/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Dinic(当前弧优化)

O(VE)~~(?)~~

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <string.h>

#include <algorithm>

#include <queue>

using namespace std;

const int VMAX = 1e5 + 5;

const int EMAX = 1e5 + 5;

const int infi = 0x3f3f3f3f;

struct edge{

int v, w;

int next;

edge(int v\_ = 0, int w\_ = 0, int n\_ = -1):v(v\_), w(w\_), next(n\_){}

};

int n, cnt;

edge e[EMAX];

int head[VMAX];

int cur[VMAX];

int level[VMAX];

void init()

{

cnt = 0;

memset(head, -1, sizeof head);

}

void addedge(int u, int v, int w)

{

/\* 正向边 \*/

e[cnt] = edge(v, w, head[u]); head[u] = cnt++;

/\* 反向边 \*/

e[cnt] = edge(u, 0, head[u]); head[v] = cnt++;

/\*

当遍历到i边时， i ^ 1 边即为其反向边

~~常出bug~~

\*/

}

bool bfs(int s, int t) // bfs分层

{

queue<int> q;

memset(level, 0, sizeof level);

level[s] = 1;

q.push(s);

while (!q.empty())

{

int u = q.front();

q.pop();

for (int i = head[u]; i != -1; i++)

{

int v = e[i].v, w = e[i].w;

if (level[v] == 0 && w > 0)

{

level[v] = level[u] + 1;

q.push(v);

}

}

}

return level[t] != 0; // 判断能否到达汇点

}

int dfs(int flow, int u, int t) // flow为当前剩余流量， u为当前点， t为汇点

{

if (u == t)

return flow;

int change = 0;

for (int& i = cur[u]; i != -1 && flow > 0; i = e[i].next)

{

int v = e[i].v;

if (level[v] == level[u] + 1 && e[i].w > 0)

{

int temp = dfs(min(flow, e[i].w), v, t);

if (temp == 0)

continue;

e[i].w -= temp;

e[i ^ 1].w += temp;

flow -= temp;

change += temp;

}

}

return change;

}

int dinic(int s, int t)

{

int ans = 0;

while (bfs(s, t))

{

memcpy(cur, head, sizeof head);

ans += dfs(infi, s, t);

}

return ans;

}