최단거리

25.05.26 (월) 오후 5시 ~ 7시

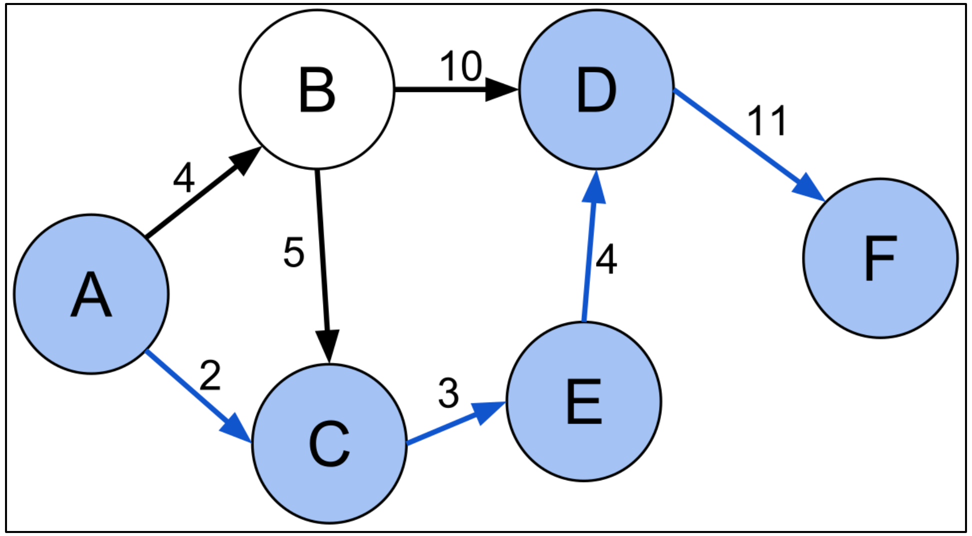
목차

- **최단 경로 알고리즘**
  - **다익스트라 알고리즘 Dijkstra’s Algorithm**
  - **플로이드 워셜 알고리즘 Floyd-Warshall Algorithm**

최단경로

최단경로 | 01 개요

- 이번 스터디에서는 가중치 그래프에서 최단 경로를 찾는 문제에 대해 다룸.
- 가중치 그래프 G = (V, E, W)에 대해 V에 속한 정점 u, v에 대해 가중치의 합이 최소인 경로 path(u, v)를 찾기.
- 문제 상황에 따라 여러 알고리즘을 사용할 수 있음.
  - 구해야 하는 값 Single Source Shortest Path, All Pair Shortest Path
  - 그래프의 형태 – 간선 방향 유무, DAG, 트리, 선인장, ...
  - 가중치의 범위 – 양수, 실수, 0/1, ...
- 대표적인 알고리즘 2가지를 스터디할 예정임.



최단경로 | 02 알고리즘 종류

- **그래프의 가중치**
  - 최단 경로 문제는 모든 간선에 가중치가 부여된 그래프를 전제로 함.
  - 다익스트라 알고리즘은 음수 가중치를 허용하지 않음.
  - 벨만-포드, 플로이드-워셜 알고리즘은 음수 가중치는 허용하나 음수 사이클이 존재하면 올바른 최단 경로를 계산할 수 없음.
- **SSSP (Single Source Shortest Path) 알고리즘**
  - 하나의 정점(Single Source)에서 모든 다른 정점까지의 최단 경로를 구하는 알고리즘
  - 그래프 G = (V, E, W)와 시작 정점 s에 대해 V에 속하는 정점 v에 대해 모든 dist(s, v)를 구함.
  - 대표적인 알고리즘에는 다익스트라, 벨만-포드, 0-1 BFS, SPFA 알고리즘이 있음
- **APSP (All Pairs Shortest Path) 알고리즘**
  - 모든 정점 쌍(All-Pairs)에 대해 최단 경로를 구하는 알고리즘
  - 그래프 G = (V, E, W)와 V에 속하는 정점 u, v에 대해 모든 dist(u, v)를 구함.
  - 대표적인 알고리즘에는 플로이드 워셜 알고리즘과 다익스트라 알고리즘을 모든 정점에 대해 반복하는 방법이 있음.

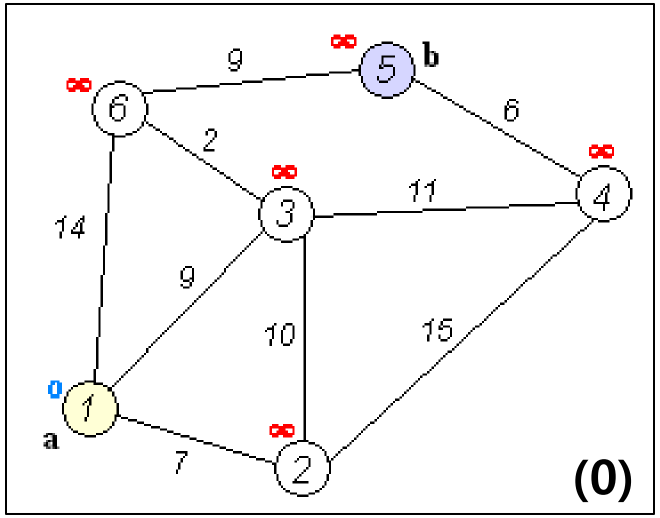
다익스트라

다익스트라 | 01 개요

- **Dijkstra’s Algorithm**
  - 가중치가 0 이상인 그래프에서 SSSP를 푸는 알고리즘.
  - (거리, 정점) 순서 쌍을 저장하는 최소 힙을 사용함.
  - **시간 복잡도 : (인접 리스트 + 우선순위 큐 사용시) O(E log V)** / (인접 행렬 + 배열 사용 시) O(V^2)
  - **그리디(Greedy) + 갱신(Relaxation) 기반 알고리즘**
    1. 시작점(S)까지의 거리는 0, 다른 모든 정점까지의 거리는 INF로 초기화
    2. 아직 거리가 “확정”되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택 (처음에는 S를 선택함)

       거리가 가장 짧은 정점을 선택하는 연산에서, 배열을 사용하면 O(V)의 시간복잡도를 가진다. 우선순위 큐를 사용하면 O(log V)의 시간복잡도를 가진다.
    3. v의 거리를 “확정”시킴
    4. v와 인접하면서 아직 거리가 “확정”되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)
    5. 모든 정점의 거리가 “확정”되었다면 종료 / 그렇지 않으면 2번으로 돌아감
  - 참고) SSSP 알고리즘을 모든 정점에 대해 수행하면 ASAP가 됨. 이때는 거리 배열을 2차원으로 잡아주면 됨.

다익스트라 | 02 동작 방식



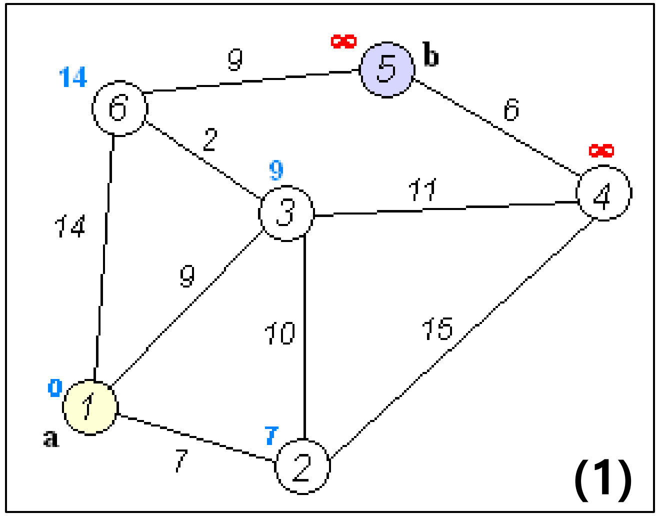
1	2	3	4	5	6
0	INF	INF	INF	INF	INF

시작점(S)까지의 거리는 0, 다른 모든 정점까지의 거리는 INF로 초기화함.

1번 정점을 우선순위 큐에 넣음. (거리, 노드 번호)의 순서쌍으로 넣어짐.

*편의 상, 거리가 확정된 정점은 초록색 표시하겠음

다익스트라 | 02 동작 방식



1	2	3	4	5	6
0	INF	INF	INF	INF	INF

↓

1	2	3	4	5	6
0	7	9	INF	INF	14

아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택 (처음에는 S를 선택함)

- 시작 정점인 정점 1을 선택함

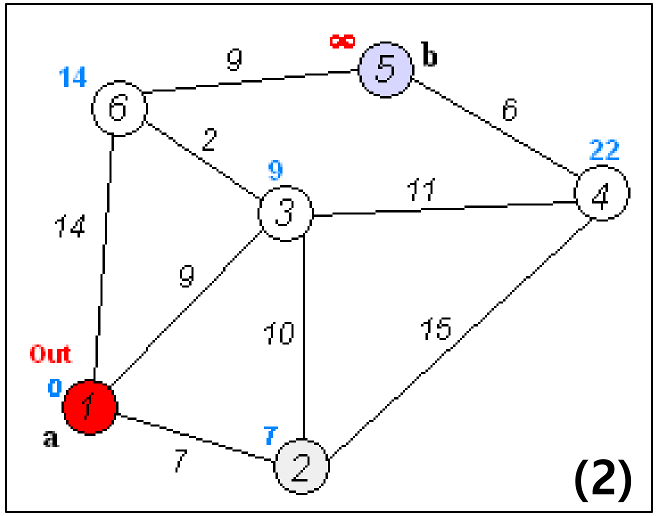
v의 거리를 "확정"시킴

- 정점 1의 거리를 0으로 확정시킴

v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)

- 정점 1과 인접하면서 거리가 확정되지 않은 정점들의 거리를 갱신함.

다익스트라 | 02 동작 방식



1	2	3	4	5	6
0	7	9	INF	INF	14

↓

1	2	3	4	5	6
0	7	9	22	INF	14

아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택

- 거리가 7인 정점 2을 선택함

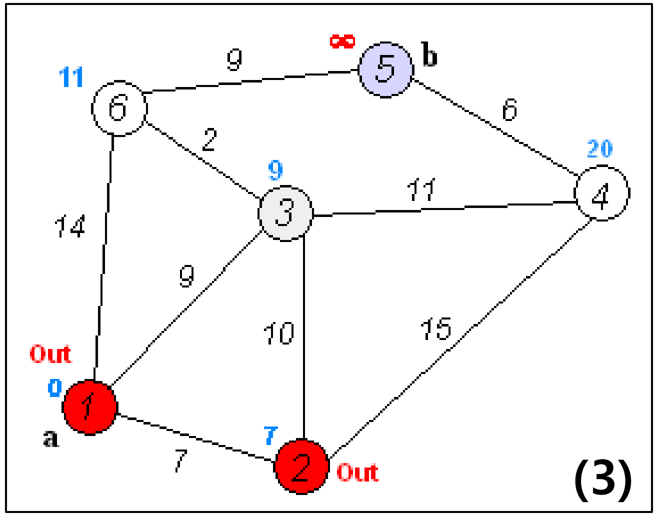
v의 거리를 "확정"시킴

- 정점 2의 거리를 7로 확정시킴

v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)

- 정점 2과 인접하면서 거리가 확정되지 않은 정점들의 거리를 갱신함.

다익스트라 | 02 동작 방식



1	2	3	4	5	6
0	7	9	22	INF	14

↓

1	2	3	4	5	6
0	7	9	20	INF	11

아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택

- 거리가 9인 정점 3을 선택함

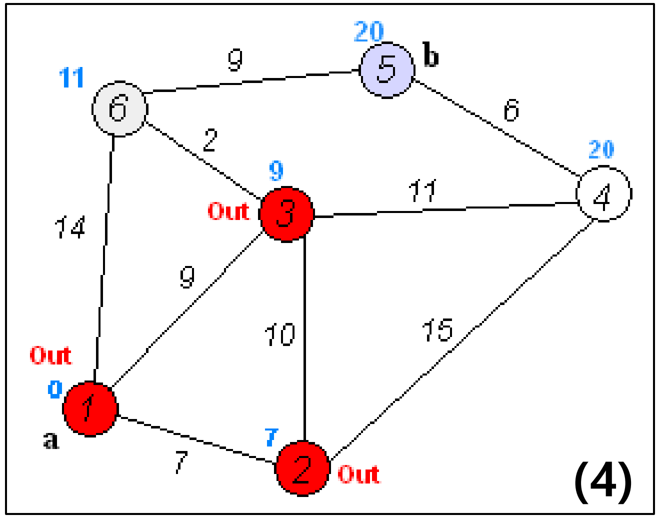
v의 거리를 "확정"시킴

- 정점 3의 거리를 9로 확정시킴

v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)

- 정점 3과 인접하면서 거리가 확정되지 않은 정점들의 거리를 갱신함.

다익스트라 | 02 동작 방식



1	2	3	4	5	6
0	7	9	20	INF	11

↓

1	2	3	4	5	6
0	7	9	20	20	11

아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택

- 거리가 11인 정점 6을 선택함

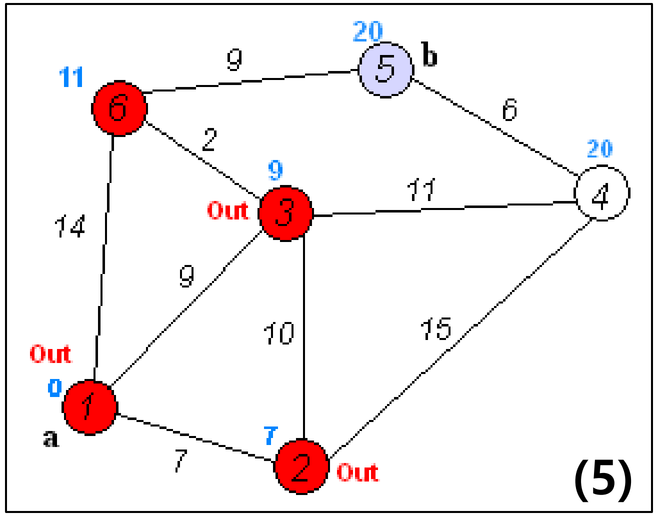
v의 거리를 "확정"시킴

- 정점 11의 거리를 6로 확정시킴

v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)

- 정점 6과 인접하면서 거리가 확정되지 않은 정점들의 거리를 갱신함.

다익스트라 | 02 동작 방식



1	2	3	4	5	6
0	7	9	20	20	11

↓

1	2	3	4	5	6
0	7	9	20	20	11

아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택

- 거리가 20인 정점 5을 선택함. 여러 개면 아무거나 선택해도 됨.

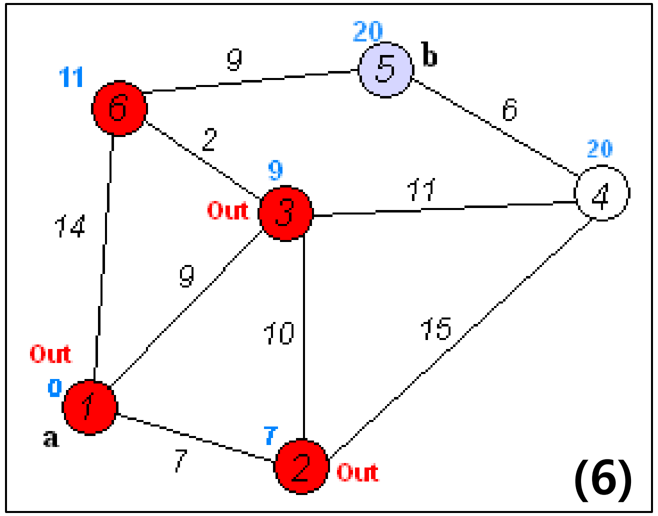
v의 거리를 "확정"시킴

- 정점 5의 거리를 20로 확정시킴

v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)

- 정점 5과 인접하면서 거리가 확정되지 않은 정점들의 거리를 갱신함.

다익스트라 | 02 동작 방식



1	2	3	4	5	6
0	7	9	20	20	11

↓

1	2	3	4	5	6
0	7	9	20	20	11

아직 거리가 "확정"되지 않은 정점 중 거리가 가장 짧은 정점(v) 선택

- 거리가 20인 정점 4을 선택함. 여러개면 아무거나 선택해도 됨.

v의 거리를 "확정"시킴

- 정점 4의 거리를 20로 확정시킴

v와 인접하면서 아직 거리가 "확정"되지 않은 정점들의 거리를 갱신(D[i] ← D[v] + weight)

- 정점 4과 인접하면서 거리가 확정되지 않은 정점들의 거리를 갱신함.

결과적으로, 배열의 값들은 시작 정점인 정점 1에 대한 최단 거리를 나타냄.

* 그림이 없어서 불가피하게 (5)를 가져옴.

다익스트라 | 03 구현

Line 7 ~ 10 : 전역 변수 및 그래프 구조 정의

- graph[i]: 정점 i에서 나가는 간선들 ({비용, 도착 정점} 형태)
- dist[i]: 시작점에서 i번 정점까지의 최소 거리 저장 배열

Line 12 ~ 16 : 다익스트라 알고리즘 : 시작 정점 초기화

- pq는 가중치가 작은 정점이 먼저 나오는 최소 힙이다. (비용, 정점)의 순서쌍이다.
- 시작 정점까지의 거리는 0으로 설정하고, pq에 시작 정점을 추가한다.

Line 17 ~ 22 : 다익스트라 알고리즘 : 비용이 작은 정점 선정

- 큐에서 가장 비용이 작은 정점을 꺼내 각각 (cost, now) 변수에 저장한다.
- 이미 더 짧은 경로로 해당 정점을 처리한 적이 있다면 스킵한다.

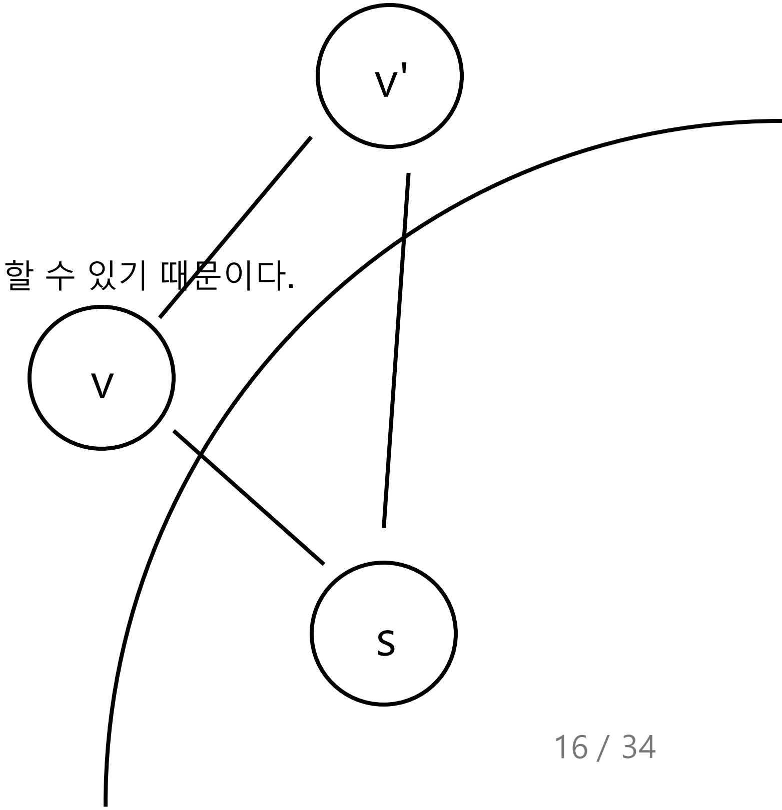
Line 24 ~ 31 : 다익스트라 알고리즘 : 연결된 정점 탐색 및 dist 배열 업데이트

- 현재 정점(now)에서 나가는 모든 간선에 대해
- next = {도착 정점}, next_cost = {시작 정점에서 현재 정점까지의 비용} + {간선 가중치}
- 누적 비용(next_cost)가 기존 최단거리보다 작으면 갱신 후 큐에 넣음.

```
7   const int INF = 1e9; // 충분히 큰 값 (10억)
8   using pii = pair<int, int>; // (비용, 정점)
9   vector<pii> graph[1000];
10  vector<int> dist(1000, INF);
11
12  void dijkstra(int start) {
13      priority_queue<pii, vector<pii>, greater<>> pq;
14      dist[start] = 0;
15      pq.push({0, start});
16
17      while (!pq.empty()) {
18          int cost = pq.top().first;
19          int now = pq.top().second;
20          pq.pop();
21
22          if (cost > dist[now]) continue;
23
24          for (auto& edge : graph[now]) {
25              int next = edge.second;
26              int next_cost = cost + edge.first;
27
28              if (next_cost < dist[next]) {
29                  dist[next] = next_cost;
30                  pq.push({next_cost, next});
31              }
32          }
33      }
34  }
```

다익스트라 | 04 정당성 증명

- 거리가 확정된 정점 집합을 U라고 하자.
  - 처음에는 시작정점 s를 집합 U에 넣는다.
- U에 D[v]가 최소인 정점 v를 추가할 때, D[v]가 v까지의 실제 최단 거리 sp[v]와 같음을 증명하면 된다.
  - v는 V - U에서 D[v]가 최소인 정점을 말한다. 즉, 거리가 확정되지 않은 정점의 집합(V - U)에서 D[v]가 최소인 정점 v를 선택한다.
  - sp[v] : 정점 v에 대해 시작 정점 s로부터 실제 최단 거리
- 귀류법을 사용한다.
  - D[v] > sp[v]라고 가정하자. 즉, D[v]는 아직 거리를 확정할 수 없는 상태라고 할 수 있다.
  - 그렇기에, s에서 v로 가는 경로에서 v를 방문하기 직전에 v’ ∈ V-U를 방문해야 한다. 그래야 D[v]를 확정할 수 있기 때문이다.
  - 하지만 이는 sp[v] = sp[v’] + w(v’, v)라는 의미이고, w(v’, v)는 양수이므로 sp[v’] < sp[v]가 되어야 한다.
  - v는 V-U에서 D[v]가 최소인 정점이 아니므로 모순이다.
  - 따라서 D[v] = sp[v]이다.



다익스트라 | 05 예제 : 백준 1753번 최단경로

- **문제 요약**
  - 방향그래프가 주어지면 주어진 시작점에서 다른 모든 정점으로의 최단 경로를 구하는 프로그램을 작성하시오. 단, 모든 간선의 가중치는 10 이하의 자연수이다.
- **입력**
  - 첫째 줄에 정점의 개수 V와 간선의 개수 E가 주어진다. (1 ≤ V ≤ 20,000, 1 ≤ E ≤ 300,000) 모든 정점에는 1부터 V까지 번호가 매겨져 있다고 가정한다.
  - 둘째 줄에는 시작 정점의 번호 K(1 ≤ K ≤ V)가 주어진다. 셋째 줄부터 E개의 줄에 걸쳐 각 간선을 나타내는 세 개의 정수 (u, v, w)가 순서대로 주어진다. 이는 u에서 v로 가는 가중치 w인 간선이 존재한다는 뜻이다. u와 v는 서로 다르며 w는 10 이하의 자연수이다. 서로 다른 두 정점 사이에 여러 개의 간선이 존재할 수도 있음에 유의한다.
- **출력**
  - 첫째 줄부터 V개의 줄에 걸쳐, i번째 줄에 i번 정점으로의 최단 경로의 경로값을 출력한다. 시작점 자신은 0으로 출력하고, 경로가 존재하지 않는 경우에는 INF를 출력하면 된다.

다익스트라 | 05 예제 : 백준 1753번 최단경로

- **구현**

```cpp
int main(){
  ios::sync_with_stdio(false);
  cin.tie(0); cout.tie(0);

  cin >> n >> m;

  for(int i = 0; i < m; i++){
    int a, b, c;
    cin >> a >> b >> c;
    graph[a].push_back({b, c});
  }

  fill(dist, dist + 1010, INF);

  cin >> start >> end;
  dijkstra(start);

  cout << dist[end];
  return 0;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;

#define INF 1e9

int n, m, start, end;
vector<pair<int, int>> graph[1010];
int dist[1010];

void dijkstra(int start){
  priority_queue<pair<int, int>> pq;
  pq.push({0, start});
  dist[start] = 0;

  while(!pq.empty()){
    int dist = -pq.top().first;
    int now = pq.top().second;
    pq.pop();

    if(dist[now] < dist) continue;
    for(int i = 0; i < graph[now].size(); i++){
      int cost = dist + graph[now][i].second;
      if(cost < dist[graph[now][i].first]){
        dist[graph[now][i].first] = cost;
        pq.push({-cost, graph[now][i].first});
      }
    }
  }
}
```

플로이드 워셜

플로이드 워셜 | 01 개요

- **Floyd-WarShall Algorithm**
  - 그래프에서 APSP를 푸는 알고리즘. 음의 가중치는 허용하나, 음의 사이클이 존재하면 올바른 최단경로를 계산할 수 없음.
  - 정점의 수가 작을 때 사용.
  - **시간 복잡도 : O(V^3)**
  - 공간 복잡도 : O(V^3)
  - **DP 기반 알고리즘 : dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])**

    *정점 i에서 정점 j로 직접 가는 것보다, 중간에 정점 k를 경유해서 가는 경로가 더 짧다면, 그 경로로 업데이트한다는 의미의 점화식

    - i → j로 가는 최단거리는 {지금까지 계산한 거리 dist[i][j]}와 {어떤 중간 정점 k를 거쳐서 가는 거리 dist[i][k] + dist[k][j]} 중 작은 값으로 갱신
    - dist[i][j]: 현재까지 알고 있는 정점 i에서 j로 가는 최단 거리
    - k: 중간에 거칠 수 있는 중간 정점

플로이드 워셜 | 02 예제 : 백준 11404번 플로이드

- **문제 요약**
  - n(2 ≤ n ≤ 100)개의 도시가 있다. 그리고 한 도시에서 출발하여 다른 도시에 도착하는 m(1 ≤ m ≤ 100,000)개의 버스가 있다. 각 버스는 한 번 사용할 때 필요한 비용이 있다.
  - 모든 도시의 쌍 (A, B)에 대해서 도시 A에서 B로 가는데 필요한 비용의 최솟값을 구하는 프로그램을 작성하시오.
- **입력**
  - 첫째 줄에 도시의 개수 n이 주어지고 둘째 줄에는 버스의 개수 m이 주어진다. 그리고 셋째 줄부터 m+2줄까지 다음과 같은 버스의 정보가 주어진다. 먼저 처음에는 그 버스의 출발 도시의 번호가 주어진다. 버스의 정보는 버스의 시작 도시 a, 도착 도시 b, 한 번 타는데 필요한 비용 c로 이루어져 있다. 시작 도시와 도착 도시가 같은 경우는 없다. 비용은 100,000보다 작거나 같은 자연수이다.
  - 시작 도시와 도착 도시를 연결하는 노선은 하나가 아닐 수 있다.
- **출력**
  - n개의 줄을 출력해야 한다. i번째 줄에 출력하는 j번째 숫자는 도시 i에서 j로 가는데 필요한 최소 비용이다. 만약, i에서 j로 갈 수 없는 경우에는 그 자리에 0을 출력한다.

플로이드 워셜 | 02 예제 : 백준 11404번 플로이드

- 구현

```cpp
#include <bits/stdc++.h>
using namespace std;

#define INF 20202020
int d[101][101];
int n, m;
int u, v, cost;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0); cout.tie(0);

  cin >> n;
  for(int i = 1; i <= n; i++){
    fill(d[i] + 1, d[i] + n + 1, INF);
    d[i][i] = 0;
  }

  cin >> m;
  while(m--){
    cin >> u >> v >> cost;
    d[u][v] = min(d[u][v], cost);
  }
```

```cpp
  for(int k = 1; k <= n; k++){
    for(int i = 1; i <= n; i++){
      for(int j = 1; j <= n; j++){
        d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
      }
    }
  }

  for(int i = 1; i <= n; i++){
    for(int j = 1; j <= n; j++){
      if(d[i][j] == INF) cout << 0 << " ";
      else cout << d[i][j] << " ";
    }
    cout << "\n";
  }

  return 0;
}
```

문제

- **기초**

  백준 14938번 서강그라운드

  백준 21940번 가운데에서 만나기

  백준 1504번 특정한 최단 경로

- **실력**

  백준 11780번 플로이드 2

  백준 11779번 최소비용 구하기 2

종강!