알고리즘 정리

- 날짜: 2020. 02. 10. (월) 16:10 ~
- 작성자: 김지형, 김록원, 장주아

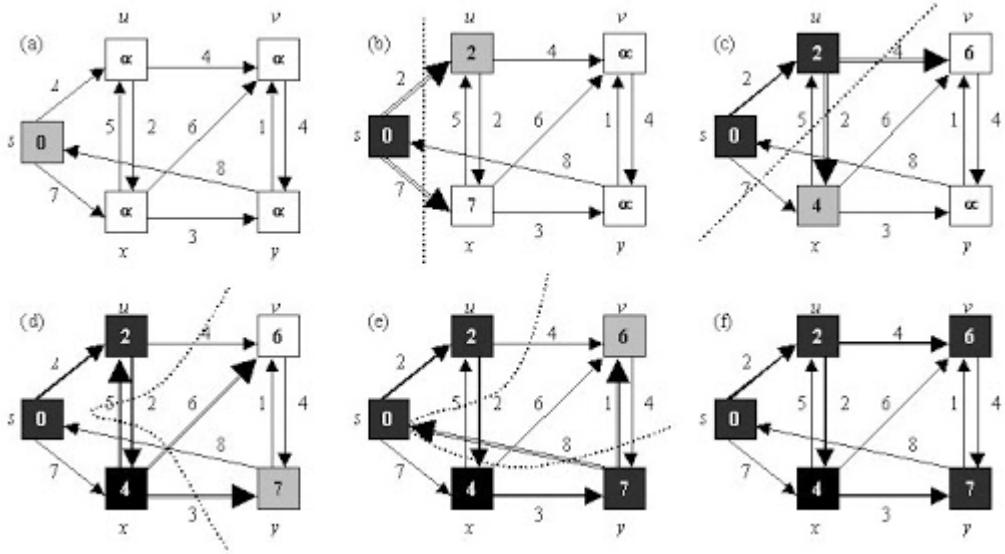
알고리즘 종류

1. BFS
2. Dijkstra
3. Bellman-Ford
4. SPFA
5. Floyd-Warshall
6. Kruskal
7. Prim

Dijkstra Algorithm

- 목적: 한 점에서 다른 모든 점까지 각각의 최단 경로를 찾고 싶을때
- 시간복잡도: $O(Elog(V))$
- 한계: 음수의 가중치 포함된 경우에는 X
- 수도 코드

```
#include<queue>

int arr[V];

void Dijkstra() {

    priority_queue<pair<int,int>> q;
    q.push({1,0});

    while(!q.empty()) {
        int x = q.top().first;
        int count = q.top().second;

        if ( count < arr[x] ) {
            arr[x] = count;
            for (auto n : arr[x])
                q.push({n.first, n.second + count});
        }
        q.pop();
    }
}
```



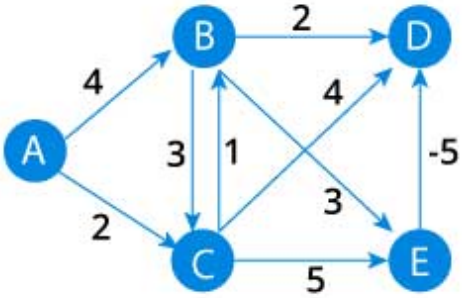
Bellman-Ford Algorithm

- 목적: 음수가 포함된 경로를 찾고 싶을때
- 시간복잡도: $O(VE)$
- 한계: dense graph의 경우 엣지 수가 대개 노드 수의 제곱에 근사하므로 O(|V|^3)
- 수도 코드

```
function bellmanFord(G, S)
    for each vertex V in G
            distance[V] <- infinite
            previous[V] <- NULL
    distance[S] <- 0
    for each vertex V in G
        for each edge (U,V) in G
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
              distance[V] <- tempDistance
              previous[V] <- U
    for each edge (U,V) in G
        If distance[U] + edge_weight(U, V) < distance[V}
            Error: Negative Cycle Exists
    return distance[], previous[]
```

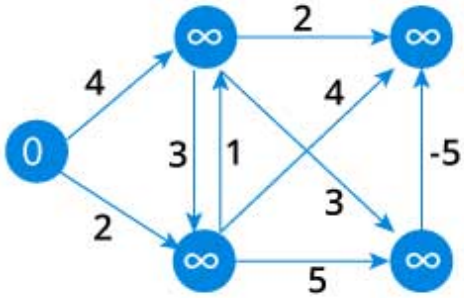
1

Start with a weighted graph



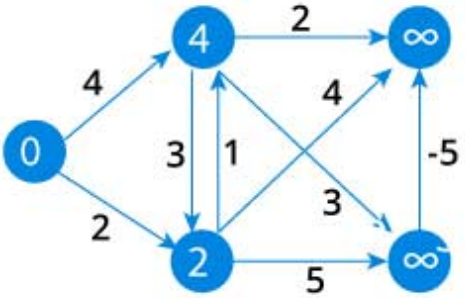
2

Choose a starting vertex and assign infinity path values to all other vertices



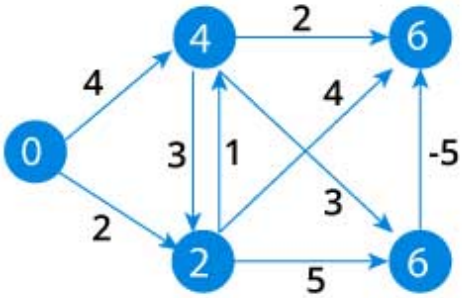
3

Visit each edge and relax the path distances if they are inaccurate



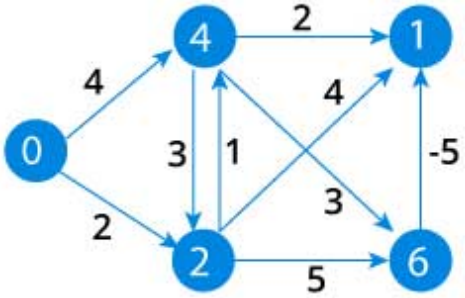
4

We need to do this V times because in the worst case, a vertex's path length might need to be readjusted V times



5

Notice how the vertex at the top right corner had its path length adjusted



6

After all the vertices have their path lengths, we check if a negative cycle is present.

A	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

SPFA

- 목적: 음수가 포함된 경로를 찾고 싶을때
- 시간복잡도: 최악의 경우: $O(VE)$ 평균: $O(V+E)$
- SPFA는 바뀐 정점과 연결된 간선에 대해서만 업데이트하므로 벨만보다 성능이 좋음
- 수도 코드

```
procedure Shortest-Path-Faster-Algorithm(G,s)
  for each vertex v != s in V(G)
    d(v) = INF
d(s) = 0
offer s into Q
while Q is not empty
  u = poll Q
  for each edge (u, v) in E(G)
    if d(u)+ w(u,v) < d(v) then
      d(v) = d(u) + w(u,v)
      if v is not in Q then
        cycle[v] += 1 //cycle체크
        if cycle[v] >= n: return
        //한 정점이 n번 이상 방문되면 그 정점을 포함하여 음수 사이클이 존재
        offer v into Q
```

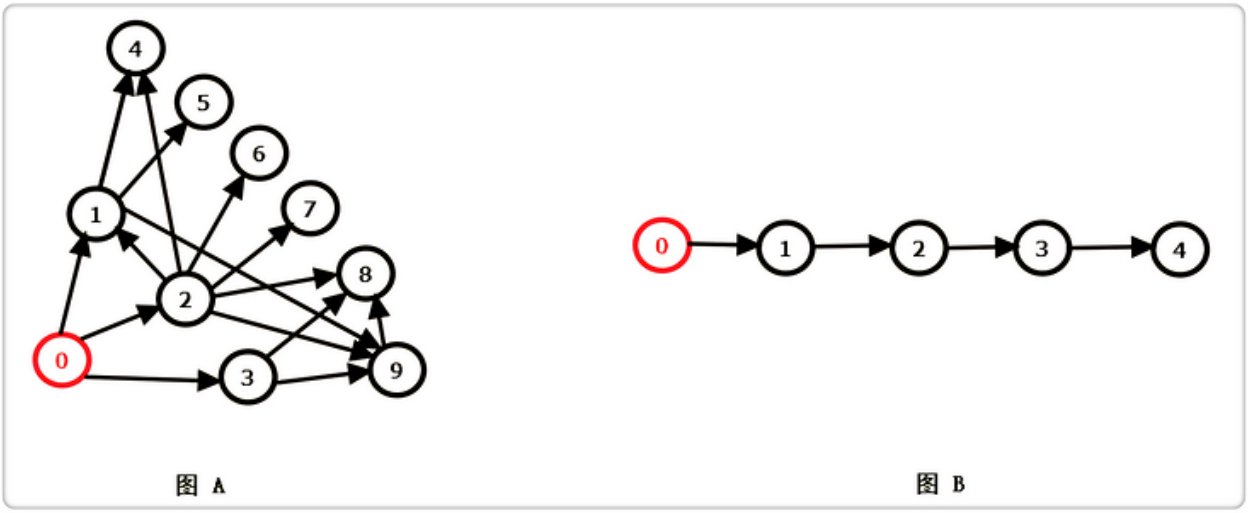
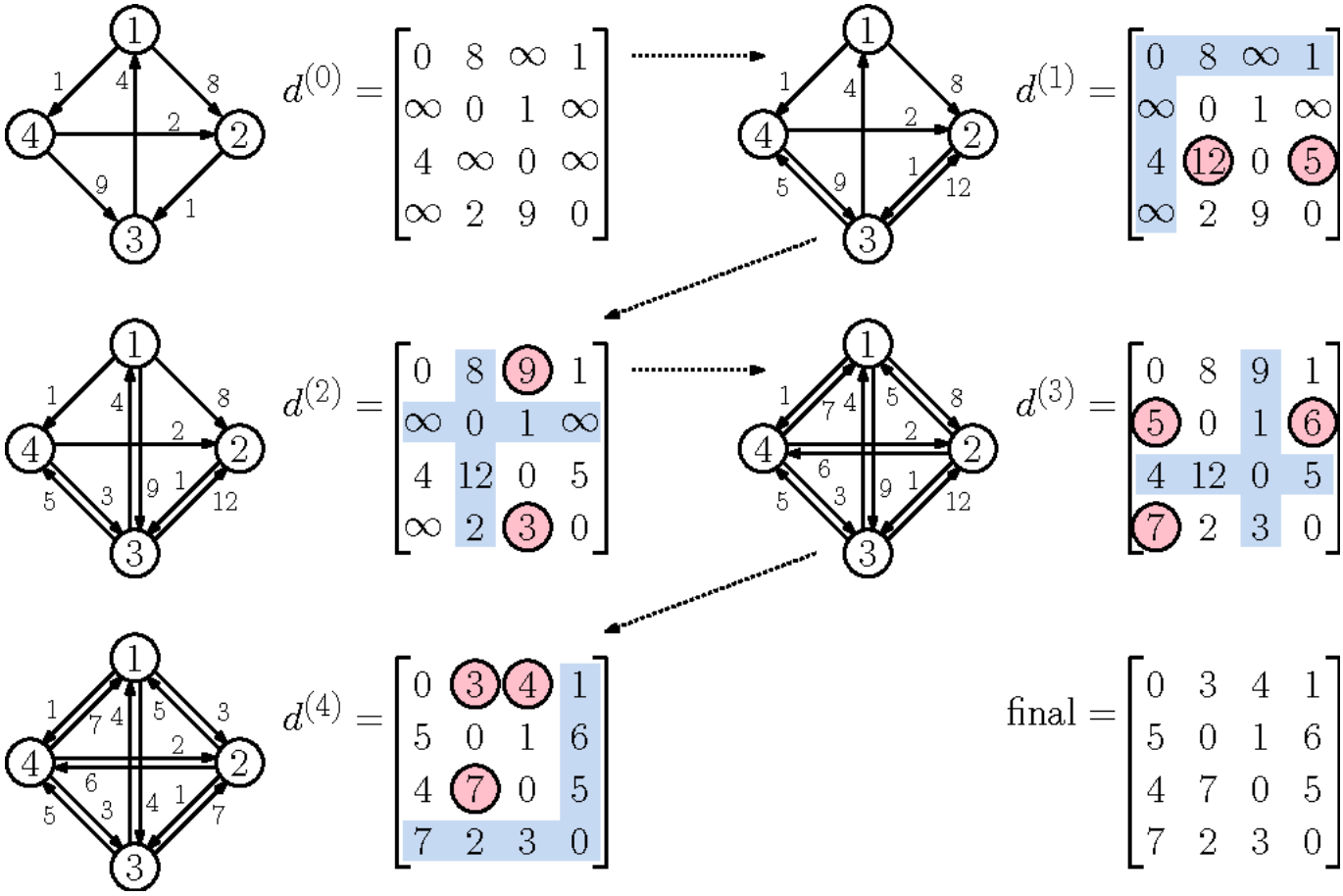


图 A 图 B

Floyd-Warshall Algorithm

- 목적: 모든 노드 쌍에 대한 최단 경로를 찾고 싶을 때
- 시간복잡도: $O(|V|^3)$
- 한계:
- 수도 코드

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u,v)
   dist[u][v] ← w(u,v)  // 변 (u,v)의 가중치
for each vertex v
   dist[v][v] ← 0
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][j] > dist[i][k] + dist[k][j]
            dist[i][j] ← dist[i][k] + dist[k][j]
         end if
```

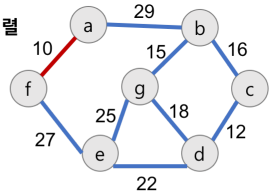


Kruskal Algorithm

- 목적: MST
- 한계: 완_벽
- 복잡도
  - $O(ElogV)$
- 수도 코드:

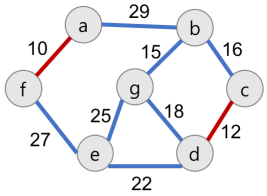
```
//유니온 파인드 알고리즘 사용

int arr[V]; //각각 index랑 똑같은 수로 초기화 되어 있음
struct info{
    int x;
    int y;
    int count;
};

int find(int i) {
    if (i == arr[i])
        return i;
    return arr[i] = find(arr[i]);
}

void merge(int i, int j) {
    arr[i] = find(j);
}

void Kruskal() {
    vector< info > v[V];
    int ans = 0;

    //pushback({x,y,count})로 입력값 집어 넣기

    //kruskal의 사간복잡도 그 자체
    sort(v.begin(), v.end(), compare); // count값으로 sort

    for (int i = 0; i < E; i++) {
        info value = v[i];
        // 두 점의 root가 다르면 같은 root로 이어줌
        // 같은 root면 서로 이어져 있다는 것!
        if (find(value.x) != find(value.y)) {
            merge(value.x, value.y)
            ans += value.count
        }
    }

}
```

1) 간선들의 가중치 오름차순 정렬



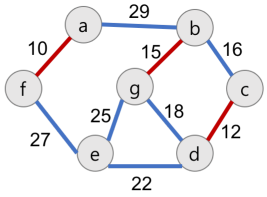
	af	cd	bg	bc	dg	de	eg	ef	ab
간선의 가중치	10	12	15	16	18	22	25	27	29

2)



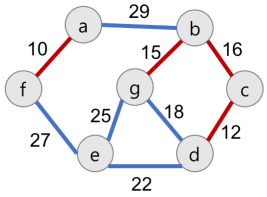
	af	cd	bg	bc	dg	de	eg	ef	ab
간선의 가중치	10	12	15	16	18	22	25	27	29

3)



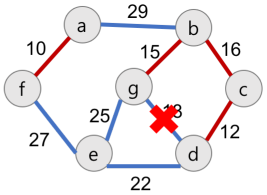
	af	cd	bg	bc	dg	de	eg	ef	ab
간선의 가중치	10	12	15	16	18	22	25	27	29

4)



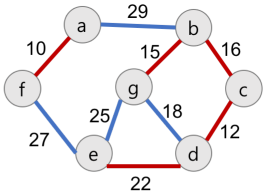
	af	cd	bg	bc	dg	de	eg	ef	ab
간선의 가중치	10	12	15	16	18	22	25	27	29

5) 사이클 형성. dg는 제외



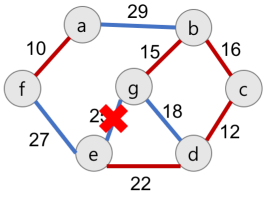
	af	cd	bg	bc	dg	de	eg	ef	ab
간선의 가중치	10	12	15	16	18	22	25	27	29

6)



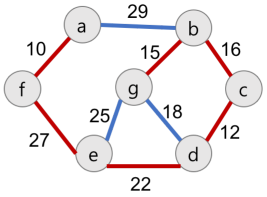
	af	cd	bg	bc	dg	de	eg	ef	ab
간선의 가중치	10	12	15	16	18	22	25	27	29

7) 사이클 형성. eg는 제외



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

8) N-1개의 간선 생성. 종료



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

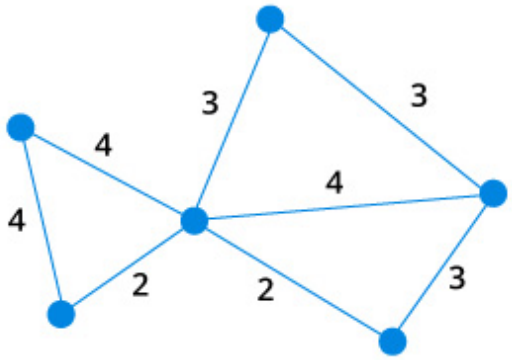
Prim Algorithm

- 목적: MST
- 복잡도
  - $O(ElogV)$ with binary heap
  - $O(E + VlogV)$ with fibonacci heap
  - 한계: 피보나치 힙은 구조의 복잡성 때문에 일반적인 프로그램에서는
- 수도 코드

```
T = ∅;
U = { 1 };
while (U ≠ V)
    let (u, v) be the lowest cost edge such that u ∈ U and v ∈ V - U;
    T = T ∪ {(u, v)}
    U = U ∪ {v}
```

1

Start with a weighted graph



2

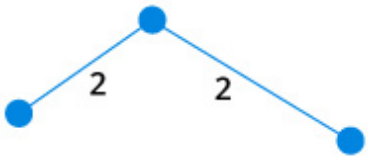
Choose a vertex

3

Choose the shortest edge from this vertex and add it

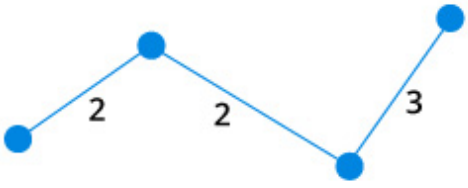
4

Choose the nearest vertex not yet in the solution



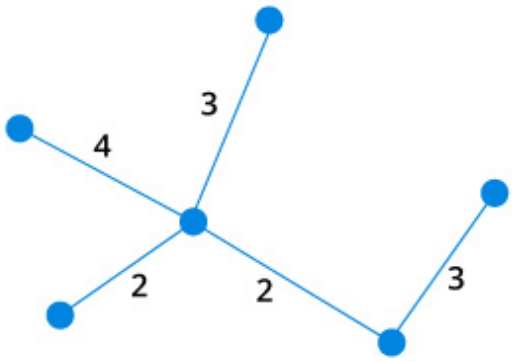
5

Choose the nearest edge not yet in the solution, if there are multiple choices, choose one at random

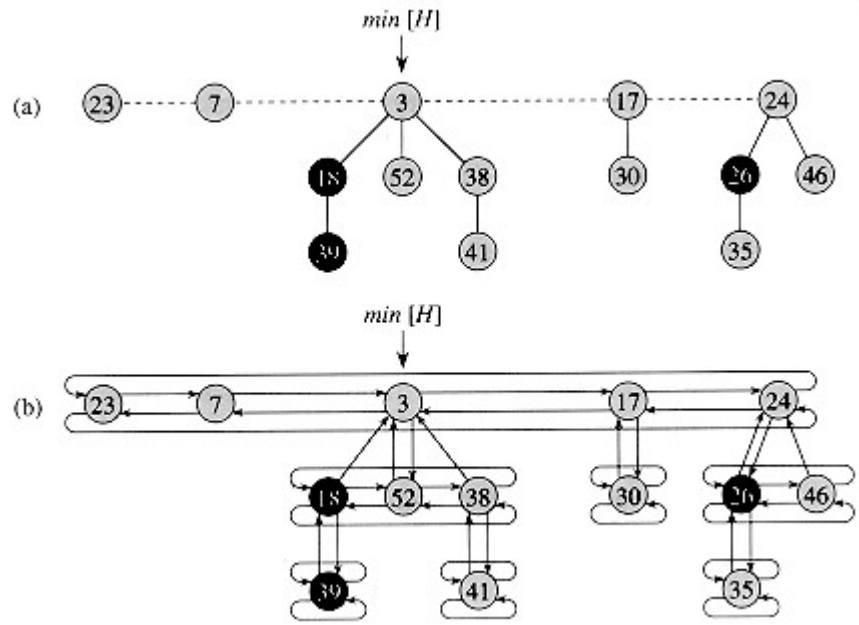


6

Repeat until you have a spanning tree



Fibonacci Heap



- **목적**

정리

알고리즘명	목적	복잡도	장점
BFS	단일점의 가중치 없는 최단경로	$O(\|V\| + \|E\|)$	
Dijkstra	단일점의 최단경로	$O(Elog(V))$	
Bellman-Ford	단일점의 음수 가중치 포함 경로	$O(VE)$	단순, 쉬움
SPFA	단일점의 음수 가중치 포함 경로	최악의 경우: $O(VE)$ 평균: $O(V + E)$	빠름, 어려움
Floyd-Warshall	모든점의 최단 경로	$O(\|V\|^3)$	
Kruskal	MST	$O(ElogV)$	
Prim	MST	$O(ElogV)$ for binary heap, $O(E + VlogV)$ for fibonnacci heap	간선이 많을 때 유리