网络流

dinic(层次图贪心预流推进)

```
//按通常方法连边后，在执行网络流前先执行
//  ans=greedy();
int greedy() {
    for (int i = 1; i <= N; i++) rank[i] = i;
    sort (rank + 1, rank + N + 1, cmp);
    memset (in, 0, sizeof (in) );
    memset (out, 0, sizeof (out) );
    in[S] = oo;
    for (int i = 1; i <= N; i++) {
        int x = rank[i];
        for (int j = st[x]; j; j = ne[j])
            if ( (! (j & 1) ) && in[x] > out[x]) {
                int t = min (in[x] - out[x], c[j]);
                in[go[j]] += t;
                out[x] += t;
            }
    }
    memset (in, 0, sizeof (in) );
    in[T] = oo;
    for (int i = N; i; i--) {
        int x = rank[i];
        for (int j = st[x]; j; j = ne[j])
            if ( (j & 1) && out[go[j]] > in[go[j]]) {
                int t = min (min (out[go[j]] - in[go[j]], in[x]),
c[j ^ 1]);
                in[go[j]] += t;
                in[x] -= t;
                c[j] += t;
                c[j ^ 1] -= t;
            }
    }
    return in[S];
}
```

Dinic 上下限

```
void Add (int x, int y, int cc) {
    ne[++pt] = st[x];
    st[x] = pt;
    go[pt] = y;
```

```
    c[pt] = cc;
}

bool build() {
    for (int i = S; i <= T; i++) dist[i] = -1;
    dist[S] = 0;
    q[l = r = 0] = S;
    for (; l <= r; l++) {
        int x = q[l];
        for (int i = st[x]; i; i = ne[i])
            if (c[i] && dist[go[i]] == -1) {
                dist[go[i]] = dist[x] + 1;
                if (go[i] == T) return 1;
                q[++r] = go[i];
            }
    }
    return 0;
}

int dinic (int x, int up) {
    if (x == T) return up;
    int w = 0;
    for (int i = st[x]; w < up && i; i = ne[i])
        if (c[i] && dist[go[i]] == dist[x] + 1) {
            int t = dinic (go[i], min (up - w, c[i]) );
            w += t;
            c[i] -= t;
            c[i ^ 1] += t;
        }
    if (!w) dist[x] = -1;
    return w;
}

int main() {
    scanf ("%d%d", &N, &M);
    pt = 1;
    for (int i = 1; i <= M; i++) {
        scanf ("%d%d%d%d", &P[i].x, &P[i].y, &P[i].l, &P[i].r);
        Add (P[i].x, P[i].y, P[i].r - P[i].l);
        Add (P[i].y, P[i].x, 0);
        in[P[i].y] += P[i].l;
        out[P[i].x] += P[i].l;
    }
    S = 0;
```

```
    T = N + 1;
    for (int i = 1; i <= N; i++) {
        if (in[i] > out[i])
            Add (S, i, in[i] - out[i]), Add (i, S, 0);
        if (in[i] < out[i])
            Add (i, T, out[i] - in[i]), Add (T, i, 0);
    }
    while (build() )
        ans += dinic (S, oo);
}
```

Dinic 退流

```
void antiflow (int x) {
    for (int i = S; i <= T; i++) dist[i] = -1;
    dist[x] = 0;
    for (q[l = r = 0] = x; l <= r && dist[T] == -1; l++) {
        int x = q[l];
        for (int i = st[x]; i; i = ne[i])
            if (c[i ^ 1] && dist[go[i]] == -1) {
                dist[go[i]] = dist[x] + 1;
                p[go[i]] = i;
                if (go[i] == T) break;
                q[++r] = go[i];
            }
    }
    for (int i = T; i != x; i = go[p[i] ^ 1])
        c[p[i]]++, c[p[i] ^ 1]--;
    c[num[x]]++;
    c[num[x] ^ 1]--;
}
```

Dinic - LQY

```
const int MaxN=
const int MaxM=
const int oo=1000000000;

typedef struct {int adj,flo,nxt,bck;} node;

int N,s,t,maxflow;
node edg[MaxM];
int tot,fir[MaxN];
int level[MaxN],fr,re,q[MaxN];
```

```
int tp,st[MaxN],lnk[MaxN];

void AddEdge(int u,int v,int c)
{
    tot++;
    edg[tot].adj=v; edg[tot].flo=c;
    edg[tot].nxt=fir[u]; fir[u]=tot;
    edg[tot].bck=tot+1;
    tot++;
    edg[tot].adj=u; edg[tot].flo=0;
    edg[tot].nxt=fir[v]; fir[v]=tot;
    edg[tot].bck=tot-1;
    return;
}

bool Makelevel()
{
    int i,u;
    for(i=1;i<=N;i++) level[i]=-1;
    level[s]=1;
    fr=re=1; q[fr]=s;
    while(fr<=re)
    {
        u=q[fr++];
        for(i=fir[u];i;i=edg[i].nxt)
        {
            if(edg[i].flo>0&&level[edg[i].adj]==-1)
            {
               level[edg[i].adj]=level[u]+1;
               if(edg[i].adj==t) return true;
               q[++re]=edg[i].adj;
            }
        }
    }
    return false;
}

void Dinic()
{
    int i,u,f;
    for(i=1;i<=N;i++) lnk[i]=fir[i];
    st[tp=1]=s;
    while(tp)
    {
```

```
        u=st[tp];
        if(u!=t)
        {
           for(i=lnk[u];i;i=edg[i].nxt)
              if(edg[i].flo>0&&level[edg[i].adj]==level[u]+1)
                 break;
           if(i) lnk[u]=i,st[++tp]=edg[i].adj;
           else level[u]=-1,tp--;
        }
        else
        {
            f=oo;
            for(i=1;i<tp;i++) f=min(f,edg[lnk[st[i]]].flo);
            maxflow+=f;
            for(i=1;i<tp;i++)
            {
                edg[lnk[st[i]]].flo-=f;
                edg[edg[lnk[st[i]]].bck].flo+=f;
            }
            for(tp=1;edg[lnk[st[tp]]].flo>0;tp++);
        }
    }
    return;
}
```

Dinic 复杂度

dinic 普通图最坏复杂度 O(V^2*E)

边的容量为 1 时，复杂度 O(min(V^(2/3),E^(1/2)) * E)

二分图上，阶段数为 O(V^(1/2))

并且只需要除了源、汇的结点满足只有一条容量为 1 的出边或只有一条容量为 1 的入边，复杂度同样为 O(V^(1/2) * E)

可以尝试用 LCT 维护有效边（分层图中的边）的生成森林

在一个阶段中，分层图的反向边不会造成影响

参考周以凡 2014 年浙江省选二试讲课

可以将 dinic 复杂度优化到 O(VElogV)

实际效果不明显

部分最小割方案求法

求必然是最小割边的边集：

求一遍最大流，如果某条边符合我们的要求，那么这条边必然是满流的，否则增加容量也没有用，然后假设端点为 u->v，那么必然存在从源点到 u 的不饱和的路径，也就是路径

上的每条边都是不满的，同样从 v 到汇点也应该有这样的路径，也只有这样才会在该边增加容量后形成一条增广路。
那么我们搞定这个问题的方法就是 DFS 了，从源点出发，沿着那些不满的边进行 DFS，这样遍历到的点均是可以与源点有一条不饱和的路径的。
同理从汇点出发(反向走不饱和的正边)，进行逆向的 DFS，这时最好先建个逆图。

求任一最小割方案：
计算方案： 对于构图后跑一次最大流，然后对于残留网络进行处理，首先从源点 S 出发，标记所有能访问到的顶点，这些顶点即为 S 割点集中的顶点。其他则为 T 集合中顶点，然后从所有边中筛选出(A 属于 S,B 属于 T,且（A,B)容量为 0)的边，即为割边。因为我们的 W+/W-边都只有一条，且都分开了。比较容易处理。

KM

```
bool path(int u) {
    vis[u] = 1;
    for (int i = fst[u]; i; i = edge[i].nex) {
        int v = edge[i].v;
        if (sig(edge[i].len - Mark[u] - Mark[v])) {
            if (vis[v]) continue;
            vis[v] = 1;
            if (!nex[v] || path(nex[v])) {
                nex[v] = u, nex[u] = v;
                return true;
            }
        } else {
            slk[v] = min(slk[v], edge[i].len - Mark[u] - Mark[v]);
        }
    }
    return false;
}

void km() {
    for (int i = 1; i <= nl; i++) Mark[lp[i]] = INF;
    for (int i = 1; i <= nr; i++) Mark[rp[i]] = 0;
    for (int i = 1; i <= nl; i++) {
        int u = lp[i];
        for (int j = fst[u]; j; j = edge[j].nex) {
            Mark[u] = min(Mark[u], edge[j].len);
        }
    }

    for (int i = 1; i <= nl; i++) {
        int u = lp[i];
```

```
        for (int j = 1; j <= nr; j++)
            slk[rp[j]] = 1 << 28;
        for (int j = 1; j <= nl; j++)
            vis[rp[j]] = vis[lp[j]] = 0;

        while (!path(u)) {
            double sing = 1 << 28;
            for (int j = 1; j <= nr; j++) {
                int v = rp[j];
                if (!vis[v]) sing = min(sing, slk[v]);
            }

            for (int j = 1; j <= nl; j++) if (vis[lp[j]])
Mark[lp[j]] += sing;
            for (int j = 1; j <= nr; j++) if (vis[rp[j]])
Mark[rp[j]] -= sing;
            for (int j = 1; j <= nr; j++) slk[rp[j]] = 1 << 28,
vis[rp[j]] = 0;
        }
    }

    for (int i = 1; i <= nl; i++) ans += Mark[lp[i]];
    for (int i = 1; i <= nr; i++) ans += Mark[rp[i]];
}
```

单纯形 - someone

```
//代码来源于BZOJ3112
//题意：
//战线可以看作一个长度为 n 的序列，现在需要在这个序列上建塔来防守敌兵
//在序列第i号位置上建一座塔有 Ci 的花费，且一个位置可以建任意多的塔，费用累加计算
//有 m 个区间[Li, Ri]，在第i个区间的范围内要建至少 Di 座塔 。求最少花费。

#include <cstdio>
int a[1005][10005],b[10005][2],N,M;

void pivot(int x,int y)
{
    int l=0;
    for (int i=0; i<=M; i++) if (a[x][i]&&i!=y)
b[++l][0]=a[x][i],b[l][1]=i;
    for (int i=0,t; i<=N; i++) if (i!=x&&(t=a[i][y]))
    {
```

```
        for (int j=0; j<=l; j++) a[i][b[j][1]]-=t*b[j][0];
        a[i][y]*=-1;
    }
}

int simplex()
{
    for (int x,y,m;;)
    {
        m=-int(1e9);for (int i=1; i<=M; i++) if (a[0][i]>m)
m=a[0][i],y=i;
        if (m<=0) return -a[0][0];
        m=int(1e9); for (int i=1; i<=N; i++) if
(a[i][y]>0&&a[i][0]<m) m=a[i][0],x=i;
        pivot(x,y);
    }
}

void doit()
{
    scanf("%d%d",&N,&M);
    for (int i=1; i<=N; i++) scanf("%d",&a[i][0]);//Ci 数组
    for (int i=1,l,r; i<=M; i++)
    {
        scanf("%d%d%d",&l,&r,&a[0][i]);
        for (int j=l; j<=r; j++) a[j][i]=1;
    }
    printf("%d\n",simplex());
}

int main()
{
    doit();
    return 0;
}
```

线性规划 对偶问题

线性规划问题

标准型：

最大化

sigma(Cj * Xj) 1 <= j <= N

满足约束

sigma(Aij * Xj) <= Bi 1 <= i <= M

```
    Xj >= 0                1 <= j <= N
```

对偶问题：

最小化

```
    sigma(Bi * Yi)         1 <= i <= M
```

满足约束

```
    sigma(Aij * Yi) >= Cj  1 <= j <= N
    Yi >= 0   1 <= i <= M
```

费用流_SPFA 多路增广_WP

```
bool bfs() {
    for (int i = 1; i <= t; i++) dis[i] = 1 << 29;
    dis[s] = 0;
    for (int i = 1; i <= t; i++) mark[i] = 0;
    que.push (s);
    mark[s] = 1;

    while (!que.empty() ) {
        int u = que.front();
        que.pop();
        mark[u] = 0;
        for (int i = fst[u]; i; i = edge[i].nex) {
            int v = edge[i].v;
            if (!edge[i].cap) continue;
            if (dis[v] > dis[u] + edge[i].cost) {
                dis[v] = dis[u] + edge[i].cost;
                if (!mark[v]) {
                    mark[v] = 1;
                    que.push (v);
                }
            }
        }
    }

    return (dis[t] != (1 << 29) );
}

int dfs (int u, int ma) {
    if (vis[u]) return 0;
    if (u == t || !ma) return ma;
    vis[u] = 1;
    int f, flow = 0;
```

```
    for (int i = fst[u]; i; i = edge[i].nex) {
        int v = edge[i].v;
        if (dis[v] == dis[u] + edge[i].cost && (f = dfs (v, min
(edge[i].cap, ma) ) ) ) {
            flow += f;
            ma -= f;
            edge[i].cap -= f;
            edge[i ^ 1].cap += f;
            //ans += edge[i].cost * f;
            if (!ma) {
                vis[u] = 0;
                return flow;
            }
        }
    }
    vis[u] = 0;
    dis[u] = INF;
    return flow;
}

void solve() {
    while (bfs() ) {
        int flow = dfs (s, INF);
        ans += flow * dis[t];
    }
}
```

最小费用最大流_LQY

```
#include<iostream>
#include<cstdio>
using namespace std;

const int MaxN=6000;
const int MaxM=300000;
const int oo=1000000000;

typedef struct {int u,v,f,w,nxt,bck;} node;

int N,s,t,maxflow,mincost;
int tot,fir[MaxN];
node edg[MaxM];
int d[MaxN],fr,re,q[MaxN],lnk[MaxN];
bool inq[MaxN];
```

```
void AddEdge(int u,int v,int c,int w)
{
    //cout<<u<<' '<<v<<' '<<c<<' '<<w<<endl;
    tot++;
    edg[tot].u=u; edg[tot].v=v;
    edg[tot].f=c; edg[tot].w=-w;
    edg[tot].nxt=fir[u]; fir[u]=tot;
    edg[tot].bck=tot+1;
    tot++;
    edg[tot].u=v; edg[tot].v=u;
    edg[tot].f=0; edg[tot].w=w;
    edg[tot].nxt=fir[v]; fir[v]=tot;
    edg[tot].bck=tot-1;
    return;
}

bool BFS()
{
    int i,j,u;
    for(i=1;i<=N;i++) d[i]=oo;
    d[s]=0; fr=0; re=1;
    q[re]=s; inq[s]=true;
    while(fr!=re)
    {
        fr=(fr+1)%MaxN; u=q[fr];
        for(i=fir[u];i;i=edg[i].nxt)
        {
            j=edg[i].v;
            if(edg[i].f>0&&d[j]>d[u]+edg[i].w)
            {
               d[j]=d[u]+edg[i].w;
               lnk[j]=i;
               if(!inq[j])
               {
                  re=(re+1)%MaxN;
                  q[re]=j; inq[j]=true;
               }
            }
        }
        inq[u]=false;
    }
    return (d[t]!=oo);
}
```

```
void Adjust()
{
    int i,f,w;
    for(i=t,f=oo;i!=s;i=edg[lnk[i]].u)
        f=min(f,edg[lnk[i]].f);
    for(i=t,w=0;i!=s;i=edg[lnk[i]].u)
    {
        edg[lnk[i]].f-=f;
        edg[edg[lnk[i]].bck].f+=f;
        w+=edg[lnk[i]].w;
    }
    maxflow+=f;
    mincost+=f*w;
    return;
}

int n,k;

void Init()
{
    int i,j,w;
    scanf("%d%d",&n,&k);
    N=n*n*2; s=++N,t=++N;
    AddEdge(s,1,k,0);
    AddEdge(n*n*2,t,k,0);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            scanf("%d",&w);
            AddEdge(2*((i-1)*n+j)-1,2*((i-1)*n+j),1,w);
            AddEdge(2*((i-1)*n+j)-1,2*((i-1)*n+j),k-1,0);
        }
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            if(j<n) AddEdge(2*((i-1)*n+j),2*((i-1)*n+j)+1,k,0);
            if(i<n) AddEdge(2*((i-1)*n+j),2*(i*n+j)-1,k,0);
        }
    return;
}

void Solve()
{
```

```
    while(BFS()) Adjust();
    printf("%d",-mincost);
    return;
}

int main()
{
   Init();
   Solve();
   return 0;
}
```

匈牙利树_LQY

```
#include <iostream>
#include <fstream>
using  namespace  std;
const  int  MAXN  =  100 ;
int  uN, vN;  // u,v 数目
bool  g[MAXN][MAXN]; // g[i][j] 表示 xi 与 yj 相连
int  xM[MAXN], yM[MAXN];  //  输出量
bool  chk[MAXN];  // 辅助量 检查某轮 y[v]是否被 check

bool  SearchPath( int  u )
  {
    int  v;
    for(v = 0; v < vN; v ++ )
     {
       if(g[u][v] && ! chk[v])
        {
          chk[v] = true ;
           if  (yM[v]== -1 || SearchPath(yM[v]))
            {
              yM[v] = u;
              xM[u] = v;
              return  true ;
            }
        }
     }
    return  false ;
  }

int  MaxMatch()
  {
```

```
    int  u;
    int  ret  =   0 ;
    memset(xM,  -1 ,  sizeof (xM));
    memset(yM,  -1 ,  sizeof (yM));
    for(u = 0 ; u < uN; u ++ )
      {
         if (xM[u] == - 1 )
          {
            memset(chk, false , sizeof (chk));
            if(SearchPath(u)) ret++ ;
        }
    }
     return  ret;
}

int  main(void)
  {
     int  i, k;
     int  tU, tV;
    ifstream cin("test.in");
    ofstream cout("test.out");
    cin  >>  uN  >>  vN  >>  k;
    memset(g,  false ,  sizeof (g));
     for  (i = 0 ; i < k; i ++ )
      {
        cin >> tU >> tV;
        g[tU][tV]=true ;
    }
     int  M = MaxMatch();
    cout  <<" Total Match:  " << M <<endl;
     for  (i = 0 ; i < MAXN; i ++ )
        if  (xM[i]  !=   - 1 )
           cout<<i<<' '<<xM[i]<<endl;
     return   0 ;
}
```