# 4

# 최단 경로 알고리즘

## 4.1 최단경로 BOJ #1753

$V \leq 20000$, $E \leq 300000$이고 음수 간선이 없는 최단 경로 문제이므로 이 상황에서는 데이크스트라 알고리즘을 쓰는 게 최선입니다.

### 정답 코드

실행 시간 112ms, 메모리 9,152KB

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

using namespace std;
using pii = pair<int, int>;

int inf = 987654;
int dist[20001];
vector<pii> graph[20001]; // destination, cost

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

    fill(dist, dist + 20001, inf);

    int n, m, k;
    cin >> n >> m >> k;

    dist[k] = 0;

    while (m--) {
        int u, v, w;
        cin >> u >> v >> w;
```

```cpp
        graph[u].emplace_back(pii(v, w));
    }

    priority_queue<pii, vector<pii>, greater<>> pq;
    // minimum heap; cost, destination
    pq.emplace(pii(0, k));

    while (pq.size()) {
        int d = pq.top().first, u = pq.top().second;
        pq.pop();
        if (dist[u] < d) continue;
        for (pii v : graph[u]) {
            if (dist[u] + v.second >= dist[v.first]) continue;
            dist[v.first] = dist[u] + v.second;
            pq.emplace(pii(dist[v.first], v.first));
        }
    }

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

    return 0;
}
```

## 4.2 최소비용 구하기 BOJ #1916

도시 $N \leq 1000$개, 버스 $M \leq 100000$개이고 비용이 음수인 버스가 없는 최단 경로 문제이므로 역시 이 상황에서는 데이크스트라 알고리즘을 쓰는 게 최선입니다. 시작 지점과 도착 지점이 지정되어 있음에 유의합시다.

### 정답 코드

실행 시간 24ms, 메모리 3,208KB

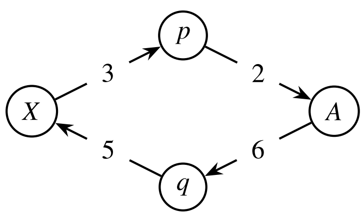
```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;
using pii = pair<int, int>;

int inf = 987654321;
int dist[1001];
vector<pii> graph[1001]; // destination, cost

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

```cpp
    cout.tie(nullptr);

    fill(dist, dist + 1001, inf);

    int n, m;
    cin >> n >> m;

    while (m--) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].emplace_back(pii(v, w));
    }

    int start, end;
    cin >> start >> end;
    dist[start] = 0;

    priority_queue<pii, vector<pii>, greater<>> pq;
    // minimum heap; cost, destination
    pq.emplace(pii(0, start));

    while (pq.size()) {
        int d = pq.top().first, u = pq.top().second;
        pq.pop();
        if (dist[u] < d) continue;
        for (pii v : graph[u]) {
            if (dist[u] + v.second >= dist[v.first]) continue;
            dist[v.first] = dist[u] + v.second;
            pq.emplace(pii(dist[v.first], v.first));
        }
    }

    cout << dist[end] << '\n';

    return 0;
}
```

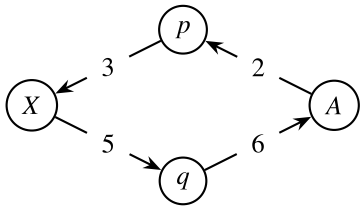
## 4.3 파티 BOJ #1238

학생 $i$에 대해 $i \to X$과 $X \to i$가 최대가 되는 학생의 최대 소요 시간을 출력하는 문제입니다.

$X \to i$는 $X$를 시작점으로 하는 한 번의 데이크스트라로 전부 구할 수 있습니다. 하지만 $i \to X$는 데이크스트라를 최대 1000번 돌려야 하는데, 이렇게 하면 실행 시간이 초과될 수 있습니다.



위 그래프에서 $A \to X$로 가는 최단 경로가 $A \to q \to X$라고 가정합시다. 이제 위 그래프에서 간선들의 방향만 전부 뒤집은 그래프를 하나 만듭니다.




기존 그래프에서 $A \to X$로 가는 최단 경로가 $A \to q \to X$이었다면, 새 그래프에서 $X \to A$로 가는 최단 경로는 $X \to q \to A$가 됩니다.

간선들의 방향만 바꾸어 주었으므로 새 그래프에서 $X \to q \to A$의 총 거리는 $A \to q \to X$의 총 거리와 같습니다. 따라서 새 그래프에서 데이크스트라를 한 번만 돌려 주면 기존 그래프에서 $i \to X$로 가는 최단 거리들이 전부 구해지게 됩니다.

결국 데이크스트라 두 번으로 해결 가능합니다. $N \le 1000$이라서 플로이드–와샬 알고리즘으로도 최적화를 잘 하면 아슬아슬하게 통과할 수 있습니다.

### 정답 코드

실행 시간 0ms, 메모리 2,256KB

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

using namespace std;
using pii = pair<int, int>;

int n, m, x;
int inf = 987654321;

void dijkstra(vector<vector<pii>> &graph, vector<int> &dist) {
    priority_queue<pii> pq;
    pq.emplace(pii(0, x)); // -distance, node
    dist[x] = 0;

    while (pq.size()) {
        int u = pq.top().second, d = -pq.top().first;
        pq.pop();
        if (dist[u] < d) continue;
        for (auto vc : graph[u]) {
            int v = vc.first, c = vc.second;
            if (dist[v] <= dist[u] + c) continue;
            dist[v] = dist[u] + c;
            pq.emplace(pii(-dist[v], v));
        }
    }
}

```

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

    cin >> n >> m >> x;

    vector<vector<pii>> from_x(n + 1), to_x(n + 1);

    while (m--) {
        int u, v, c;
        cin >> u >> v >> c;
        from_x[u].emplace_back(pii(v, c));
        to_x[v].emplace_back(pii(u, c));
    }

    vector<int> dist_from(n + 1, inf), dist_to(n + 1, inf);

    dijkstra(from_x, dist_from);
    dijkstra(to_x, dist_to);

    int mx = 0;
    for (int i = 1; i <= n; i++) {
        if (i == x) continue;
        mx = max(mx, dist_from[i] + dist_to[i]);
    }

    cout << mx;

    return 0;
}
```

## 4.4 플로이드 BOJ #11404

문제 이름에서 보이듯이, $n \leq 100$으로 플로이드–와샬로 풀 수 있는 문제입니다. $u \to v$의 간선이 여러 개 들어온다면 그 중 최솟값만 하나 저장하는 데에 유의합시다.

### 정답 코드

실행 시간 68ms, 메모리 2,028KB

```cpp
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

int dp[101][101];
int inf = 98765432;

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

```

```cpp
    for (int i = 1; i <= n; i++) {
        fill(dp[i], dp[i] + 101, inf);
        dp[i][i] = 0;
    }

    while (m--) {
        int u, v, c;
        cin >> u >> v >> c;

        dp[u][v] = min(dp[u][v], c);
    }

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

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (dp[i][j] >= inf)
                dp[i][j] = 0;
            cout << dp[i][j] << ' ';
        }
        cout << '\n';
    }

    return 0;
}
```

## 4.5 특정한 최단 경로 BOJ #1504

1번 정점에서 $N$번 정점까지 가야 하는데, 임의로 주어진 두 정점 $X, Y$를 통과해야 한다고 합니다. 간단하게 $1 \to X \to Y \to N$의 최단 거리와 $1 \to Y \to X \to N$의 최단 거리 중 작은 것을 선택하면 됩니다.

1번 정점, $X$번 정점, $Y$번 정점에서 시작하는 데이크스트라를 각각 한 번씩 돌리고 마지막에서 위에서 언급한 주 경로 중 짧은 것을 구해 출력하면 됩니다.

데이크스트라를 여러 번 돌릴 필요가 있는 문제라면 함수를 만들어 두는 편이 좋습니다.

### 정답 코드

실행 시간 48ms, 메모리 9,376KB

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

```cpp

using namespace std;
using ll = long long;
using pll = pair<ll, ll>;

void dijkstra(vector<vector<pll>>& g, vector<ll>& dist, int x) {
    priority_queue<pll> pq; // -dist, idx
    dist[x] = 0;
    pq.emplace(pll(0, x));
    while (pq.size()) {
        ll d = -pq.top().first, u = pq.top().second;
        pq.pop();
        if (d > dist[u]) continue;

        for (pll vc : g[u]) {
            ll v = vc.first, c = vc.second;
            if (dist[v] > d + c) {
                dist[v] = d + c;
                pq.emplace(pll(-dist[v], v));
            }
        }
    }
}

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

    ll inf = 987654321987;

    int n, m;
    cin >> n >> m;

    vector<vector<pll>> g(n + 1);

    while (m--) {
        int u, v, c;
        cin >> u >> v >> c;
        g[u].emplace_back(pll(v, c));
        g[v].emplace_back(pll(u, c));
    }

    int x, y;
    cin >> x >> y;

    vector<ll> dist_1(n + 1, inf), dist_x(n + 1, inf), dist_y(n + 1, inf);
    dijkstra(g, dist_1, 1);
    dijkstra(g, dist_x, x);
    dijkstra(g, dist_y, y);

    ll mn = inf;
    if (dist_1[x] != inf && dist_x[y] != inf && dist_y[n] != inf) {
        mn = min(mn, dist_1[x] + dist_x[y] + dist_y[n]);
    }
    if (dist_1[y] != inf && dist_y[x] != inf && dist_x[n] != inf) {
        mn = min(mn, dist_1[y] + dist_y[x] + dist_x[n]);
    }

    if (mn != inf) {
        cout << mn;
    } else {
        cout << -1;
```

```
    }

    return 0;
}
```

## 4.6 웜홀 BOJ #1865

시간을 역행하는 웜홀이 있습니다. 이는 음의 가중치를 가진 간선으로 생각합니다. 음의 간선이 있는 경우는 벨만–포드 알고리즘을 활용해 풀 수 있습니다.

### 정답 코드

실행 시간 20ms, 메모리 2,120KB

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

using namespace std;
using pii  = pair<int, int>;

int inf = 987654321;

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

    int t;
    cin >> t;

    while (t--) {
        int n, m, w;
        cin >> n >> m >> w;

        vector<vector<pii>> gr(n + 1); // dest, cost
        vector<int> dist(n + 1, inf);
        dist[1] = 0;

        while (m--) {
            int u, v, c;
            cin >> u >> v >> c;
            gr[u].emplace_back(pii(v, c));
            gr[v].emplace_back(pii(u, c));
        }

        while (w--) {
            int u, v, c;
            cin >> u >> v >> c;
            gr[u].emplace_back(pii(v, -c));
        }

        bool flag = false;
        for (int i = 0; i < n; i++) {
```

```
            for (int u = 1; u <= n; u++) {
                if (dist[u] == inf) continue;

                for (pii vc : gr[u]) {
                    int v = vc.first, c = vc.second;
                    if (dist[v] <= dist[u] + c) continue;
                    dist[v] = dist[u] + c;

                    if (i != n - 1) continue;
                    flag = true;
                    break;
                }
            }
        }

        if (flag) {
            cout << "YES\n";
        } else {
            cout << "NO\n";
        }
    }

    return 0;
}
```

## 4.7 최소비용 구하기 2 BOJ #11779

데이크스트라로 푸는 거까진 좋은데, 경로까지 출력해야 하는 문제입니다.

경로를 출력해야 하는 경우 trace 등의 배열을 만들어서 이 정점을 갱신해 준 정점 정보를 저장해 둡니다.

trace 배열을 이용해 마지막 정점에서 첫 정점으로 되돌아오면서 정점들을 스택에 저장하고, 하나씩 출력하면 됩니다. 스택은 출력되는 정점들의 순서를 뒤집어주는 역할을 합니다.

### 정답 코드

실행 시간 28ms, 메모리 3,328KB

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <queue>

using namespace std;
using pii = pair<int, int>;

int inf = 98765;
int tems[101], dp[101][101];

int main() {
```

```cpp
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n, m, r;
    cin >> n >> m >> r;

    for (int i = 1; i <= n; i++) {
        cin >> tems[i];
        for (int j = 1; j <= n; j++) {
            dp[i][j] = inf;
        }
        dp[i][i] = 0;
    }

    while (r--) {
        int a, b, l;
        cin >> a >> b >> l;

        dp[a][b] = min(dp[a][b], l);
        dp[b][a] = min(dp[b][a], l);
    }

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

    int ans = 0;

    for (int i = 1; i <= n; i++) {
        int s = 0;
        for (int j = 1; j <= n; j++) {
            if (dp[i][j] <= m) s += tems[j];
        }
        ans = max(ans, s);
    }

    cout << ans;

    return 0;
}
```

## 4.8 서강그라운드 BOJ #14938

지역 $u$에서 시작한다면, 다른 지역 $v$로 가는 모든 최단 거리를 계산하고 이 거리가 $m$보다 작거나 같다면 아이템 수를 누적해 주면 됩니다.

플로이드–와샬을 사용하는 것이 이상적이지만, 데이터 크기가 작아 세 개의 알고리즘 중 어느 알고리즘이든 사용해도 괜찮은 문제입니다.

### 정답 코드

실행 시간 0ms, 메모리 2,028KB

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

using namespace std;
using pii = pair<int, int>;

int inf = 98765;
int tems[101], dp[101][101];

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

    int n, m, r;
    cin >> n >> m >> r;

    for (int i = 1; i <= n; i++) {
        cin >> tems[i];
        for (int j = 1; j <= n; j++) {
            dp[i][j] = inf;
        }
        dp[i][i] = 0;
    }

    while (r--) {
        int a, b, l;
        cin >> a >> b >> l;

        dp[a][b] = min(dp[a][b], l);
        dp[b][a] = min(dp[b][a], l);
    }

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

    int ans = 0;

    for (int i = 1; i <= n; i++) {
        int s = 0;
        for (int j = 1; j <= n; j++) {
            if (dp[i][j] <= m) s += tems[j];
        }
        ans = max(ans, s);
    }

    cout << ans;

    return 0;
}
```

## 4.9 맥주 마시면서 걸어가기 BOJ #9205

편의점들 간의 거리가 주어지는 대신 편의점들 사이의 좌표가 주어집니다. 편의점의 수가 100곳 이하이므로 이들의 거리를 미리 전부 계산해 둘 수 있습니다.

문제를 잘 읽어보면, 결국엔 $20 \times 50\text{m} = 1000\text{m}$보다 먼 거리는 갈 수 없음을 알 수 있습니다. 따라서 간선 $u \to v$가 1000m가 넘지 않는 경우 1, 아닌 경우 0인 인접 행렬을 만들고 다음과 같은 플로이드–와샬 알고리즘을 수행할 수 있습니다.

$$D_{uv} = D_{uk} \text{ and } D_{kv} \qquad \forall k$$

$u \to k$로 가는 경로가 있고, $k \to v$로 가는 경로가 하나라도 있으면 $u \to v$로 가는 경로가 있다는 뜻입니다.

한편 이 문제는 꼭 최단 경로 문제로 접근하지 않아도, 1000m가 넘지 않는 간선들만 가중치 없이 남긴 그래프에서 단순히 BFS/DFS를 하는 것만으로도 해결할 수 있습니다. 저는 플로이드–와샬로 해결했습니다.

### 정답 코드

실행 시간 8ms, 메모리 2,000KB

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;
using pii = pair<int, int>;

bool dist[102][102];
pii pos[102];

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

    int t;
    cin >> t;

    while (t--) {
        int n;
        cin >> n;

        memset(dist, 0, sizeof(dist));

        for (int i = 0; i < n + 2; i++) {
            dist[i][i] = true;
        }
        for (int i = 0; i < n + 2; i++) {
```

```cpp
            cin >> pos[i].first >> pos[i].second;
        }

        for (int i = 0; i < n + 2; i++) {
            for (int j = 0; j < n + 2; j++) {
                int d = abs(pos[i].first - pos[j].first)
                    + abs(pos[i].second - pos[j].second);
                if (d <= 1000) dist[i][j] = true;
            }
        }

        for (int k = 0; k < n + 2; k++) {
            for (int u = 0; u < n + 2; u++) {
                for (int v = 0; v < n + 2; v++) {
                    if (!dist[u][k] || !dist[k][v]) continue;
                    dist[u][v] = 1;
                }
            }
        }

        if (dist[0][n + 1]) {
            cout << "happy\n";
        } else {
            cout << "sad\n";
        }
    }

    return 0;
}
```

## 4.10 거의 최단 경로 BOJ #5719

최단 경로에 포함되지 않은 도로로 이루어진 경로 중 가장 짧은 경로를 찾아야 합니다.

'최소비용 구하기 2'에서 했던 것처럼, 데이크스트라를 한 번 돌리고 최단 경로를 이루는 간선들을 전부 마킹합니다. 이후 데이크스트라를 한 번 더 돌리는데, 이 번에는 최단 경로를 이루는 간선들을 무시하고 탐색합니다. 줄 90–91을 참고합시다.

### 정답 코드

실행 시간 44ms, 메모리 2,492KB

```cpp
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 #include <queue>
5 #include <cstring>
6 
7 using namespace std;
8 using pii = pair<int, int>;
```

```cpp

int inf = 987654321;
int dist1[501], dist2[501];
bool visit1[501], visit2[501], shortest[501][501];

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

    while (true) {
        int n, m;
        cin >> n >> m;

        if (!n && !m) break;

        vector<vector<pii>> g(n); // cost, dest
        vector<vector<int>> prev(n);

        fill(dist1, dist1 + 501, inf);
        fill(dist2, dist2 + 501, inf);
        memset(visit1, 0, sizeof(visit1));
        memset(visit2, 0, sizeof(visit2));
        memset(shortest, 0, sizeof(shortest));

        int s, d;
        cin >> s >> d;

        dist1[s] = dist2[s] = 0;

        while (m--) {
            int u, v, p;
            cin >> u >> v >> p;
            g[u].push_back(pii(p, v));
        }

        // 1st dijkstra
        priority_queue<pii> pq; // cost, dest
        pq.emplace(pii(0, s));
        while (!pq.empty()) {
            int x = -pq.top().first, u = pq.top().second;
            pq.pop();

            for (pii pv : g[u]) {
                int p = pv.first, v = pv.second;
                int y = x + p;

                if (dist1[v] > y) {
                    dist1[v] = y;
                    prev[v].clear();
                    prev[v].push_back(u);
                    pq.emplace(pii(-y, v));
                } else if (dist1[v] == y) {
                    prev[v].push_back(u);
                }
            }
        }

        // reverse bfs
        queue<pii> q; // prev, curr
        q.emplace(pii(-1, d));

        while (!q.empty()) {
```

```cpp
            auto[v, u] = q.front();
            q.pop();

            if (v != -1) shortest[u][v] = true;

            for (int w : prev[u]) {
                q.emplace(pii(u, w));
            }
        }

        // 2nd dijkstra
        pq.emplace(pii(0, s));
        while (!pq.empty()) {
            int x = -pq.top().first, u = pq.top().second;
            pq.pop();

            for (pii pv : g[u]) {
                int p = pv.first, v = pv.second;
                // discard 1st shortest path
                if (shortest[u][v]) continue;
                int y = x + p;

                if (dist2[v] > y) {
                    dist2[v] = y;
                    pq.emplace(pii(-y, v));
                }
            }
        }

        if (dist2[d] == inf) {
            cout << "-1\n";
        } else {
            cout << dist2[d] << '\n';
        }
    }

    return 0;
}
```