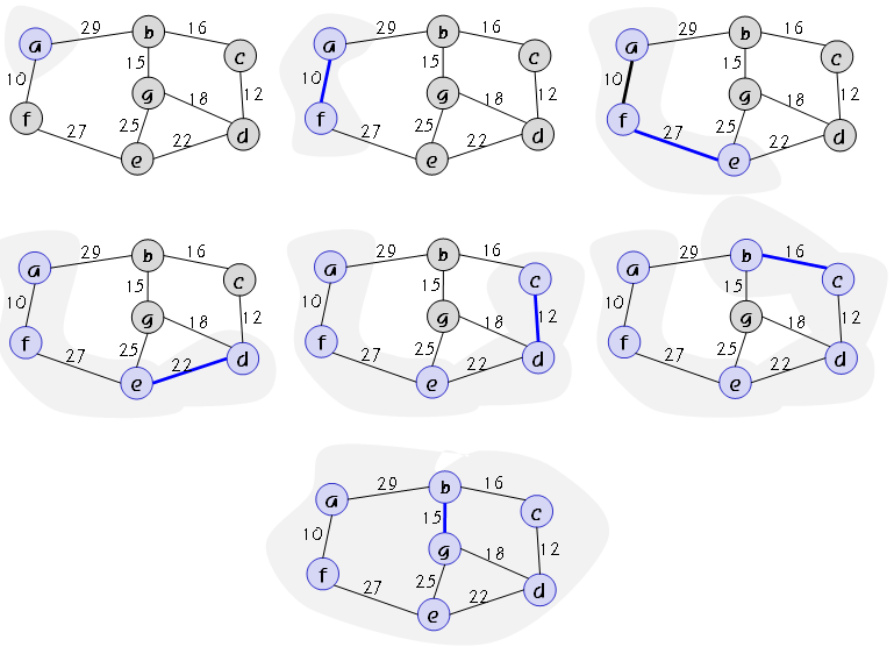
- 대부분 간선들을 정렬하는 시간에 좌우됨
  - 사이클 테스트 등의 작업은 정렬에 비해 매우 신속하게 수행됨
- 간선 e개
  - 퀵정렬과 같은 효율적인 알고리즘으로 정렬한다면
  - Kruskal 알고리즘의 시간 복잡도는 O(e*log(e))

Prim의 MST 알고리즘

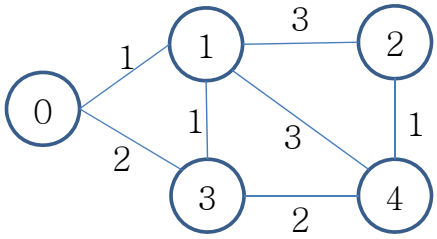
- 기본 개념
  - 처음에는 시작 정점만이 신장 트리 집합에 포함됨
  - 시작 정점부터 신장 트리 집합을 단계적으로 확장해나감
  - 현재의 신장 트리 집합에 인접한 정점 중 최저 간선으로 연결된 정점 선택하여 신장 트리 집합에 추가
  - 이 과정을 n-1개의 간선을 가질 때까지 반복

```
Prim()
1. 그래프에서 시작 정점을 선택하여 초기 트리를 만든다.
2. 현재 트리의 정점들과 인접한 정점들 중에서 간선의 가중치가 가장
작은 정점 v를 선택한다.
3. 이 정점 v와 이때의 간선을 트리에 추가한다.
4. 모든 정점이 삽입될 때 까지 2번으로 이동한다.
```

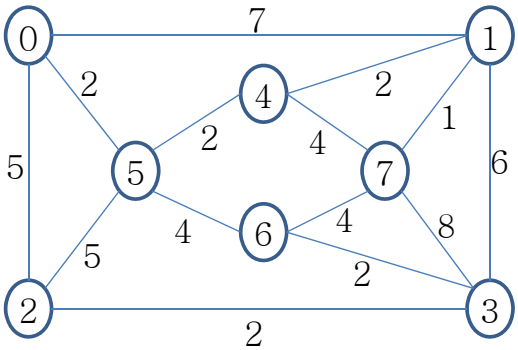

Prim의 MST 알고리즘



- 아래의 네트워크에 대하여 Prim의 MST 알고리즘을 이용해서 최소 비용 신장 트리가 구성되는 과정을 보여라 (0번 정점으로 시작).



- 아래 그래프에 Kruskal의 알고리즘을 이용해 최소 비용 신장 트리를 만드는 과정을 보여라.



- 위의 그래프에서 0번 정점을 시작 정점으로 Prim 알고리즘을 이용해 최소 비용 신장 트리를 만드는 과정을 보여라.

Prim의 MST 구현

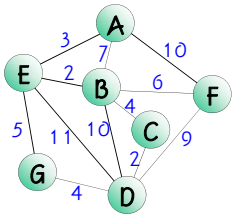
```
void Prim()
{
        int i, u, v;
        for(i=0 ; i<vsize ; i++) {
                dist[i] = INF;
                selected[i] = 0;
        }
        dist[0]=0;
        for(i=0 ; i<vsize ; i++){
                u = getMinVertex();
                selected[u] = 1;
                if( dist[u] == INF ) return;
                printf("%c ", vdata[u]);
                for( v=0; v<vsize; v++)
                        if( adj[u][v] != INF)
                                if( !selected[v] && adj[u][v]< dist[v] )
                                        dist[v] = adj[u][v];
        }
        printf("\n");
}
```


Prim의 MST 알고리즘 복잡도

- Prim의 알고리즘은 $O(n^2)$ 의 복잡도
  - 주 반복문이 정점의 수 n만큼 반복하고
  - 내부 반복문이 n번 반복
- 희박한 그래프
  - O(e*log(e)) 인 Kruskal의 알고리즘이 유리
- 밀집한 그래프 (간선이 매우 많은 그래프)
  - $O(n^2)$ 인 Prim의 알고리즘이 유리

12.4 최단 경로(shortest path) 문제

- 최단경로란?
  - 네트워크에서 정점 u와 정점 v를 연결하는 경로 중에서 간선들의 가중치 합이 최소가 되는 경로
  - 간선의 가중치는 비용, 거리, 시간 등

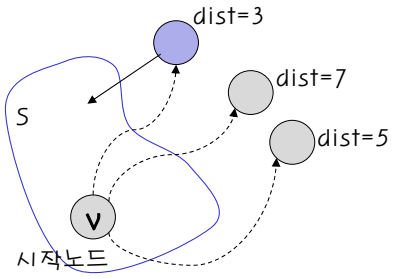


	A	B	C	D	E	F	G
A	0	7	∞	∞	3	10	∞
B	7	0	4	10	2	6	∞
C	∞	4	0	2	∞	∞	∞
D	∞	10	2	0	11	9	4
E	3	2	∞	11	0	∞	5
F	10	6	∞	9	∞	0	∞
G	∞	∞	∞	4	5	∞	0

- 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

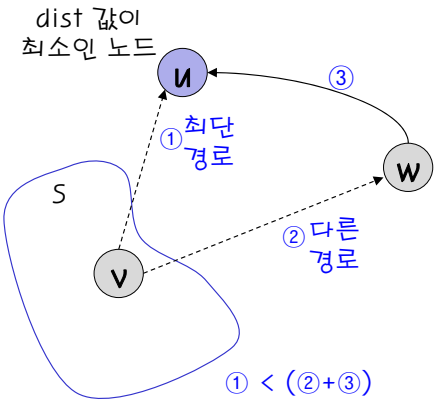
Dijkstra의 최단경로 알고리즘

- 시작 정점 v에서 모든 다른 정점까지의 최단 경로 찾음
  - 집합 S: v에서부터의 최단경로가 이미 발견된 정점들의 집합
  - dist 배열: 최단경로가 알려진 정점들을 이용한 다른 정점들까지의 최단경로 길이
    - 시작정점 v, 다른 정점들 u
    - 초기값
      - dist[v] = 0
      - dist[u] = 시작정점 v와 u간의 가중치 값



  - 매 단계에서 최소 distance인 정점을 S에 추가

최단경로 증명



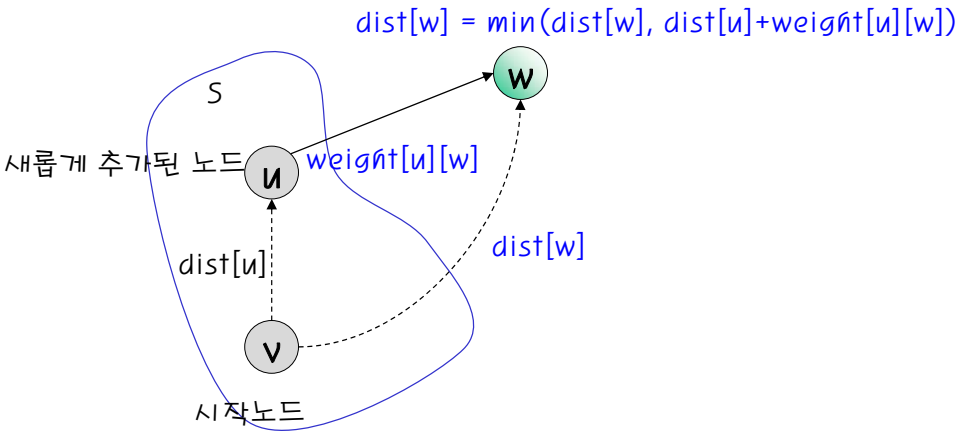
- distance 값이 가장 작은 정점이 u
- v에서 u까지의 최단경로는 ①

- 경로 ②는 ①보다 항상 길 수 밖에 없음.
- Why? ③ 은 항상 양수이므로…

- 따라서 매 단계에서 distance 값이 가장 작은 정점들을 추가하면 모든 정점까지의 최단거리를 구할 수 있다.

distance값 갱신

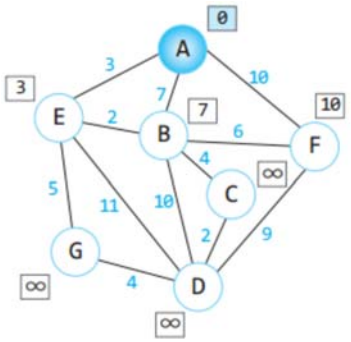
- 새로운 정점이 S에 추가되면 distance값 갱신



Dijkstra의 최단경로 알고리즘

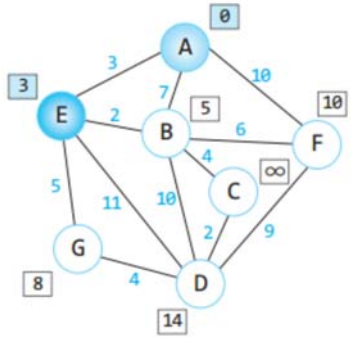
```
// 입력: 가중치 그래프 G, 가중치는 음수가 아님.
// 출력: dist 배열, dist[u]는 v에서 u까지의 최단 거리이다.
shortestPath(v)
    S←{v}
    for 각 정점 w∈G do
        dist[w]←weight[v][w];
    while 모든 정점이 S에 포함되지 않으면 do
        u←집합 S에 속하지 않는 정점 중에서 최소 distance 정점;
        S←S∪{u}
        for u에 인접하고 S에 있지 않은 각 정점 z do
            if dist[u]+weight[u][z] < dist[z]
                then dist[z]←dist[u]+weight[u][z];
```


알고리즘 실행 과정: Step1 - 2



S={A}

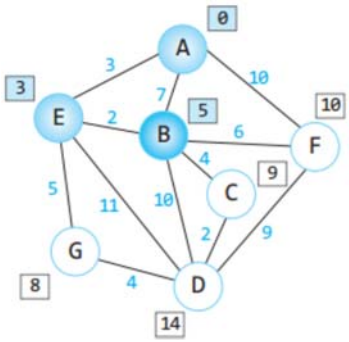
dist(A)=w(A,A)=0
dist(B)=w(A,B)=7
dist(C)=w(A,C)=∞
dist(D)=w(A,D)=∞
dist(E)=w(A,E)=3
dist(F)=w(A,F)=10
dist(G)=w(A,G)=∞



S={A, E}

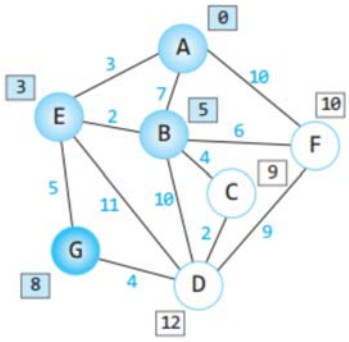
dist(A)=0
dist(B)=min(dist(B),dist(E)+w(E,B))=min(7,3+2)=5
dist(C)=min(dist(C),dist(E)+w(E,C))=min(∞,3+∞)=∞
dist(D)=min(dist(D),dist(E)+w(E,D))=min(∞,3+11)=14
dist(E)=w(A,E)=3
dist(F)=min(dist(F),dist(E)+w(E,F))=min(10,3+∞)=10
dist(G)=min(dist(G),dist(E)+w(E,G))=min(∞,3+5)=∞

알고리즘 실행 과정: Step3 - 4



S={A, E, B}

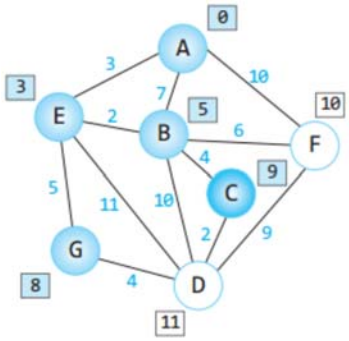
dist(A)=0
dist(B)=5
dist(C)=min(dist(C),dist(B)+w(B,C))=min(∞,5+4)=9
dist(D)=min(dist(D),dist(B)+w(B,D))=min(14,5+10)=14
dist(E)=3
dist(F)=min(dist(F),dist(B)+w(B,F))=min(10,5+6)=10
dist(G)=min(dist(G),dist(B)+w(B,G))=min(8,5+∞)=10



S={A, E, B, G}

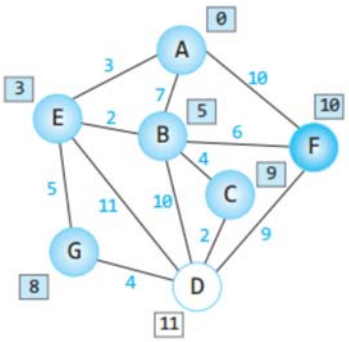
dist(A)=0
dist(B)=5
dist(C)=min(dist(C),dist(G)+w(G,C))=min(9,8+∞)=9
dist(D)=min(dist(D),dist(G)+w(G,D))=min(14,8+4)=12
dist(E)=3
dist(F)=min(dist(F),dist(G)+w(G,F))=min(10,8+∞)=10
dist(G)=8

알고리즘 실행 과정: Step5 - 6



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

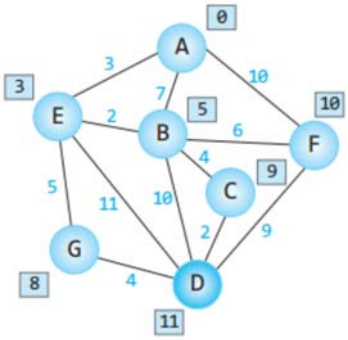
dist(A)=0
dist(B)=5
dist(C)=9
dist(D)=min(dist(D),dist(C)+w(C,D))=min(12,9+2)=11
dist(E)=3
dist(F)=min(dist(F),dist(C)+w(C,F))=min(10,9+∞)=10
dist(G)=8



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

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

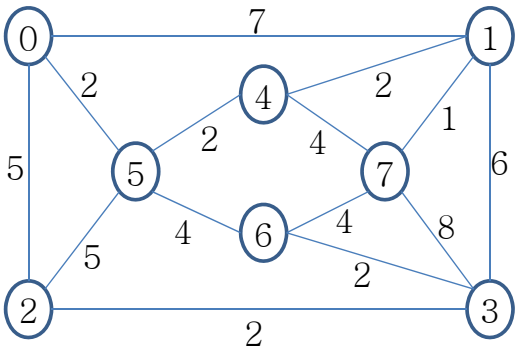
알고리즘 실행 과정: 최종



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

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

- 아래의 그래프에 0번 정점을 시작 정점으로 Dijkstra의 알고리즘을 이용해 최단 경로를 구하는 과정을 배열 distance[]의 변화 과정으로 보여라.



32

Dijkstra의 최단경로 함수

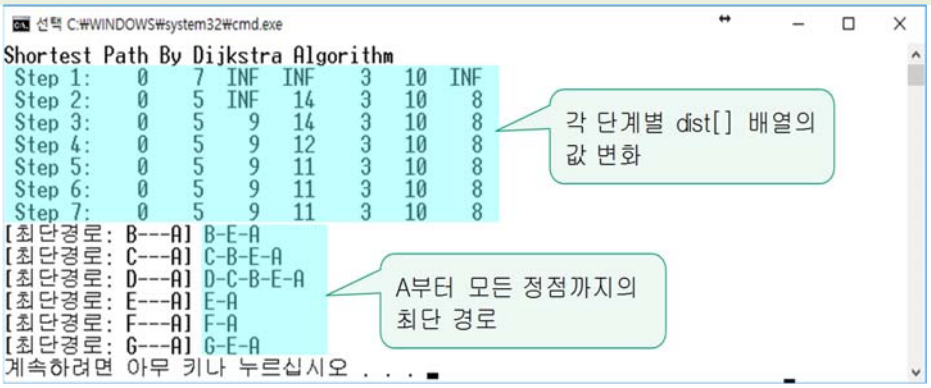
```
int path[MAX_VTXS];
int dist[MAX_VTXS];
int found[MAX_VTXS];

int choose_vertex()
{
  int i, min = INF, minpos = -1;
  for( i=0 ; i<vsize ; i++ )
    if(dist[i]< min && !found[i]){
        min = dist[i];
        minpos=i;
    }
  return minpos;
}
```

```
void shortest_path_dijkstra( int start )
{
  int i, u, w;
  for( i=0 ; i<vsize ; i++) {
    dist[i] = adj[start][i];
    path[i] = start;
    found[i] = 0;
  }
  found[start] = 1;
  dist[start] = 0;
  for( i=0 ; i<vsize ; i++ ){
    u = choose_vertex();
    found[u] = 1;
    for( w=0 ; w<vsize ; w++) {
      if( found[w] == 0 ) {
        if(dist[u]+adj[u][w] < dist[w]){
          dist[w] = dist[u] + adj[u][w];
          path[w] = u;
        }
      }
    }
  }
}
```


전체 프로그램

```
void main()
{
        int i;
        load_wgraph( "wgraph_sp.txt" );
        print_wgraph( "최단거리(wgraph_sp.txt)\n");
        printf("Shortest Path By Dijkstra Algorithm\n");
        shortest_path_dijkstra( 0 );
        for (i = 1; i<vsize; i++)                    // 진행상황 출력용
                print_shortest_path(0, i);           // 진행상황 출력용
}
```



Dijkstra 알고리즘 시간 복잡도

- 네트워크에 n개의 정점이 있다면
  - 주 반복문을 n번 반복
  - 내부 반복문을 2n번 반복

➔ Dijkstra의 최단경로 알고리즘은 $O(n^2)$의 시간 복잡도

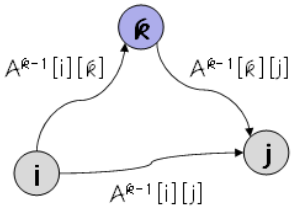
Floyd의 최단경로 알고리즘

- 기본 개념
  - 모든 정점 사이의 최단경로를 찾음
  - 2차원 배열 A를 이용하여 3중 반복을 하는 루프로 구성
    - 배열 A의 초기 값은 인접 행렬 weight
    - 인접 행렬 weight 구성
      - i==j이면, 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
       for i ← 0 to  n - 1
          for j ← 0 to n - 1
              A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

알고리즘 증명

- 알고리즘
  - A^k [i][j]: 0부터 k까지의 정점만을 이용한 정점 i에서 j까지의 최단 경로 길이
  - $A^{-1} \rightarrow A^0 \rightarrow A^1 \rightarrow \ldots \rightarrow A^{n-1}$ 순으로 최단 경로 구해감
  - A^{k-1} 에서 k번째 정점이 추가되는 상황
    - 다음의 2가지의 경우로 나뉘어짐



(1) 정점 k를 거치지 않는 경우: $A^k[i][j] \leftarrow A^{k-1}[i][j]$.

(2) 정점 k를 통과하는 경우: $A^k[i][j] \leftarrow A^{k-1}[i][k] + A^{k-1}[j][k]$

따라서 최종적인 최단거리는 다음과 같이 당연히 (1)과 (2)중에서 더 적은 값이 될 것이다.

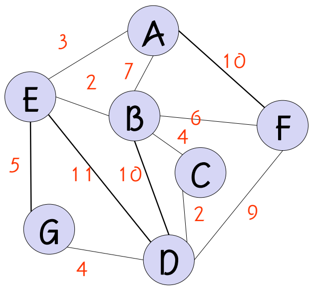
$$A^k[i][j] \leftarrow \min(A^{k-1}[i][j], A^{k-1}[i][k] + A^{k-1}[j][k])$$

Floyd의 최단경로 함수

```
static int A[MAX_VTXS][MAX_VTXS];  // 최단경로 거리
static void printA( ) { … }

void shortest_path_floyd()
{
        int i, j, k;
        for( i=0 ; i<vsize ; i++ )
        for( j=0 ; j<vsize ; j++ )
                A[i][j] = adj[i][j];
        for(k=0 ; k<vsize ; k++ ){
                for( i=0; i<vsize ; i++ )
                for (j = 0; j < vsize; j++) {
                        if (A[i][k]+A[k][j] < A[i][j])
                                A[i][j] = A[i][k] + A[k][j];
                }
                printA();        // 진행상황 출력용
        }
}
```

Floyd 알고리즘 실행 결과

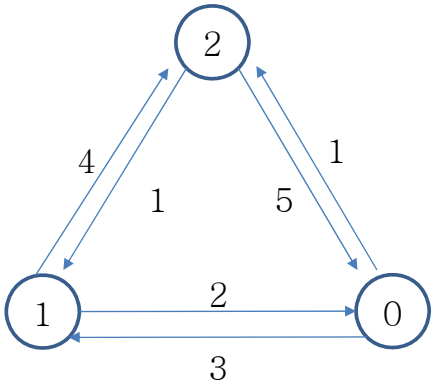



```
C:\WINDOWS\system32\cmd.e...
최단거리(wgraph_sp.txt)
7
A     0   7   -   -   3  10   -
B     7   0   4  10   2   6   -
C     -   4   0   2   -   -   -
D     -  10   2   0  11   9   4
E     3   2   -  11   0  13   5
F    10   6   -   9  13   0   -
G     -   -   -   4   5   -   0
Shortest Path By Floyd 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 . . .
```


Floyd의 알고리즘 복잡도

- 네트워크에 n개의 정점이 있다면,
  - Floyd의 최단경로 알고리즘은 3중 반복문을 실행
  - 시간 복잡도는 $O(n^3)$
- Dijkstra 알고리즘과의 비교
  - 모든 정점쌍의 최단경로를 구하려면 Dijkstra의 알고리즘 $O(n^2)$을 n번 반복해도 되며, 이 경우 전체 복잡도는 $O(n^3)$ 이 된다
  - 모든 정점쌍의 최단경로를 구하는데 있어 두 알고리즘 모두 동일한 $O(n^3)$의 복잡도를 가지지만 Floyd의 알고리즘은 매우 간결한 반복구문을 사용하므로 Dijkstra의 알고리즘 보다 효율적이다

- 아래의 그래프에 Floyd의 알고리즘을 이용해 최단 경로를 구할 때 2차원 배열 A[]가 변경되는 모습을 보여라.



41