出题人：_LiWenX_,hukk。

题出的好！难度适中，覆盖知识点广，题目又着切合实际的背景，解法比较自然。给出题人点赞！

期望大众分：$[60,100]+[40,100]+[44,100]+[20,100]=[164,400]$。

全都 AK 了记得来骂出题人 lf。

团子制作

纯送分题，显然是对着 NOIP2022 的 T1 出的（

考虑分三种情况计算答案。

1. 两个人选的都是横着的。
2. 两个人选的都是竖着的。
3. 两个人选的一横一竖。

假设我们已经预处理了每一个点 左边，右边，上面，下面分别有多少空位。

前两种情况本质相同，我们只考虑第一种。

首先如果两个人所选的不是同一行，这很简单处理，如果选到了同一行，我们钦定第一个人选的位置靠前，那么枚举第二个人所选的左端点即可，这可以简单计算。

第一种情况就处理完了。

第三种情况，我们可以看做两个人选的东西没有交，不妨考虑容斥，它们的交只在一个点上，我们直接枚举这一个点就不重不漏了。

具体来说，先计算出一个人横着选的方案数 A 和一个人竖着选的方案总数 B。

先算出 $A\times B$ 作为答案，再枚举每一个点，钦定这个点是交，直接减去同包涵这个点的横着和竖着的数量的乘积即可。

代码实现非常容易，**注意多测清空，致敬传奇 NOIP2022 不清多测题。**

```
#include<bits/stdc++.h>
#define int long long
#define mod 998244353
using namespace std;
int n,m;
char a[1005][1005];
int pre1[1005][1005],suf1[1005][1005],s1[1005],s2[1005];
int pre2[1005][1005],suf2[1005][1005];
void solve(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            cin>>a[i][j];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            if(a[i][j]=='#'){
                pre1[i][j]=0;continue;
            }
            pre1[i][j]=pre1[i][j-1]+1;
```

```
            s1[i]+=pre1[i][j];
        }
    }
    for(int i=1;i<=n;i++){
        for(int j=m;j;j--){
            if(a[i][j]=='#'){
                suf1[i][j]=0;continue;
            }
            suf1[i][j]=suf1[i][j+1]+1;
        }
    }
    for(int j=1;j<=m;j++){
        for(int i=1;i<=n;i++){
            if(a[i][j]=='#'){
                pre2[i][j]=0;continue;
            }
            pre2[i][j]=pre2[i-1][j]+1;
            s2[j]+=pre2[i][j];
        }
    }
    for(int j=1;j<=m;j++){
        for(int i=n;i;i--){
            if(a[i][j]=='#'){
                suf2[i][j]=0;continue;
            }
            suf2[i][j]=suf2[i+1][j]+1;
        }
    }
    int ans=0,S1=0,S2=0;
    for(int i=1;i<=n;i++){
        S1+=s1[i];if(S1>=mod) S1-=mod;
    }
    for(int i=1;i<=n;i++){
        ans+=(S1-s1[i])*s1[i]%mod;
        int sum=0;
        for(int j=1;j<=m;j++){
            ans+=sum*suf1[i][j]%mod*2%mod;
            sum+=pre1[i][j];
            if(sum>=mod) sum-=pre1[i][j];
        }
    }ans=(ans%mod+mod)%mod;
    for(int j=1;j<=m;j++){
        S2+=s2[j];if(S2>=mod) S2-=mod;
    }
    for(int j=1;j<=m;j++){
        ans+=(S2-s2[j])*s2[j]%mod;
        int sum=0;
        for(int i=1;i<=n;i++){
            ans+=sum*suf2[i][j]%mod*2%mod;
            sum+=pre2[i][j];
        }
    }ans=(ans%mod+mod)%mod;
    ans+=S1*S2%mod*2%mod;
    for(int i=1;i<=n;i++){
        for(int j=1;j<=m;j++){
            ans-=pre1[i][j]*suf1[i][j]%mod*pre2[i][j]%mod*suf2[i][j]%mod*2%mod;
```

```
        }
    }
    ans=(ans%mod+mod)%mod;
    cout<<ans<<'\n';
    memset(pre1,0,sizeof(pre1));
    memset(pre2,0,sizeof(pre2));
    memset(suf1,0,sizeof(suf1));
    memset(suf2,0,sizeof(suf2));
    memset(s1,0,sizeof(s1));
    memset(s2,0,sizeof(s2));
}
signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    int tt;cin>>tt;
    while(tt--) solve();
}
```

石樱异变

感觉我给的这个 NTT 模板还是比较好用的吧，毕竟都是重载了 `vector<int>` 的乘法了（

$O(n|t|)$ 的 dp 做法是很简单的，但是和正解毫无关系，所以这里不多赘述。

首先我们要发现一个性质，答案和这个 t 字母的顺序完全无关，我们只在意每一个字符的出现次数。

所以让我们考虑 t 中只有一种字符 `a` 的情况怎么做，假设 $m=|t|$。

不妨先把概率转化成计数，我们认为有 w_x 种不同的字符 x，然后考虑计算合法的字符串的数量，依然设 $S=\sum w_x$。

设 f_n 表示长度为 $n+m$ 的字符串 s 在第 $n+m$ 位**恰好**拥有了 m 个 `a` 的方案数，根据插板法，可以发现答案是 $(S-w_a)^n w_a^m\binom{n+m-1}{m-1}$，这是显然的，那么对于 n，答案就是 $\sum\limits_{i=1}^{n-m} f_i S^{n-i-m}$。这是可以简单计算的。

如何推广到 26 种字符呢?

推广状态，直接设 $f_{x,n}$ 表示对于字符 x 长度为 $n+t_x$ 的字符串的最后一位恰好拥有了 t_x 个字符 `x` 的方案数（t_x 为 x 在 t 中的出现次数）。

不难发现，设 F 为 $f_1\sim f_{26}$ 的卷积，F 就和刚刚的问题中的 f 等价了，这个过程大家可以自行理解一下，和背包的原理是一样的。

我们直接使用下发的卷积代码，就可以得到一个复杂度为 $O(|\Sigma| n\log n)$ 的做法了，其中 $|\Sigma|$ 是字符集（有人可以把这个当常数做到 $O(n\log n)$ 是谁我不说），可以通过。

不知道大家能不能给出更好的做法 qwq。

```
#include<bits/stdc++.h>
#define int long long
#define mod 998244353
using namespace std;
#define poly vector<int>
int quickpow(int a,int b){
    int ret=1;
    while(b){
```

```
        if(b&1) ret=ret*a%mod;
        a=a*a%mod;
        b>>=1;
    }return ret;
}
int G=3;
int ri[400005];
void NTT(poly &f,int N,int op){
    for(int i=0;i<N;i++){
        if(i<ri[i]) swap(f[i],f[ri[i]]);
    }
    for(int len=2,k=1;len<=N;len<<=1,k<<=1){
        int Wn=quickpow((op==1?G:quickpow(G,mod-2)),(mod-1)/len);
        for(int i=0;i<N;i+=len){
            for(int j=0,w=1;j<k;j++,w=w*Wn%mod){
                int x=f[i+j],y=f[i+j+k]*w%mod;
                f[i+j]=x+y;if(f[i+j]>=mod) f[i+j]-=mod;
                f[i+j+k]=x-y;if(f[i+j+k]<0) f[i+j+k]+=mod;
            }
        }
    }
    if(op==-1){
        int Inv=quickpow(N,mod-2);
        for(int i=0;i<N;i++) f[i]=f[i]*Inv%mod;
    }
}
poly operator *(poly A,poly B){
    int N=1,lim=0;int len=A.size()+B.size()-1;
    while(N<=(int)A.size()+(int)B.size()) N<<=1,lim++;
    A.resize(N),B.resize(N);
    for(int i=0;i<N;i++){
        ri[i]=(ri[i>>1]>>1)|((i&1)<<(lim-1));
    }
    NTT(A,N,1),NTT(B,N,1);
    for(int i=0;i<N;i++) A[i]=A[i]*B[i]%mod;
    NTT(A,N,-1);A.resize(len);
    return A;
}
int fac[200005],inv[200005];
void init(int n=200000){
    fac[0]=1;for(int i=1;i<=n;i++) fac[i]=fac[i-1]*i%mod;
    inv[n]=quickpow(fac[n],mod-2);
    for(int i=n-1;~i;i--) inv[i]=inv[i+1]*(i+1)%mod;
}
int C(int n,int m){
    return fac[n]*inv[m]%mod*inv[n-m]%mod;
}
int n,w[26];
int t[26];
poly f[26];
signed main(){
    init();
    int S=0;
    for(int i=0;i<26;i++){
        cin>>w[i];
        S+=w[i];
```

```
    }
    S%=mod;
    string T;cin>>T;int len=T.size();
    cin>>n;
    int ti=1;
    for(int i=0;i<len;i++){
        t[T[i]-'a']++;
        ti=ti*w[T[i]-'a']%mod;
    }
    for(int c=0;c<26;c++){
        if(!t[c]) continue;
        f[c].resize(n+1-len);
        for(int i=0;i<=n-len;i++){
            f[c][i]=quickpow((S-w[c]),i)*C(i+t[c]-1,t[c]-1)%mod;
        }
    }
    poly ans;ans.resize(n+1-len);
    ans[0]=1;
    for(int c=0;c<26;c++){
        if(t[c]) ans=ans*f[c];
        ans.resize(n+1-len);
    }
    for(int i=1;i<len;i++){
        cout<<0<<'\n';
    }
    int sum=0;
    for(int i=0;i<=n-len;i++){
        sum=sum*S+ans[i];
        sum%=mod;
        cout<<sum*ti%mod*quickpow(quickpow(S,i+len),mod-2)%mod<<'\n';
    }
}
```

魔理沙偷走了重要的东西

关于题目背景，这是一个很洗脑的曲子，赶紧去洗脑吧！

~~题目背景的话出自MV而不是歌词。~~

回到题目上来，确保你没有看错题，首先简要题意应该是：

给出一颗 n 个点构成的树 T 以及 m 个二元组 (u_i, v_i)，设 L_i 为第 i 个二元组对应的树上路径包涵的点构成的点集。

你需要给这棵树上的每一个节点黑白染色，一个染色方案的权值为满足以下条件的集合 S 数量。

1. $S \subseteq \{1, 2, 3, \dots, n\}$。
2. 对于所有 $1 \le i \le m$，若 $L_i \subseteq S$，则必须存在一个点 $x \in L_i$，满足 x 为黑色。
3. S 内的点在树上连通。

现在的问题是，对于所有的染色方案，求它们的权值和，答案对 998244353 取模。

让我们先考虑暴力怎么做，首先我们可以想到的一点是拆贡献，不难发现对于一个连通块 S，它的贡献是独立的，所以我们只用对每个 S 去计数它在哪些染色方案中是合法的再加起来即可。（其实干了这一步之后就很像这个题，但做法没一点关系。）

对于一个连通块 S，改如何计算答案呢，首先连通块外部的点都是随便填的，所以最后要乘一个 $2^{n-|S|}$，接下来，考虑内部情况，假设所有被 S 包涵的链构成了集合 A，我们考虑容斥，通过容斥的手段去计数，具体来说，我们枚举子集 $B\subseteq A$，钦定 B 中元素全部都是白色，其他位置随便填，那么可以发现，答案是：

$$2^{n-|S|}\sum_{B\subseteq A}(-1)^{|B|}2^{|S|-|\bigcup_{i\in B}L_i|}$$

明显可以把一个 $2^{|S|}$ 提出来。

$$2^n\sum_{B\subseteq A}(-1)^{|B|}2^{-|\bigcup_{i\in B}L_i|}$$

让我们再提前枚举 S，把答案写出来。

$$2^n\sum_S\sum_{B\subseteq 被S包涵的链}(-1)^{|B|}2^{-|\bigcup_{i\in B}L_i|}$$

改变求和顺序，其中对 B 的限制是 B 是 $\{1,2,3,\dots,m\}$ 的子集：

$$2^n\sum_B\sum_{S包涵了B中的所有链}(-1)^{|B|}2^{-|\bigcup_{i\in B}L_i|}$$

$$2^n\sum_B(-1)^{|B|}2^{-|\bigcup_{i\in B}L_i|}\sum_{S包涵了B中的所有链}1$$

这个 $\sum_{S包涵了B中的所有链}1$ 很有意思，它其实就是包涵了 B 中所有链的连通块数，由于 m 很小，所以对 B 的枚举量只有 2^m 可以接受，关键在于快速计算包涵了 B 中所有链的连通块数，我们可以使用树形 dp 解决。

考虑一个朴素的实现，不难想到一个 $O(2^m mn)$ 的做法，我们需要将这个 n 变成 $\text{polylog}(n)$，所以考虑树剖维护，发现问题是两个部分，对于一个集合 S，求出 S 中链的并的大小，这个部分很简单，直接树剖线段树解决，麻烦的在于包涵 S 中所有链的连通块数量，我们需要一个比较好的做法。

设状态 $f_{i,j}$ 表示 $i\to j$ 这个方向，j 为连通块的 LCA 的方案数，显然状态数是 $O(n)$ 的，转移显然：$f_{i,j}=\prod_{u\in e_j,u\neq i}f_{j,i}+1$。

我们令 $val_i=\dfrac{\prod_{u\in e_i}f_{i,u}+1}{(f_{i,fa_i}+1)\times(f_{fa_i,i}+1)}$，那么对于一个连通块 S 来说，设 p 为 S 中所有点的 lca，包涵它的连通块数就是 $(f_{p,fa_p}+1)\times(f_{fa_p,p}+1)\times\text{lca}(S)\prod_{i\in S}val_i$，具体原因可以手推一下，应该是不难理解的。

那么依然使用树剖线段树即可维护这个东西，具体来说对于现有的连通块 S，考虑加入链 (x_i,y_i) 得到 S'，向 S 中添加链 (x_i,y_i) 以及链 $(\text{lca}(S),\text{lca}(x_i,y_i))$ 即可，使用扫描线那样的线段树可以维护这种并信息。

时间复杂度 $O(2^m m\log^2 n)$，我不知道卡卡常能不能过，考虑继续优化掉那个复杂度中的 m，由于每一次都要重构 S 中的链，但其实由 S 到 $S+1$ 对于二进制位的变化其实是 $O(1)$ 的，甚至从 S 和 $S+1$ 从高到低第一个不同的位开始把后面的每一位都撤销，再重构，均摊下来也是每次修改 $O(1)$ 次的，所以我们对于一条链，记录修改前的 $\text{lca}(S)$，第一次出现不同位时，直接撤销，撤销的 -1 操作，在线段树上也可以实现，需要用也直接仿照刚刚的做法加上即可，这部分的时间复杂度变为了 $O(2^m\log^2 n+2^m m\log n)$（因为还要把现在的 $\text{lca}(S)$ 推出来，不过精细实现可以省掉这部分）。

这样的话，总复杂度是 $O(n+2^m\log^2 n+2^m m\log n)$，可以通过。

```
#include<bits/stdc++.h>
#define int long long
```

```
#define mod 998244353
#define pb push_back
using namespace std;
int quickpow(int a,int b){
    int ret=1;
    while(b){
        if(b&1) ret=ret*a%mod;
        a=a*a%mod;
        b>>=1;
    }return ret;
}
int n,m;
vector<int> e[100005];
int
dep[100005],fa[100005],siz[100005],son[100005],id[100005],D,top[100005],wt[100005
];
void dfs1(int now,int F){
    fa[now]=F;
    dep[now]=dep[F]+1;
    siz[now]=1;
    for(int u:e[now]){
        if(u==F) continue;
        dfs1(u,now);
        siz[now]+=siz[u];
        if(siz[u]>siz[son[now]]) son[now]=u;
    }
}
void dfs2(int now,int TOP){
    top[now]=TOP;
    id[now]=++D;wt[D]=now;
    if(son[now]) dfs2(son[now],TOP);
    for(int u:e[now]){
        if(u==son[now]||u==fa[now]) continue;
        dfs2(u,u);
    }
}
int lca(int x,int y){
    if(!x||!y) return x^y;
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        x=fa[top[x]];
    }if(dep[x]<dep[y]) return x;
    return y;
}
int val[100005],Val[100005],bitnum[1<<20];
#define ls (now<<1)
#define rs (now<<1|1)
#define mid ((l+r)>>1)
int tree[100005<<2],St[100005<<2],tag1[100005<<2],num[100005<<2],tag2[100005<<2];
void build(int now,int l,int r){
    tree[now]=1;
    if(l==r){
        St[now]=val[wt[l]];
        return ;
    }
    build(ls,l,mid);
```

```
    build(rs,mid+1,r);
    St[now]=St[ls]*St[rs]%mod;
}
void pushup(int now,int l,int r){
    if(tag1[now]) tree[now]=St[now];
    else if(l==r) tree[now]=1;
    else tree[now]=tree[ls]*tree[rs]%mod;
    if(tag2[now]) num[now]=r-l+1;
    else if(l==r) num[now]=0;
    else num[now]=num[ls]+num[rs];
}
void add(int now,int l,int r,int x,int y,int k1,int k2){
    if(l>=x&&r<=y){
        tag1[now]+=k1;
        tag2[now]+=k2;
        pushup(now,l,r);
        return ;
    }
    if(mid>=x) add(ls,l,mid,x,y,k1,k2);
    if(mid<y) add(rs,mid+1,r,x,y,k1,k2);
    pushup(now,l,r);
}
#undef ls
#undef rs
#undef mid
void add(int x,int y,int k1,int k2){
    while(top[x]!=top[y]){
        if(dep[top[x]]<dep[top[y]]) swap(x,y);
        add(1,1,n,id[top[x]],id[x],k1,k2);
        x=fa[top[x]];
    }
    if(id[x]<id[y]) add(1,1,n,id[x],id[y],k1,k2);
    else add(1,1,n,id[y],id[x],k1,k2);
}
unordered_map<int,int> f[100005];
int dp(int x,int y){
    if(f[x][y]) return f[x][y];
    int ret=1;
    for(int u:e[y]){
        if(u==x) continue;
        ret=ret*(dp(y,u)+1)%mod;
    }return f[x][y]=ret;
}
int P[100005];
int x[100005],y[100005],L[100005],pre[100005];
signed main(){
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);
    cin>>n>>m;
    P[0]=1;for(int i=1;i<=n;i++) P[i]=P[i-1]*2%mod;
    for(int i=1;i<n;i++){
        int x,y;cin>>x>>y;
        e[x].pb(y);e[y].pb(x);
    }
    dfs1(1,1);
    dfs2(1,1);
```

```
    for(int i=1;i<=n;i++){
        for(int u:e[i]) dp(i,u);
    }
    int s=0;
    for(int i=1;i<=n;i++){
        int sum=1;
        for(int u:e[i]){
            if(u==fa[i]) continue;
            sum=sum*(f[i][u]+1)%mod;
        }s+=sum;
    }
    s++;
    s%=mod;
    int ans=s*P[n]%mod;
    for(int i=1;i<=m;i++){
        cin>>x[i]>>y[i];
        L[i]=lca(x[i],y[i]);
    }
    for(int i=1;i<=n;i++){
        val[i]=1;
        for(int u:e[i]){
            val[i]=val[i]*(f[i][u]+1)%mod;
        }
        if(i!=1){
            Val[i]=quickpow((f[i][fa[i]]+1)*(f[fa[i]][i]+1)%mod,mod-2);
        }
        else Val[i]=1;
        val[i]=val[i]*Val[i]%mod;
    }
    int now=0;
    build(1,1,n);
    for(int S=1;S<(1<<m);S++){
        bitnum[S]=bitnum[S>>1]+(S&1);
        int Lca=0;
        bool flag=0;
        for(int i=m-1;~i;i--){
            if(((S>>i)&1)!=(((now>>i)&1))){
                flag=1;
            }
            if(flag){
                if((now>>i)&1){
                    add(x[i+1],y[i+1],-1,-1);
                    if(pre[i+1]) add(L[i+1],pre[i+1],-1,0);
                }
                if((S>>i)&1){
                    int nxt=lca(L[i+1],Lca);
                    add(x[i+1],y[i+1],1,1);
                    if(Lca) add(L[i+1],Lca,1,0);
                    pre[i+1]=Lca;
                    Lca=nxt;
                }
            }
            else{
                if(((S>>i)&1)==0) continue;
                Lca=lca(Lca,L[i+1]);
            }
```

```
        }
        if(bitnum[S]&1){
            ans-=P[n-num[1]]*tree[1]%mod*quickpow(Val[Lca],mod-2)%mod;
        }
        else{
            ans+=P[n-num[1]]*tree[1]%mod*quickpow(Val[Lca],mod-2)%mod;
        }
        now=S;
    }
    ans=(ans%mod+mod)%mod;
    cout<<ans<<'\n';
}
```

妖魔夜行

首先一看就是要 dp 的，具体咋 dp 让我们先枚举一个比较好转移的状态。

不难发现，每一次向前移动都必然走到一个当前到过的最靠前的位置，所以直接设 f_i 表示目前在 i 这个位置，且到过最靠前的位置也是 i 的方案数，显然 $f_n=1$，答案是 f_1，为了方便转移，我们还要求，要么走过 f_{i+1}，要么到达 i 是由 13 而来，也就是说，考虑 f 转移的时候，它不会出现任何向后的操作了。

我们先考虑一个相当暴力的做法，对于 f_i，使用它去更新其他地方的 dp 值。

考虑从 i 开始第一步使用操作 3 我们可以枚举一个 $j|i$，让 i 使用操作 3 到达 j 点，给 f_j 加上 f_i，然后，枚举一个 $k:j+1\sim i-1$，表示此时使用操作 1 向后走到的地方，k 的下一步必然是操作 3，到达一个 k 的因子 d 上去，而且由于你执行了 13 操作，你必然不可再使用 1 操作，所以这样的情况不会递归出现，到 d 就结束了这种贡献。优化这个做法，考虑转置一下贡献，我们直接在枚举 i,j 后枚举 d，对于 d 考虑合法的 k 的数量，也就是 $j+1\sim i-1$ 中 d 的倍数数量，这个值当然是 $\frac{i-1}{d}-\frac{j}{d}$，（默认除法都是下取整的）。

再考虑 $i\to i-1$ 的转移，我们对于 i 来说，直接 $f_i\leftarrow f_i+f_{i+1}$ 即可。

那么可以得到一段这样的代码：

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
int n,mod;
int f[100005];
vector<int> ins[100005];
signed main(){
    cin>>n>>mod;
    f[n]=1;
    for(int i=1;i<=n;i++){
        for(int j=i+i;j<=n;j+=i) ins[j].push_back(i);
    }
    for(int i=n;i>=1;i--){
        f[i]=(f[i]+f[i+1])%mod;
        for(int u:ins[i]){
            f[u]+=f[i];
            for(int j=1;j<u;j++){
                f[j]+=((i-1)/j-u/j)*f[i]%mod;
            }
        }
    }
```

```
    cout<<f[1];
}
```

因为 (i,j) 对数是 $O(n\log n)$ 的，做法复杂度为 $O(n^2\log n)$，常数很小，跑 10000 都不在话下。

接下来去优化这个做法，发现这个形式很像整除分块，具体来说把 $\frac{i-1}{d}-\frac{j}{d}$ 拆成两个部分分别考虑，每个部分都可以写成 $O(\sqrt{n})$ 个区间加的形式，使用数据结构做这个区间修改，可以得到复杂度为 $O(n\sqrt{n}\log^2 n)$ 的复杂度。我们把数据结构换成差分，就直接复杂度变成了 $O(n\sqrt{n}\log n)$。

考虑继续优化，首先考虑 $f_d\leftarrow f_d+\frac{j}{d}f_i$，枚举 j 显然必不可少，注意到此时下标 d 可以跑到 $j-1$ ，于是对于任意的 j 来说，修改的区别只在系数，可以对差分数组打标记 $O(1)$ 完成，查询一个差分数组点值直接把上面标记推回来即可，那么第一部分做到了 $O(n\sqrt{n})$。

第二部分，$f_d\leftarrow f_d+\frac{i-1}{d}f_i$，注意到此时分母都是固定的，并且 j 只影响 d 的范围，我们从小到的枚举 j，然后 d 的范围也是单调的了，我们按照整除分块的块还有 j 去跑双指针，位置之前的区别也是只有系数，直接差分数组上修改就对了，复杂度也是 $O(n\sqrt{n})$，但是空间复杂度依然为 $O(n\sqrt{n})$。

本来这就是这个题的标算做法了，奈何要跑 $2s$，如果时限开标算两倍就变成 $4s$ 了，实在不美观，所以我进行了一些精细化实现（打表），做到了更优。

首先展示一下刚刚做法的代码：

```
#include<bits/stdc++.h>
#define ll long long
#define mp make_pair
using namespace std;
int n,mod;
ll f[100005],c[100005];
vector<int> ins[100005];
int tag[100005];
vector<int>vec[100005];
signed main(){
    cin>>n>>mod;
    f[n]=1;
    for(int i=1;i<=n;i++){
        for(int j=i+i;j<=n;j+=i) ins[j].push_back(i);
    }
    for(int i=1;i<=n;i++){
        int l=1,r;
        while(l<=i){
            r=min(i,i/(i/l));
            vec[r].push_back(i);
            l=r+1;
        }
    }
    for(int i=n;i>=1;i--){
        for(int u:vec[i]){
            int tii=u/i-(u/(i+1));
            c[i]+=1ll*tag[u]*tii%mod;
        }
        c[i]+=c[i+1];
        c[i]=(c[i]%mod+mod)%mod;
        f[i]=(f[i]+f[i+1]+c[i])%mod;
        for(int u:ins[i]){
```

```
            f[u]+=f[i]*2;
            tag[u]-=f[i];
            if(tag[u]<0) tag[u]+=mod;
        }
        int l=1,r;
        for(int j=0;j<ins[i].size();j++){
            int R=ins[i][j],siz=ins[i].size()-j;
            int tii=1ll*f[i]*siz%mod;
            while(l<R){
                int val=(i-1)/l;
                r=min(R-1,(i-1)/val);
                c[l-1]-=1ll*tii*val%mod;
                c[r]+=1ll*tii*val%mod;
                l=r+1;
            }
        }
    }
    cout<<f[1];
}
```

发现空间瓶颈在于那个 `vec` 数组的预处理，打表发现你开了 4×10^8 个位置，这实在是太浪费了！

所以我们不妨对于一个 `vec[i]` 的元素取出来，你发现元素都是一段一段出现的，而段数很少，仅仅 $O(n\log n)$ 段，那么我们记录段的两个端点，空间复杂度就变成了 $O(n\log n)$，卡常后可以通过。

至于证明这个结论：发现对于任意 $x\in vec_i$，我们只关心 $\frac{x}{i},\frac{x}{i+1}$ 的值，发现段中的元素必然这两者有一个全部相同，而这个两个东西取值总和是 $O(\sum\limits_{i=1}^{n}\frac{n}{i})=O(n\log n)$ 的。

但是既然我时限开了 $2s$，也就是说我的标算进入了 $1s$，这是怎么一回事呢？

将这些段的表打出来，发现对于 i 来说，vec_i 形如 $[i,i],[2i,i+(i+1)],[3i,i+2(i+1)]\dots,[i^2,n]$，所以你不用整除分块来预处理表了，而是可以直接利用规律得到所有区间，直接可以卡进 $1s$。至于这个结论的证明，我表示我也不会，但是貌似不利用这个结论也是可以通过这个题目的，所以无所谓了。

展示标算代码，不过由于大量卡常，已经失去了可读性：

```
#include<bits/stdc++.h>
#define ll long long
#define mod 998244353
using namespace std;
int n,tag[100005];
ll f[100005],c[100005];
vector<int> ins[100005];
signed main(){
    freopen("rumia.in","r",stdin);
    freopen("rumia.out","w",stdout);
    cin>>n;f[n]=1;
    for(int i=1;i<=n;i++){
        for(int j=i+i;j<=n;j+=i) ins[j].push_back(i);
    }
    for(int i=n;i>=1;i--){
        int l=i,r=i;
        ll I=1ull*i*i;
        while(r<=n){
```

```
            if(l==I) r=n;
            for(int u=l;u<=r;u++){  //也许这里还可以再套一个整除分块，可惜瓶颈不在于此
                int tii=u/i-(u/(i+1));
                c[i]+=1ull*tag[u]*tii;
            }
            l+=i,r+=i+1;
        }
        c[i]+=c[i+1];
        c[i]=(c[i]%mod+mod)%mod;
        f[i]+=f[i+1]+c[i];
        while(f[i]>=mod) f[i]-=mod;
        for(int u:ins[i]){
            f[u]+=f[i]*2;
            tag[u]-=f[i];
            if(tag[u]<0) tag[u]+=mod;
        }
        l=1,r=0;
        for(int j=0;j<(int)ins[i].size();j++){
            int R=ins[i][j],siz=ins[i].size()-j;
            int tii=1ull*f[i]*siz%mod;
            while(l<R){
                int val=(i-1)/l;
                r=min(R-1,(i-1)/val);
                ll C=1ull*tii*val;
                c[l-1]-=C,c[r]+=C;
                l=r+1;
            }
        }
    }
    cout<<f[1];
}
```

然而蛋哥给出了一个爆标的 $O(n\log^2 n)$ 做法。

考虑反过来走，从 $1\to n$。

那么设 f_i 表示到 i 且下一步不往前走的方案数。

此时对于 i，枚举 i 的倍数 j，那么所有小于等于 i 的 k，标记 $i\sim j-1$ 中 k 的倍数数量倍的 f_k 的贡献，所以直接可以使用树状数组维护转移。

相当于是把之前做法的东西转置了一下，可以做到更好的复杂度。