Standard Source code Library

mayf3

April 4, 2014

Contents

1 图论

1.1 最短路算法

1.1.1 Dijkstra最短路

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>

/*
 * name     :     dijkstra(STL)
 * usage     :    single-source shortest path(only non-negative weight)
 * develop    :    small label first optimization, negative circle
 * space complexity    :    O(M)
 * time complexity    :    O(NlogN)
 * checked    :    no
 */

const int N = 111111; //number of the vertices

int n, m;
int dist[N];
vector<PII> E[N];

int calc(int s, int e) {
    priority_queue<PII, vector<PII>, greater<PII> > Q;
    rep(i, n) dist[i] = -1;
    dist[s] = 0;
    Q.push(MP(0, s));
    int x, y, cost;
    while (!Q.empty()) {
        x = Q.top().Y, cost = Q.top().X;
        Q.pop();
        if (cost > dist[x]) continue;
        rep(i, E[x].size()){
            y = E[x][i].X, cost = E[x][i].Y;
            if (dist[y] == -1 || dist[y] != -1 && dist[x] + cost > dist[y]){
                dist[y] = dist[x] + cost;
                Q.push( make_pair(dist[y], y) );
            }
        }
    }
    return dist[e];
}

int main(){
    while(~scanf("%d%d", &n, &m)){
        rep(i, n) E[i].clear();
        int x, y, c;
        rep(i, m){
            scanf("%d%d%d", &x, &y, &c);
            x--, y--;
            E[x].PB(MP(y, c));
            E[y].PB(MP(x, c));
        }
        printf("%d\n", calc(0, n - 1));
    }
    return 0;
}
```

1.1.2 spfa 最短路

```
#include "template.cpp"

/*
 * name     :     spfa
 * usage     :    single-source shortest path, differential restraint system
 * develop    :    small label first optimization, negative circle
 * space complexity    :    O(M)
 * time complexity    :    O(k * M) (where k is usually less than 2)
 * checked    :    no
 */

const int N = 10000;

int n, m;
vector<PII> E[N];
int dist[N];

int spfa(int s, int e){
    static deque<int> Q;
    static bool inque[N];
    Cls(inque);
    memset(dist, -1, sizeof dist);
    dist[s] = 0;
    Q.PB(s);
    inque[s] = true;
    int x, y, c;
    while(!Q.empty()){
        x = Q.front();
        Q.pop_front();
        inque[x] = false;
        rep(i, E[x].size()){
            y = E[x][i].X;
            c = E[x][i].Y;
            if (dist[y] == -1 || dist[y] != -1 && dist[y] > dist[x] + c){
                dist[y] = dist[x] + c;
                if (!inque[y]){
                    Q.PB(y);
                    inque[y] = true;
                }
            }
        }
    }
    return dist[e];
}

int main(){
    while(~scanf("%d%d", &n, &m)){
        rep(i, n) E[i].clear();
        int x, y, c;
        rep(i, m){
            scanf("%d%d%d", &x, &y, &c);
            x--, y--;
            E[x].PB(MP(y, c));
            E[y].PB(MP(x, c));
        }
        printf("%d\n", spfa(0, n - 1));
    }
    return 0;
}
```

1.1.3 k最短路（无环）

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <map>

using namespace std;

const int MAXN = 50 + 10; //number of vertices
const int MAXK = 200 + 10;
const int INF = 1000000000; //max dist

struct Tpath {
    int cnt, len, pos;
    int v[MAXN];
};

Tpath path[MAXK];
int g[MAXN][MAXN];
int len[MAXK], pos[MAXK], ans[MAXK];
bool used[MAXN];
int dist[MAXN], prev[MAXN], List[MAXN];
int N, M, K, S, T, cnt;

void Dijkstra() {
    int visited[MAXN];
    for (int i = 0; i <= N; ++i) dist[i] = INF, visited[i] = 0;
    dist[T] = 0;
    for (int k, i = T; i != N; i = k) {
        visited[i] = 1; k = N;
        for (int j = 0; j < N; ++j) {
            if (visited[j] || used[j]) continue;
            if (g[j][i] > -1 && dist[i] + g[j][i] < dist[j]) {
                dist[j] = dist[i] + g[j][i];
                prev[j] = i;
            }
            if (dist[j] < dist[k]) k = j;
        }
    }
}

void setPath(int v, Tpath &p) {
    p.len = 0;
    while (1) {
        p.v[p.cnt++] = v;
        if (v == T) return;
        p.len += g[v][prev[v]]; v = prev[v];
    }
}

void solve() {
    memset(used, 0, sizeof(used));
    Dijkstra();
    memset(ans, -1, sizeof(ans));
    if (dist[S] == INF)    return;
    multimap<int, int> Q; Q.clear();
   path[0].cnt = 0; path[0].pos = 0; setPath(S, path[0]); Q.insert( make_pair(path[0].len, 0) )
    int tot = 1;
    for (int i = 0; i < K; ++i) {
        if (Q.empty()) return;
        multimap<int, int> :: iterator p = Q.begin();
        int x = (*p).second;
        ans[i] = path[x].len;
        if (i == K - 1) break;
        memset(used, 0, sizeof(used));
        Tpath cur; cur.cnt = 0; cur.len = 0;
        for (int sum = 0, j = 0; j + 1 < path[x].cnt; ++j) {
            cur.v[cur.cnt++] = path[x].v[j]; used[path[x].v[j]] = 1;
            if (j) sum += g[path[x].v[j - 1]][path[x].v[j]];
            if (j >= path[x].pos) {
                Dijkstra();
                int u = path[x].v[j];
                for (int v = 0; v < N; ++v)
                if (g[u][v] > -1 && !used[v] && dist[v] < INF && v != path[x].v[j + 1]) {
                 Tpath tp = cur; tp.pos = j + 1; setPath(v, tp); tp.len += sum + g[u][v];
                    if (tot < K) path[tot] = tp, Q.insert( make_pair(tp.len, tot++) );
                        else {
                            multimap<int, int> :: iterator p = Q.end(); --p;
                            if (tp.len >= (*p).first) continue;
                   path[(*p).second] = tp; Q.insert( make_pair(tp.len, (*p).second) );
                            Q.erase(p);
                        }
                    }
            }
        }
        Q.erase(p);
    }
}

void DFS(int step, int u, int len) {
    if (!cnt) return;
    if (u == T) {
        if (len == ans[K - 1]) {
            if (!(--cnt)) {
                for (int j = 0; j < step; ++j) {
                    if (j) printf("-");
                    printf("%d", List[j] + 1);
                }
                printf("\n");
            }
        }
        return;
    }
    Dijkstra();
    int tmp[MAXN];
    for (int i = 0; i < N; ++i) tmp[i] = dist[i];
    for (int i = 0; i < N; ++i)
     if (g[u][i] > -1 && !used[i] && tmp[i] < INF && len + g[u][i] + tmp[i] <= ans[K - 1]) {
            used[i] = 1; List[step] = i;
            DFS(step + 1, i, len + g[u][i]);
            if (!cnt) return;
            used[i] = 0;
        }
}
```

```
int main() {
    scanf("%d%d%d%d%d", &N, &M, &K, &S, &T);
    --S; --T;
    if (S == T) ++K;
    memset(g, -1, sizeof(g));
    for (int i = 0; i < M; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        g[u][v] = w;
    }
    solve();
    if (ans[K - 1] == -1) printf("None\n");
    else {
        cnt = 0;
        for (int i = 0; i < K; ++i)
            if (ans[i] == ans[K - 1]) ++cnt;
        memset(used, 0, sizeof(used));
        used[S] = 1; List[0] = S;
        DFS(1, S, 0);
    }
    return 0;
}
```

1.1.4 k最短路

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <queue>

using namespace std;

const int MAXN = 1000 + 10; //number of vertices
const int MAXM = 100000 + 10; //number of edges
const int MAXK = 1000 + 10;
const int MAXH = 200000; //M + N log N
const int INF = 1000000000; //max dist

struct Theap {
    int idx, dep;
    int chd[3];
};

struct Tedge {
    int u, v, w, delta;
    bool inT;
```

```
};

Theap heap[MAXH];
Tedge edge[MAXM];
int first[MAXN], rfirst[MAXN], outdeg[MAXN], dist[MAXN], nextT[MAXN], list[MAXN], H1[MAXN], H2
int next[MAXM], rnext[MAXM];
int ans[MAXK];
int N, M, K, S, T, nlist, H, curedge;

void Dijkstra() {
  priority_queue < pair<int, int>, vector< pair<int, int> >, greater< pair<int, int> > > Q;
    for (int i = 0; i < N; ++i)    dist[i] = INF;
    dist[T] = 0; Q.push( make_pair(0, T) );
    while (!Q.empty()) {
        int u = Q.top().second, d = Q.top().first;
        Q.pop();
        if (d > dist[u]) continue;
        for (int i = rfirst[u]; i != -1; i = rnext[i]) {
            int v = edge[i].u, w = edge[i].w;
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                Q.push( make_pair(dist[v], v) );
            }
        }
    }
}

void DFS(int u) {
    list[nlist++] = u;
    for (int i = rfirst[u]; i != -1; i = rnext[i]) {
        int v = edge[i].u, w = edge[i].w;
        if (!edge[i].delta && nextT[v] == -1) {
            nextT[v] = u; edge[i].inT = 1;
            DFS(v);
        }
    }
}

int buildH1(int Size, int dep) {
    if (!Size) return 0;
    if (edge[curedge].inT) curedge = next[curedge];
    int cur = H++;
    heap[cur].idx = curedge; curedge = next[curedge];
    heap[cur].chd[2] = 0;
    if (!dep) heap[cur].chd[0] = buildH1(Size - 1, dep + 1), heap[cur].chd[1] = 0;
    else {
        int half = (Size - 1) / 2;
     heap[cur].chd[0] = buildH1(half, dep + 1); heap[cur].chd[1] = buildH1(Size - 1 - half, dep
    }
    int i = cur;
    while (1) {
        int k = i;
        for (int j = 0; j < 2; ++j)
        if (heap[i].chd[j] && edge[heap[heap[i].chd[j]].idx].delta < edge[heap[k].idx].delta
        if (k == i) break;
        swap(heap[k].idx, heap[i].idx); i = k;
    }
    return cur;
}

int buildH2(int a, int b) {
    if (!a) {
        heap[b].chd[0] = heap[b].chd[1] = 0; heap[b].dep = 1;
        return b;
    }
    int Next = heap[heap[a].chd[0]].dep >= heap[heap[a].chd[1]].dep;
    int cur = H++;
    heap[cur] = heap[a];
    if (edge[heap[b].idx].delta < edge[heap[a].idx].delta) {
        heap[b].chd[0] = heap[a].chd[0]; heap[b].chd[1] = heap[a].chd[1];
        heap[b].chd[Next] = buildH2(heap[b].chd[Next], cur);
        heap[b].dep = min(heap[heap[b].chd[0]].dep, heap[heap[b].chd[1]].dep) + 1;
        return b;
    }
    else {
        heap[cur].chd[Next] = buildH2(heap[cur].chd[Next], b);
        heap[cur].dep = min(heap[heap[cur].chd[0]].dep, heap[heap[cur].chd[1]].dep) + 1;
        return cur;
    }
}

void solve() {
    Dijkstra();
    memset(ans, -1, sizeof(ans));
    if (dist[S] == INF) return;
    for (int i = 0; i < M; ++i) edge[i].delta = edge[i].w - dist[edge[i].u] + dist[edge[i].v];
    memset(nextT, -1, sizeof(nextT));
    nextT[T] = -2; nlist = 0;
    DFS(T);
    H = 1; heap[0].dep = 0;
    memset(H1, 0, sizeof(H1));
    for (int i = 0; i < N; ++i)
        if (dist[i] < INF) {
            int Size = outdeg[i];
            if (i != T) --Size;
            curedge = first[i];
            H1[i] = buildH1(Size, 0);
            if (H1[i]) {
                heap[H1[i]].chd[2] = heap[H1[i]].chd[0];
                heap[H1[i]].chd[0] = 0;
                heap[H1[i]].dep = 1;
            }
        }
    memset(H2, 0, sizeof(H2));
    H2[T] = H1[T];
    for (int i = 1; i < nlist; ++i) {
        int j = list[i];
        if (!H1[j]) H2[j] = H2[nextT[j]];
        else H2[j] = buildH2(H2[nextT[j]], H1[j]);
    }
    ans[0] = dist[S];
   priority_queue < pair<int, int>, vector< pair<int, int> >, greater< pair<int, int> > > Q;
    if (H2[S]) Q.push( make_pair(edge[heap[H2[S]].idx].delta, H2[S]) );
    for (int i = 1; i < K; ++i) {
        if (Q.empty()) break;
        int u = Q.top().second, d = Q.top().first;
        ans[i] = dist[S] + d;
        Q.pop();
        for (int j = 0; j < 3; ++j) {
            int v = heap[u].chd[j];
        if (v) Q.push( make_pair(d + edge[heap[v].idx].delta - edge[heap[u].idx].delta, v) );
        }
        int v = H2[edge[heap[u].idx].v];
        if (v) Q.push( make_pair(d + edge[heap[v].idx].delta, v) );
    }
}

int main() {
    memset(first, -1, sizeof(first));
    memset(rfirst, -1, sizeof(rfirst));
    memset(outdeg, 0, sizeof(outdeg));
    scanf("%d%d", &N, &M);
    for (int i = 0; i < M; ++i) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        --u; --v;
        edge[i].u = u; edge[i].v = v; edge[i].w = w; edge[i].inT = 0;
        next[i] = first[u]; first[u] = i; rnext[i] = rfirst[v]; rfirst[v] = i;
        ++outdeg[u];
    }
    scanf("%d%d%d", &S, &T, &K);
    --S; --T;
    if (S == T) ++K;
    solve();
    printf("%d\n", ans[K - 1]);
    return 0;
}
```

1.2 生成树

1.3 网络流

1.4 匹配

1.5 图

2 数据结构

3 字符串算法

4 计算几何

4.1 基础

4.2 凸包

4.3 平面

4.4 多边形和圆

4.5 其他

5 理论

5.1 数学

5.2 数论

5.3 博弈论

5.4 特殊数列

6 其他