정윤성

최단 경로

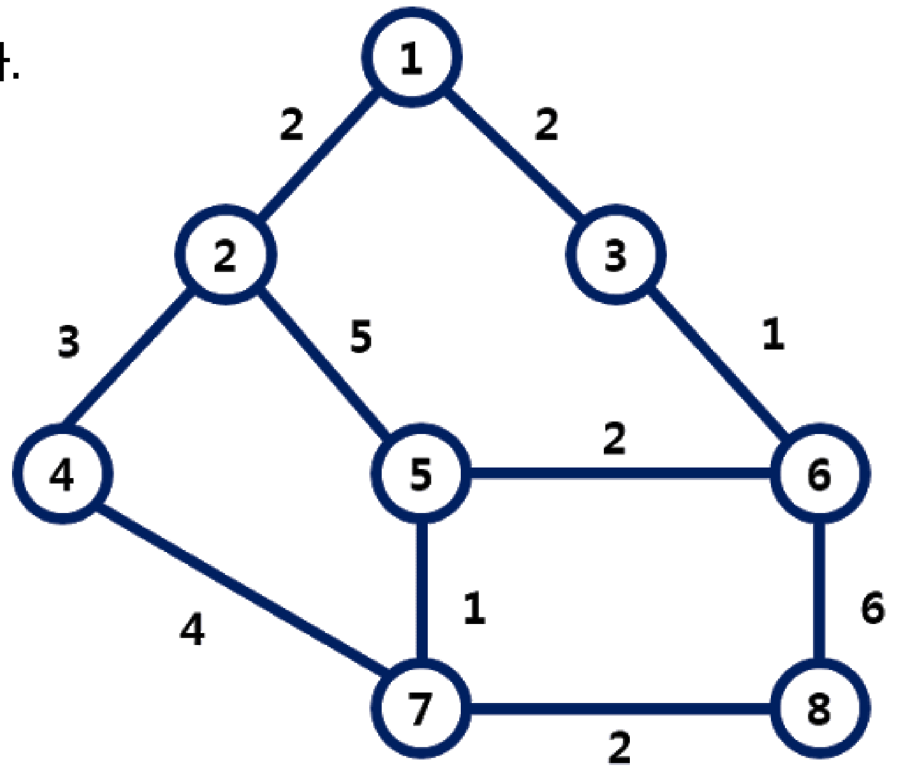
Shortest Path

By POSCAT

그래프 최단경로 알고리즘

- 그래프에서 노드들 간의 최단경로를 구하자.
- 간선의 가중치가 주어지며 경로의 길이는 포함된 간선 가중치의 총합이다.

- 그래프는 G = (V, E)로 표현하며
- V는 vertex(노드)의 집합, E는 edge(간선)의 집합이다.
- Edge는 {s,e,c} (시작점 s, 끝점 e, 가중치 c) 와 같이 표현하며
- 방향 그래프는 edge = {s,e,c}가 s->e로만 갈 수 있고 (화살표 있음)
- 양방향 그래프는 s->e, e->s 모두 가능하다 (선분으로 그림)



그래프 최단경로 알고리즘

- 단일 출발(single-source) 최단경로
  - 단일 노드 v에서 출발하여 그래프 내의 모든 다른 노드에 도착하는 가장 짧은 경로를 찾는 문제.
- 단일 도착(single-destination) 최단경로
  - 모든 노드로부터 출발하여 그래프 내의 한 단일 노드 v로 도착하는 가장 짧은 경로를 찾는 문제.
  - 그래프 내의 노드들을 거꾸로 뒤집으면 단일 출발 최단경로문제와 동일.
- 단일 쌍(single-pair) 최단 경로
  - 주어진 두 노드 u에서 v까지의 최단 경로를 찾는 문제.
- 전체 쌍(all-pair) 최단 경로
  - 그래프 내 모든 노드 쌍들 사이의 최단 경로를 찾는 문제.

Bellman-Ford 알고리즘

- 우선 간선 가중치가 양수일때만 생각하자.
- 임의의 S->T 최단경로는 최대 몇 개의 edge를 포함 가능할까?
- 이를 통해 어떤 시작점 S로부터 모든 다른 노드까지의 최단경로를 O(VE)에 구할 수 있다.

- V개 이상의 edge 포함 -> cycle 존재 -> cycle을 제거할 수 있음 (최단경로임에 모순)

Bellman-Ford 알고리즘

- dist[v]를 최대 i개의 간선만 사용해서 S->v로 도달하는 최단경로의 길이라고 정의하다.
- dist[S] = 0, dist[v] = INF (v!=S)

- 이제 i+1번째 간선을 사용하면, dist[v]에서 v에 달려있는 edge = {u, v}를 사용해서 인접한 dist[u]를 갱신할 수 있다.
- 즉, 다음과 같은 edge relaxation을 수행할 수 있다.
- for every edge {s,e,c}, dist[e] = min(dist[e], dist[s] + c);

- 이를 V-1번 반복하면 된다.

Bellman-Ford 알고리즘

- 만약 음의 가중치를 가지는 간선이 존재한다면?
- 앞의 edge relaxation은 통하지만, V-1번으로 해결되지 않을 수 있다.

- 최단경로에 i(i>V)개의 edge가 필요하다면, cycle이 존재하면서 최단경로이므로 cycle의 총합이 음수 (Negative Cycle)
- 생각해보면 Negative Cycle이 존재하기만 하면 최단경로의 길이가 –INF이므로 해결된다.

Bellman-Ford 코드

Code Explanation

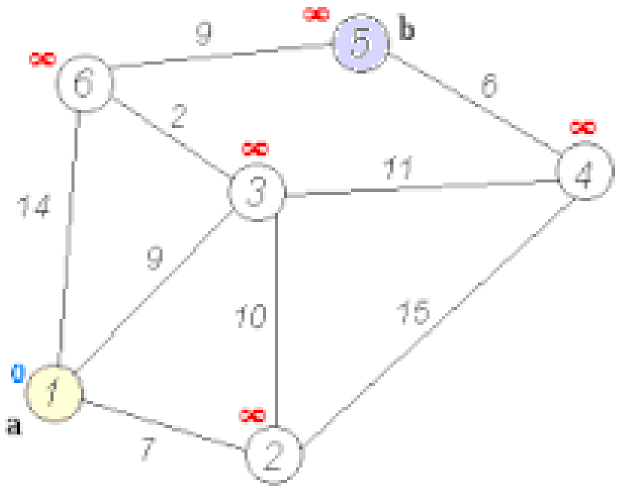
```
for (int i=1; i<=V; i++) dist[i] = INF;

dist[S] = 0;

for(int i=1;i<=V;i++){ // 업데이트 횟수

    bool flag = 0; //만약 V번째에도 update가 일어나면 Negative Cycle 존재

    for all edge {s,e,c}, {

        if(dist[e] > dist[s] + c) dist[e] = dist[s] + c, flag = 1;

    }

    if(i == V && flag) for(int v=1;v<=V;v++) dist[v] = -INF;

}
```

Dijkstra

- 마찬가지로 어떤 점 v에 대해 인접한 dist[u] = dist[v] + c 로 edge relaxation을 한다.
- 이때, 모든 edge cost가 양수이면 더 빠르게 수행할 수 있다.
- S에서 dist[v]가 짧은 순으로 보면서 edge relaxation을 하면, 이후에 한 노드에 의해 이전의 노드가 업데이트되지 않음

Dijkstra

- BFS와 유사하게, 처음에 S에서 시작해서 인접한 노드의 dist[]를 업데이트한다.
- 이때 후보들 중 가장 dist[v]가 짧은 노드에서 Edge Relaxation을 시작해야 한다.
- (이 노드의 dist[]가 더 갱신된다면 위해서는 dist[]가 더 짧은 것이 있어야 하므로 모순)



- 갱신 가능성이 있는 모든 점들 중에서, dist[v]가 가장 작은 v를 얻을 때
- 단순히 dist[v]를 모두 탐색하면 한번당 O(V)
- Priority Queue를 도입하면 한번당 O(log V)

Priority Queue

Code Explanation

```
#include <queue>

using namespace std;

struct cmp{

    bool operator()(int a, int b){ return dist[a] < dist[b]; }

};

priority_queue<int, vector<int>, cmp> pq;   //자료형, 구현체, 정렬기준 (struct cmp 없이 < operator overloading도 가능하다)

pq.push(x) : x를 추가

x = pq.top(); pq에서 최댓값 뽑기

Pq.pop(); pq의 최댓값 제거
```

Dijkstra

Code Explanation

```
Edge를 adjacent list로 저장 (adj[s]에 s에서 도착하는 점의 list 저장)

for all v != S, dist[v] = INF

pq.push(S);

for(!pq.empty()){

    int v = pq.top(); pq.pop(); (if visited[v], continue)

    for(auto edge: adj[v]){

        int u = edge.endpoint, c = edge.cost;

        if(dist[u] > dist[v] + c) {dist[u] = dist[v]+c; pq.push(u);}

    }

}
```

Floyd-Warshall

- 모든 쌍 최단경로를 구할 때 사용
- dp[i][j][k]를 vertex {1, 2, ⋯, k}만 사용하는 i->j 최단경로라 하자.
- dp[i][j][k] = min((k 사용 안함)dp[i][j][k-1], (k 사용함)dp[i][k][k-1] + dp[k][j][k-1])
- 음수 간선도 처리가 불가능하지는 않지만 까다로워 생략한다.
- (dist[i][i] < 0 이면 Negative Cycle이 존재하는건 맞지만 overflow 문제를 처리해야 한다)

Floyd-Warshall 코드

Code Explanation

O(V^3)인데 무엇보다 코드가 간단해서 좋다.

```
for all i,j, dp[i][j] = INF;

for all i, dp[i][j] = 0;

for(int k=1;k<=V;k++)

    for(int i=1;i<=V;i++)

        for(int j=1;j<=V;j++)

            dp[i][j] = min(dp[i][j], dp[i][k]+dp[k][j]);
```

연습 문제

- https://www.acmicpc.net/problem/1865 Bellman-Ford
- https://www.acmicpc.net/problem/1753 Dijkstra
- https://www.acmicpc.net/problem/11404 Floyd-Warshall