```
#include <cstring>
#include <cmath>
#include <algorithm>
#include <complex>
#include <random>
#define CONFIG_POLY_DATA int
//#define CONFIG_FFT is not set
#define mod 998244353
#define g 3
#define invg 332748118
#define CONFIG_NTT y
inline int pow(int a,int k){
  long long res=1;
  while(k){
    if(k&1){
      #ifdef mod
      res=1ll*res*a%mod;
      #else
      res*=a;
      #endif
    }
    #ifdef mod
    a=1ll*a*a%mod;
    #else
    a*=a;
    #endif
    k>>=1;
  }
  return res;
}
namespace Poly{
class Poly{
  public:
    CONFIG_POLY_DATA xi[1000005];
    int ci;
    Poly(int n){
      ci=n;
      memset(xi,0,sizeof(CONFIG_POLY_DATA)*n);
      return ;
    }
    Poly(){
      ci=0;
      xi[0]=0;
      return ;
    }
    Poly(CONFIG_POLY_DATA *seq,int len){
      ci=len;
```

```
      memcpy(xi,seq,len*sizeof(CONFIG_POLY_DATA));
      return ;
    }
};
double PI;
namespace warn{
  #ifdef CONFIG_FFT
  static void fft(std::complex <double> *f,int lim,int type,int *r){
    for(int i=0;i<=lim;i++){
      if(i<r[i]){
        std::swap(f[i],f[r[i]]);
      }
    }
    for(int len=2;len<=lim;len<<=1){
      std::complex <double> omega(std::cos(PI*2/len),std::sin(type*PI*2/len));
      for(int i=0;i<=lim;i+=len){
        std::complex<double>wn1(1,0);
        for(int k=i;k<len/2+i;k++,wn1*=omega){
          std::complex<double>x=f[k],y=f[k+len/2]*wn1;
          f[k]=x+y;
          f[k+len/2]=x-y;
        }
      }
    }
    return ;
  }
  #endif
  #ifdef CONFIG_NTT
  static void ntt(int *a,int lim,int type,int *r){
    for(int i=0;i<=lim;i++){
      if(i<r[i]){
        std::swap(a[i],a[r[i]]);
      }
    }
    for(int i=1;i<lim;i<<=1){
      int omega=pow(type?g:invg,(mod-1)/(i<<1));
      for(int j=0;j<lim;j+=(i<<1)){
        int g0=1;
        for(int k=0;k<i;k++){
          int x=a[j+k];
          int y=1ll*a[i+j+k]*g0%mod;
          a[j+k]=(x+y)%mod;
          a[i+j+k]=(((x-y)%mod)+mod)%mod;
          g0=(1ll*g0*omega)%mod;
        }
      }
    }
    return ;
```

```
  }
  #endif
}
#ifdef CONFIG_FFT
Poly operator - (Poly a,const Poly &b){
  for(int i=a.ci+1;i<=b.ci;i++){
    a.xi[i]=0;
  }
  a.ci=std::max(a.ci,b.ci);
  for(int i=0;i<=a.ci;i++){
    a.xi[i]-=b.xi[i];
  }
  return a;
}
Poly operator - (Poly a,const Poly &b){
  for(int i=a.ci+1;i<=b.ci;i++){
    a.xi[i]=0;
  }
  a.ci=std::max(a.ci,b.ci);
  for(int i=0;i<=a.ci;i++){
    a.xi[i]+=b.xi[i];
  }
  return a;
}
Poly FFT(const Poly &a,const Poly &b){
  PI=acos(-1);
  int tol=1;
  int k=0;
  while(tol<std::min(a.ci+b.ci+2,1000002)){
    tol<<=1;
    k++;
  }
  int *r=new int[2*tol+5];
  r[0]=0;
  std::complex <double> *aa=new std::complex<double> [2*tol+5],*bb=new
std::complex<double> [2*tol+5];
  for(int i=0;i<=a.ci;i++){
    aa[i]=a.xi[i];
  }
  for(int i=0;i<=b.ci;i++){
    bb[i]=b.xi[i];
  }
  Poly ans(tol+5);
  ans.ci=0;
  for(int i=0;i<=tol;i++){
    r[i]=(r[i>>1]>>1)|((i&1)<<(k-1));
  }
  warn::fft(aa,tol,1,r);
```

```
  warn::fft(bb,tol,1,r);
  for(int i=0;i<=tol;i++){
    aa[i]*=bb[i];
  }
  warn::fft(aa,tol,-1,r);
  for(int i=0;i<=a.ci+b.ci;i++){
    ans.xi[i]=(aa[i].real()/tol+0.5);
  }
  ans.ci=a.ci+b.ci;
  delete[] r;
  delete[] aa;
  delete[] bb;
  return ans;
}
#endif
#ifdef CONFIG_NTT
Poly operator - (Poly a,const Poly &b){
  for(int i=a.ci+1;i<=b.ci;i++){
    a.xi[i]=0;
  }
  a.ci=std::max(a.ci,b.ci);
  for(int i=0;i<=a.ci;i++){
    a.xi[i]-=b.xi[i];
    a.xi[i]%=mod;
    a.xi[i]+=mod;
    a.xi[i]%=mod;
  }
  return a;
}
Poly operator + (Poly a,const Poly &b){
  for(int i=a.ci+1;i<=b.ci;i++){
    a.xi[i]=0;
  }
  a.ci=std::max(a.ci,b.ci);
  for(int i=0;i<=a.ci;i++){
    a.xi[i]+=b.xi[i];
    a.xi[i]%=mod;
  }
  return a;
}
Poly ans;
Poly NTT(const Poly &a,const Poly &b){
  int tol=1;
  int k=0;
  while(tol<std::min(a.ci+b.ci+2,1000002)){
    tol<<=1;
    k++;
  }
```

```
  ans.ci=0;
  int *r=new int[2*tol+5];
  r[0]=0;
  int *aa=new CONFIG_POLY_DATA [2*tol+5],*bb=new CONFIG_POLY_DATA[2*tol+5];
  for(int i=0;i<=a.ci;i++){
    aa[i]=a.xi[i];
  }
  for(int i=a.ci+1;i<=tol*2;i++){
    aa[i]=0;
  }
  for(int i=0;i<=b.ci;i++){
    bb[i]=b.xi[i];
  }
  for(int i=b.ci+1;i<=tol*2;i++){
    bb[i]=0;
  }
  for(int i=0;i<=tol;i++){
    r[i]=(r[i>>1]>>1)|((i&1)<<(k-1));
  }
  warn::ntt(aa,tol,1,r);
  warn::ntt(bb,tol,1,r);
  for(int i=0;i<=tol;i++){
    aa[i]=1ll*aa[i]*bb[i]%mod;
  }
  warn::ntt(aa,tol,0,r);
  int inv=pow(tol,mod-2);
  ans.ci=a.ci+b.ci;
  for(int i=0;i<=a.ci+b.ci;i++){
    ans.xi[i]=1ll*aa[i]*inv%mod;
  }
  delete [] aa;
  delete [] bb;
  delete [] r;
  return ans;
}
#endif
const Poly operator * (const Poly &a,const Poly &b){
  #ifdef CONFIG_FFT
  return FFT(a,b);
  #endif
  #ifdef CONFIG_NTT
  return NTT(a,b);
  #endif
}
#ifdef CONFIG_NTT
namespace modd{
  std::mt19937 ran;
  class complex{
```

```
public:
  long long real,imag;
  complex(int a=0,int b=0){
    real=a;
    imag=b;
    return ;
  }
};
inline int pow(int a,int k){
  long long res=1;
  while(k){
    if(k&1){
      #ifdef mod
      res=1ll*res*a%mod;
      #else
      res*=a;
      #endif
    }
    #ifdef mod
    a=1ll*a*a%mod;
    #else
    a*=a;
    #endif
    k>>=1;
  }
  return res;
}
int i2;
const complex operator * (const complex &a,const complex &b){
  complex ans(0,0);
  ans.real=a.real*b.real;
  ans.real%=mod;
  ans.real+=((a.imag*b.imag)%mod*i2)%mod;
  ans.real%=mod;
  ans.real+=mod;
  ans.real%=mod;
  ans.imag+=(a.imag*b.real)%mod;
  ans.imag%=mod;
  ans.imag+=(b.imag*a.real)%mod;
  ans.imag%=mod;
  return ans;
}
complex pow(complex a,int b){
  complex ans(1,0);
  complex cur=a;
  while(b){
    if(b%2==1){
      ans=ans*cur;
```

```
        ans.real=ans.real%mod;
        ans.imag=ans.imag%mod;
      }
      cur=cur*cur;
      cur.real=cur.real%mod;
      cur.imag=cur.imag%mod;
      b/=2;
    }
    return ans;
  }
  int sqrtt(long long n){
    if(n==0){
      return 0;
    }
    if(pow(n,(mod-1)/2)==mod-1){
      return -1;
    }
    long long a;
    while(1){
      a=ran()%mod;
      long long b=(((a*a)%mod-n)%mod+mod)%mod;
      i2=b;
      if(pow(b,(mod-1)/2)==mod-1){
        break;
      }
    }
    complex base(a,1);
    long long ans1=pow(base,(mod+1)/2).real;
    long long ans2=mod-ans1;
    return std::min(ans1,ans2);
  }
};
static void inv_work(Poly &b,Poly &a,int len){
  if(len==1){
    b.xi[0]=pow(a.xi[0],mod-2);
    b.ci=0;
    return ;
  }
  inv_work(b,a,((len+1)>>1));
  int tt=a.ci;
  a.ci=len-1;
  int tol=1;
  int k=0;
  while(tol<(len*2)){
    tol<<=1;
    k++;
  }
  ans.ci=0;
```

```
  int *r=new int[2*tol+5];
  r[0]=0;
  int *aa=new CONFIG_POLY_DATA [2*tol+5],*bb=new CONFIG_POLY_DATA[2*tol+5];
  for(int i=0;i<=a.ci;i++){
    aa[i]=a.xi[i];
  }
  for(int i=a.ci+1;i<=tol*2;i++){
    aa[i]=0;
  }
  for(int i=0;i<=b.ci;i++){
    bb[i]=b.xi[i];
  }
  for(int i=b.ci+1;i<=tol*2;i++){
    bb[i]=0;
  }
  for(int i=0;i<=tol;i++){
    r[i]=(r[i>>1]>>1)|((i&1)<<(k-1));
  }
  warn::ntt(aa,tol,1,r);
  warn::ntt(bb,tol,1,r);
  for(int i=0;i<=tol;i++){
    aa[i]=1ll*((((2-1ll*aa[i]*bb[i])%mod)+mod)%mod)*bb[i]%mod;
  }
  warn::ntt(aa,tol,0,r);
  int inv=pow(tol,mod-2);
  ans.ci=len-1;
  for(int i=0;i<len;i++){
    ans.xi[i]=1ll*aa[i]*inv%mod;
  }
  delete [] aa;
  delete [] bb;
  delete [] r;
  b=ans;
  a.ci=tt;
  return ;
}
Poly inv(Poly &a){
  Poly ans(a.ci);
  inv_work(ans,a,a.ci+1);
  return ans;
}
void Dao(Poly &b){
  if(b.ci==0){
    b.xi[0]=0;
    b.ci=0;
    return ;
  }
  for(int i=1;i<=b.ci;i++){
```

```
    b.xi[i-1]=(1ll*(i)*b.xi[i])%mod;
  }
  b.ci--;
  return ;
}
void Ji(Poly &b){
  b.ci++;
  for(int i=b.ci;i>=0;i--){
    b.xi[i+1]=(1ll*(pow(i+1,mod-2))*b.xi[i])%mod;
  }
  b.xi[0]=0;
  return ;
}
Poly ln(Poly a){
  Poly b=inv(a);
  Dao(a);
  b=b*a;
  for(int i=a.ci+2;i<=b.ci;i++){
    b.xi[i]=0;
  }
  b.ci=std::min(b.ci,a.ci+1);
  Ji(b);
  return b;
}
Poly a,temp,temp2;
void exp_work(Poly &ans,Poly &b,int len){
  if(len==1){
    ans.ci=0;
    ans.xi[0]=1;
    return ;
  }
  exp_work(ans,b,(len+1)>>1);
  for(int i=ans.ci+1;i<=(len)*2;i++){
    ans.xi[i]=0;
  }
  ans.ci=len*2;
  a=ln(ans);
  temp=Poly(len-1);
  temp.xi[0]=1;
  for(int i=0;i<len;i++){
    temp.xi[i]+=b.xi[i];
    temp.xi[i]%=mod;
  }
  for(int i=0;i<len;i++){
    temp.xi[i]+=mod-a.xi[i];
    temp.xi[i]%=mod;
  }
  ans=temp*ans;
```

```
  for(int i=len;i<=ans.ci;i++){
    ans.xi[i]=0;
  }
  ans.ci=len-1;
  return ;
}
Poly exp(Poly &a){
  Poly ans(a.ci);
  exp_work(ans,a,a.ci+1);
  return ans;
}
void sqrt_work(Poly &ans,Poly &a,int len){
  if(len==1){
    ans.ci=0;
    ans.xi[0]=modd::sqrtt(a.xi[0]);
    return ;
  }
  sqrt_work(ans,a,(len+1)/2);
  temp2=ans*ans;
  for(int i=0;i<=ans.ci;i++){
    ans.xi[i]*=2;
    ans.xi[i]%=mod;
  }
  for(int i=ans.ci+1;i<=len;i++){
    ans.xi[i]=0;
  }
  ans.ci=len-1;
  temp=inv(ans);
  for(int i=0;i<=temp.ci;i++){
    temp.xi[i]=(1ll*temp.xi[i])%mod;
    temp2.xi[i]+=a.xi[i];
    temp2.xi[i]%=mod;
  }
  temp2.ci=len-1;
  ans=temp*temp2;
  for(int i=len;i<=ans.ci;i++){
    ans.xi[i]=0;
  }
  ans.ci=len-1;
  return ;
}
Poly sqrt(Poly &a){
  Poly ans(a.ci);
  sqrt_work(ans,a,a.ci+1);
  return ans;
}
Poly chu(Poly &F,Poly G,Poly &Lf){
  for(int i=0;i<=F.ci;i++){
```

```
    if(F.ci-i>i){
      std::swap(F.xi[i],F.xi[F.ci-i]);
    }
  }
  a=G;
  for(int i=0;i<=G.ci;i++){
    if(G.ci-i>i){
      std::swap(G.xi[i],G.xi[G.ci-i]);
    }
  }
  int bk=G.ci;
  for(int i=F.ci-G.ci+1;i<=G.ci;i++){
    G.xi[i]=0;
  }
  G.ci=F.ci-G.ci;
  G=inv(G);
  temp=F*G;
  for(int i=F.ci-bk+1;i<=a.ci;i++){
    temp.xi[i]=0;
  }
  temp.ci=std::min(temp.ci,F.ci-bk);
  for(int i=0;i<=temp.ci;i++){
    if(temp.ci-i>i){
      std::swap(temp.xi[i],temp.xi[temp.ci-i]);
    }
  }
  for(int i=0;i<=F.ci;i++){
    if(F.ci-i>i){
      std::swap(F.xi[i],F.xi[F.ci-i]);
    }
  }
  for(int i=0;i<=G.ci;i++){
    if(G.ci-i>i){
      std::swap(G.xi[i],G.xi[G.ci-i]);
    }
  }
  Lf=F-a*temp;
  return temp;
}
#endif
}
```

```
int s,t;
int fir[35005];
int nxt[5000005];
int v[5000005];
int now=1;
int can[5000005];
int used[5000005];
void add(int x,int y,int z){
  v[++now]=y;
  nxt[now]=fir[x];
  fir[x]=now;
  can[now]=z;
  used[now]=0;
  return ;
}
namespace Dinic{
  std::queue<int>qu;
  int nfir[550015];
  int dep[550015];
  bool bfs(int now){
    for(int i=1;i<=t;i++){
      dep[i]=0;
    }
    dep[now]=1;
    qu.push(now);
    while(qu.size()>0){
      int tt=qu.front();
      qu.pop();
      for(int i=fir[tt];i!=-1;i=nxt[i]){
        if(dep[v[i]]||can[i]==used[i]){
          continue;
        }
        dep[v[i]]=dep[tt]+1;
        qu.push(v[i]);
      }
    }
    return dep[t];
  }
  int dfs(int now,int flow){
    if(flow==0||now==t){
      return flow;
    }
    int nflow=0;
    for(int &i=nfir[now];i!=-1;i=nxt[i]){
      if(dep[v[i]]!=dep[now]+1||can[i]==used[i]){
        continue;
      }
```

```
      int tt=dfs(v[i],std::min(flow-nflow,can[i]-used[i]));
      if(tt==0){
        continue;
      }
      nflow+=tt;
      used[i]+=tt;
      used[i^1]-=tt;
      if(nflow==flow){
        return nflow;
      }
    }
    return nflow;
  }
  int solve(int ans=0){
  while(bfs(s)){
      for(int i=1;i<=t;i++){
        nfir[i]=fir[i];
      }
      ans+=dfs(s,0x3f3f3f3f);
    }
    return ans;
  }
};
```

```
int fir[10005];
int nxt[400005];
int u[400005];
int v[400005];
int now=1;
int can[400005];
int used[400005];
int cost[400005];
void add(int x,int y,int z,int a){
  v[++now]=y;
  u[now]=x;
  nxt[now]=fir[x];
  fir[x]=now;
  can[now]=z;
  used[now]=0;
  cost[now]=a;
  return ;
}
int s,t;
namespace mcflow{
  bool vis[5000005];
  int h[5000005];
  int dis[5000005];
  int pre[5000005];
  int usedge[5000005];
  class node{
    public:
      int id,data;
      node(int x,int y){
        id=x;
        data=y;
        return ;
      }
  };
  bool operator < (node a,node b){
    return a.data>b.data;
  }
  void spfa(){
    std::queue<int>qu;
    for(int i=1;i<=t;i++){
      vis[i]=0;
      h[i]=0x3f3f3f3f3f3f3f3f;
    }
    h[s]=0;
    vis[s]=1;
    qu.push(s);
    while(qu.size()>0){
```

```
      int tp=qu.front();
      qu.pop();
      vis[tp]=0;
      for(int i=fir[tp];i!=-1;i=nxt[i]){
        if(can[i]!=used[i]&&h[v[i]]>h[tp]+cost[i]){
          h[v[i]]=h[tp]+cost[i];
          if(vis[v[i]]==0){
            vis[v[i]]=1;
            qu.push(v[i]);
          }
        }
      }
    }
    return ;
  }
  bool dij(){
    std::priority_queue<node>qu;
    for(int i=1;i<=t;i++){
      dis[i]=0x3f3f3f3f3f3f3f3f;
      vis[i]=0;
      pre[i]=0;
    }
    dis[s]=0;
    qu.push(node(s,0));
    while(qu.size()>0){
      node temp=qu.top();
      qu.pop();
      if(vis[temp.id]){
        continue;
      }
      vis[temp.id]=1;
      for(int i=fir[temp.id];i!=-1;i=nxt[i]){
        int edge=cost[i]+h[temp.id]-h[v[i]];
        if(can[i]!=used[i]&&dis[v[i]]>dis[temp.id]+edge){
          dis[v[i]]=dis[temp.id]+edge;
          usedge[v[i]]=i;
          pre[v[i]]=temp.id;
          if(vis[v[i]]==0){
            qu.push(node(v[i],dis[v[i]]));
          }
        }
      }
    }
    return dis[t]!=0x3f3f3f3f3f3f3f3f;
  }
  int ans(){
    spfa();
    int ans=0;
```

```
    while(dij()){
    for(int i=1;i<=t;i++){
        h[i]+=dis[i];
      }
      int minf=0x3f3f3f3f3f3f3f;
      for(int i=t;i!=s;i=pre[i]){
        minf=std::min(minf,can[usedge[i]]-used[usedge[i]]);
      }
      for(int i=t;i!=s;i=pre[i]){
        used[usedge[i]]+=minf;
        used[usedge[i]^1]-=minf;
      }
      ans+=minf*h[t];
    }
    return ans;
  }
};
```