10. Dijkstra Algorithm

이화여자대학교 황채원

최단 경로

그래프에서 두 정점 사이를 연결하는 경로들 중에서 간선의 가중치 합이 최소가 되는 경로를 찾는 문제입니다.

Single-Source Shortest Path (SSP)

: 하나의 출발점에서 각 정점까지 도달하는데 드는 비용을 계산하여 최단경로를 구하는 방법입니다.

- 다익스트라 알고리즘: 모든 간선의 가중치가 음수가 아닐 때 사용할 수 있습니다. -> O(ElogV)
- 벨만-포드 알고리즘: 간선의 가중치가 음수일 때도 사용할 수 있습니다. 다익스트라 알고리즘보다 시간 복잡도가 높습니다. -> O(VE)

최단 경로

All-Pairs Shortest Path (ASP)

: 그래프 내 가능한 모든 정점 쌍 사이의 최단 경로를 찾는 문제입니다.

- 플로이드-워셜 알고리즘 : 동적 프로그래밍을 이용하여 모든 노드 간 최단 경로를 구하는 알고리즘입니다.

  시간복잡도 O(V^3)

Dijkstra

- 하나의 출발점으로부터 각 정점까지의 최단 경로를 구하는 Single Source Shortest Path 알고리즘입니다.
- 음수 가중치가 없는 그래프에서 동작합니다.

  가중치가 음수인 간선이 있다면 알고리즘이 무한 루프에 빠질 수 있습니다.
- 시작 정점으로부터 가장 가까운 정점부터 탐색하는 그리디 기반의 알고리즘

  그리디 : 주어진 상황에서 가장 좋은 경로를 선택한다.

Relaxation

- 특정 간선을 통해 다른 정점까지의 거리를 줄일 수 있는지 확인하고,

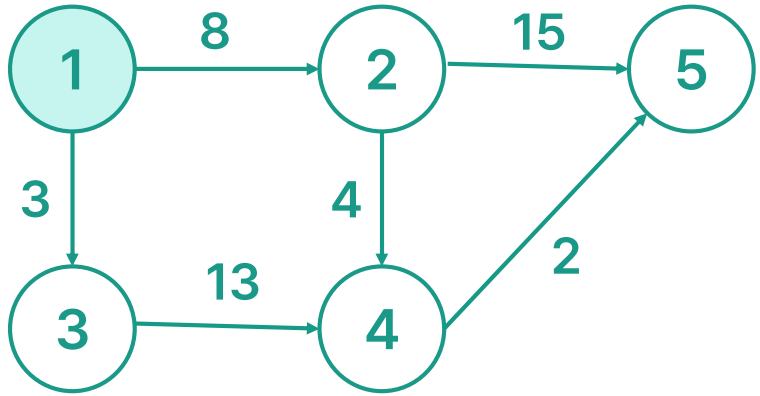
  가능하다면 해당 거리를 업데이트하는 과정입니다.

1. 간선 (u, v)에 대해, u를 거쳐 v로 가는 경로가 현재 알려진 v까지의 최단 거리보다 짧을 경우,
2. v까지의 거리(가중치)를 업데이트합니다.

   즉, distance[v] = distance[u] + weight(u, v)로 설정합니다.

Dijkstra

- 최단거리 배열의 시작 정점은 0, 나머지 정점은 무한대로 초기화합니다.

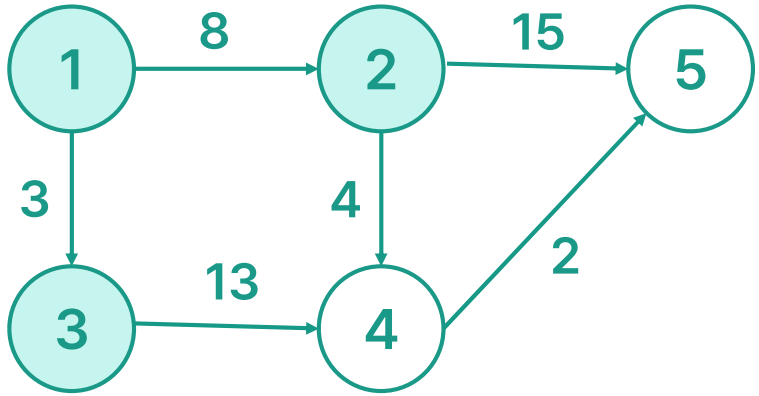


1	2	3	4	5
0	∞	∞	∞	∞

출처 : Do it! 알고리즘 코딩 테스트

Dijkstra

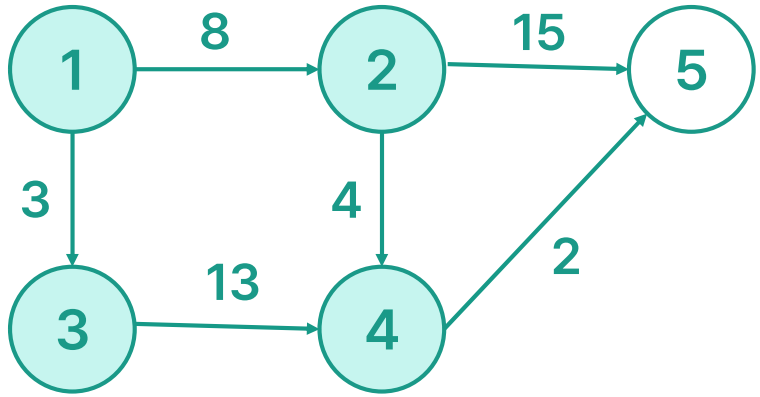
- 최단거리 배열에서 가장 값이 작은 노드를 선택하고, 배열을 업데이트합니다.



1	2	3	4	5
0	8	3	∞	∞

출처 : Do it! 알고리즘 코딩 테스트

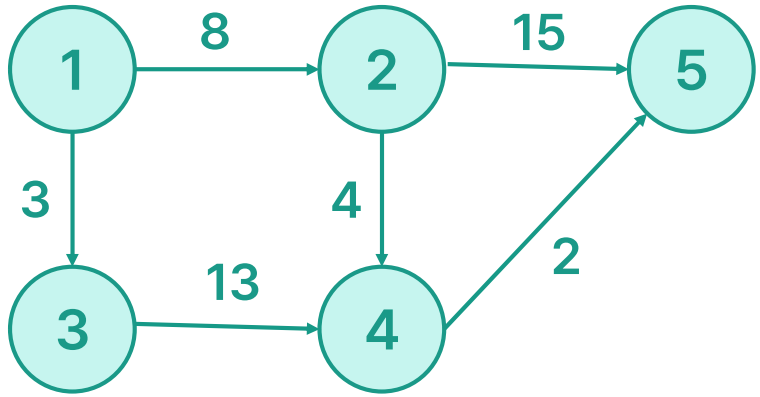
Dijkstra



1	2	3	4	5
0	8	3	16	∞

출처 : Do it! 알고리즘 코딩 테스트

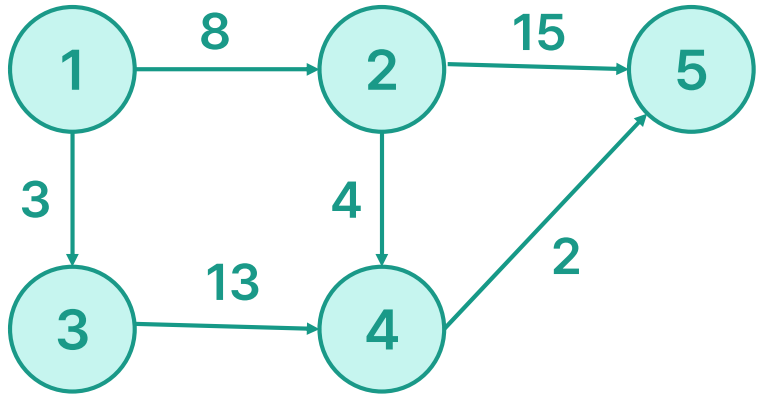
Dijkstra



1	2	3	4	5
0	8	3	12	23

출처 : Do it! 알고리즘 코딩 테스트

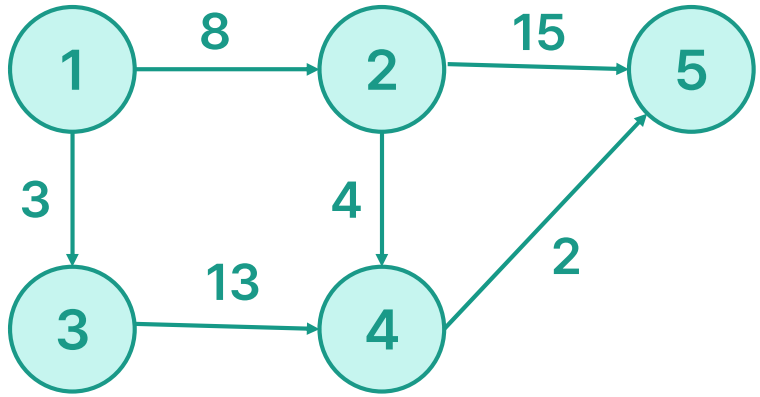
Dijkstra



1	2	3	4	5
0	8	3	12	14

출처 : Do it! 알고리즘 코딩 테스트

Dijkstra



1	2	3	4	5
0	8	3	12	14

출처 : Do it! 알고리즘 코딩 테스트

Pseudo Code

1. 모든 정점들의 거리 값을 무한대로 설정한다. 시작 정점의 거리 값은 0으로 설정한다.

2. 모든 정점들을 미방문 상태로 표시한다.

3. 우선순위 큐를 초기화하고, 시작 정점만을 넣는다(거리 값이 0).

Pseudo Code

반복:

a. 우선순위 큐에서 거리 값이 가장 작은 정점을 꺼낸다. 이 정점을 현재 정점으로 한다.

b. 현재 정점을 방문한 것으로 표시한다.

c. 현재 정점과 인접한 모든 미방문 정점에 대해, 다음을 수행한다:

i. (현재 정점의 거리 값 + 현재 정점에서 인접 정점까지의 간선 가중치)를 계산한다.

ii. 새로운 거리 값이 기존의 거리 값보다 작다면, 인접 정점의 거리 값을 새로운 거리 값으로 업데이트한다.

iii. 업데이트된 거리 값으로 인접 정점을 우선순위 큐에 다시 넣는다.

Code

```cpp
const int INF = numeric_limits<int>::max();
typedef pair<int, int> pii; // 정점 번호와 가중치를 저장하는 pair 정의
typedef vector<vector<pii>> Graph; // 그래프를 인접 리스트로 표현

// 다익스트라 알고리즘 함수
vector<int> dijkstra(const Graph& graph, int src) {
    int n = graph.size();
    vector<int> dist(n, INF); // 각 정점까지의 최단 거리를 저장하는 배열
    priority_queue<pii, vector<pii>, greater<pii>> pq; // 최소 힙 우선순위 큐

    dist[src] = 0; // 시작 정점까지의 거리는 0
    pq.push({0, src});
```

Code


```
    while (!pq.empty()) {
        int currentDist = pq.top().first; // 현재 정점까지의 거리
        int current = pq.top().second; // 현재 정점
        pq.pop();

        // 이미 더 짧은 경로를 찾은 경우 무시
        if (dist[current] < currentDist) continue;

        // 현재 정점과 인접한 모든 정점을 확인
        for (auto& edge : graph[current]) {
            int next = edge.first; // 인접 정점
            int weight = edge.second; // 현재 정점에서 인접 정점까지의 가중치

            // 현재 정점을 거쳐 인접 정점으로 가는 거리가 더 짧은 경우 업데이트
            if (dist[next] > currentDist + weight) {
                dist[next] = currentDist + weight;
                pq.push({dist[next], next}); // 업데이트된 거리로 우선순위 큐에 삽입
            }
        }
    }

    return dist; // 모든 정점까지의 최단 거리 반환
}
```

시간복잡도

인접리스트 + 우선순위 큐를 사용했을 때 다익스트라 알고리즘의 시간복잡도는 O((V+E)logV) 입니다.

- 우선순위 큐에서 요소를 삽입/추출하는 시간 복잡도 O(logV)
- 모든 정점을 우선순위 큐에 삽입/추출하는 데 O(VlogV) 시간이 소요
- 모든 간선을 한 번씩 검사하는 데 O(E) 시간이 소요
- 각 인접 정점을 우선순위 큐에 다시 삽입하는 과정에서 추가로 O(ElogV) 시간이 소요

다른 표현들?

O(ElogV)

- V <= E
- 중복 간선을 포함하지 않는 경우, E는 항상 V^2 이하
- logE < log(V^2)
- log(V^2)은 2logV이기 때문에 O(logV)
- 따라서, 다익스트라 알고리즘의 전체 시간 복잡도를 O(ElogV)로도 표현할 수 있습니다.

우선순위 큐 말고 배열에 저장한다면?

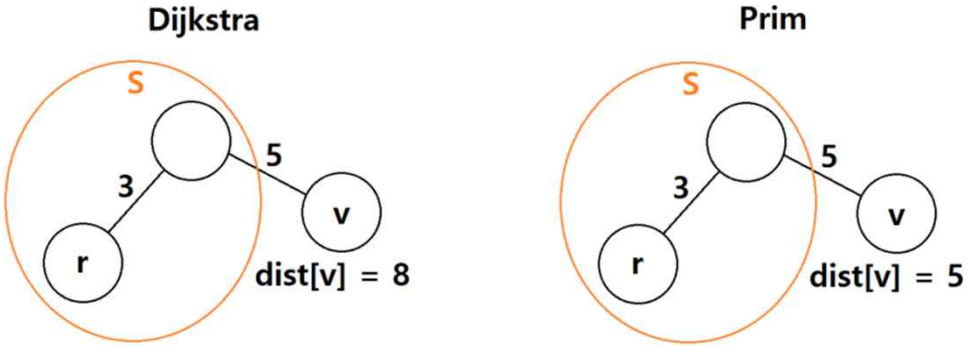
O(V^2)

- 가장 작은 값을 찾으려면 배열을 순차적으로 탐색해야한다.

Prim Algorithm

- 그래프 상에 존재하는 모든 노드들을 최소 비용으로 연결시키는 알고리즘입니다.
- 임의의 시작 정점에서 출발하여 트리에 인접한 간선 중 가장 가중치가 낮은 간선을 선택하고, 해당 간선이 연결하는 정점을 트리에 추가합니다. 이 과정을 모든 정점이 트리에 포함될 때까지 반복합니다.
- 즉, 최소 신장 트리(MST)를 찾아내는 알고리즘입니다.

Prim Algorithm



- 다익스트라 알고리즘에서 r, v 사이의 거리는 8로 계산되지만

  프림 알고리즘에서 r, v 사이의 거리는 5로 계산된다.

https://m.blog.naver.com/zbqmgldjfh/222380542936

Prim Algorithm


```cpp
int prim(const vector<vector<pii>>& graph, int V) {
    int totalWeight = 0; // MST의 총 가중치
    vector<bool> selected(V, false); // MST에 포함된 정점들
    priority_queue<pii, vector<pii>, greater<pii>> pq; // 최소 힙

    // 시작 정점을 0으로 선택
    pq.push({0, 0}); // (가중치, 정점 번호)

    while (!pq.empty()) {
        int weight = pq.top().first; // 현재 간선의 가중치
        int u = pq.top().second; // 현재 정점
        pq.pop();

        // 이미 MST에 포함된 정점이면 스킵
        if (selected[u]) continue;

        // 정점을 MST에 포함시키고 가중치를 추가
        selected[u] = true;
        totalWeight += weight;

        // 현재 정점과 인접한 모든 정점을 확인
        for (auto& edge : graph[u]) {
            int v = edge.second; // 인접 정점
            int nextWeight = edge.first; // 인접 정점까지의 가중치

            // MST에 아직 포함되지 않은 정점이라면 우선순위 큐에 추가
            if (!selected[v]) {
                pq.push({nextWeight, v});
            }
        }
    }

    return totalWeight; // MST의 총 가중치 반환
}
```

BOJ 1753 최단경로

최단경로 성공

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	197567	59290	30259	25.426%

문제

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.

BOJ 1753 최단경로


```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);

    int V, E, K;
    cin >> V >> E >> K;
    Graph graph(V + 1);

    for (int i = 0; i < E; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
    }

    vector<int> min_distance = dijkstra(graph, K);

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

    return 0;
}
```

BOJ 1753 최단경로


```cpp
vector<int> dijkstra(const Graph& graph, int src) {
    int n = graph.size();
    vector<int> dist(n, INF);
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    dist[src] = 0;
    pq.push({0, src});

    while (!pq.empty()) {
        int currentDist = pq.top().first;
        int current = pq.top().second;
        pq.pop();

        if (dist[current] < currentDist) continue;

        for (auto& edge : graph[current]) {
            int next = edge.first, nextDist = currentDist + edge.second;
            if (dist[next] > nextDist) {
                dist[next] = nextDist;
                pq.push({nextDist, next});
            }
        }
    }

    return dist;
}
```

BOJ 1504 특정한 최단 경로

특정한 최단 경로

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	82727	22069	14955	24.998%

문제

방향성이 없는 그래프가 주어진다. 세준이는 1번 정점에서 N번 정점으로 최단 거리로 이동하려고 한다. 또한 세준이는 두 가지 조건을 만족하면서 이동하는 특정한 최단 경로를 구하고 싶은데, 그것은 바로 임의로 주어진 두 정점은 반드시 통과해야 한다는 것이다.

세준이는 한번 이동했던 정점은 물론, 한번 이동했던 간선도 다시 이동할 수 있다. 하지만 반드시 최단 경로로 이동해야 한다는 사실에 주의하라. 1번 정점에서 N번 정점으로 이동할 때, 주어진 두 정점을 반드시 거치면서 최단 경로로 이동하는 프로그램을 작성하시오.

입력

첫째 줄에 정점의 개수 N과 간선의 개수 E가 주어진다. ($2 \le N \le 800$, $0 \le E \le 200,000$) 둘째 줄부터 E개의 줄에 걸쳐서 세 개의 정수 a, b, c가 주어지는데, a번 정점에서 b번 정점까지 양방향 길이 존재하며, 그 거리가 c라는 뜻이다. ($1 \le c \le 1,000$) 다음 줄에는 반드시 거쳐야 하는 두 개의 서로 다른 정점 번호 v_1과 v_2가 주어진다. ($v_1 \neq v_2$, $v_1 \neq N$, $v_2 \neq 1$) 임의의 두 정점 u와 v사이에는 간선이 최대 1개 존재한다.

출력

첫째 줄에 두 개의 정점을 지나는 최단 경로의 길이를 출력한다. 그러한 경로가 없을 때에는 -1을 출력한다.

BOJ 1504 특정한 최단 경로

- 양방향 그래프 : a번 정점에서 b번 정점으로 가는 경로와 b번 정점에서 a번 정점으로 가는 경로 모두를 그래프에 추가합니다.
- 다익스트라 알고리즘 활용:
  - 1번 정점 -> v1, v2
  - v1 -> v2
  - N번 정점 -> v1, v2
- 최단 경로는 (1 -> v1 -> v2 -> N) 또는 (1 -> v2 -> v1 -> N) 경로 중 더 짧은 경로입니다.

BOJ 1504 특정한 최단 경로


```
int main() {
    int N, E, v1, v2;
    cin >> N >> E;
    Graph graph(N + 1);

    for (int i = 0; i < E; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
        graph[v].push_back({u, w}); // 양방향 그래프
    }

    cin >> v1 >> v2;

    // 다익스트라 알고리즘으로 최단 경로 길이를 계산
    vector<int> distFrom1 = dijkstra(graph, 1);
    vector<int> distFromV1 = dijkstra(graph, v1);
    vector<int> distFromV2 = dijkstra(graph, v2);

    // 1->v1->v2->N과 1->v2->v1->N 중 최단 경로 찾기
    long long path1 = (long long)distFrom1[v1] + distFromV1[v2] + distFromV2[N];
    long long path2 = (long long)distFrom1[v2] + distFromV2[v1] + distFromV1[N];
    long long shortestPath = min(path1, path2);

    if (shortestPath >= INF) cout << "-1\n";
    else cout << shortestPath << "\n";

    return 0;
}
```

BOJ 1504 특정한 최단 경로


```cpp
vector<int> dijkstra(const Graph& graph, int src) {
    int n = graph.size();
    vector<int> dist(n, INF);
    priority_queue<pii, vector<pii>, greater<pii>> pq;

    dist[src] = 0;
    pq.push({0, src});

    while (!pq.empty()) {
        int currentDist = pq.top().first;
        int current = pq.top().second;
        pq.pop();

        if (dist[current] < currentDist) continue;

        for (auto& edge : graph[current]) {
            int next = edge.first, nextDist = currentDist + edge.second;
            if (dist[next] > nextDist) {
                dist[next] = nextDist;
                pq.push({nextDist, next});
            }
        }
    }

    return dist;
}
```

BOJ 4485 녹색 옷 입은 애가 젤다지?

녹색 옷 입은 애가 젤다지? 성공 다국어

★ 한국어

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	26975	14470	9883	51.136%

문제

젤다의 전설 게임에서 화폐의 단위는 루피(rupee)다. 그런데 간혹 '도둑루피'라 불리는 검정색 루피도 존재하는데, 이걸 획득하면 오히려 소지한 루피가 감소하게 된다!

젤다의 전설 시리즈의 주인공, 링크는 지금 도둑루피만 가득한 N x N 크기의 동굴의 제일 왼쪽 위에 있다. [0][0]번 칸이기도 하다. 왜 이런 곳에 들어왔냐고 묻는다면 밖에서 사람들이 자꾸 "젤다의 전설에 나오는 녹색 애가 젤다지?"라고 물어봤기 때문이다. 링크가 녹색 옷을 입은 주인공이고 젤다는 그냥 잡혀있는 공주인데, 게임 타이틀에 젤다가 나와 있다고 자꾸 사람들이 이렇게 착각하니까 정신병에 걸릴 위기에 놓인 것이다.

하여튼 젤다...아니 링크는 이 동굴의 반대편 출구, 제일 오른쪽 아래 칸인 [N-1][N-1]까지 이동해야 한다. 동굴의 각 칸마다 도둑루피가 있는데, 이 칸을 지나면 해당 도둑루피의 크기만큼 소지금을 잃게 된다. 링크는 잃는 금액을 최소로 하여 동굴 건너편까지 이동해야 하며, 한 번에 상하좌우 인접한 곳으로 1칸씩 이동할 수 있다.

링크가 잃을 수밖에 없는 최소 금액은 얼마일까?

BOJ 4485 녹색 옷 입은 애가 젤다지?

입력

입력은 여러 개의 테스트 케이스로 이루어져 있다.

각 테스트 케이스의 첫째 줄에는 동굴의 크기를 나타내는 정수 N이 주어진다. (2 ≤ N ≤ 125) N = 0인 입력이 주어지면 전체 입력이 종료된다.

이어서 N개의 줄에 걸쳐 동굴의 각 칸에 있는 도둑루피의 크기가 공백으로 구분되어 차례대로 주어진다. 도둑루피의 크기가 k면 이 칸을 지나면 k루피를 잃는다는 뜻이다. 여기서 주어지는 모든 정수는 0 이상 9 이하인 한 자리 수다.

출력

각 테스트 케이스마다 한 줄에 걸쳐 정답을 형식에 맞춰서 출력한다. 형식은 예제 출력을 참고하시오.

예제 입력 1 복사

```
3
5 5 4
3 9 1
3 2 7
5
3 7 2 0 1
2 8 0 9 1
1 2 1 8 1
9 8 9 2 0
3 6 5 1 5
7
9 0 5 1 1 5 3
4 1 2 1 6 5 3
0 7 6 1 6 8 5
1 1 7 8 3 2 3
9 4 0 7 6 4 1
5 8 3 2 4 8 3
7 4 8 4 8 3 4
0
```

예제 출력 1 복사

```
Problem 1: 20
Problem 2: 19
Problem 3: 36
```

BOJ 4485 녹색 옷 입은 애가 젤다지?

- 2차원 그리드 형태의 맵에서 각 칸을 정점으로, 상하좌우 이동을 간선으로 간주해봅시다.
- 각 칸에서의 도둑루피의 값을 간선의 가중치로 고려할 수 있습니다.

BOJ 4485 녹색 옷 입은 애가 젤다지?


```cpp
int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);

    int N, problem = 1;
    while (true) {
        cin >> N;
            if (N == 0) break;

        Matrix graph(N, vector<int>(N));
        for (int i = 0; i < N; ++i) {
            for (int j = 0; j < N; ++j) {
                cin >> graph[i][j];
            }
        }

        vector<vector<int>> min_cost = dijkstra(graph, N); // 다익스트라 알고리즘
        cout << "Problem " << problem++ << ": " << min_cost[N-1][N-1] << "\n"; // 결과 출력
    }

    return 0;
}
```

BOJ 4485 녹색 옷 입은 애가 젤다지?


```cpp
const int INF = numeric_limits<int>::max();
typedef pair<int, int> pii; // 위치 좌표
typedef pair<int, pii> Node; // 비용과 위치 좌표
typedef vector<vector<int>> Matrix;

vector<vector<int>> dijkstra(const Matrix& graph, int N) {
    vector<vector<int>> dist(N, vector<int>(N, INF));
    priority_queue<Node, vector<Node>, greater<Node>> pq;

    // 시작 위치(0, 0)의 비용을 초기화
    dist[0][0] = graph[0][0];
    pq.push({graph[0][0], {0, 0}});

    // 상하좌우 이동을 위한 방향 벡터
    int dx[4] = {0, 1, 0, -1};
    int dy[4] = {-1, 0, 1, 0};
```

BOJ 4485 녹색 옷 입은 애가 젤다지?


```
    while (!pq.empty()) {
        int cost = pq.top().first; // 현재 칸까지의 비용
        pii pos = pq.top().second; // 현재 칸의 위치 (y, x)
        pq.pop();

        if (dist[pos.first][pos.second] < cost) continue; // 이미 더 작은 비용으로 방문한

        // 상하좌우 칸을 확인
        for (int i = 0; i < 4; ++i) {
            int ny = pos.first + dy[i];
            int nx = pos.second + dx[i];

            // 맵 범위 내에 있는지 확인
            if (ny >= 0 && ny < N && nx >= 0 && nx < N) {
                int nextCost = cost + graph[ny][nx]; // 다음 칸까지의 총 비용
                if (nextCost < dist[ny][nx]) { // 최소 비용 갱신이 가능한 경우
                    dist[ny][nx] = nextCost;
                    pq.push({nextCost, {ny, nx}}); // 우선순위 큐에 삽입
                }
            }
        }
    }

    return dist;
}
```

BOJ 11404 플로이드

플로이드

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
1 초	256 MB	67052	28612	20114	41.857%

문제

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.

BOJ 11404 플로이드

- 다익스트라 알고리즘을 n개의 도시에 대해 실행해보면 되지 않을까?

  -> O(n*(mlogn))의 시간 복잡도

- 모든 출발지로부터 모든 도착지까지의 최단 경로를 구해야하는 이 문제는

  플로이드-워셜 알고리즘을 사용해서 해결해야 합니다.

  -> 시간복잡도 O(n^3)

BOJ 11404 플로이드

- 다익스트라를 사용해도 맞출 수는 있습니다.(위: 플로이드 / 아래: 다익스트라)

```cpp
int main() {
    int n, m;
    cin >> n >> m;
    vector<vector<pii>> graph(n+1);

    for (int i = 0; i < m; ++i) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
    }

    for (int i = 1; i <= n; ++i) {
        vector<int> min_cost = dijkstra(graph, i);
        for (int j = 1; j <= n; ++j) {
            if (i == j) cout << "0 ";
            else if (min_cost[j] == INF) cout << "0 ";
            else cout << min_cost[j] << " ";
        }
        cout << "\n";
    }

    return 0;
}
```

결과	메모리	시간	언어	코드 길이
맞았습니다!! ✔	2020 KB	92 ms	C++17 / 수정	1479 B
맞았습니다!!	3104 KB	104 ms	C++17 / 수정	1391 B

BOJ 11404 플로이드


```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;

const int INF = numeric_limits<int>::max();

// 플로이드-워셜 알고리즘
void floydWarshall(vector<vector<int>>& dist, int n) {
    for (int k = 0; k < n; ++k) { // 경유하는 정점
        for (int i = 0; i < n; ++i) { // 시작 정점
            for (int j = 0; j < n; ++j) { // 도착 정점
                // 경유해서 가는 비용이 더 저렴한 경우, 비용을 업데이트
                if (dist[i][k] < INF && dist[k][j] < INF) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}
```

```cpp
int main() {
    int n, m;
    cin >> n >> m;

    vector<vector<int>> dist(n, vector<int>(n, INF));

    for (int i = 0; i < n; ++i) dist[i][i] = 0;

    for (int i = 0; i < m; ++i) {
        int a, b, c;
        cin >> a >> b >> c;
        a--; b--; // 인덱스를 0부터 시작하도록 조정
        dist[a][b] = min(dist[a][b], c);
    }

    // 플로이드-워셜 알고리즘을 사용하여 모든 정점 쌍의 최단 거리를 계산
    floydWarshall(dist, n);

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

문제

필수 문제

- BOJ 5972 (택배배송)
- BOJ 1916 (최소비용 구하기)
- BOJ 1446 (지름길)
- BOJ 14284 (간선 이어가기2)
- BOJ 23793 (두 단계 최단 경로 1)

심화 문제

- BOJ 1238 (파티)
- BOJ 10282 (해킹)
- BOJ 17396 (백도어)