챕터 5. Dynamic Programming (동적 계획법) - (3) 최단경로 문제와 플로이드 알고리즘

최단 경로 문제

- 주어진 그래프에서 모든 정점의 쌍에 대한 최단 경로를 구하시오.

- 엄밀한 문제 정의:
  - G = (V, E): G는 그래프, V는 정점(vertex)의 집합 E는 간선(edge)의 집합
  - 그래프 G는 방향성(directed), 가중치(weighted) 그래프임
  - 최단 경로는 단순 경로(simple path): 같은 정점을 두 번 거치지 않음(acyclic)

최단 경로 문제의 이해

- 단순 무식한 방법으로 해결하기
  - 각 정점에서 다른 정점으로 가는 모든 경로를 구한 뒤
  - 그 경로들 중에서 가장 짧은 경로를 찾는 방법
  - 효율성 분석 (최악의 경우 = 모든 정점간의 간선이 존재할 때)
    - (n - 2)(n - 3) --- 1 = (n - 2)! (지수 시간 복잡도)
      - 완전 그래프를 가정하면, O(n!)로 나와서 비효율적이다.

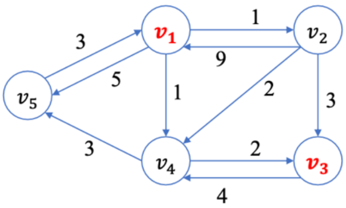
- 최단 경로 문제는 최적화 문제(optimization problem)
  - 최적화 문제는 하나 이상의 해답 후보가 있을 수 있고,
  - 해답 후보 중에서 가장 최적의 값(optimal value)을 가진 해답을 가진 문제

최단 경로 문제의 입력 사례

- 그래프의 표현: 인접 행렬(adjacency matrix)

W	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

D	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0



- Shortest path from v_1 to v_3?
  - $length[v_1, v_2, v_3] = 1 + 3 = 4$
  - $length[v_1, v_4, v_3] = 1 + 2 = 3$
  - $length[v_1, v_2, v_4, v_3] = 1 + 2 + 2 = 5$

- v1에서 v3로 가는 가능한 단순 경로의 수는? 3

- v1에서 v3로 가는 최단 경로와 경로 길이는? [v1, v4, v3], length = 3.

최단 경로: 동적계획(Dynamic Programming)

- 1단계: 재귀 관계식을 찾는다.
  - D: 각 정점의 쌍이 가지는 최단 경로의 길이를 나타내는 행렬
  - D[i][j]: vi에서 vj로 가는 최단 경로의 길이
  - 목표: 인접 행렬 W에서 최단 경로의 행렬 D와의 재귀 관계식 구하기

- 2단계: 상향식 방법으로 해답을 구한다.
  - 초기화: $D^0 = W$
  - 최종 목표: $D^n = D$.
  - 상향식 계산: D^0, D^1, ---, D^n

최단 경로 행렬의 이해

- Dk: k개의 중간 정점을 지나는 최단 경로 길이의 행렬
- $D^k[i][j]$: v_i에서 v_j로 k개의 중간 정점을 지나는 최단 경로의 길이
- D^0: 다른 어떤 정점도 지나지 않는 최단 경로의 길이 (= W)
- D^n: 다른 모든 정점을 지날 수 있는 최단 경로의 길이(= D)

W	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

D	1	2	3	4	5
1	0	1	**3**	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

재귀 관계식 구하기

- $D^0 = W$, $D^k = D^{k-1}$로부터 구함 ($1 <= k <= n$)
- $D^{k-1}[i][j]$: v_i에서 v_j로 k - 1개의 중간 정점을 지남
- $D^k[i][j]$: 다음과 같은 두 가지의 경우를 고려
  - 경우 1: 하나의 정점을 더 지나게 해 줘도 새로운 최단 경로가 없는 경우
    - $D^k[i][j] = D^{k-1}[i][j]$:
  - 경우 2: 하나의 정점(v_k)을 더 지나면 새로운 최단 경로가 있는 경우
    - $D^k[i][j] = D^{k-1}[i][k] + D^{k-1}[k][j]$

최단 경로의 재귀 관계식

- $D^0 = W$

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

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        if (i == j) D[i][j] = 0;
        else if (W[i][j] == 0) D[i][j] = INF;
        else D[i][j] = W[i][j];

        P[i][j] = 0;
    }
}
```

```
for (int k = 1; k <= n; k++) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (D[i][k] != INF && D[k][j] != INF && D[i][j] > D[i][k] + D[k][j]) {
                D[i][j] = D[i][k] + D[k][j];
                P[i][j] = k;
            }
        }
    }
}
```

최단 경로 구하기

- 최단 경로를 구하기 위해서는 그 과정을 기록해야 함.

- P[i][j]: vi에서 vj로 가는 최단 경로가 거쳐야 하는 새로운 정점

    - vi에서 vj로 가는 최단 경로의 중간에 놓여있는 정점이 최소한 하나가 있는 경우에는 그 놓여있는 정점 중에서 가장 큰 인덱스

- 최단 경로의 중간에 놓여있는 정점이 없는 경우에는 -1

```cpp
// Floyd Algorithm
/* input case
5 10
1 2 1
1 4 1
1 5 5
2 1 9
2 3 3
2 4 2
3 4 4
4 3 2
4 5 3
5 1 3
25
1 1
1 2
1 3
1 4
1 5
2 1
2 2
2 3
2 4
2 5
3 1
3 2
3 3
3 4
3 5
4 1
4 2
4 3
4 4
4 5
5 1
5 2
5 3
5 4
5 5
*/

#include <iostream>
#include <vector>
using namespace std;

typedef vector<vector<int> > Matrix;

const int INF = 999;

void floyd2(int n, Matrix& W, Matrix& D, Matrix& P) {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (i == j) D[i][j] = 0;
            else if (W[i][j] == 0) D[i][j] = INF;
            else D[i][j] = W[i][j];

            P[i][j] = 0;
        }
    }

    for (int k = 1; k <= n; k++) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (D[i][k] != INF && D[k][j] != INF && D[i][j] > D[i][k] + D[k][j]) {
                    D[i][j] = D[i][k] + D[k][j];
                    P[i][j] = k;
                }
            }
        }
    }
}

void path(Matrix& P, int u, int v, vector<int>& p) {
    int k = P[u][v];
    if (k != 0) {
        path(P, u, k, p);
        p.push_back(k);
        path(P, k, v, p);
    }
}
```

```cpp
int main () {
    int N, M, u, v, w;
    cin >> N >> M;

    Matrix W(N+1, vector<int>(N+1, 0));

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            if (i != j) W[i][j] = INF;
        }
    }

    for (int i = 0; i < M; i++) {
        cin >> u >> v >> w;
        W[u][v] = w;
    }

    Matrix D(N+1, vector<int>(N+1, INF));
    Matrix P(N+1, vector<int>(N+1, 0));

    floyd2(N, W, D, P);

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            if (D[i][j] == INF) cout << INF;
            else cout << D[i][j];
            if (j != N) cout << " ";
        }
        cout << endl;
    }

    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= N; j++) {
            if (P[i][j] == INF) cout << INF;
            else cout << P[i][j];
            if (j != N) cout << " ";
        }
        cout << endl;
    }

    int K, start, end;
    cin >> K;

    for (int i = 0; i < K; i++) {
        cin >> start >> end;
        vector<int> p;
        path(P, start, end, p);

        if (D[start][end] == INF) {
            cout << "NONE" << endl;
            continue;
        }

        cout << start << " ";
        for (int node : p) {
            cout << node << " ";
        }
        cout << end << endl;

    }

    return 0;
}
```