**网络流：**

**Dinic:**

O(n^2\*m)

int level[maxn];

bool makelevel(int s,int t)

{

memset(level,0,sizeof(level));

level[s]=1;

queue<int> que;

que.push(s);

int now;

while(!que.empty())

{

now=que.front();

que.pop();

for(int i=box[now];i!=-1;i=edge[i].next)

{

if(edge[i].v==0) continue;

if(level[edge[i].to]) continue;

level[edge[i].to]=level[now]+1;

que.push(edge[i].to);

if(edge[i].to==t) return 1;

}

}

return false;

}

int makeflow(int s,int t,int maxc)

{

if(s==t) return maxc;

int rec=0,flow;

for(int i=box[s];i!=-1;i=edge[i].next)

{

if(!edge[i].v)continue;

if(level[edge[i].to]<=level[s]) continue;

flow=makeflow(edge[i].to,t,min(edge[i].v,maxc-rec));

edge[i].v-=flow;

edge[i^1].v+=flow;

rec+=flow;

if(rec==maxc) return maxc;

}

return rec;

}

int dinic(int s,int t)

{

int res=0;

while(makelevel(s,t))

res+=makeflow(s,t,inf);

return res;

}

非递归：

int que[maxn\*10];

int dinic(int s,int t)

{

int ans = 0;

while(makelevel(s,t))

{

int now, x, y, back, iter = 1;

while(iter)

{

x = (iter == 1) ? s : edge[que[iter - 1]].to;

if (x == t)

{

int minCap = inf;

for (int i = 1; i < iter; i++)

{

now = que[i];

if (edge[now].v < minCap)

{

minCap = edge[now].v;

back = i;

}

}

for (int i = 1; i < iter; i++)

{

now = que[i];

edge[now].v -= minCap;

edge[now ^ 1].v += minCap;

}

ans += minCap;

iter = back;

}

else

{

for (now = box[x]; now + 1; now = edge[now].next)

{

y = edge[now].to;

if (edge[now].v && level[y] >= level[x] + 1)

break;

}

if (now+1)

que[iter++] = now;

else

{

level[x] = -1;

iter--;

}

}

}

}

return ans;

}

**SAP:**

O(N^2\*M)

int dis[maxn],pre[maxn];

int cur[maxn],aug[maxn];

int gap[maxn];

int sap(int s, int e, int n)

{

int max\_flow = 0, v, u = s;

int id, mindis;

aug[s] = inf;

pre[s] = -1;

memset(dis, 0, sizeof(dis));

memset(gap, 0, sizeof(gap));

gap[0] = n;

for (int i = 0; i <= n; ++i){//初始化当前弧为第一条弧

cur[i] = box[i];

}

while (dis[s] < n)

{

bool flag = false;

if (u == e)

{

max\_flow += aug[e];

for (v = pre[e]; v != -1; v = pre[v])//路径回溯更新残留网络

{

id = cur[v];

edge[id].v -= aug[e];

edge[id^1].v += aug[e];

aug[v] -= aug[e]; //修改可增广量，以后会用到

if (edge[id].v == 0) u = v; //不回退到源点，仅回退到容量为0的弧的弧尾

}

}

for (id = cur[u]; id != -1; id = edge[id].next)

{ // 从当前弧开始查找允许弧

v = edge[id].to;

if (edge[id].v > 0 && dis[u] == dis[v] + 1) //找到允许弧

{

flag = true;

pre[v] = u;

cur[u] = id;

aug[v] = min(aug[u], edge[id].v);

u = v;

break;

}

}

if (flag == false)

{

if (--gap[dis[u]] == 0) break; /\*gap优化层次树出现断层则结束算法\*/

mindis = n;

cur[u] = box[u];

for (id = box[u]; id != -1; id = edge[id].next)

{

v = edge[id].to;

if (edge[id].v > 0 && dis[v] < mindis)

{

mindis = dis[v];

cur[u] = id; //修改标号的同时修改当前弧

}

}

dis[u] = mindis + 1;

gap[dis[u]]++;

if (u != s) u = pre[u]; //回溯继续寻找允许弧

}

}

return max\_flow;

}

**求最小割**

bool vis[maxn];

void dfs(int now)

{

vis[now]=1;

for(int i=box[now]; i!=-1; i=edge[i].next)

{

if(edge[i].v&&!vis[edge[i].to])

dfs(edge[i].to);

}

}

for(int i=0; i<4\*N; i++)

{ if(!edge[i].v&&(vis[edge[i].from]^vis[edge[i].to]))

{

……

}

}

**最小费用最大流：**

int in[maxn];

int dis[maxn];

int from[maxn];

queue<int> que;

bool spfa(int f,int t)

{

int now;

for(int i=0; i<2\*N+2; i++)

dis[i]=inf;

memset(in,0,sizeof(in));

memset(from,-1,sizeof(from));

que.push(f);

in[f]=1;

dis[f]=0;

while(!que.empty())

{

now=que.front();

que.pop();

for(int i=box[now]; i!=-1; i=edge[i].next)

{

if(edge[i].cap&&dis[edge[i].to]>dis[now]+edge[i].cost)

{

dis[edge[i].to]=dis[now]+edge[i].cost;

from[edge[i].to]=i;

if(!in[edge[i].to])

{

que.push(edge[i].to);

in[edge[i].to]=1;

}

}

}

in[now]=0;

}

if(dis[t]==inf) return 0;

return dis[t];

}

int fyl(int f,int t)

{

int cost=0;

int mincap;

while(spfa(f,t))

{

mincap=inf;

for(int i=from[t]; i!=-1; i=from[edge[i].from])

mincap=min(mincap,edge[i].cap);

for(int i=from[t]; i!=-1; i=from[edge[i].from])

{

edge[i].cap-=mincap;

edge[i^1].cap+=mincap;

}

cost+=dis[t]\*mincap;

}

return cost;

}

**zkw版费用流：**

bool vis[maxn];

int cost,ans;

int T=1;

int aug(int u, int f)//dinic

{

if(u == T)

{

ans += cost \* f;

return f;

}

vis[u] = 1;

int tmp = f;

for(int i = box[u]; i != -1; i = edge[i].next)

if(edge[i].cap &&!edge[i].cost && !vis[edge[i].to])//当前cost为0 才可以走

{

int delta = aug(edge[i].to, tmp < edge[i].cap ? tmp : edge[i].cap);

edge[i].cap -= delta;

edge[i^1].cap += delta;

tmp -= delta;

if(!tmp) return f;

}

return f - tmp;

}

bool modlabel()

{

int delta = inf;

for(int u = 0; u < NN; u++)//找最小可增广费用

if(vis[u])

for(int i = box[u]; i != -1; i = edge[i].next)

if(edge[i].cap && !vis[edge[i].to] && edge[i].cost < delta)

delta = edge[i].cost;

if(delta == inf) return false;

for(int u = 0; u < NN; u++)

if(vis[u])

for(int i = box[u]; i != -1; i = edge[i].next)//对于两个点都在里面的，正向修改一次，反向修改一次，即没修改

edge[i].cost -= delta, edge[i^1].cost += delta;

cost += delta;

return true;

}

void costflow(int s,int t)

{

T=t;

cost=0;

do

{

do

{

memset(vis, 0, sizeof(vis));

}while(aug(s, inf));

}while(modlabel());

}

**费用流消圈:**

用于找一个更优解而不是找最优解。先建立残余网络，再用spfa判断是否存在负环，如果存在，则说明存在更优解。

LL dis[maxn];

bool in[maxn];

queue<int> que;

int cnt[maxn];

int from[maxn];

int spfa(int s)

{

memset(in,0,sizeof(in));

for(int i=0;i<2+N+M;i++)

dis[i]=inf;

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

memset(from,-1,sizeof(from));

que.push(s);

in[s]=1;

dis[s]=0;

int now;

while(!que.empty())

{

now=que.front();

que.pop();

for(int i=box[now];i!=-1;i=edge[i].next)

{

if(dis[edge[i].to]>dis[now]+edge[i].cost)

{

from[edge[i].to]=i;

dis[edge[i].to]=dis[now]+edge[i].cost;

if(!in[edge[i].to])

{

que.push(edge[i].to);

in[edge[i].to]=1;

cnt[edge[i].to]++;

if(cnt[edge[i].to]>N+M+2)

{

while(!que.empty())que.pop();

return edge[i].to;

}

}

}

}

in[now]=0;

}

return -1;

}

int vis[maxn];

void findCircle(int now)

{

int i;

memset(vis,0,sizeof(vis));

while(!vis[now]) //遍历两边，确保点都在环中

{

vis[now]=1;

now=edge[from[now]].from;

}

while(vis[now]==1)

{

vis[now]=2;

i=from[now];

if(edge[i].to>=2&&edge[i].to<2+N&&edge[i].from>=2+N)

kplan[edge[i].to-2][edge[i].from-N-2]--;

if(edge[i].from>=2&&edge[i].from<2+N&&edge[i].to>=2+N)

kplan[edge[i].from-2][edge[i].to-N-2]++;

now=edge[i].from;

}

}

**上下界网络流经典构图：**

1.无源汇：

对于每一个点i，令

Mi= sum(i点所有流进来的下界流)– sum(i点所有流出去的下界流)

如果Mi大于0，代表此点必须还要流出去Mi的自由流，那么我们从源点连一条Mi的边到该点。

如果Mi小于0，代表此点必须还要流进来Mi的自由流，那么我们从该点连一条Mi的边到汇点。

（即附加网络）

如果求S->T的最大流，看是否满流(S的相邻边都流满)。

满流则有解，否则无解。

2.有源汇最大流：

连一条T-S的inf边，再按无源汇的做法构建附加网络，看SS是否满流，如果满流则有解。若有解，再做S-T的最大流，答案就是这个最大流加上inf边的反悔流

3.有源汇最小流：

不连T-S边构建附加网络，做一次最大流（消除循环流），再连T-S的inf边，做一次最大流，即为答案

**无向图最小割 Stoer-Wagner:**

O(n^3)

int S, T, minCut, N;

void Search(){

     int i, j, Max, tmp;

     memset(vis, 0, sizeof(vis));

     memset(wet, 0, sizeof(wet));

     S = T = -1;

     for (i = 0; i < N; i++){

         Max = -INF;

         for (j = 0; j < N; j++){

             if (!combine[j] && !vis[j] && wet[j] > Max){

                tmp = j;

                Max = wet[j];

             }

         }

         if (T == tmp) return;

         S = T; T = tmp;

         minCut = Max;

         vis[tmp] = 1;

         for (j = 0; j < N; j++){

             if (!combine[j] && !vis[j]){

                wet[j] += map[tmp][j];

             }

         }

     }

}

int Stoer\_Wagner(){

    int i, j;

    memset(combine, 0, sizeof(combine));

    int ans = INF;

    for (i = 0; i < N - 1; i++){

        Search();

        if (minCut < ans) ans = minCut;

        if (ans == 0) return 0;

        combine[T] = 1;

        for (j = 0; j < N; j++){

            if (!combine[j]){

               map[S][j] += map[T][j];

               map[j][S] += map[j][T];

            }

        }

    }

    return ans;

}