20221208 数据结构与算法 解题报告

All pairs shortest path

使用 Floyd 或 Dijkstra 进行最短路求解即可。

All-Pairs-Shortest-Path_Floyd.cpp

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

#define INF 1E9

int main(int argc, char const *argv[])
{
    freopen("init.in", "r", stdin);
    int n, m, q;
    cin >> n >> m >> q;
    vector<vector<int>> dis(n + 1, vector<int>(n + 1, INF));
    for (; m--;)
    {
        int u, v, w;
        cin >> u >> v >> w;
        dis[u][v] = min(dis[u][v], w);
    }

    for (int i = 1; i <= n; i++)
        dis[i][i] = 0;

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

    for (; q--;)
    {
        int u, v;
        cin >> u >> v;
        cout << (dis[u][v] == INF ? -1 : dis[u][v]) << endl;
    }

    return 0;
}
```

All-Pairs-Shortest-Path_Dijkstra.cpp

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

#define INF 1E9

struct Edge
{
    int vec, cost;
    Edge *next;
    Edge(int _vec = 0, int _cost = INF, Edge *_next = nullptr) : vec(_vec), cost(_cost),
next(_next) {}
    ~Edge()
    {
        if (next != nullptr)
            delete next;
    }
};

void dijkstra(int start, const vector<Edge *> &finalEdge, vector<int> &dis)
{
    int n = dis.size() - 1;
    priority_queue<pair<int, int>> hep;
    vector<bool> sgn(n + 1, false);
    hep.push(pair<int, int>(0, start));
    for (; !hep.empty();)
    {
        int cur;
        for (; sgn[hep.top().second]; hep.pop())
            ;
        dis[cur = hep.top().second] = hep.top().first;
        sgn[cur] = true;
        hep.pop();
        for (Edge *edge = finalEdge[cur]; edge != nullptr; edge = edge->next)
            if (!sgn[edge->vec])
                hep.push(pair<int, int>(dis[cur] + edge->cost, edge->vec));
    }

    for (int i = 1; i <= n; i++)
        if (dis[i] == INF)
            dis[i] = -1;
}

int main(int argc, char const *argv[])
{
    freopen("init.in", "r", stdin);
    int n, m, q;
    cin >> n >> m >> q;
    vector<Edge *> finalEdge(n + 1, nullptr);
    vector<vector<int>> dis(n + 1, vector<int>(n + 1, INF));
```

```
    for (; m--;)
    {
        int u, v, w;
        cin >> u >> v >> w;
        finalEdge[u] = new Edge(v, w, finalEdge[u]);
    }

    for (int i = 1; i ⩽ n; i++)
        dijkstra(i, finalEdge, dis[i]);

    for (; q--;)
    {
        int u, v;
        cin >> u >> v;
        cout << dis[u][v] << endl;
    }

    for (auto &edge : finalEdge)
        delete edge;

    return 0;
}
```

All-Pairs-Shortest-Path_SPFA.cpp

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

#define INF 1E9

struct Edge
{
    int vec, cost;
    Edge *next;
    Edge(int _vec = 0, int _cost = INF, Edge *_next = nullptr) : vec(_vec), cost(_cost),
next(_next) {}
    ~Edge()
    {
        if (next ≠ nullptr)
            delete next;
    }
};

void SPFA(int start, const vector<Edge *> &finalEdge, vector<int> &dis)
{
    int n = dis.size() - 1;
    queue<int> que;
```

```cpp
    vector<bool> sgn(n + 1, false);
    que.push(start), sgn[start] = true, dis[start] = 0;
    for (; !que.empty();)
    {
        int cur = que.front();
        for (Edge *edge = finalEdge[cur]; edge ≠ nullptr; edge = edge→next)
            if (dis[cur] + edge→cost < dis[edge→vec])
            {
                dis[edge→vec] = dis[cur] + edge→cost;
                if (!sgn[edge→vec])
                    que.push(edge→vec), sgn[edge→vec] = true;
            }
        que.pop(), sgn[cur] = false;
    }

    for (int i = 1; i ⩽ n; i++)
        if (dis[i] == INF)
            dis[i] = -1;
}

int main(int argc, char const *argv[])
{
    freopen("init.in", "r", stdin);
    int n, m, q;
    cin >> n >> m >> q;
    vector<Edge *> finalEdge(n + 1, nullptr);
    vector<vector<int>> dis(n + 1, vector<int>(n + 1, INF));
    for (; m--;)
    {
        int u, v, w;
        cin >> u >> v >> w;
        finalEdge[u] = new Edge(v, w, finalEdge[u]);
    }

    for (int i = 1; i ⩽ n; i++)
        SPFA(i, finalEdge, dis[i]);

    for (; q--;)
    {
        int u, v;
        cin >> u >> v;
        cout << dis[u][v] << endl;
    }

    for (auto &edge : finalEdge)
        delete edge;

    return 0;
}
```

Robot

将目的格子的 Terrain 值视作每一步的花费，进行最短路求解即可。

Robot.cpp

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

#define INF 1E9

pair<int, int> direction[4] = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};

pair<int, int> operator+(const pair<int, int> &a, const pair<int, int> &b) { return
pair<int, int>(a.first + b.first, a.second + b.second); }

int SPFA(const vector<vector<int>> &ter, int sx, int sy, int ex, int ey)
{
    int r = ter.size(), c = ter[0].size();
    vector<vector<int>> dis(r, vector<int>(c, INF));
    queue<pair<int, int>> que;
    vector<vector<bool>> sgn(r, vector<bool>(c, false));
    dis[sx][sy] = ter[sx][sy], que.push(pair<int, int>(sx, sy)), sgn[sx][sy] = true;

    for (; !que.empty();)
    {
        auto cur = que.front();
        for (int i = 0; i < 4; i++)
        {
            auto vec = cur + direction[i];
            if (vec.first ⩾ 0 && vec.first < r && vec.second ⩾ 0 && vec.second < c)
                if (dis[cur.first][cur.second] + ter[vec.first][vec.second] <
dis[vec.first][vec.second])
                {
                    dis[vec.first][vec.second] = dis[cur.first][cur.second] +
ter[vec.first][vec.second];
                    if (!sgn[vec.first][vec.second])
                        que.push(vec), sgn[vec.first][vec.second] = true;
                }
        }
        sgn[cur.first][cur.second] = false, que.pop();
    }

    return dis[ex][ey];
}
```

```cpp
int main(int argc, char const *argv[])
{
    freopen("init.in", "r", stdin);
    int T;
    cin >> T;
    for (; T--;)
    {
        int r, c;
        cin >> r >> c;
        vector<vector<int>> ter(r, vector<int>(c, 0));
        for (int i = 0; i < r; i++)
            for (int j = 0; j < c; j++)
                cin >> ter[i][j];

        int sx, sy, ex, ey;
        cin >> sx >> sy >> ex >> ey;
        cout << SPFA(ter, --sx, --sy, --ex, --ey) << endl;
    }
    return 0;
}
```