目录

图论

总结一下图论算法的模板

图的储存

临接表，用这种方法比"链式前向星"，好，主要是写起来简单，而且在空间上的效率是一样的 $O(|V|+|E|)$

```
int ne,nv;
struct Edge{
  int from,to,weight;
  Edge(int u=0,int v=0,int w=0):from(u),to(v),weight(w){};
};

int tot=0;
Edge E[maxn*2];
std::vector<int> G[MAX_V];
int dist[MAX_V];
void add_edge(int u,int v,int w){
  E[tot++] = Edge{u,v,w};
  G[u].push_back(tot-1);
}
```

最短路

1. dijkstra

在没有负边的时候使用，$O(|E|lg|V|)$

```
void dijkstra(int s){
  priority_queue<Pair,std::vector<Pair>  > pq;
  memset(dist,INF,sizeof(dist));
  pq.push(Pair(0,s));
  dist[s] = 0;
  while (!pq.empty()) {
    Pair p = pq.top();pq.pop();
    int d = p.fi,u = p.se;
    if(dist[u]<d)continue;
    for(int i=0 ; i<G[u].size() ; ++i)
    {
      int e = G[u][i];
      int v = E[e].to;
      if(dist[v]>dist[u]+E[e].weight){
        dist[v] = dist[u]+E[e].weight;
```

```
        pq.push(Pair(dist[v],v));
      }
    }
  }
}
```

2. Bellman_Ford

实现简单，复杂度 $O(|V|*|E|)$

```
bool Bellman_Ford(s)
{
  memset(d,INF,sizeof(INF));
  for(int i=0 ; i<nv ; ++i){
    for(int j=0 ; j<E.size() ; ++j)
    d[E[i].to] = min(d[E[i].to],d[E[i].from]+E[i].weight);
  }
  //负环判定
  for(int i=0 ; i<E.size() ; ++i){
    if(d[E[i].to]<d[E[i].from]+E[i].weight)return false;
  }
  return true;
}
```

3. spfa

```
bool spfa(int s){
  memset(d,INF,sizeof(d));
  memset(inq,false,sizeof(inq));
  memset(cnt,0,sizeof(cnt));
  queue<int> Q;
  Q.push(s);
  d[s] = 0;
  inq[s] = true;
  while(!Q.empty())
  {
    int u = Q.front();
    inq[u] = false;
    for(int i=0 ; i<G[u].size() ; ++i){
      Edge &e = E[G[u][i]];
      if(d[e.to]>d[e.from]+e.weight){
        d[e.to] = d[e.from]+e.weight;
        if(!inq[e.to]){
          Q.push(e.to);inq[e.to] = true;
          if(++cnt[e.to]>n)return false;
        }
      }
    }
  }
  return true;
}
```

次短路

1. dijkstra

仿照求最短路，我们记录一下他的次短路，再做相应的更新，（允许重复走，所以肯定有次短路）

```
void dijkstra2(int s){
  memset(dist,INF,sizeof(dist));
  memset(dist2,INF,sizeof(dist2));
  priority_queue<PII,std::vector<PII> ,greater<PII> > pq;
  pq.push(PII(0,s));
  dist[s] = 0;
  while (!pq.empty()) {
    int d = pq.top().fi; int u = pq.top().se;
    pq.pop();
    if(dist2[u]<d)continue;
    for(int i=0 ; i<G[u].size() ; ++i){
      int e = G[u][i];
      int v = E[e].to;
      int tmp = d+E[e].weight;
      if(dist[v]>tmp){
        swap(dist[v],tmp);//可能 d 成为次短路
        pq.push(PII(dist[v],E[e].to));
      }
      if(dist2[v]>tmp && tmp>dist[v])
      {
        dist2[v] = tmp;
        pq.push(PII(tmp,E[e].to));
      }
    }
  }
}
```

最小生成树

```
int kruskal(){
    sort(E.begin(),E.end());
    int res = 0;
    for(int i=0 ; i<E.size() ; ++i){
      int u = E[i].from,v = E[i].to;
      if(find(u)!=find(v)){
        UNION(u,v);
        res+=E[i].weight;
      }
    }
    return res;
}
```

所有节点对的最短路

```
void floyd(){
  for(int k=1 ; k<=nv ; ++k)
    for(int i =1 ; i<=nv ; ++i)
      for(int j=1 ; j<=nv ; ++j){
        d[i][j] = min(d[i][j],d[i][k]+d[k][j]);
      }
}
```

传递闭包

上面的算法稍微修改以下就可以求传递传递闭包了。

```
void floyd(){
  for(int k=1 ; k<=nv ; ++k)
    for(int i =1 ; i<=nv ; ++i)
      for(int j=1 ; j<=nv ; ++j){
        d[i][j] |=d[i][k]&d[k][j];
      }
}
```

若干和 dfs 相关的算法

二分图判定

```
std::vector<int> G[maxn];//G[u][i]表示第 i 个邻居

int color[maxn];//0:未着色,1:黑色,2:白色.
//同找奇环
bool bipartite(int u){
    //仿照 dfs
    for(auto v: G[u]){
        if(color[v]==color[u])return false;
        if(!color[v]){
            color[v] = 3-color[u]//对立颜色
            if(!bipartite(v))return false;
        }
    }
    return true;
}
```

割点和桥

```
//判断重边可以开个 map 来判断.然后置为-1
int dfs_clock = 0;//时间戳
int pre[maxn];//顶点的访问顺序
int low[maxn];//子树的最低访问时间
int cut[maxn];//割点后能增加的联通分量数
void init(){
```

```
    memset(pre,0,sizeof(pre));
    memset(cut,0,sizeof(cut));
    dfs_clock = 0;
}
//只能跑一个连通分量中的割点
void dfs(int u,int fa){
    low[u] = pre[u] = ++dfs_clock;
    int cl = 0;//孩子数目
    for(int i=0 ; i<G[u].size() ; ++i){
        Edge & e = E[G[u][i]];
        int v = e.to;
        if(!pre[v]){
            cl++;
            dfs(v,u);
            low[u]= min(low[u],low[v]);
            //判断割点
            if(low[v] >= pre[u])cut[u]++;//增加(u,v)的联通集
            //判断割边
            if(low[v] > pre[u] && e.qiao !=-1)e.qiao = 1;
        }else if(pre[v] < pre[u] && v != fa){
            low[u] = min(low[u],pre[v]);
        }
    }
    if(fa == -1 && cl == 1)cut[u] = 0;//根节点
}
```

双连通分量 bcc

```
std::vector<int> G[maxn];
int dfs_clock = 0;//时间戳
int pre[maxn];//顶点的访问顺序
int low[maxn];//子树的最低访问时间
int cut[maxn];//割点后能增加的联通分量数
int bcc_cnt;
std::vector<int> bcc[maxn];//双连通分量点集
int bcc_no[maxn];//每个点的临时编号

stack<Pair> S;

void init(){
    memset(pre,0,sizeof(pre));
    memset(cut,0,sizeof(cut));
    dfs_clock = 0;
    bcc_cnt = 0;
    memset(bcc_no,0,sizeof(bcc_no));
}
//求双连通分量
void dfs(int u,int fa){
    low[u] = pre[u] = ++dfs_clock;
```

```
    int cl =0;
    for(auto v : G[u]){
        Pair e = mp(u,v);
        if(!pre[v]){
            ++cl;
            S.push(e);
            dfs(v,u);
            low[u] = min(low[u],low[v]);
            if(low[v]>=pre[u]){
                cut[u]++;//割点
                ++bcc_cnt;bcc[bcc_cnt].clear();
                while (true) {
                    Pair x = S.top();S.pop();
                    if(bcc_no[x.fi] != bcc_cnt){bcc[bcc_cnt].pb(x.fi);b
cc_no[x.fi] = bcc_cnt;}
                    if(bcc_no[x.se]!= bcc_cnt){bcc[bcc_cnt].pb(x.se);bc
c_no[x.se] = bcc_cnt;}
                    if(x == e)break;
                }
            }
        }else if(pre[v]<pre[u] && v!=fa){
            S.push(e);
            low[u] = min(low[u],pre[v]);
        }
    }
    if(fa<0 && cl==1)cut[u] = 0;//根节点
}

void get_bcc(int n){
    for(int i=0 ; i<n ; ++i){
        if(!pre[i])dfs(i,-1);
    }
}
```

强联通分量分解

```
std::vector<int> G[MAX_V];
std::vector<int> rG[MAX_V];
bool vis[MAX_V];
int scc[MAX_V];
std::vector<int> vs;//访问结束时间栈
void add_edge(int u,int v){
  G[u].push_back(v);
  G[v].push_back(u);
}
void dfs(int u) {
```

```
  vis[u] = true;
  for(int i=0 ; i<G[u].size() ;++i)
    if(!vis[G[u][i]])dfs(G[u][i]);
  vs.push_back(u);
}

void rdfs(int u,int scc_cnt) {
  vis[u] = true;
  scc[u] = scc_cnt;
  for(int i=0 ; i<rG[u].size() ; ++i)
    if(!vis[rG[u][i]])rdfs(rG[u][i]);
}

int Kosaraju(int nv){
  int scc_cnt = 0;
  memset(vis,false,sizeof(vis));
  vs.clear();
  for(int i=0 ; i<v ; ++i)
    if(!vis[i])dfs(i);
  memset(vis,false,sizeof(vis));
  for(int i= vs.size()-1 ; i>=0 ; --i)
    if(!vis[vs[i]])rdfs(vs[i],++scc_cnt);
  return scc_cnt;
}
```

强联通分量 tarjan 版

一定有 scc[i]>scc[j]则缩点之后一定有 i->j

```
std::vector<int> G[maxn];

int pre[maxn],low[maxn];
int dfs_clock;
int scc[maxn],scc_cnt;

stack<int> S;//辅助栈

void dfs(int u) {
    low[u] = pre[u] = ++dfs_clock;
    S.push(u);
    for(auto v:G[u]){
        if(!pre[v]){
            //未访问
            dfs(v);
            low[u] = min(low[v],low[u]);
        }else if(!scc[v])low[u] = min(low[v],low[u]);
    }

    //计算出 low 值之后看是否满足起始条件
```

```
    if(low[u] == pre[u]){
        //标记
        scc_cnt++;
        while (true) {
            int v = S.top();S.pop();
            scc[v] = scc_cnt;
            if(v == u)break;
        }
    }
}
```

网络流

Dinic

```
struct Edge{
  int from,to,cap;
  Edge(int u,int v,int c = 0):from(u),to(v),cap(c){};
};

//残量网络
void add_edge(int u,int v,int cap){
  E.push_back(Egde(u,v,cap));G[u].push_back(E.size()-1);
  E.push_back(Edge(v,u,0));  G[v].push_back(E.size()-1);
}

struct Dinic{
  std::vector<Edge> E;
  std::vector<int> G[MAX_V];
  int level[MAX_V],cur[MAX_V];//分层，当前弧；
  void bfs(int s){
    memset(level,-1,sizeof(level));
    queue<int> Q;Q.push(s);
    level[s] = 0;
    while (!Q.empty()) {
      int u = Q.front();Q.pop();
      for(int i=0 ; i<G[u].size() ; ++i){
        Edge & e = E[G[u][i]];
        if(e.cap>0 && level[e.to]<0){
          level[e.to] = level[u]+1;
          Q.push(e.to);
        }
      }
    }
  }

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

```
    if(v==t || f == 0)return f;
    for(int& i = cur[v] ; i<G[v].size() ; ++i){
      Edge & e = E[G[v][i]];Edge & rev = E[G[v][i]^1];
      if(e.cap>0 && level[v]<level[e.to]){
        int a = dfs(e.to,t,min(f,e.cap));
        if(a>0){
          e.cap-=a;
          rev.cap+=a;
          return a;
        }
      }
    }
    return 0;
  }

  int max_flow(int s,int t){
    int flow = 0;
    for(;;){
      bfs(s);
      if(level[t]<0)break;
      memset(cur,0,sizeof(cur));
      int f;
      while ((f = dfs(s,t,INF))>0) {
        flow+=f;
      }
    }
    return flow;
  }
}
```

对于上面的建边的方式,e 的反向边就是 e^1,这个可以自行枚举证明.

最大流最小切割定理

以下三个条件等价: 1. f 是 G 的最大流. 2. 残存网络中没有增广路 3. $|f| = c(S,T)$ 其中(S,T)是最小切割

切割

G的某个切割 $\{S,T\}$ 是指，将图分为两个不相交的集合，$\{S,T\}$ 的容量为，从 S d 到 T 的最大容量，即割边的最大容量

上下界网络流

区域赛在急，看看了上下界网络流,引水思源，这里给个出处　上下界网络流 https://www.cnblogs.com/liu-runda/p/6262832.html　这里只简单记录建图方法

1. 无源汇可行流

建图

1. u->v upper-lower 在源边上建容量上限为 upper-lower 的边
2. 计算每条边的$totflow[u] = \sum in[u] - \sum out[u]$
3. 对于每个点 $totflow[u] > 0, s -> u, totflow[u]$,反之建 $i -> t, -totflow[i]$
4. 跑$s - t$ 最大流，如果最大流等于总需要流入的流量($\sum^{totflow[u]>0} t\ otflow[u]$) 说明可行.
5. 每条边的实际流量为 low[i] + s-t 最大流跑完后边的流量

例题 zoj 2314

```
#include <bits/stdc++.h>
using namespace std;
#define ms(x,v) (memset((x),(v),sizeof(x)))
#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define INF 0x3f3f3f3f
typedef long long LL;
typedef pair<int,int > Pair;
const int MAX_V = 200+10;
const int MAX_E = 1e5+10;
int n,m;

namespace dinic{
    int ne =0,s,t;
    struct Edge{
        int from ,to,cap;
        Edge(int u=0,int v=0,int c =0):from(u),to(v),cap(c){};
    } E[MAX_E<<2];
    std::vector<int> G[MAX_V];
    int level[MAX_V],cur[MAX_V];
    inline void add_edge(int u,int v,int cap) {
        E[ne] = Edge(u,v,cap);G[u].pb(ne++);
        E[ne] = Edge(v,u,0); G[v].pb(ne++);
    }
    void bfs(int s) {
        ms(level,-1);
        queue<int > Q;Q.push(s);
        level[s] =0;
        while (!Q.empty()) {
            int u = Q.front();Q.pop();
            for(int i=0 ; i< G[u].size() ; ++i){
                Edge &e = E[G[u][i]];
                if(e.cap >0 && level[e.to] <0){level[e.to] = level[u]+1
 ; Q.push(e.to);}
            }
        }
```

```
    }
    int dfs(int v,int t,int f){
        if(v ==t || f==0)return f;
        for(int &i = cur[v] ; i<G[v].size() ; ++i){
            Edge & e = E[G[v][i]];Edge & rev = E[G[v][i]^1];
            if(e.cap>0 && level[v] <level[e.to]){
                int a = dfs(e.to,t,min(f,e.cap));
                if(a > 0){e.cap -=a ; rev.cap += a;return a;}
            }
        }
        return 0;
    }
    int max_flow(int s,int t){
        int flow = 0;
        for(;;){
            bfs(s);
            if(level[t] < 0 )break;
            ms(cur,0);
            int f;
            while ((f = dfs(s,t,INF)) > 0)flow += f;
        }
        return flow;
    }
}
using namespace dinic;
int totflow[MAX_V];
int low[MAX_E];
int main(int argc, char const *argv[]) {
    int T;
    scanf("%d",&T);
    while (T--) {
        ne =0;
        ms(totflow,0);
        scanf("%d%d",&n,&m);
        s = 0;t = n+1;
        for(int i=0 ; i<m ; ++i){
            int u,v,c;
            scanf("%d%d%d%d",&u,&v,&low[i],&c);
            add_edge(u,v,c-low[i]);totflow[u] -=low[i];totflow[v] += lo
w[i];
        }
        int sum=0;
        for(int i=1 ; i<=n ; ++i)
            if(totflow[i]>0)add_edge(s,i,totflow[i]),sum += totflow[i];
            else add_edge(i,t,-totflow[i]);
        if(max_flow(s,t) == sum){
            printf("YES\n");
            for(int i=1 ; i<2*m ; i+=2)printf("%d\n",low[(i>>1)]+ E[i].
cap);
```

```
        }else printf("NO\n");
        for(int i=s ; i<=t ; ++i)G[i].clear();
    }
    return 0;
}
```

2. 有源汇上下界可行流.

只需连一条 $t-s$ 容量无限的边就是 上面的模型了.

建图

1. u->v upper-lower 在源边上建容量上限为 upper-lower 的边
2. 计算每条边的$totflow[u]=\sum in[u]-\sum out[u]$
3. 建新的原点，汇点 ss,tt
4. 对于每个点 $totflow[u]>0, ss->u, totflow[u]$,反之建 $i->tt, -totflow[i]$
5. 跑 $ss-tt$ 最大流，判断是否可行,

这个是下面两个的基础

3. 有源汇上下界最大流

建图 > 1. 同上．先求可行流设为 $f=E[t-s].flow$ (残量网络中的容量) > 2. 将新边$(t-s,inf)$中 $s-t$ 的容量设为 0， 这就等价于上面那片 blog 说的重新重 $s-t$ 增广． 得出最大流 $f', f+f'$ 为实际最大流， > 不过对于我的模板来说，其实不必有第一步，因为我们只需在 $ss-tt$ 的残量网络上直接增广就行因此可以 直接求 $s-t$ 的最大流

loj 116

```
#include <bits/stdc++.h>
using namespace std;
#define ms(x,v) (memset((x),(v),sizeof(x)))
#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define INF 0x3f3f3f3f
typedef long long LL;
typedef pair<int,int > Pair;
const int MAX_V = 300;
const int MAX_E = 1e5+10;
int n,m;

namespace dinic{
    int ne =0,s,t;
    struct Edge{
        int from ,to,cap;
```

```
        Edge(int u=0,int v=0,int c =0):from(u),to(v),cap(c){};
    } E[MAX_E<<2];
    std::vector<int> G[MAX_V];
    int level[MAX_V],cur[MAX_V];
    inline void add_edge(int u,int v,int cap) {
        E[ne] = Edge(u,v,cap);G[u].pb(ne++);
        E[ne] = Edge(v,u,0); G[v].pb(ne++);
    }
    void bfs(int s) {
        ms(level,-1);
        queue<int > Q;Q.push(s);
        level[s] =0;
        while (!Q.empty()) {
            int u = Q.front();Q.pop();
            for(int i=0 ; i< G[u].size() ; ++i){
                Edge &e = E[G[u][i]];
                if(e.cap >0 && level[e.to] <0){level[e.to] = level[u]+1
 ; Q.push(e.to);}
            }
        }
    }
    int dfs(int v,int t,int f){
        if(v ==t || f==0)return f;
        for(int &i = cur[v] ; i<G[v].size() ; ++i){
            Edge & e = E[G[v][i]];Edge & rev = E[G[v][i]^1];
            if(e.cap>0 && level[v] <level[e.to]){
                int a = dfs(e.to,t,min(f,e.cap));
                if(a > 0){e.cap -=a ; rev.cap += a;return a;}
            }
        }
        return 0;
    }
    int max_flow(int s,int t){
        int flow = 0;
        for(;;){
            bfs(s);
            if(level[t] < 0 )break;
            ms(cur,0);
            int f;
            while ((f = dfs(s,t,INF)) > 0)flow += f;
        }
        return flow;
    }
}
using namespace dinic;
int totflow[MAX_V];
int low[MAX_E];
int main(int argc, char const *argv[]) {
    scanf("%d%d%d%d",&n,&m,&s,&t );
```

```
    for(int i=0 ; i<m ; ++i){
        int u,v,l,up;
        scanf("%d%d%d%d",&u,&v,&l,&up );
        add_edge(u,v,up-l);
        totflow[u]-=l;totflow[v] += l;low[i] = l;
    }
    int sum =0;
    int ss = 0,tt = n+1;
    for(int i=1 ; i<=n ; ++i)
        if(totflow[i]>0)add_edge(ss,i,totflow[i]),sum+=totflow[i];
        else add_edge(i,tt,-totflow[i]);
    add_edge(t,s,INF);
    if(max_flow(ss,tt) == sum){
        int f = E[ne-1].cap;
        E[ne-1].cap =0;
        printf("%d\n",f+ max_flow(s,t));
        //or printf("%d\n",max_flow(s,t));
    }else printf("please go home to sleep\n");
    return 0;
}
```

5. 有源汇上下界最小流

建图

1. 同上，先求可行流，求出流量 $f = E[t-s].flow$
2. 将$t-s$ 的容量设为 0 重跑$f' = max_flow(t,s)$ 表示回退流量,最小流为 $f - f'$

loj 117

```
#include <bits/stdc++.h>
using namespace std;
#define ms(x,v) (memset((x),(v),sizeof(x)))
#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define INF 0x3f3f3f3f
typedef long long LL;
typedef pair<int,int > Pair;
const int MAX_V = 50000+100;
const int MAX_E = 125000+10;
int n,m;
typedef LL cap_type;
namespace dinic{
    int ne =0,s,t;
    struct Edge{
        int from ,to;
        cap_type cap;
        Edge(int u=0,int v=0,cap_type c =0):from(u),to(v),cap(c){};
```

```
    } E[MAX_E<<2];
    std::vector<int> G[MAX_V];
    int level[MAX_V],cur[MAX_V];
    inline void add_edge(int u,int v,cap_type cap) {
        E[ne] = Edge(u,v,cap);G[u].pb(ne++);
        E[ne] = Edge(v,u,0); G[v].pb(ne++);
    }
    void bfs(int s) {
        ms(level,-1);
        queue<int > Q;Q.push(s);
        level[s] =0;
        while (!Q.empty()) {
            int u = Q.front();Q.pop();
            for(int i=0 ; i< G[u].size() ; ++i){
                Edge &e = E[G[u][i]];
                if(e.cap >0 && level[e.to] <0){level[e.to] = level[u]+1
 ; Q.push(e.to);}
            }
        }
    }
    cap_type dfs(int v,int t,cap_type f){
        if(v ==t || f==0)return f;
        for(int &i = cur[v] ; i<G[v].size() ; ++i){
            Edge & e = E[G[v][i]];Edge & rev = E[G[v][i]^1];
            if(e.cap>0 && level[v] <level[e.to]){
                int a = dfs(e.to,t,min(f,e.cap));
                if(a > 0){e.cap -=a ; rev.cap += a;return a;}
            }
        }
        return 0;
    }
    cap_type max_flow(int s,int t){
        cap_type flow = 0;
        for(;;){
            bfs(s);
            if(level[t] < 0 )break;
            ms(cur,0);
            cap_type f;
            while ((f = dfs(s,t,INF)) > 0)flow += f;
        }
        return flow;
    }
}
using namespace dinic;
int totflow[MAX_V];
int low[MAX_E];
int main(int argc, char const *argv[]) {
    scanf("%d%d%d%d",&n,&m,&s,&t );
    for(int i=0 ; i<m ; ++i){
```

```
        int u,v,l,up;
        scanf("%d%d%d%d",&u,&v,&l,&up );
        add_edge(u,v,up-l);
        totflow[u]-=l;totflow[v] += l;low[i] = l;
    }
    int sum =0;
    int ss = 0,tt = n+1;
    for(int i=1 ; i<=n ; ++i)
        if(totflow[i]>0)add_edge(ss,i,totflow[i]),sum+=totflow[i];
        else add_edge(i,tt,-totflow[i]);
    add_edge(t,s,0x7fffffffff);
    if(max_flow(ss,tt) == sum){
        //std::cout << E[ne].cap << " " << E[ne-2].cap << '\n';
        cap_type f = E[ne-1].cap;
        E[ne-1].cap = 0;
        printf("%lld\n",f - max_flow(t,s));
    }else printf("please go home to sleep\n");
    return 0;
}
```

二分图匹配

二分图的一个匹配是指二分图中的一些没有公共顶点的边集，匹配数就是边集的数目，最大匹配是指，使得这样的边集的数目最大.

算法

当作网络流来处理，将U与V 的边改成从 U流入V 的一条边，其容量为一.对于这个多元多汇问题，我们只需要连一个超级节点 S 到每一个源点，容量为 1，再从每一个汇点连到一个超级汇点 容量也为一，这样这个图的最大流就是最大匹配数.

不过由于是二分图我们可不必真的这样实现

Code

```
int V;
std::vector<int> G[MAX_V];
bool used[MAX_V];
int match[MAX_V];

bool dfs(int u){
  used[u] = true;
  for(int i=0 ; i<G[u].size() ; ++i){
    int v = G[u][i];int w = match[v];
    if(w<0 || !used[w] && dfs(w)){
      match[u] = v;match[v] = u;
      return true;
    }
  }
```

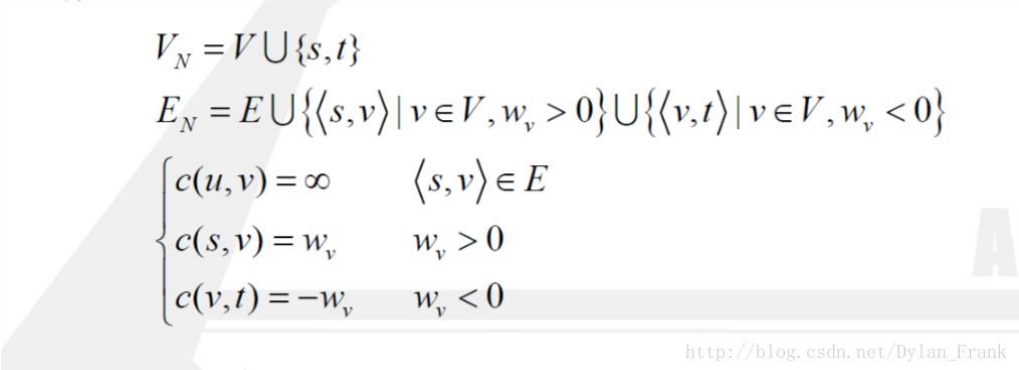
```
  return false;
}

int bipartite_match(){
  int res = 0;
  memset(match,-1,sizeof(match));
  for(int i = 1 ; i<=V ; ++i){
    if(match[i]<0){
      memset(used,false,sizeof(used));
      if(dfs(i))res++;
    }
  }
  return res;
}

void add_edge(int u,int v){
  G[u].push_back(v);
  G[v].push_back(u);
}
```

最大权闭合图

参见文献: 最小割模型在比赛中的运用



这里写图片描述

从源点搜到的满流边就是解

这是最大权闭合子图问题，参见以上文献. 建边， s->i if b[i]>0; i->t if b[i]<0; i->j if j 是 i 的下属 费用就等于 $\sum^{b[i]>0} b\,[i] - (max - flow)$ 与S相连接的点为裁掉的人员

```
#define INF64 0x3f3f3f3f3f3f3f3f
using namespace std;

typedef long long LL;
typedef pair<int,int> Pair;

const int maxn = 5000+10;
```

```
const int MAX_V = 5000+10;
struct Edge{
  int from,to;//原图的边
  LL cap;
  Edge(int u,int v,LL c = 0):from(u),to(v),cap(c){};
};

std::vector<Edge> E;
std::vector<int> G[MAX_V];

//残量网络
void add_edge(int u,int v,LL cap){
  E.push_back(Edge(u,v,cap));G[u].push_back(E.size()-1);
  E.push_back(Edge(v,u,0));  G[v].push_back(E.size()-1);
}

struct Dinic{
  int level[MAX_V],cur[MAX_V];//分层，当前弧；
  void bfs(int s){
    memset(level,-1,sizeof(level));
    queue<int> Q;Q.push(s);
    level[s] = 0;
    while (!Q.empty()) {
      int u = Q.front();Q.pop();
      for(int i=0 ; i<G[u].size() ; ++i){
        Edge & e = E[G[u][i]];
        if(e.cap>0 && level[e.to]<0){
          level[e.to] = level[u]+1;
          Q.push(e.to);
        }
      }
    }
  }

  LL dfs(int v,int t,LL f){
    if(v==t || f == 0)return f;
    for(int& i = cur[v] ; i<G[v].size() ; ++i){
      Edge & e = E[G[v][i]];Edge & rev = E[G[v][i]^1];
      if(e.cap>0 && level[v]<level[e.to]){
        LL a = dfs(e.to,t,min(f,e.cap));
        if(a>0){
          e.cap-=a;
          rev.cap+=a;
          return a;
        }
      }
    }
    return 0;
  }
```

```
  LL max_flow(int s,int t){
    LL flow = 0;
    for(;;){
      bfs(s);
      if(level[t]<0)break;
      memset(cur,0,sizeof(cur));
      LL f;
      while ((f = dfs(s,t,INF64))>0) {
        flow+=f;
      }
    }
    return flow;
  }
};

LL b[maxn];

Dinic Flow;

bool mark[MAX_V];
int cnt =0;
void dfs(int s){
  mark[s] = true;cnt++;
  for(int i=0 ; i<G[s].size() ; ++i){
    Edge & e = E[G[s][i]];
    if( e.cap>0&& !mark[e.to])dfs(e.to);
  }
}


int main() {

  int n,m;
  scanf("%d%d",&n,&m );
    for(int i=1 ; i<=n ; ++i)scanf("%lld",&b[i] );
    while (m--) {
      int x,y;
      scanf("%d%d",&x,&y );
      add_edge(x,y,INF64);
    }

    int s = 0,t = n+1;
    LL sum = 0;
    for(int i=1 ; i<=n ; ++i)
    {
        if(b[i]>0){add_edge(s,i,b[i]);sum+=b[i];}
        else add_edge(i,t,-b[i]);
    }
```

```
    LL profit = sum-Flow.max_flow(s,t);

    memset(mark,false,sizeof(mark));
    cnt = 0;
    dfs(s);


    printf("%d %lld\n",--cnt,profit );


  return 0;
}
```

参考文献 * 《最小割模型在信息学竞赛中的应用》

分数规划

分数规划问题

$$\text{Minimize} \quad \lambda = f(\mathbf{x}) = \frac{a(\mathbf{x})}{b(\mathbf{x})} \quad (\mathbf{x} \in S)$$

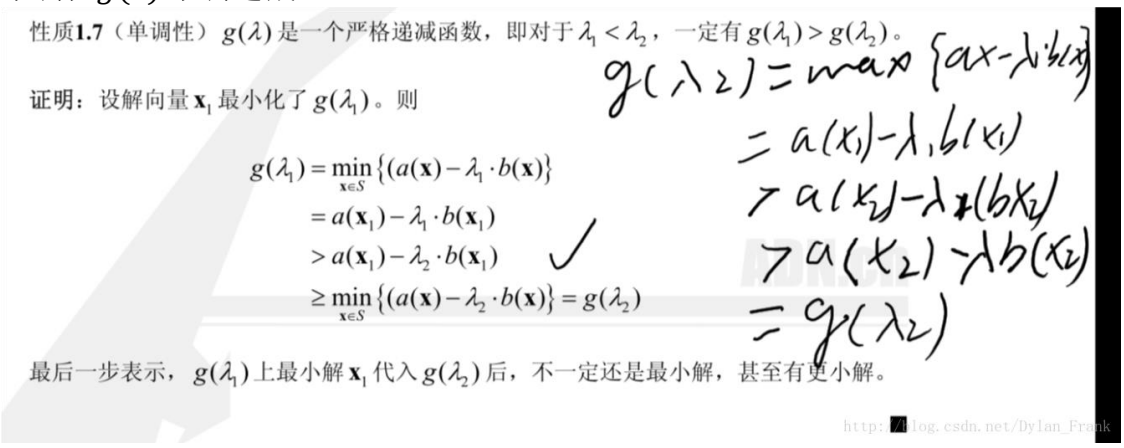
$$s.t. \quad \forall \mathbf{x} \in S, \quad b(\mathbf{x}) > 0$$

分数规划中有几个重要的结论，详细的证明过程不在给出,请参考论文

原问题等价于 $g(\lambda) = min\{a(X) - \lambda b(X)\}$

重要性质

1. $g(\lambda') = 0 \Leftrightarrow \lambda'$*为原分数规划的最优解*
2. 单调性$g(\lambda)$ 单调递减



性质1.7（单调性） $g(\lambda)$ 是一个严格递减函数，即对于 $\lambda_1 < \lambda_2$，一定有 $g(\lambda_1) > g(\lambda_2)$。

证明：设解向量 $\mathbf{x}_1$ 最小化了 $g(\lambda_1)$。则

$$\begin{aligned} g(\lambda_1) &= \min_{\mathbf{x}\in S}\{(a(\mathbf{x}) - \lambda_1 \cdot b(\mathbf{x})\} \\ &= a(\mathbf{x}_1) - \lambda_1 \cdot b(\mathbf{x}_1) \\ &> a(\mathbf{x}_1) - \lambda_2 \cdot b(\mathbf{x}_1) \\ &\geq \min_{\mathbf{x}\in S}\{(a(\mathbf{x}) - \lambda_2 \cdot b(\mathbf{x})\} = g(\lambda_2) \end{aligned}$$

最后一步表示，$g(\lambda_1)$ 上最小解 $\mathbf{x}_1$ 代入 $g(\lambda_2)$ 后，不一定还是最小解，甚至有更小解。

$g(\lambda_2) = \max\{ax - \lambda_2 b(x)\}$
$= a(x_1) - \lambda_2 b(x_1)$
$> a(x_2) - \lambda_2 b(x_2)$
$\geq a(x_2) - \lambda_1 b(x_2)$
$= g(\lambda_1)$

更一般的我们有

推论1.9 设λ^*为该规划的最优解，则

$$\begin{cases} g(\lambda)=0 \Leftrightarrow \lambda=\lambda^* \\ g(\lambda)<0 \Leftrightarrow \lambda>\lambda^* \\ g(\lambda)>0 \Leftrightarrow \lambda<\lambda^* \end{cases}$$

这个对于最大化分数规划也是同样试用的.证略

最大密度子图

最大密度子图可以转化为分数规划问题

1. 边权为 1

建图

$$V_N = V \cup \{s,t\}$$

$$E_N = \{\langle u,v\rangle \mid (u,v)\in E\} \cup \{\langle s,v\rangle \mid v\in V\} \cup \{\langle v,t\rangle \mid v\in V\}$$

$$\begin{cases} c(u,v)=1 & (u,v)\in E \\ c(s,v)=U & v\in V \\ c(v,t)=U+2g-d_v & v\in V \end{cases}$$

$h(g)=\frac{U*n-(min-cut(S,T))}{2}$

二分找g使得$hg=0$复炸度为$O(k*maxflow)$对于详细的证明过程参见论文，我认为没有人可以讲的比他还详尽了

poj 3155code 模板

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
#include <cmath>
#include <cstring>
#include <map>
#include <set>
#include <stack>
#define fi first
#define se second
#define INF 0x3f3f3f3f
#define INF64 0x3f3f3f3f3f3f3f3f
using namespace std;
```

```
typedef long long LL;
typedef pair<int,int> Pair;
const double eps = 1e-8;

const int maxn = 1000+10;
const int MAX_V = 100+10;
const int MAX_E = 1000+10;
struct Edge{
  int from,to;//原图的边
  double cap;
  Edge(int u,int v,double c = 0):from(u),to(v),cap(c){};
};

std::vector<Edge> E;
std::vector<int> G[MAX_V];
Pair p[MAX_E];
int d[MAX_V];
int n,m,s,t;
//残量网络
void add_edge(int u,int v,double cap){
  E.push_back(Edge(u,v,cap));G[u].push_back(E.size()-1);
  E.push_back(Edge(v,u,0));  G[v].push_back(E.size()-1);
}

struct Dinic{
  int level[MAX_V],cur[MAX_V];//分层，当前弧；
  void bfs(int s){
    memset(level,-1,sizeof(level));
    queue<int> Q;Q.push(s);
    level[s] = 0;
    while (!Q.empty()) {
      int u = Q.front();Q.pop();
      for(int i=0 ; i<G[u].size() ; ++i){
        Edge & e = E[G[u][i]];
        if(e.cap>eps && level[e.to]<0){
          level[e.to] = level[u]+1;
          Q.push(e.to);
        }
      }
    }
  }

  double dfs(int v,int t,double f){
    if(v==t || f == 0)return f;
    for(int& i = cur[v] ; i<G[v].size() ; ++i){
      Edge & e = E[G[v][i]];Edge & rev = E[G[v][i]^1];
      if(e.cap>eps && level[v]<level[e.to]){
        double a = dfs(e.to,t,min(f,e.cap));
        if(a>0){
```

```
          e.cap-=a;
          rev.cap+=a;
          return a;
        }
      }
    }
    return 0;
  }

  double max_flow(int s,int t){
    double flow = 0;
    for(;;){
      bfs(s);
      if(level[t]<0)break;
      memset(cur,0,sizeof(cur));
      double f;
      while ((f = dfs(s,t,INF64))>0) {
        flow+=f;
      }
    }
    return flow;
  }
};

Dinic Flow;

bool mark[MAX_V];
int cnt =0;
set<int> vertex;

void dfs(int s){
  mark[s] = true;cnt++;
  vertex.insert(s);
  for(int i=0 ; i<G[s].size() ; ++i){
    Edge & e = E[G[s][i]];
    if( e.cap>eps&& !mark[e.to])dfs(e.to);
  }
}



void build_graph(double g){
  for(int i=0 ; i<=n+1 ; ++i)G[i].clear();
  E.clear();
  for(int i=1 ; i<=n ; ++i){
    add_edge(s,i,m);
    add_edge(i,t,m+2*g-d[i]);
  }
  for(int i=0 ; i<m; ++i)
```

```
  {
    add_edge(p[i].fi,p[i].se,1.0);
    add_edge(p[i].se,p[i].fi,1.0);
  }
}

int main() {


  scanf("%d%d",&n,&m );
   s = 0,t = n+1;
   memset(d,0,sizeof(d));
  for(int i=0 ; i<m ; ++i){
    scanf("%d%d",&p[i].fi,&p[i].se );
    d[p[i].fi]++;d[p[i].se]++;
  }
  if(m==0){std::cout << 1 << '\n'<<1<<'\n';}
  else{
    double left = 0,right = m,mid,hg;
    double precise = 1.0/n/n;
    while (right-left>=precise) {
      mid = (right+left)/2;
      build_graph(mid);
      hg = ((double)m*n-Flow.max_flow(s,t))/2;
      (hg>eps?left:right) = mid;
    }
    build_graph(left);
    Flow.max_flow(s,t);
    cnt =0;
    memset(mark,false,sizeof(mark));
    dfs(s);
    printf("%d\n", --cnt);
    for(set<int>:: iterator it = ++vertex.begin() ; it!=vertex.end() ;
++it)
      printf("%d\n",*it );
  }

  return 0;
}
```

2. 边带权图

只需

$$ d_u = \sum_{(u,v)\in E}w_e\\ U = \sum_{e}w_e\\ c(u,v) = w_e $$

3. 点边均带权的图

$$ d_u = \sum_{(u,v)\in E}w_e\\ c(u,v) = w_e\\ U = \sum w_e+2\sum p_v\\ c(u,t) = U+2g-2p_u+d_u $$

最小费用流

在概念上最小费用流只是在最大流的边上在附加一个费用，即求出从源点到汇点的给定流量的最小费用.

算法

先从源点找一条到汇点的最短路，然后沿着最短路增广.建图的时候将反向边的费用设为-cost.(退流退费用)

Code

```
struct Edge{
  int from,to,cap,cost;
  Edge(int f,int t,int c,int co):from(f),to(t),cap(c),cost(co){}
};
std::vector<Edge> E;
std::vector<int> G[MAX_V];
void add_edge(int u,int v,int cap,int cost){
  E.push_back(Edge(u,v,cap,cost));G[u].push_back(E.size()-1);
  E.push_back(Edge(v,u,0,-cost)) ;G[v].push_back(E.size()-1);
}
struct MCMF{
  int V;
  int dist[MAX_V];
  int pre_E[MAX_V];//最短路径弧
  bool inq[MAX_V];//spfa 判断
  //未判断负圈
  void spfa(int s){
    memset(dist,INF,sizeof(dist));
    memset(inq,false,sizeof(inq));
    queue<int> Q;
    Q.push(s);
    dist[s] = 0;
    inq[s] = true;
    pre_E[s] = -1;
    while(!Q.empty())
    {
      int u = Q.front();Q.pop();
      inq[u] = false;
      for(int i=0 ; i<G[u].size() ; ++i){
        Edge &e = E[G[u][i]];
        if(e.cap>0&&dist[e.to]>dist[u]+e.cost){
          dist[e.to] = dist[u]+e.cost;
          pre_E[e.to] = G[u][i];
          if(!inq[e.to]){Q.push(e.to);inq[e.to] = true;}
        }
      }
```

```
    }
  }

  int min_cost_flow(int s,int t,int f){
    int res = 0;
    while (f>0) {
      spfa(s);
      if(dist[t]==INF)return break;//不能增广
      //沿着最短路增广
      int d = f;
      for(int i = pre_E[t] ; i!=-1 ;i = pre_E[E[i].from])d = min(d,E[i].
cap);
      f-=d;
      res+=d*dist[t];
      for(int i = pre_E[t] ; i!=-1 ;i = pre_E[E[i].from]){
        E[i].cap-=d;
        E[i^1].cap+=d;
      }
    }
    return res;
  }
};
```

树

LCA

```
int st[MAX_V][32];
//返回下标的RMQ;
void  RMQ_init(const int *A,int n)
{
  for(int i=0 ; i<n ; ++i)st[i][0] = i;

  for(int j = 1 ; (1<<j)<=n ; ++j)
    for(int i =0 ; i+(1<<j)-1<n ; ++i)
    if(A[st[i][j-1]]<A[st[i+1<<(j-1)][j-1]])
      st[i][j] = st[i][j-1];
    else st[i][j] = st[i+1<<(j-1)][j-1];
}
//最小值的下标
int query(const int *A,int L,int R){
  if(R<L)return -1;
  int k=0;
  while(1<<(k+1)<=R-L+1)k++;
  if(A[st[L][k]]<A[st[R-(1<<k)+1][k]])
    return st[L][k];
  else return st[R-(1<<k)+1][k];
}
```

```
//LCA
std::vector<int> G[MAX_V];

struct LCA{
  int root;
  int vs[MAX_V<<1];//访问数组
  int depth[MAX_V<<1]//深度数组
  int pos[MAX_V];//位置数组

  void dfs(int v,int p,int d,int& k){
    pos[v] = k;
    vs[k] = v;
    depth[k++] = d;
    for(int i=0 ; i<G[v].size() ; ++i)
      if(G[v][i] !=p){
        dfs(G[v][i],v,d+1,k);
        vs[k] = v;//回退
        depth[k++] = d;
      }
  }

  //预处理
  void init(int V,int r){
    int k=0 ;
    root = r;
    dfs(root,-1,0,k);
    RMQ_init(depth,2*V-1);
  }

  int lca(int u,int v){
    if(pos[u]>pos[v])swap(u,v);
    return vs[query(depth,pos[u],pos[v])];
  }
};
```

clique

版本 1

```
#define ctz(x) ((x)? __builtin_ctzll(x):64)

/*计算右边第一个1之后的0的个数
*最大计算为64,从右边起序号依次为0,1,2,...
*/
int ans,n;
ULL g[maxn];
//最大团
```

```
void BronKerbosch(ULL clique,ULL allow,ULL forbid){
    if(!allow && !forbid){
        ans = max(ans,__builtin_popcountll(clique));return;
    }
    if(!allow)return;
    int pivot = ctz(allow | forbid);//选择轴点
    ULL choose = allow & ~g[pivot];
    for(int u = ctz(choose) ; u<n ; u += ctz(choose>>(u+1))+1){
        BronKerbosch(clique|(1ULL<<u),allow&g[u],forbid&g[u]);
        allow ^=1ULL<<u;
        forbid|=1ULL<<u;
    }
}
```

版本 2

```
const int MAX_V = 55;
bitset<MAX_V> g[MAX_V],clique,allow,forbid;


//暴力枚举
int ans,n;
void BronKerbosch(int sz,bitset<MAX_V> allow, bitset<MAX_V> forbid,int
begin=0) {
    if(allow.none()&&forbid.none()){
        ans = max(ans,sz);
        return;
    }
    if(allow.none())return;
    int pivot = 0;
    for(int i=0 ; i<n ; ++i)
        if(allow[i]|forbid[i]){pivot = i;break;}
    bitset<MAX_V> choose = allow & (~g[pivot]);
    for(int u=begin ; u<n ; ++u){
        if(choose[u]){
            sz++;
            BronKerbosch(sz,allow&g[u],forbid&g[u],begin=0);
            sz--;
            allow.set(u,0);
            forbid.set(u);
        }
    }
}
```

版本 3

```
std::vector<int> g[MAX_V];

int mat[maxn][maxn];
//暴力枚举
```

```
int ans = 0;
int n,m,s;
int cnt;
int Stack[maxn];
bool ok(int u){
    for(int i=0 ; i<cnt ; ++i)
        if(!mat[u][Stack[i]])return 0;
    return 1;
}
void dfs(int u,int idx){
    if(cnt ==s){ans++;return;}
    if(g[u].size()+cnt <s|| idx >=g[u].size())return;
    if(ok(g[u][idx])){
        Stack[cnt++] = g[u][idx];
        dfs(u,idx+1);
        cnt--;
    }
    dfs(u,idx+1);
}

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(0);
    // cout.tie(0);

    int T;
    cin>>T;

    int allow[100];
    while (T--) {
        cin>>n>>m>>s;
        for(int i=0 ; i<n ; ++i)g[i].clear();
        ms(mat,0);
        for(int i=0 ; i<m ; ++i){
            int u,v;
            scanf("%d%d",&u,&v );
            u--;v--;
            if(u==v)continue;
            mat[u][v] = mat[v][u] =1;
            g[u].pb(v);
            g[v].pb(u);
        }
        ans =0;
        for(int i=0 ; i<n ; ++i){
            cnt =0;
            Stack[cnt++] = i;
            dfs(i,0);
            for(int j=0 ; j<n ; ++j){
```

```
                if(mat[i][j])mat[i][j]=mat[j][i] = 0;
            }
        }
        std::cout << ans << '\n';
    }
    return 0;
}
```

最小费用流

```
#define fi first
#define se second

using namespace std;

typedef long long LL;
typedef pair<int,int> Pair;

const int maxn = 50;
const int MAX_V = 2500;

struct Edge{
  int from,to,cap,cost;
  Edge(int f,int t,int c,int co):from(f),to(t),cap(c),cost(co){}
};
std::vector<Edge> E;
std::vector<int> G[MAX_V];
void add_edge(int u,int v,int cap,int cost){
  E.push_back(Edge(u,v,cap,cost));G[u].push_back(E.size()-1);
  E.push_back(Edge(v,u,0,-cost)) ;G[v].push_back(E.size()-1);
}
struct MCMF{
  int V;
  int dist[MAX_V];
  int pre_E[MAX_V];//最短路径弧
  bool inq[MAX_V];//spfa 判断
  //未判断负圈
  void spfa(int s){
    memset(dist,INF,sizeof(dist));
    memset(inq,false,sizeof(inq));
    queue<int> Q;
    Q.push(s);
    dist[s] = 0;
    inq[s] = true;
    pre_E[s] = -1;
    while(!Q.empty())
    {
      int u = Q.front();Q.pop();
```

```
      inq[u] = false;
      for(int i=0 ; i<G[u].size() ; ++i){
        Edge &e = E[G[u][i]];
        if(e.cap>0&&dist[e.to]>dist[u]+e.cost){
          dist[e.to] = dist[u]+e.cost;
          pre_E[e.to] = G[u][i];
          if(!inq[e.to]){Q.push(e.to);inq[e.to] = true;}
        }
      }
    }
  }

  int min_cost_flow(int s,int t,int f){
    int res = 0;
    while (f>0) {
      spfa(s);
      if(dist[t]==INF) break;//不能增广
      //沿着最短路增广
      int d = f;
      for(int i = pre_E[t] ; i!=-1 ;i = pre_E[E[i].from])d = min(d,E[i].
cap);
      f-=d;
      res+=d*dist[t];
      for(int i = pre_E[t] ; i!=-1 ;i = pre_E[E[i].from]){
        E[i].cap-=d;
        E[i^1].cap+=d;
      }
    }
    return res;
  }
};

MCMF mcf;

int a[maxn][maxn];
int main(int argc, char const *argv[]) {
  int N,M;
  while (scanf("%d%d",&N,&K )!=EOF) {
    for(int i=0 ; i<N ; ++i)
      for(int j=0 ; j<M ; ++j)
        scanf("%d",&a[i][j] );
    int ans  = a[0][0]+a[N-1][M-1];
    int s = N*M,t = s+1;
    E.clear();
    for(int i=0 ; i<= t  ; ++i)G[i].clear();
    for(int i=0 ; i<N ; ++i){
      for(int j = 0 ; j<M ; ++j){
        int u = i*M+j,v1 = u+1,v2 = (i+1)*N+j;
        if(j!=M-1){
```

```
            add_edge(u,v1,1,-a[i][j+1]);
            add_edge(u,v1,INF,0);
          }
          if(i!=N-1){
            add_edge(u,v2,1,-a[i+1][j]);
            add_edge(u,v2,INF,0);
          }
        }
      }
      add_edge(N*M-1,t,K,0);
      add_edge(s,0,K,0);
      ans+=(-mcf.min_cost_flow(s,t,K));
      std::cout << ans << '\n';
    }


    return 0;
}
```

数学

计数

BEST 定理 和这里不加证明的给出 BEST 定理，有向图$G, d_i = d_i^- = d_i^+$的欧拉回路数目为

$$T_v * \prod^{i \in V(G)} (d_i - 1)!$$

i 其中T_v 为任意顶点v 的 in_tree 或者 out_tree 数目

注意： d 定理中的欧拉回路和这个题里的不一样，定理中由于欧拉回路有环结构，与第一条边选取无关，而这个问题是算作不同的.

mobius

基本公式定理

$$n = \sum_{d|n} \phi(d)$$

$$\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$$

1. mobius 反演公式

$$
\begin{aligned}
F(n) &= \sum_{d|n} f(d) \\
f(n) &= \sum_{d|n} \mu(n/d) F(d) \\
&= \sum_{d|n} \mu(d) F(n/d)
\end{aligned}
$$

另一种描述:

```
$$
\begin{align}
F(d)&=\sum_{d\mid n}f(n)\\
f(d)&=\sum_{d\mid n}F(n)\mu(n/d)
\end{align}
$$
```

2. 经典公式

$$
\begin{aligned}
n &= \sum_{d|n} \phi(d) \\
\phi(n) &= \sum_{d|n} \mu(d) \frac{n}{d} \\
\sum_{d|n} \mu(d) &= (n == d)
\end{aligned}
$$

分块求和

如果说计算式中出现了 $\sum^{i} f(i) * g(\lfloor (n/i) \rfloor)$,则由于 $\lfloor \frac{n}{i} \rfloor$的取值只有 $O(\sqrt{n})$ 种显然我们可以运用分段求和(可以打印出这样的值来看一下)记录f的前缀和，然后 g 就进行分段求和.

```
ll F(int n, int m, int d) {
    if (n > m) swap(n, m);
    ll ans = 0;
    n /= d, m /= d;
    for (int i = 1, last = 1; i <= n; i = last + 1) {
        last = min(n / (n / i), m / (m / i));
        ans += (ll)(sum[last] - sum[i - 1]) * (n / i) * (m / i);
    }
    return ans;
}
```

线性筛法笔记整理

线性筛法处理积性函数

```
void monius(){
    cnt =0;
    mu[1] = 1;
    memset(prime,0,sizeof(prime));
    for(int i = 2 ; i<maxn ; ++i){
        if(!prime[i]){
            prime[cnt++] = i;
            mu[i] =-1;
        }
        for(int j=0 ; j<cnt && i*prime[j]<maxn ; ++j){
            prime[i*prime[j]] = 1;
            if(i%prime[j])mu[prime[j]*i] = -mu[i];
            else {
                mu[i*prime[j]] = 0;
                break;
            }
        }
    }
    sum_mu[0] = 0;
    for(int i=1 ; i<maxn ; ++i)
        sum_mu[i] = sum_mu[i-1]+mu[i];
}

void phi_table(){
    cnt =0;
    phi[1] = 0;
    memset(prime,0,sizeof(prime));
    for(int i = 2 ; i<maxn ; ++i){
        if(!prime[i]){
            prime[cnt++] = i;
            phi[i] =i-1;
        }
        for(int j=0 ; j<cnt && i*prime[j]<maxn ; ++j){
            prime[i*prime[j]] = 1;
            if(i%prime[j])phi[prime[j]*i] = phi[i]*(prime[j]-1);
            else {
                phi[i*prime[j]]= phi[i]*prime[j];
                break;
            }
        }
    }
}
```

参考文献:

贾智鹏 线性筛

ex_gcd

```
//返回值为最大公约数
LL ex_gcd(LL a,LL b,LL &x,LL &y)
{
    LL d = a;
    if(!b){x = 1,y = 0;}
    else{
      d = ex_gcd(b,a%b,y,x);
      y-=a/b*x;
    }
    return d;
}
```

素数

素因子分解

朴素算法$O(\sqrt{n})$

```
void prime_factor(int n,map<int,int> &pf)//
{
    for(int i =2 ; i*i<=n ; ++i)//n 为素数时!
    {
        while(n%i==0)
        {
            ++pf[i];
            n/=i;
        }
    }
    if(n!=1)pf[n] = 1;
}
```

Eratosthenes 筛法

```
void Eratosthenes(int n)
{
    memset(is_prime,true,sizeof(is_prime));

    for(int i = 2 ; i*i<=n; ++i)
        if(is_prime[i])
        for(int j=i*i ; j<=n ; j+=i)is_prime[j] = false;
}
```

区间筛法

```
void segment_sieve(LL a,LL b)//[a,b]
{
    memset(is_prime_ab,true,sizeof(is_prime_ab[0])*(b-a+1));
    memset(is_prime_sqrtb,true,sizeof(is_prime_sqrtb[0])*(sqrt(b)+2));
    for(LL i = 2 ; i*i<=b ; ++i)
```

```
    if(is_prime_sqrtb[i]){
        for( LL j = i*i ; j*j<=b ; j+=i)is_prime_sqrtb[j] = false;
        for(LL j = max(i*i,(a-1)/i+1)*i ; j<=b ; j+=i)is_prime_ab[j-a]
= false;
    }
}
```

大素数分解与大素数测试

1. miller_rabin

已知最快的素数分解算法.$O(lgV)$

```
bool witness(LL a,LL n,LL u,LL t){
    LL x0 = power_mod(a,u,n),x1;
    for(int i=1 ;i<=t ; ++i){
        x1 = mulmod(x0,x0,n);
        if(x1==1 && x0!=1 && x0!=n-1)
            return false;
        x0 = x1;
    }
    if(x1 !=1)return false;
    return true;
}

bool miller_rabin(LL n, int times = 20){
    if(n<2)return false;
    if(n==2)return true;
    if(!(n&1))return false;
    LL u = n-1,t =0;
    while (u%2==0) {
        t++;u>>=1;
    }
    while (times--) {
        LL a = random(1,n-1);
        //if(a == 0)std::cout << a << " "<<n<< " "<<u<<" " << t<<'\n';
        if(!witness(a,n,u,t))return false;
    }
    return true;
}
```

2. pollard_rho

分解一个合数V的运行时间$O(V^{1/4})$

```
/*
*pollard_rho 分解n,
*c : 随机迭代器，每次运行设置为随机种子往往更快.
*/
LL pollard_rho(LL n,LL c = 1){
```

```
    LL x  = random(1,n);
    LL i =1,k =2,y = x;
    while (1) {
        i++;
        x = (mulmod(x,x,n)+c)%n;
        LL d = gcd(y-x>=0?y-x:x-y,n);
        if(d!=1 && d!=n)return d;//非平凡因子.
        if(y==x)return n;//重复.
        if(i==k){ y = x ; k<<=1;}//将x_1,2,4,8,16,..赋值为y.
    }
}
```

- 找出因子分解 $O(V^{1/4}lgV)$

```
void find_factor(LL n,std::map<LL, int>  & m){
    if(n<=1)return ;
    if(miller_rabin(n)){
        ++m[n];
        return ;
    }
    LL p = n;

    while (p==n)p = pollard_rho(p,random(1,n));
    find_factor(p,m);
    find_factor(n/p,m);
}
```

euler phi 函数

$\phi(n) = n\prod_{i=1}^{k}(1-\frac{1}{p_i})$ 证明详见《初等数论及其应用》

```
int  euler_phi(int  n)
{
    int  ans = n;
    for(int i=2 ; i*i<=n ; ++i)
        if(ans%i ==0)
        {
            ans = ans/i*(i-1);
            while(n%i==0)n/=i;
        }
        if(n>1)ans = ans/n*(n-1);
    return ans;
}
```

模运算

大数乘法取模

```
inline ll mul_mod_ll(ll a,ll b){
   ll d=(ll)floor(a*(double)b/MOL+0.5);
```

```
    ll ret=a*b-d*MOL;
    if(ret<0)ret+=MOL; return ret;
}
```

模方程

```
LL MLE(LL a,LL b,LL n){
    LL d,x,y;
    d = ex_gcd(a,n,x,y);
    if(b%d !=0){
        return -1;
    }else{
        LL x0 = x*b/d%n+n;
        return x0%(n/d);//模（n/d)
    }
}
```

乘法逆元

a 在模 n 意义下的逆

```
LL inv(LL a,LL n){
    LL x,y;
    LL d = ex_gcd(a,n,x,y);
    return d==1? (x+n)%n:-1;//非负性保证.
}
```

中国剩余定理

```
//x % m[i] = a[i]
LL china(int n,int *a,int *m){
    LL M = 1,x = 0,y,z;
    for(int i=0 ; i<n ; ++i)M*=m[i];
    for(int i=0 ; i<n ; ++i){
        LL M_i = M/m[i];
        ex_gcd(M_i,m[i],y,z);//M_i*y = 1(mod m[i])
        x = (x+M_i*a[i]*y)%M;
    }
    return (x+M)%M;
}
```

朴素模方程(m_i不两两互素的时候)

```
LL MLE(int *r,int *mod,int n){
    LL lm = 0, lb = 1;
    for (int i = 0; i < n; i++)
    {
        LL k1,k2;
        LL d= exgcd(lb, mod[i],k1,k2);  // x=c1(mod r1)
        if ((lm - r[i]) % d) { return -1; } // 联立 x=r2(mod m2)，(r1-r2)
=0(mod gcd)才有解
        lb = lb / d * mod[i];                           // lcm
```

```
        LL z = k2 * ((lm - r[i]) / d);                    // 求出 k2
        lm = z * mod[i] + r[i];
// 得到方程组的一个最小解
        lm = ((lm % lb) + lb) % lb;
// 保证最小解大于0
    }
    return lm;
}
```

离散对数

阶定义

设 $(a,m)=1$, 满足 $a^x \equiv 1(mod\ m)$ 的最小的 x，称为 a 对 m 的阶，记为 $ord_m(a)$ 当 $ord_m(a)=\phi(m)$ 时称为 a 为 m 的原根.

简单性质

1. $a^x \equiv 1 \Leftrightarrow ord_m(a) \mid x$
2. $ord_m(a) \mid \phi(m)$
3. $a^0, a^1, \dots, a^{ord_m(a)-1}$ 构成摸 m 的既约剩余系.
4. $a^x \equiv a^y(mod\ m) \Leftrightarrow x \equiv y(mod\ ord_m(a))$
5. $ord_m(a^d)=\frac{ord_m(a)}{(ord_m(a),d)}$(利用这个性质可以求出所有原根)
6. m(若存在原根)的原根数目为 $\phi(\phi(m))$.
7. n*有原根* $\Leftrightarrow n=2,4,p^e,2p^e$($p$*为奇素数*).
8. 设 $\phi(m)=p_1^{r_1}p_2^{r_2}\dots p_k^{r_k}$,则$g$*是*$m$的原根当且仅当对与所有的$p_i$.

$$g^{\frac{\phi(m)}{p_i}} \neq 1(mod\ m)$$

求阶和原根的方法

上面的性质是非常容易证明的.随便找一本数论书籍都会有详细的证明.由上面的性质我们可以得到一个相对简单的求阶和原根的方法(暴力)

- **阶:** 我门可以对 m 先分解因子，设$m=p_1^{r_1}p_2^{r_2}\dots p_k^{r_k}$，然后将$r_i$逐个相减记为$p$直到再减一个之后$a^p \neq 1$
- **原根:** 这个就更加暴力了，我们可以用性质 8 逐一枚举与m互素的数然后对分解式进行验证就好.

这里有一份 51nod 模板题 1135 的代码

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
```

```cpp
#include<cmath>
#include <cstring>
#include <map>
#include <iomanip>
#define fi first
#define se second
#define INF 0x3f3f3f3f
using namespace std;
const int MOD = 1e9+7;
const int MAX_P = 2e4+10;
const int maxn =2e5+10;
const int MAX_V = 5e5+10;
const int maxv = 1e6+10;
typedef long long LL;
typedef long double DB;
typedef pair<int,int> Pair;
int p;
int prime[maxn],cnt;
int factor[maxn],fact_cnt;
void init_prime(){
    memset(prime,0,sizeof(prime));
    cnt = 0;
    for(int i = 2 ; i<maxn ; ++i){
        if(!prime[i]){
            prime[cnt++] = i;
        }
        for(int j =0 ; j<cnt && prime[j]*i<maxn ; ++j){
            prime[prime[j]*i] = 1;
            if(i % prime[j] == 0)break;
        }
    }
}
LL power_mod(LL x,LL n,LL mod){
    LL res =1;
    while (n) {
        if(n & 1)res = res*x % mod;
        x = x*x % mod;
        n >>= 1;
    }
    return res;
}
void get_fact(int n){
    fact_cnt = 0;
    for(int i = 0 ; i< cnt && prime[i]*prime[i] <=n ; ++i ){
        if(n % prime[i] == 0){
            factor[fact_cnt++] = prime[i];
            while (n%prime[i] == 0)n/=prime[i];
        }
    }
```

```
    if(n!=1)factor[fact_cnt++] = n;
}

bool check(int g){
    for(int i=0 ; i<fact_cnt ; ++i)
        if(power_mod(g,(p-1)/factor[i],p) ==1)return false;
    return true;
}

int proot(int p){
    get_fact(p-1);
    for(int i=2 ; i<p ; ++i)
        if(check(i))return i;
}

int main(int argc, char const *argv[]) {
    init_prime();
    cin>>p;
    std::cout << proot(p) << '\n';
  return 0;
}
```

bsgs algorithm

$A^x \equiv C(mod\ m)$ 大步小步算法，这个算法有一定的局限性，只有当$gcd(a,m) = 1$ *时才可以用.*

原理

简单说一下它的原理.其实由上面的性质，我们知道$0 \le x < ord_m(a)$,设$x = uT - v$ 则原方程转化为$a^{uT} \equiv Ca^v(a^{-1}$*存在*$)$ 我们可以预处理出每一个 Ca^v对每一个计算出的 a^{uT} 找一下有没有等于 Ca^v 的取$(T = \sqrt{m})$复杂度 $O(\sqrt{m}lg(m))$

扩展大步小步

上面的过程能求离散对数的前提是 a^{-1} 存在即若 $gcd(a,m) = 1$ 则大步小步是可用的，现在我们来扩展一下大步小步算法.使得即使对任意的 m 都能求出x或者不存在时返回-1. 由于模方程$ax \equiv b(mod\ m) \Leftrightarrow \frac{a}{d}x \equiv \frac{b}{d}(mod\ \frac{m}{d}), d = gcd(a,m)$(证明见《初等数论及其应用》),我们可以对$a,m,b$ 不断消因子,直到 $(a,m) = 1$ 设共进行 δ 次消因子操作，那么 δ 次操作以后就变成了 $a^{x-\delta}a^{\delta} \equiv b'(mod\ m')$,(若消因子中途遇见 b 不能整除返回-1,若中途遇见b等于a^{δ} , 直接返回 δ).现在就可以用大步小步算法来求解了.详细内容请看代码. 模板题 spoj MOD - Power Modulo Inverted

```
#include <cstdio>
#include <iostream>
#include <vector>
#include <queue>
```

```cpp
#include <algorithm>
#include<cmath>
#include <cstring>
#include <map>
#define fi first
#define se second
#define INF 0x3f3f3f3f
using namespace std;
const int MOD = 1e9+7;
const int MAX_P = 2e4+10;
const int maxn =500+10;
const int MAX_V = 5e5+10;
const int maxv = 1e6+10;
typedef long long LL;
typedef pair<int,int> Pair;

LL power_mod(LL x,LL n, int mod){
   LL res =1;
   while (n) {
      if(n&1)res = res*x % mod;
      x = x*x %mod;
      n >>=1;
   }
   return res;
}

LL bsgs(LL A,LL C,LL mod){
   A %= mod;C %= mod;
   if(C==1)return 0;
   LL cnt =0;
   LL tmp = 1;
   for(LL g = __gcd(A,mod) ; g != 1 ; g = __gcd(A,mod)){
      if(C % g)return -1;//不能整除
      C /=g ; mod/=g ; tmp = tmp*A/g%mod;
      ++cnt;
      if(C == tmp)return cnt;
   }
   //大步小步a^xa^cnt=C (mod m)a^cnt = tmp;
   LL T = (LL)sqrt(0.5+mod);
   LL b = C;
   map<LL,LL> hash;
   hash[b] = 0;
   for(int i=1 ; i<=T ; ++i){
      b = b*A%mod;//当mod 为LL 时注意溢出
      hash[b] = i;
   }
   A = power_mod(A,T,mod);
   for(int u =1 ; u<=T ; ++u){
      tmp = tmp*A %mod;
```

```
        if(hash.count(tmp))return u*T-hash[tmp]+cnt;
    }
    return -1;
}

int main(int argc, char const *argv[]) {
    LL x,y,z,k;
    while (scanf("%lld%lld%lld",&x,&z,&k ) && z) {
        y = bsgs(x,k,z);
        if(y==-1)std::cout << "No Solution" << '\n';
        else std::cout << y << '\n';
    }
    return 0;
}
```

积性函数

题目链接 Count a * b

分析

这是很有意思的积性函数问题

反过来定义

$$\begin{aligned} h(m) &= m^2 - f(m) \\ &= \sum_{m}^{a,b} [a*b\%m = 0] \\ &= \sum_{a=1,b}^{m} g\,cd(a,m)|b \\ &= \sum_{a=1}^{m} g\,cd(a,m) \\ &= \sum_{d|m} d*\phi(m/d) \end{aligned}$$

即$h(m)$ 是恒等映射与 Euler 函数的狄利克雷卷积，所以它是积性函数 对于素数p
$h(p^k) = \sum_{i=0}^{k} p^i * \phi(p^{k-i}) = p^k + \sum_{i=0}^{k-1} p^i\,(p^{k-i} - p^{k-i-1}) = p^{k-1}(k*(p-1)+p)$

所以

$$g(n) = \sum_{d|n} d^2 - h(d)$$

分别求

$$ g1(n) =\sum_{d|n}d^2\\ g2(n)=\sum_{d|n}h(d) $$

$g1, g2$均是 1 和积性函数的狄利克雷卷积，所以他们都是积性函数.以$g2$举例$n = p_1^{r1} \dots p_k^{r_k}$

$$\begin{aligned} g2(n) &= \prod_{i=1}^{k} g\,2(p_i^r) \\ &= \prod_{i=1}^{k}(\sum_{j=0}^{r_i} h\,(p_i^j)) \end{aligned}$$

AC code

```
//Problem : 5528 ( Count a * b )     Judge Status : Accepted
//RunId : 22241701    Language : G++    Author : zouzhitao
//Code Render Status : Rendered By HDOJ G++ Code Render Version 0.01 Be
ta
#include<bits/stdc++.h>
#define pb push_back
#define mp make_pair
#define PI acos(-1)
#define fi first
#define se second
#define INF 0x3f3f3f3f
#define INF64 0x3f3f3f3f3f3f3f3f
#define random(a,b) ((a)+rand()%((b)-(a)+1))
#define ms(x,v) memset((x),(v),sizeof(x))
#define sci(x) scanf("%d",&x );
#define scf(x) scanf("%lf",&x );
#define eps 1e-8
#define dcmp(x) (fabs(x) < eps? 0:((x) <0?-1:1))
#define lc o<<1
#define rc o<<1|1
using namespace std;
typedef unsigned long long ULL;
typedef long long LL;
typedef long double DB;
typedef pair<int,int> Pair;
const int maxn = 1e5+10;
int prime[maxn],cnt;//

void getPrime(/* arguments */) {
    cnt =0;
    ms(prime,0);
    for(int i=2;  i<maxn ; ++i){
        if(!prime[i]){prime[cnt++] = i;}
        for(int j=0 ; j<cnt&&prime[j]*i < maxn ; ++j){
            prime[i*prime[j]] =1;
```

```
            if(i%prime[j]==0)break;
        }
    }
}

ULL power_mod(ULL x,ULL n){
    ULL ret =1;
    while (n) {
        if(n&1)ret*=x;
        n>>=1;
        x=x*x;
    }
    return ret;
}

ULL H(ULL p,ULL k){
    return k==0?1 :power_mod(p,k-1)*(k*(p-1)+p);
}
ULL SQRT(ULL p,ULL k){
    return power_mod(p,2*k);
}

int main()
{
    // ios_base::sync_with_stdio(0);
    // cin.tie(0);
    // cout.tie(0);
    int T;
    getPrime();
    scanf("%d",&T );
    while (T--) {
        int n;
        cin>>n;
        int nn = n;
        ULL f1=1,f2=1;
        for(int i=0 ; i<cnt && prime[i]*prime[i]<=nn; ++i){
            if(n%prime[i]==0){
                int k=0;
                while (n%prime[i]==0) {
                    n/=prime[i];k++;
                }
                ULL val1 = 0;
                ULL val2 =0;
                ULL p = prime[i];
                for(int i=0 ; i<=k ;++i){
                    val1 += H(p,i);val2 += SQRT(p,i);
                }
                f1 *=val1;
                f2 *= val2;
```

```
            }
        }
        if(n!=1){
            f1 *= (H(n,0)+H(n,1));
            f2 *= (SQRT(n,0)+SQRT(n,1));
        }
        std::cout << f2-f1 << '\n';
    }
    //std::cout << "time "<< clock()/1000 <<"ms"<< '\n';
    return 0;
}
```

欧拉函数

问题

x已知，求

$$\sum_{gcd(a,x)=1} a$$

我们知道这样的 a 只有$\phi(x)$ 个，我们想能否用经典的 gauss 求等差数列的方法来解决这个问题呢? s 设 $gcd(a,x)=1$，那麽有 $gcd(x-a,x)=1$因此对于任意的数 $a, gcd(a,x)=1$ 则有 对应的$x-a$，*满足*$gcd(x-a,x)=1$,因此我们有

$$\sum_{gcd(a,x)=1} a = \frac{\phi(x)x}{2}$$

gauss 消元

gauss_jordan 对角消元

```
typedef double Matrix[maxn][maxn];
int n;
//消元为对角阵
void gauss_jordan(Matrix A,int n){
    //A 增广矩阵，第 n 列是结果列
    for(int i=0 ; i<n ; ++i){
        int r = i;//元素最大列
        for(int j = i+1 ; j<n ; ++j)
            if(abs(A[j][i]) > abs(A[r][i]))r = j;
        if(abs(A[r][i]) < eps)continue;
        if(r!=i)for(int j = 0 ; j<=n; ++j)swap(A[r][j],A[i][j]);//交换
        //与 i 行以外的所有行消元，化为阶梯阵,与 gauss 消元的不同
        for(int k=0 ; k<n ; ++k)
```

```
            if(k!=i)
            for(int j = n ; j>=i ; --j)A[k][j] -=A[k][i]/A[i][i]*A[i]
[j];//精度.
    }
}
```

异或方程组消元求秩

```
int my_rank(Matrix &A, int m,int n){
    int i=0 ,j = 0,r;
    while (i<m && j < n) {
        int r =i ;
        for(int k =i ; k<m ; ++k)
            if(A[k][j]){r=k ; break;}
        if(A[r][j]){
            if(r != i)for(int k = 0 ; k<n ; ++k)swap(A[r][k],A[i][k]);
            for(int u= i+1 ; u<m ; ++u)
                if(A[u][j])
                    for(int k = j ; k<n ; ++k)
                        A[u][k] ^= A[i][k];
            ++i;
        }
        j++;
    }
    return i;
}
```

线性基

2013.「SCOI2016」幸运数字 给一颗树，每个点有一个值a_i,求$u \sim v$路径上异或最大值

分析

涉及异或的东西，就是线性基了，如果你对这玩意不熟，可参考这个 Sengxian's Blog

然后由于线性基合并是log^2V 因此，可以暴力合并，也就是采用倍增思想，计算u,v 到公共祖先路径上的线性基，然后合并就行了

Ac code

```
#include <bits/stdc++.h>
using namespace std;
#define ms(x,v) (memset((x),(v),sizeof(x)))
#define pb push_back
typedef long long LL;
const int maxn = 20000+10;
const int LOG_MAXN = 15;
const int MOD = 998244353;
```

```
int n,q;
const int MAX_BASE = 60;
LL a[maxn];
struct LinearBase{
    LL base[MAX_BASE+1];
    LinearBase(){ms(base,0);}
    bool insert(LL val){
        for(int i=MAX_BASE ; i>=0 ; --i){
            if(val >>i &1){
                if(base[i])val ^=base[i];
                else{base[i]=val ; break;}
            }
        }
        return val>0;
    }
    LinearBase merge(LinearBase & o) {
        for(int i=MAX_BASE ; i>=0 ; --i)if(o.base[i])insert(o.base[i]);
return *this;
    }
    LL max(){
        LL ans =0;
        for(int i=MAX_BASE ;i>=0 ; --i)ans = std::max(ans,ans ^ base
[i]);return ans;
    }
};
LinearBase lb[maxn][LOG_MAXN+1];
int dep[maxn],fa[maxn][LOG_MAXN+1];
std::vector<int> G[maxn];
void dfs(int u,int father) {
    dep[u] = dep[father]+1;
    fa[u][0]=father;
    lb[u][0].insert(a[father]);lb[u][0].insert(a[u]);
    // std::cout << " lb " << u <<" " << 0 << " " <<lb[u][0].max() << '
\n';
    for(int i=1 ; (1<<i)<=dep[u] ; ++i){
        fa[u][i] = fa[fa[u][i-1]][i-1];
        lb[u][i].merge(lb[u][i-1]);lb[u][i].merge(lb[fa[u][i-1]][i-1]);
        //std::cout << " lb " << u <<" " << i << " " <<lb[u][i].max() <
< '\n';
    }
    for(auto v : G[u])
        if(v!=father)dfs(v,u);
}

LinearBase lca(int x,int y){
    LinearBase ans;
    if(dep[x] <dep[y])swap(x,y);
    int bin=dep[x]-dep[y];
```

```
    for(int i=0 ; i<LOG_MAXN ; ++i)
        if(bin>>i&1)ans.merge(lb[x][i]),x=fa[x][i];
    if(x==y){ans.insert(a[y]);return ans;}
    for(int i= LOG_MAXN-1 ; i>=0 ; --i){
        if(fa[x][i]!=fa[y][i])ans.merge(lb[x][i]),ans.merge(lb[y][i]),x
=fa[x][i],y=fa[y][i];
    }
    ans.merge(lb[x][0]);ans.insert(a[y]);
    return ans;
}


int main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin>>n>>q;
    for(int i=1 ; i<=n ; ++i)cin>>a[i];
    for(int i=1 ; i<n ; ++i){
        int u,v;
        cin>>u>>v;
        G[u].pb(v);G[v].pb(u);
    }
    dfs(1,0);
    while (q--) {
        int x, y;cin>>x>>y;
        std::cout << lca(x,y).max() << '\n';
    }
    return 0;
}
```

线性基简单表示

```
void cal() {
    for (int i = 0; i < n; ++i)
        for (int j = MAX_BASE; j >= 0; --j)
            if (a[i] >> j & 1) {
                if (b[j]) a[i] ^= b[j];
                else {
                    b[j] = a[i];
                    for (int k = j - 1; k >= 0; --k) if (b[k] && (b[j]
 >> k & 1)) b[j] ^= b[k];
                    for (int k = j + 1; k <= MAX_BASE; ++k) if (b[k] >>
 j & 1) b[k] ^= b[j];
                    break;
                }
}
```

数据结构

树上莫队

1. 点权

debug 到想吐．．．．　各种撒比错误，一晚上就没有了，　总结如下几点:

1. 两个不同参数的数组，(n,m) 的最大值不一样，最好开到同样大
2. 树上莫队注意重复节点的拆分
3. 树型数据简单生成技巧：* i rand()%i
4. 树上莫队的桶是 (q[i].l/S) not u/S(saaaa...)

题目链接 Count on a tree II:

分析　如果你学了树上莫队，对这题应该不会陌生，将树搞成链

AC code

```
#include <bits/stdc++.h>
using namespace std;
#define ms(x,v) (memset((x),(v),sizeof(x)))
#define pb push_back
#define mp make_pair
#define fi first
#define se second
typedef long long LL;
typedef pair<int,int > Pair;
const int maxn = 1e5+10;
const int max_query = 1e5+10;
const int MAX_LOG = 19;

int n,m;
int S;

struct Query{
    int id,l,r,lc,backet;
    bool operator<(const Query & o)const{
        return backet==o.backet?(backet&1?r>o.r : r <o.r) : l<o.l;
    }
};
Query q[max_query];
int ret[max_query],ans=0;
int dfn[maxn<<1],st[maxn],ed[maxn],dft=0;
int tmp[maxn],a[maxn];
int dep[maxn];
int fa[maxn][MAX_LOG];
int vis[maxn],cnt[maxn];
std::vector<int> G[maxn];
```

```
void dfs(int u,int f) {
    dep[u] = dep[f]+1;
    fa[u][0] = f;
    st[u] = ++dft;
    dfn[dft] = u;
    for(int i=1 ; (1<<i) <= dep[u] ; ++i)fa[u][i] = fa[fa[u][i-1]][i-1];
    for(auto v : G[u]){
        if(v == f)continue;
        dfs(v,u);
    }
    ed[u] = ++dft;
    dfn[dft] = u;
}
inline int lca(int u,int v){
    if(dep[u]<dep[v])swap(u,v);
    int bin = dep[u] - dep[v];
    for(int i=0 ; i<MAX_LOG ; ++i)if(bin>>i & 1)u = fa[u][i];
    if(u==v) return u;
    for(int i= MAX_LOG-1 ; i>=0 ; --i)
        if(fa[u][i]!=fa[v][i]) u = fa[u][i],v = fa[v][i];
    return fa[u][0];
}
inline int move(int node){
    if(vis[node] && --cnt[a[node]]==0)ans--;
    else if(!vis[node] && ++cnt[a[node]]==1) ans++;
    vis[node] ^=1;
}
void mo() {
    int curL = q[0].l ,curR = q[0].l-1;
    for(int i=0 ; i<m ; ++i){
        int L = q[i].l,R=q[i].r;
        while (curL > L)move(dfn[--curL]);
        while (curR < R)move(dfn[++curR]);
        while (curL < L)move(dfn[curL++]);
        while (curR > R)move(dfn[curR--]);
        ret[q[i].id] = ans;
        if(!cnt[a[q[i].lc]])ret[q[i].id]++;
    }
}


int main(int argc, char const *argv[]) {
    scanf("%d%d",&n,&m );
    ms(vis,0);
    ms(cnt,0);
    for(int i=1 ; i<=n ; ++i)scanf("%d", a+i),tmp[i] =a[i];
    sort(tmp+1,tmp+n+1);
    for(int i=1 ; i<=n ; ++i)a[i] = lower_bound(tmp+1,tmp+n+1,a[i]) - t
mp;
```

```
    for(int i=1 ; i<n ; ++i){
        int u,v;
        scanf("%d%d",&u,&v );
        G[u].pb(v);G[v].pb(u);
    }
    S = sqrt(2.0*n)+1;
    dfs(1,0);
    // for(int i=1 ; i<=dft ; ++i)std::cout << dfn[i] << ' ';std::cout
 << '\n';
    // for(int i=1 ; i<=dft ; ++i)std::cout << a[dfn[i]] << ' ';std::co
ut  << '\n';

    for(int i=0 ; i<m ; ++i){
        int u,v;
        scanf("%d%d",&u,&v );
        // std::cout << u << " " << v << '\n';
        int p = lca(u,v);
        if(st[u]>st[v])swap(u,v);
        q[i].id = i;
        q[i].lc = p;
        if(p==u){q[i].l = st[u];q[i].r = st[v];}
        else {q[i].l = ed[u];q[i].r = st[v];}
        q[i].backet = q[i].l/S;
        // std::cout << q[i].l << " "<< q[i].r<< " " << q[i].backet<< '
\n';
        // std::cout << q[i].lc << '\n';
    }
    sort(q,q+m);
    mo();
    for(int i=0 ; i<m ; ++i)printf("%d\n",ret[i]);
    return 0;
}
```

2. 边权

题目链接 Problem F. Frank Sinatra

分析

这题和前面那个题唯一不一样的地方是，这题访问的是边上的，因此可以将边上的值算做入边顶点的值，这样就 u,v 对应的区间就是 $[dfl[u]+1,dfl[v]]$, 这样开个桶记录访问到的数就行了.

code

```
#include <bits/stdc++.h>
using namespace std;
#define ms(x,v) (memset((x),(v),sizeof(x)))
#define pb push_back
#define mp make_pair
```

```
#define fi first
#define se second
typedef long long LL;
typedef pair<int,int > Pair;
const int maxn = 1e5+10;
int n,m;
const int S = 320;
struct Query{
    int id,l,r,backet;
    bool operator<(const Query & o)const{
        return backet==o.backet?(backet&1?r>o.r : r <o.r) : l<o.l;
    }
} q[maxn];
std::vector<Pair> G[maxn];
int dfl[maxn],dfr[maxn],dfn[maxn<<1],val[maxn],dft=0;
int cnt[maxn],vis[maxn],sum[maxn],ret[maxn];
void dfs(int x) {
    dfl[x] = ++dft;dfn[dft] = x;
    for(auto v : G[x])
        if(!dfl[v.fi])val[v.fi] =v.se, dfs(v.fi);
    dfr[x] = ++dft;dfn[dft] = x;
}

inline void move(int node) {
    if(val[node]>n)return;
    if(vis[node] && --cnt[val[node]]==0)--sum[val[node]/S];
    else if(!vis[node] && ++cnt[val[node]]==1)++sum[val[node]/S];
    vis[node] ^=1;
}
inline void mo(/* arguments */) {
    int l=  q[0].l,r = q[0].l-1,L,R;
    for(int i=0 ;i<m ; ++i){
        L = q[i].l,R = q[i].r;
        while (l > L)move(dfn[--l]);
        while (r < R)move(dfn[++r]);
        while (l < L)move(dfn[l++]);
        while (r > R)move(dfn[r--]);
        int j;
        for( j=0;sum[j]==S ; ++j);
        for(j*=S ; cnt[j]; ++j);
        ret[q[i].id]  = j;
    }
}
int main(int argc, char const *argv[]) {
    scanf("%d%d",&n,&m );
    for(int i=1 ; i<n ; ++i){
        int u,v,c;
        scanf("%d%d%d",&u,&v,&c );
        G[u].pb(mp(v,c));G[v].pb(mp(u,c));
```

```
    }
    dfs(1);
    for(int i=0 ; i<m ; ++i){
        int u,v;
        scanf("%d%d",&u,&v );
        if(dfl[u]> dfl[v])swap(u,v);
        q[i].id = i;
        q[i].l= dfl[u]+1;q[i].r = dfl[v];q[i].backet = q[i].l/S;
    }
    sort(q,q+m);
    mo();
    for(int i=0 ; i<m ; ++i)printf("%d\n",ret[i]);
    return 0;
}
```

其他

hash

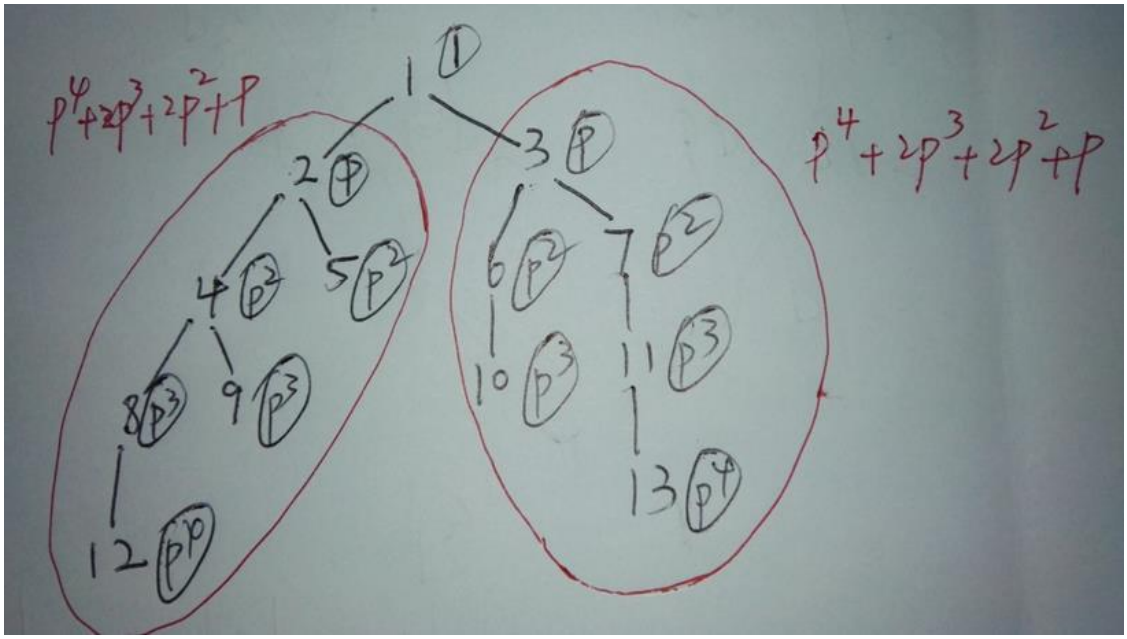
迷一样的 hash．．．．．

题目链接

2016　JAG E Similarity of Subtrees

分析

啊这位大佬的图非常到位



hash 函数的定义方式是将深度为 d 的顶点给一个权重p^d, 取　p 为素数就好

AC code

```
const int  MOD = 1e9+7;
const int p = 19;
std::map<LL, int> ma;
LL hash_val[maxn];
std::vector<int> G[maxn];
void dfs(int u,int fa) {
    hash_val[u] =1;
    for(auto v: G[u]){
        if(v == fa)continue;
        dfs(v,u);
        hash_val[u] = (hash_val[u]+hash_val[v]*p) % MOD;
    }
    ma[hash_val[u]]++;
}


int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);

    int n;
    cin>>n;
    for(int i=0 ; i<n-1 ; ++i){
        int u,v;
        cin>>u>>v;
        G[u].pb(v);G[v].pb(u);
    }
    dfs(1,-1);
    LL ans =0;
    for(auto e : ma){
        ans += (LL)e.se*(e.se -1)/2;
    }
    std::cout << ans << '\n';
    return 0;
}
```

二维离散化

分析

直接二维离散化,然后记录下各压缩了多少行和列，将其权值相乘便是离散化的图里的权重. dfs or bfs 统计一下就行了.

```
std::map<int, int> idx_x,idx_y;

LL vx[maxn],vy[maxn];
int a[maxn][maxn];
std::vector<LL> ans;
int x[maxn],y[maxn];

Pair bad[maxn];

int lishan(int * arr,int len, LL *val,std::map<int, int>& m ){
    m.clear();
    sort(arr,arr+len);
    len = unique(arr,arr+len)-arr;
    int t =1;
    val[t] = 1;
    m[arr[0]] = t++;
    for(int i=1 ; i<len ; ++i){
        if(arr[i]!=arr[i-1]+1)val[t++] = arr[i] - arr[i-1]-1;//补空白
        val[t] =1;m[arr[i]] = t++;
    }
    return t;
}
int dx[] = {1,0,0,-1};
int dy[] = {0,1,-1,0};
int vis[maxn][maxn];
LL bfs(int i,int j) {
    queue<Pair> Q;
    Q.push(mp(i,j));
    vis[i][j] =1;
    LL cnt =0;
    while (!Q.empty()) {
        Pair p = Q.front();Q.pop();
        cnt += vx[p.fi] * vy[p.se];
        for(int i=0 ; i<4 ; ++i){
            int ni = p.fi + dx[i];
            int nj = p.se+dy[i];
            if(!a[ni][nj] && !vis[ni][nj]){
                vis[ni][nj] =1;
                Q.push(mp(ni,nj));
            }
        }
    }
    return cnt;
}


int main()
{
```

```
    // ios_base::sync_with_stdio(0);
    // cin.tie(0);
    // cout.tie(0);

    int T;
    cin>>T;

    for(int kase =1 ; kase <=T ; ++kase){
        int n,m,k;
        cin>>n>>m>>k;
        for(int i=1 ; i<=k ; ++i){
            scanf("%d%d",&bad[i].fi, &bad[i].se );
            x[i] = bad[i].fi;y[i] = bad[i].se;
        }
        x[0] = 1;x[k+1] = n;
        y[0] = 1;y[k+1] = m;
        n = lishan(x,k+2,vx,idx_x);
        m = lishan(y,k+2,vy,idx_y);
        ms(a,0);
        for(int i=1;  i<=k ; ++i)a[idx_x[bad[i].fi]][idx_y[bad[i].se]]
= 1;
        for(int i=0 ; i<=m ; ++i)a[0][i] = a[n][i] =1;
        for(int i=0 ; i<=n ; ++i)a[i][0] = a[i][m] =1;
        ms(vis,0);
        ans.clear();
        for(int i=1 ; i<n ; ++i){
            for(int j=1 ; j<m ; ++j){
                if(!a[i][j] && !vis[i][j])ans.pb(bfs(i,j));
            }
        }
        printf("Case #%d:\n",kase );
        sort(ans.begin(),ans.end());
        printf("%d\n",ans.size() );
        for(unsigned int i=0 ; i<ans.size() ; ++i){
            printf("%lld%c",ans[i],i==ans.size()-1?'\n':' ' );
        }
    }

    return 0;
}
```

优化

FastIO

```
namespace
{
#define RG register

    namespace io
```

```
    {
        const int MaxBuff = 1 << 15;
        const int Output = 1 << 24;
        char B[MaxBuff], *S = B, *T = B;
        #define getc() ((S == T) && (T = (S = B) + fread(B, 1, MaxBuff,
 stdin), S == T) ? 0 : *S++)
        char Out[Output], *iter = Out;
        inline void flush() {fwrite(Out, 1, iter - Out, stdout); iter =
 Out;}
    }

    template<class Type> inline Type read()
    {
        using namespace io;
        RG char ch; RG Type ans = 0; RG bool neg = 0;
        while(ch = getc(), (ch < '0' || ch > '9') && ch != '-')  ;
        ch == '-' ? neg = 1 : ans = ch - '0';
        while(ch = getc(), '0' <= ch && ch <= '9') ans = ans * 10 + ch
- '0';
        return neg ? -ans : ans;
    }
    template<class Type> inline void print(RG Type x, RG char ch = '\n')
    {
        using namespace io;
        if(!x) *iter++ = '0';
        else
        {
            if(x < 0) *iter++ = '-', x = -x;
            static int S2[100]; RG int t = 0;
            while(x) S2[++t] = x % 10, x /= 10;
            while(t) *iter++ = '0' + S2[t--];
        }
        *iter++ = ch;
    }
}
```

爆 ll 乘法

```
inline ll mul_mod_ll(ll a,ll b){
   ll d=(ll)floor(a*(double)b/MOL+0.5);
   ll ret=a*b-d*MOL;
   if(ret<0)ret+=MOL; return ret;
}
```