ACM模板

By:EndlessLethe

**最短路:**

#include <stdio.h>

#include <stdlib.h>

#include <iostream>

#include <string.h>

#include <vector>

#include <queue>

using namespace std;

//读入的边 序号从0开始

//q, inq, cnt, d都在函数中初始化，main里面只用初始化G和edge

struct Edge{

int from, to, dist;

};

const int MAX\_V = 10100;

const int INF = 0x7fffffff;

int inq[MAX\_V];//是否在队列中

int d[MAX\_V];//s到各点的距离

//int p[MAX\_V];//最短路中的上一条弧

int cnt[MAX\_V];//进队次数

vector <int> G[MAX\_V];

vector <Edge> edge;

int V;

void add\_edge(int from, int to, int dist) {//add\_edge(a-1, b-1, c)

edge.push\_back((Edge){from, to, dist});

int m = edge.size();

G[from].push\_back(m-1);

}

bool bellman\_ford(int s) {

queue <int> q;

memset(inq, 0, sizeof(inq));

memset(cnt, 0, sizeof(cnt));

for (int i = 0; i < V; i++) d[i] = INF;

d[s] = 0;

inq[s] = true;

q.push(s);

while (!q.empty()) {

int u = q.front();

q.pop();

inq[u] = false;

for (int i = 0; i < G[u].size(); i++) {

Edge &e = edge[G[u][i]];

if (d[u] < INF && d[e.to] > d[u] + e.dist) {//u已经访问且e.to可以更新

d[e.to] = d[u] + e.dist;

// p[e.to] = G[u][i];

if (!inq[e.to]) {

q.push(e.to);//只对没有访问的加入松弛队列//可能已经加入了队列

inq[e.to] = true;

if (++cnt[e.to] > V) return false;//判断负圈

}

}

}

}

return true;

}

int main() {

int N, M, u, v, temp;

while (scanf("%d%d", &N, &M) == 2) {

if (M == 0 && N == 0) break;

V = N;//初始化全局变量

edge.clear();

for (int i = 0; i < N; i++) G[i].clear();

for (int i = 0; i < M; i++) {

scanf("%d%d%d", &u, &v, &temp);

add\_edge(u-1, v-1, temp);

add\_edge(v-1, u-1, temp);

}

bellman\_ford(0);

if (N != 1) printf("%d\n", d[N-1]);

else printf("0\n");

}

}

// bool whether = bellman\_ford(0);

// if (!whether) printf("-1\n");//有负环

// else {

// if (d[n-1] - d[0] >= INF) printf("-2\n");//无法到达（无约束）

// else printf("%d\n", d[n-1]-d[0]);

**最大流:**

const int MAX\_V = 20010;

const int INF = 0x7fffffff;

struct edge {int to, cap, rev;};

vector <edge> G[MAX\_V];

int level[MAX\_V];//距原点的距离

int iter[MAX\_V];//当前弧

void add\_edge(int from, int to, int cap) {

G[from].push\_back((edge){to, cap, G[to].size()});

G[to].push\_back((edge){from, 0, G[from].size()-1});

}

void bfs(int s) {//建立当前残余网络的层次图

memset(level, -1, sizeof(level));//注意是-1

queue<int> q;

level[s] = 0;

q.push(s);

while (!q.empty()) {

int v = q.front();

q.pop();

for (int i = 0; i < G[v].size(); i++) {

edge &e = G[v][i];

if (e.cap > 0 && level[e.to] < 0) {

level[e.to] = level[v] +1;

q.push(e.to);

}

}

}

}

int dfs(int v, int t, int f) {

if (v == t) return f;//f为当前能传输的最大值

for (int &i = iter[v]; i < G[v].size(); i++) {

edge &e = G[v][i];

if (e.cap > 0 && level[v] < level[e.to]) {//保证离汇点更近（not only+1）

int d = dfs(e.to, t, min(f, e.cap));//min求一个瓶颈

if (d > 0) {

e.cap -= d;

G[e.to][e.rev].cap += d;

return d;

}

}

}

return 0;

}

int max\_flow(int s, int t) {

int flow = 0;

for (;;) {

bfs(s);

if (level[t] < 0) return flow;

memset(iter, 0, sizeof(iter));

int f = dfs(s, t, INF);

while (f > 0) {

flow += f;

f = dfs(s, t, INF);

}

}

}

int main()

{

int N, M, A, B, a, b, c;

scanf("%d%d", &N, &M);

int s = N, t = s + 1;//顶点编号从0开始

for (int i = 0; i < N; i++) {

scanf("%d%d", &A, &B);

add\_edge(s, i, A);

add\_edge(i, t, B);

}

for (int i = 0; i < M; i++){

scanf("%d%d%d", &a, &b, &c);

add\_edge(a-1, b-1, c);

add\_edge(b-1, a-1, c);

}

printf("%d\n", max\_flow(s, t));

return 0;

}

**线段树**

#include <stdio.h>

#define MID ((l+r)>>1)

const int MAX\_N = 820000;

int n;//注意读入不是n

int dat[2 \* MAX\_N - 1], mark[2\*MAX\_N-1];

//当dat有初始值的时候，通过递归建树（还是要先调用init清零）

int get(int k, int l, int r) {

if (l == r-1) return dat[k];

int v1 = get(2\*k+1, l, MID);

int v2 = get(2\*k+2, MID, r);

return (dat[k] = v1 + v2);

}

void init(int n\_) {

n = 1;

while (n < n\_) n \*= 2;

for (int i = 0; i < 2 \* n - 1; i++) {

dat[i] = 0;

mark[i] = 0;

}

}

void update(int k, int a) {//单点修改 把第k个更新为a

k += n-1;

dat[k] = a;

while (k > 0) {

k = (k - 1) / 2;

dat[k] = dat[2\*k+1]+dat[2\*k+2];

}

}

//步骤：1.修改儿子标记 2.下传（修改儿子数据）3.修改自身标记

void push\_down(int k, int l, int r) {

if (mark[k] == 1) {

mark[2\*k+1] = 1;

dat[2\*k+1] = MID-l;

mark[2\*k+2] = 1;

dat[2\*k+2] = r-MID;

mark[k] = 0;

}

else if (mark[k] == -1) {

mark[2\*k+1] = -1;

dat[2\*k+1] = 0;

mark[2\*k+2] = -1;

dat[2\*k+2] = 0;

mark[k] = 0;

}

}

void maintain(int k) {

dat[k] = dat[2\*k+1] + dat[2\*k+2];

}

//全空0 x=-1or全满0 x = 1

//把区间[a,b)都修改 mark可同时成为多个标记

void updata(int a, int b, int x, int k, int l, int r) {

if (r <= a || b <= l) return;

if (a <= l && r <= b) {//则打标记并修改当前值

mark[k] = x;

if (x == -1) dat[k] = 0;

else dat[k] = r-l;//区间长度为r-l，因为区间是左闭右开的

}

else {//这里肯定不是叶节点

push\_down(k, l, r);

updata(a, b, x, 2\*k+1, l, MID);//不需要 if (a <= MID)

updata(a, b, x, 2\*k+2, MID, r);

maintain(k);

}

}

int query(int a, int b, int k, int l, int r) {//查询区间 [a,b)的0的个数

if (r <= a || b <= l) return 0;//区间不相交

if (a <= l && r <= b) return dat[k];//查询区间完全包含当前区间

else {

push\_down(k, l, r);

int v1 = query(a, b, 2\*k+1, l, MID);

int v2 = query(a, b, 2\*k+2, MID, r);

maintain(k);

return v1+v2;

}

}

//查询在[a,b)区间内dat为x的位置，返回在0~n-1的下标 //dat为满足区间减法的东西 eg:sum\rank

int find(int a, int b, int x, int k, int l, int r) {

if (r == l+1) return l;

else {

push\_down(k, l, r);

int pos;

if (b < MID) pos = find(a, b, x, 2\*k+1, l, MID);

else if (a > MID) pos = find(a, b, 2\*k+2, x, MID, r);

else {

if (dat[2\*k+1] >= x) pos = find(a, b, x, 2\*k+1, l, MID);

else pos = find(a, b, x-dat[2\*k+1], 2\*k+2, MID, r);

}

maintain(k);

return pos;

}

}

// for (int i = 0; i < 2 \* n; i++) cout << dat[i] << " " << endl;