网络流

EK算法

- 时间复杂度O(V^2E)，适合稠密图
- 对于容量为0的通路，网络里**不会存在**该有向边，但是在一般图里**存在**权值为0的有向边。
- **不要忘记初始化！！！！！**

```
const int inf=0x3f3f3f3f;
const int N=205;
const int M=205;
int cnt;
int head[N],pre[N];
bool vis[N];
struct Edge{
   int v,next;
   int cap,flow;
}E[M<<1];//双边

void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}

void add(int u,int v,int c){
    E[cnt].v=v;
    E[cnt].cap=c;
    E[cnt].flow=0;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

bool bfs(int s,int t){
    memset(pre,-1,sizeof(pre));
    memset(vis,0,sizeof(vis));
    queue<int>q;
```

```
    vis[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];~i;i=E[i].next){
            int v=E[i].v;
            if(!vis[v]&&E[i].cap>E[i].flow){
                vis[v]=1;
                pre[v]=i;//边下标
                q.push(v);
                if(v==t)  return 1;//找到一条可增广路
            }
        }
    }
    return 0;
}

int EK(int s,int t){
    int maxflow=0;
    while(bfs(s,t)){
        int v=t,d=inf;
        while(v!=s){//找可增量
            int i=pre[v];
            d=min(d,E[i].cap-E[i].flow);
            v=E[i^1].v;
        }
        maxflow+=d;
        v=t;
        while(v!=s){//沿可增广路增流
            int i=pre[v];
            E[i].flow+=d;
            E[i^1].flow-=d;
            v=E[i^1].v;
        }
    }
    return maxflow;
}
```

Dinic算法

- 时间复杂度O(VE^2)，适合稀疏图
- 对于容量为0的通路，网络里**不会存在**该有向边，但是在一般图里**存在**权值为0的有向边。
- **不要忘记初始化！！！！！**

```
const int inf=0x3f3f3f3f;
const int N=205;
const int M=205;
int cnt;
int head[N],d[N];
struct Edge{
   int v,next;
   int cap,flow;
}E[M<<1];//双边
```

```
void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}

void add(int u,int v,int c){
    E[cnt].v=v;
    E[cnt].cap=c;
    E[cnt].flow=0;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

bool bfs(int s,int t){//分层
    memset(d,0,sizeof(d));
    queue<int>q;
    d[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];~i;i=E[i].next){
            int v=E[i].v;
            if(!d[v]&&E[i].cap>E[i].flow){
                d[v]=d[u]+1;
                q.push(v);
                if(v==t)  return 1;
            }
        }
    }
    return 0;
}

int dfs(int u,int flow,int t){//在分层的基础上dfs
    if(u==t) return flow;
    int rest=flow;
    for(int i=head[u];~i&&rest;i=E[i].next){
        int v=E[i].v;
        if(d[v]==d[u]+1&&E[i].cap>E[i].flow){
            int k=dfs(v,min(rest,E[i].cap-E[i].flow),t);
            if(!k) d[v]=0;
            E[i].flow+=k;
            E[i^1].flow-=k;
            rest-=k;
        }
    }
    return flow-rest;
}

int Dinic(int s,int t){
    int maxflow=0;
    while(bfs(s,t)){
        maxflow+=dfs(s,inf,t);
    }
    return maxflow;
}
```

ISAP算法

- 时间复杂度O(V^2E)，适合稠密图
- 对于容量为0的通路，网络里**不会存在**该有向边，但是在一般图里**存在**权值为0的有向边。
- 可以提前结束，所以**速度非常快**（比之前的算法快100倍）
- **不要忘记初始化！！！！！**

```
const int inf=0x3f3f3f3f;
const int N=205;
const int M=205;
int cnt;
int head[N],pre[N],h[N],g[N];
struct Edge{
   int v,next;
   int cap,flow;
}E[M<<1];//双边

void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}

void add(int u,int v,int c){
    E[cnt].v=v;
    E[cnt].cap=c;
    E[cnt].flow=0;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

void set_h(int t,int n){//标高
    queue<int> q;
    memset(h,-1,sizeof(h));
    memset(g,0,sizeof(g));
    h[t]=0;
    q.push(t);
    while(!q.empty()){
        int u=q.front();q.pop();
        ++g[h[u]];//高度为h[u]的节点个数
        for(int i=head[u];~i;i=E[i].next){
            int v=E[i].v;
            if(h[v]==-1){
                h[v]=h[u]+1;
                q.push(v);
            }
        }
    }
}

int ISAP(int s,int t,int n){
    set_h(t,n);
    int ans=0,u=s,d;
    while(h[s]<n){
        int i=head[u];
        if(u==s)
```

```
            d=inf;
        for(;~i;i=E[i].next){
            int v=E[i].v;
            if(E[i].cap>E[i].flow&&h[u]==h[v]+1){
                u=v;
                pre[v]=i;
                d=min(d,E[i].cap-E[i].flow);
                if(u==t){
                    while(u!=s){
                        int j=pre[u];
                        E[j].flow+=d;
                        E[j^1].flow-=d;
                        u=E[j^1].v;
                    }
                    ans+=d;
                    d=inf;
                }
                break;
            }
        }
        if(i==-1){
            if(--g[h[u]]==0)
                break;
            int hmin=n-1;
            for(int j=head[u];~j;j=E[j].next)
                if(E[j].cap>E[j].flow)
                    hmin=min(hmin,h[E[j].v]);
            h[u]=hmin+1;
            ++g[h[u]];
            if(u!=s)
                u=E[pre[u]^1].v;
        }
    }
    return ans;
}
```

二分图最大匹配（匈牙利算法）

- 最小点覆盖等于最大匹配数

一重匹配

- 注意要初始化

```
#include<cstdio>
#include<cstring>
using namespace std;
const int inf=0x3f3f3f3f;
const int N=405;//注意节点数牛+牛棚
const int M=40500;
int cnt;
int head[N],match[N];
bool vis[N];
struct Edge{
   int v,next;
}E[M<<1];//双边
```

```
void init(){//初始化
    memset(head,-1,sizeof(head));
    memset(match,0,sizeof(match));
    cnt=0;
}

void add(int u,int v){
    E[cnt].v=v;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

bool Find(int u){
    for(int i=head[u];~i;i=E[i].next){
        int v=E[i].v;
        if(!vis[v]){
            vis[v]=1;
            if(!match[v]||Find(match[v])){
                match[v]=u;
                return true;
            }
        }
    }
    return false;
}

int main(){
    int n,m,k,v;
    while(~scanf("%d%d",&n,&m)){
        init();
        for(int i=1;i<=n;i++){
            scanf("%d",&k);
            for(int j=1;j<=k;j++){
                scanf("%d",&v);
                add(i,n+v);
            }
        }
        int ans=0;
        for(int i=1;i<=n;i++){
            memset(vis,0,sizeof(vis));
            if(Find(i))
                ans++;
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

二分图多重匹配

```
#include<cstdio>//二分图多重匹配，匈牙利算法，1279ms
#include<cstring>
using namespace std;
const int N=100002;
int cap[12],g[N][12],cnt[12],match[12][N];
bool vis[12];
int n,m;
```

```
int dfs(int u){
    for(int i=0;i<m;i++){
        if(g[u][i]&&!vis[i]){
            vis[i]=true;
            if(cnt[i]<cap[i]){//匹配次数小于容量
                match[i][cnt[i]++]=u;
                return 1;
            }
            for(int j=0;j<cnt[i];j++){
                if(dfs(match[i][j])){
                    match[i][j]=u;
                    return 1;
                }
            }
        }
    }
    return 0;
}

int main(){
    while(~scanf("%d%d",&n,&m)){
        for(int i=0;i<n;i++)
            for(int j=0;j<m;j++)
                scanf("%d",&g[i][j]);
        for(int i=0;i<m;i++)
            scanf("%d",&cap[i]);
        memset(cnt,0,sizeof(cnt));
        bool flag=true;
        for(int i=0;i<n;i++){
            memset(vis,0,sizeof(vis));
            if(!dfs(i)){
                flag=false;
                break;
            }
        }
        if(flag) printf("YES\n");
        else printf("NO\n");
    }
    return 0;
}
```

多分图匹配

- 用最大流即可，但是注意中间的点需要拆点，保证只匹配一次

最大流最小割

- 最大流的值等于最小割的容量

最小边割集

- **删除最少的边使得图不连通（且收益最大）**
- 对于怎么判断割边，即怎么判断一条边的两个端点分别在集合S,T中，在Dinic和ISAP,EK有不同的体现。如果采用Dinic算法求最大流，则可以直接根据**最后一次分层进行判断，层次为真的节点属于S集合，其他节点属于T集合。**如果采用EK或者ISAP算法求最大流，则**需要从原点出发，沿着**`cap>flow`**的边进行dfs，标记已访问的节点**，源点和已访问的节点为S集合，其余点和汇点为T集合。

Dinic

```
#include<cstdio>//ISAP 1279ms,Dinic 1326ms,Dinic2 1248ms当前弧优化
#include<cstring>
#include<queue>
#include<algorithm>
using namespace std;
const int inf=0x3f3f3f3f;
const int N=1010;
const int M=101010;
int cnt,tot;
int head[N],d[N],ans[N];
bool vis[N];
struct Edge{
   int v,next;
   int cap,flow;
}E[M<<1];//双边

void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}

void add(int u,int v,int c){
    E[cnt].v=v;
    E[cnt].cap=c;
    E[cnt].flow=0;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

void adde(int u,int v,int c){
    add(u,v,c);
    add(v,u,0);
}

bool bfs(int s,int t){//分层
    memset(d,0,sizeof(d));
    queue<int>q;
    d[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];~i;i=E[i].next){
            int v=E[i].v;
            if(!d[v]&&E[i].cap>E[i].flow){
```

```
                d[v]=d[u]+1;
                q.push(v);
                if(v==t)  return 1;
            }
        }
    }
    return 0;
}

int dfs(int u,int flow,int t){//在分层的基础上dfs
    if(u==t) return flow;
    int rest=flow;
    for(int i=head[u];~i&&rest;i=E[i].next){
        int v=E[i].v;
        if(d[v]==d[u]+1&&E[i].cap>E[i].flow){
            int k=dfs(v,min(rest,E[i].cap-E[i].flow),t);
            if(!k) d[v]=0;
            E[i].flow+=k;
            E[i^1].flow-=k;
            rest-=k;
        }
    }
    return flow-rest;
}

int Dinic(int s,int t){
    int maxflow=0;
    while(bfs(s,t)){
        maxflow+=dfs(s,inf,t);
    }
    return maxflow;
}

int main(){
    int T,n,m,f,u,v,w,cas=0;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d",&n,&m,&f);
        int s=1,t=n+1;
        init();
        for(int i=1;i<=m;i++){
            scanf("%d%d%d",&u,&v,&w);
            adde(u,v,w);
        }
        int tot=0;
        for(int i=1;i<=f;i++){
            scanf("%d%d",&u,&w);
            tot+=w;
            adde(u,t,w);
        }
        printf("Case %d: %d\n",++cas,tot-Dinic(s,t));
        tot=0;
        for(int i=0;i<2*m;i+=2){//注意是0~2m，不包括点权到汇点的边，每次增2
            int u=E[i^1].v,v=E[i].v;
            if(d[u]&&!d[v])//最后一次bfs,d[]为真的属于S集合
                ans[tot++]=i/2;
        }
        printf("%d",tot);
```

```
        for(int i=0;i<tot;i++)
            printf(" %d",ans[i]+1);
        printf("\n");
    }
    return 0 ;
}
```

ISAP

```
#include<cstdio>//ISAP 1279ms,Dinic 1326ms,Dinic2 1248ms
#include<cstring>
#include<queue>
#include<algorithm>
using namespace std;
const int inf=0x3f3f3f3f;
const int N=1010;
const int M=101010;
int cnt,tot;
int head[N],ans[N],pre[N],h[N],g[N];
bool vis[N];
struct Edge{
   int v,next;
   int cap,flow;
}E[M<<1];//双边

void init(){//初始化
    memset(head,-1,sizeof(head));
    cnt=0;
}

void add(int u,int v,int c){
    E[cnt].v=v;
    E[cnt].cap=c;
    E[cnt].flow=0;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

void adde(int u,int v,int c){
    add(u,v,c);
    add(v,u,0);
}

void set_h(int t){//标高
    queue<int> q;
    memset(h,-1,sizeof(h));
    memset(g,0,sizeof(g));
    h[t]=0;
    q.push(t);
    while(!q.empty()){
        int u=q.front();q.pop();
        ++g[h[u]];//高度为h[u]的节点个数
        for(int i=head[u];~i;i=E[i].next){
            int v=E[i].v;
            if(h[v]==-1){
                h[v]=h[u]+1;
                q.push(v);
```

```
            }
        }
    }
}

int ISAP(int s,int t,int n){
    set_h(t);
    int ans=0,u=s,d;
    while(h[s]<n){
        int i=head[u];
        if(u==s)
           d=inf;
        for(;~i;i=E[i].next){
            int v=E[i].v;
            if(E[i].cap>E[i].flow&&h[u]==h[v]+1){
                u=v;
                pre[v]=i;
                d=min(d,E[i].cap-E[i].flow);
                if(u==t){
                    while(u!=s){
                        int j=pre[u];
                        E[j].flow+=d;
                        E[j^1].flow-=d;
                        u=E[j^1].v;
                    }
                    ans+=d;
                    d=inf;
                }
                break;
            }
        }
        if(i==-1){
            if(--g[h[u]]==0)
                break;
            int hmin=n-1;
            for(int j=head[u];~j;j=E[j].next)
                if(E[j].cap>E[j].flow)
                    hmin=min(hmin,h[E[j].v]);
            h[u]=hmin+1;
            ++g[h[u]];
            if(u!=s)
                u=E[pre[u]^1].v;
        }
    }
    return ans;
}

void dfs(int u){
    for(int i=head[u];~i;i=E[i].next){
        int v=E[i].v;
        if(!vis[v]&&E[i].cap>E[i].flow){
            vis[v]=1;
            dfs(v);
        }
    }
}

int main(){
```

```
    int T,n,m,f,u,v,w,cas=0;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d",&n,&m,&f);
        int s=1,t=n+1;
        init();
        for(int i=1;i<=m;i++){
            scanf("%d%d%d",&u,&v,&w);
            adde(u,v,w);
        }
        int tot=0;
        for(int i=1;i<=f;i++){
            scanf("%d%d",&u,&w);
            tot+=w;
            adde(u,t,w);
        }
        printf("Case %d: %d\n",++cas,tot-ISAP(s,t,t));
        memset(vis,0,sizeof(vis));
        vis[s]=true;
        dfs(s);
        tot=0;
        for(int i=0;i<2*m;i+=2){
            int u=E[i^1].v,v=E[i].v;
            if(vis[u]&&!vis[v])
                ans[tot++]=i/2;
        }
        printf("%d",tot);
        for(int i=0;i<tot;i++)
            printf(" %d",ans[i]+1);
        printf("\n");
    }
    return 0 ;
}
```

最小点割集

- **删除权值之和尽量小的点，使得源点到汇点不连通。**
- 对于无向带权图点连通性的网络流问题中，**点权需要转化为边权**，可以将**每个点都拆成两个点`u`和`u'`，容量为点权。**将**原图的无向边`(u,v)`拆成两条边`(u',v)`和`(u,v')`，容量为无穷大**，转化为最小割问题，根据最大流最小割定理，**求解`S'`到`T`的最大流即可。**

```
#include<cstdio>
#include<cstring>
#include<queue>
#include<algorithm>
using namespace std;
const int inf=0x3f3f3f3f;
const int N=210;
const int M=15000;
int cnt;
int head[N],d[N];
struct Edge{
   int v,next;
   int cap,flow;
}E[M<<1];//双边

void init(){//初始化
```

```
    memset(head,-1,sizeof(head));
    cnt=0;
}

void add(int u,int v,int c){
    E[cnt].v=v;
    E[cnt].cap=c;
    E[cnt].flow=0;
    E[cnt].next=head[u];
    head[u]=cnt++;
}

void adde(int u,int v,int c){
    add(u,v,c);
    add(v,u,0);
}

bool bfs(int s,int t){//分层
    memset(d,0,sizeof(d));
    queue<int>q;
    d[s]=1;
    q.push(s);
    while(!q.empty()){
        int u=q.front();
        q.pop();
        for(int i=head[u];~i;i=E[i].next){
            int v=E[i].v;
            if(!d[v]&&E[i].cap>E[i].flow){
                d[v]=d[u]+1;
                q.push(v);
                if(v==t)  return 1;
            }
        }
    }
    return 0;
}

int dfs(int u,int flow,int t){//在分层的基础上dfs
    if(u==t) return flow;
    int rest=flow;
    for(int i=head[u];~i&&rest;i=E[i].next){
        int v=E[i].v;
        if(d[v]==d[u]+1&&E[i].cap>E[i].flow){
            int k=dfs(v,min(rest,E[i].cap-E[i].flow),t);
            if(!k) d[v]=0;
            E[i].flow+=k;
            E[i^1].flow-=k;
            rest-=k;
        }
    }
    return flow-rest;
}

int Dinic(int s,int t){
    int maxflow=0;
    while(bfs(s,t)){
        maxflow+=dfs(s,inf,t);
    }
```

```
    return maxflow;
}

int main(){
    int T,a,b,n,m,s,t;
    scanf("%d",&T);
    while(T--){
        scanf("%d%d%d%d",&n,&m,&s,&t);
        s+=n;
        init();
        for(int i=1;i<=n;i++){
            scanf("%d",&a);
            adde(i,i+n,a);
        }
        for(int i=1;i<=m;i++){
            scanf("%d%d",&a,&b);
            adde(a+n,b,inf);
            adde(b+n,a,inf);
        }
        printf("%d\n",Dinic(s,t));
    }
    return 0;
}
```