水管系统

每次找到流速最慢的建筑i。假设它两侧的建筑流速分别为a和b。如果a>b，则我们可以发现，i一定是先和b连通，和b连通之后再和a连通，而不可能跳过b，直接和a连通。

所以，我们可以用一个优先队列维护所有连通块的流速最小值，每次取出最小的连通块，把它的流速增加到两侧更小的那一栋建筑，然后连通。

```
#include<bits/stdc++.h>
#define rep(i,j,k) for(int i=j;i<=k;i++)
#define repp(i,j,k) for(int i=j;i>=k;i--)
#define ls(x) (x<<1)
#define rs(x) ((x<<1)|1)
#define mp make_pair
#define sec second
#define fir first
#define pii pair<int,int>
#define lowbit(i) i&-i
#define int long long
#define double long double
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
const int N=1e5+5,M=6,S=(1<<15)+5,inf=(ll)1e18+7,mo=1e9+7;
const double eps=1e-8;
void read(int &p){
    int x=0,w=1;
    char ch=0;
    while(!isdigit(ch)){
        if(ch=='-')w=-1;
        ch=getchar();
    }
    while(isdigit(ch)){
        x=(x<<1)+(x<<3)+ch-'0';
        ch=getchar();
    }
    p=x*w;
}
int n;
struct node{
    int id,v;
    friend bool operator<(node x,node y){
        return x.v<y.v;
    }
}a[N];
int ans=0;
struct bcj{
    int val[N],fa[N];
    void init(){
        rep(i,1,n)
            val[i]=a[i].v,fa[i]=i;
    }
    int find(int x){
        if(x==fa[x])return fa[x];
```

```
        return fa[x]=find(fa[x]);
    }
    void merge(int x,int y){
        x=find(x),y=find(y);
        if(x==y)return;
        if(val[x]<val[y])swap(x,y);
        ans+=val[x]-val[y],val[y]=val[x],fa[y]=x;
    }
    int getv(int x){
        x=find(x);
        return val[x];
    }
}B;
int pos[N];
signed main(){
    //freopen("A.in","r",stdin);
    read(n);
    rep(i,1,n)
        read(a[i].v),a[i].id=i;
    B.init();
    sort(a+1,a+n+1);
    rep(i,1,n)
        pos[a[i].id]=i;
    rep(i,1,n){
        int x=a[i].id;
        int y=x==1?n:x-1;
        if(B.getv(y)<a[i].v)B.merge(x,y);
        y=(x==n?1:x+1);
        if(B.getv(y)<a[i].v)B.merge(x,y);
    }
    printf("%lld\n",ans);
    return 0;
}
```

升级

如果没有捆绑活动，我们可以写出一个朴素的背包DP，用$f[i][j]$表示前i种能量丸，花j元能获得的最大提升等级

转移就是：$f[i][j]=max(f[i-1][j],f[i-1][j-p[i]]+r[i])$

那么，有了捆绑活动以后怎么做呢？

一个很直观的想法是，我们肯定会把买的能量丸中最便宜的和最贵的捆绑。

那这相当于什么呢？相当于是，最便宜的能量丸价格*2，最贵的能量丸免费。

那么，背包问题如何让最便宜的能量丸*2，最贵的免费呢？

我们考虑把能量丸按照价格排序DP，这样，最便宜的能量丸就是我们购买的第一个能量丸。

那如何判断哪个能量丸是第一个呢？在转移的时候，我们就控制一下，如果一个状态是从0转移出来的，就表示是第一个，如果不是从0转移出来的，就不是第一个。

也就是说，我们在原先的背包的基础上，不允许从$f[i-1][0]$转移到$f[i][p[i]]$，同时增加一条从$f[i-1][0]$到$f[i][2*p[i]]$的转移，这样就能保证选的第一个能量丸价格翻倍了。

最贵的能量丸免费就比较好办，枚举能量丸i免费，然后用$f[i-1][j]+r[i]$更新答案就行了

```cpp
#include <bits/stdc++.h>
using namespace std;

const int M = 10005, N = 10005;
int f[M], n, m, ans;
pair<int,int>p[N];
inline void Mn(int &x, int y) { if (x < y) x = y;  }

int main() {
    cin>>m>>n;
    for (int i = 1; i <= n; i++)
        cin>>p[i].first>>p[i].second;
    sort(p+1,p+1+n);
    memset(f,-0x3f,sizeof(f));
    for (int i = 1, v, w; i <= n; ++i) {
        w=p[i].first,v=p[i].second;

        ans=max(ans,v);
        for (int j = 1; j <= m; j++)
            ans=max(ans,f[j]+v);

        for (int j = m; j > w; j--) {
            if (j == w*2)
                f[j]=max(f[j],v);
            f[j]=max(f[j],f[j-w]+v);
        }
    }
    cout<<ans;
    return 0;
}
```

文本查找

考虑递推出答案

用$ans[j]$表示询问串T_i在S_j中的出现次数。

按照题目的规则，S_j是在S_{j-1}的基础上复制一遍，中间加一个分隔符。

这样，我们可以用w_j表示T_i在S_j的跨越分割符的部分出现了几次，从而得到公式：

$ans[j] = w_j + ans[j-1] \times 2$

最终，我们可以推出$ans[m]$的公式：

$ans[m] = ans[m-1] \times 2 + w_m = ans[m-2] \times 4 + w_{j-1} \times 2 + w_m = \ldots = \sum_{j=0}^{m} w_j \times 2^{m-j}$

但是，我们每次询问都做一遍递推也会超时，怎么办呢？

注意到，我们只需要先递推到$|S_j| \geq |T_i|$就够了，从此以后，S_j的前T_i位和后T_i位就固定了。也就是说，新出现的次数，就只跟分隔符有关。

这样，我们可以直接对26种分隔符，都求出出现次数。同时，在询问前，预处理每一种字符的$\sum_{j=0}^{m} w_j \times 2^{m-j}$

这样就可以直接推出答案了

```cpp
#include<bits/stdc++.h>
```

```
using namespace std;
const int N=2e5+100;
const int mod = 1e9+7;
char s[20][N],x[N],T[N],ss[N];
int cnt[26][N];
int len[20];
int nxt[N];
int pow2[N];
int kmp(char s[],char t[],int n,int m) {
    if (m>n) return 0;
    int ans = 0,i,j;
    for (i=2,j=0;i<=m;i++) {
        while (j&&t[j+1]!=t[i])
            j=nxt[j];
        nxt[i]= t[j+1]==t[i] ? ++j : 0;
    }
    for (i=1,j=0;i<=n;i++) {
        while (j&&t[j+1]!=s[i])
            j=nxt[j];
        if (t[j+1]==s[i]) {
            j++;
            if (j==m)
                ans++,j=nxt[j];
        }
    }
    return ans;
}
int main() {
    int i,k,m,c,Q,ans;
    scanf("%s",s[0]+1);
    scanf("%s",x+1); m = strlen(x+1);

    len[0] = strlen(s[0]+1);
    for (i = 0; len[i] < 1e5; i++) {
        strcpy(s[i+1]+1,s[i]+1);
        s[i+1][len[i]+1] = x[i+1];
        strcpy(s[i+1]+1+len[i]+1,s[i]+1);
        len[i+1] = len[i]*2+1;
    }

    for (i=pow2[0]=1;i<=m;i++)
        pow2[i]=(pow2[i-1]<<1)%mod;

    for (c = 0; c < 26; c++)
        for (i = m; i; i--) {
            cnt[c][i]=cnt[c][i+1];
            if (x[i]-'a'==c)
                cnt[c][i]=(cnt[c][i]+pow2[m-i])%mod;
        }

    scanf("%d",&Q);
    while (Q--) {
        scanf("%s",T+1);
        k = strlen(T+1);

        for (i=0; len[i]<k; i++);

        if (i<=m) {
```

```
            ans=1LL*kmp(s[i],T,len[i],k)*pow2[m-i]%mod;

            memcpy(ss+1,s[i]+len[i]-k+2,k-1);
            memcpy(ss+k+1,s[i]+1,k-1);
            for (c=0;c<26;c++) {
                ss[k]=c+'a';
                ans = (ans + 1LL*kmp(ss,T,k*2-1,k)*cnt[c][i+1])%mod;
            }
            printf("%d\n",ans);
        } else {
            printf("%d\n",kmp(s[m],T,len[m],k));
        }
    }
    return 0;
}
```

单身村庄

先考虑40分暴力：

每次询问，以x为根做一遍树形DP

用$f[i][j]$表示在i的子树中选一个包含i的大小为j的连通块的方案数。

转移就是把孩子一个一个加进来：

枚举每个孩子k，再枚举这个孩子中选t个点，然后$f[i][j]+=f[k][t]*f[i][j-t]$

那么，怎么优化到满分呢？

对于满分，显然不可能每次询问重新DP了，所以我们得先确定一个根（假设是1），预处理一个DP

那怎么修改呢？

注意到个题的$j\leq 12$，那么，修改了一条边以后，最多只会影响到这条边往上走12个点的DP值，我们把这12个点的DP值重新算一下就好了。

怎么重新算呢？注意到，我们DP既可以加一个孩子，也可以删一个孩子：
$f[i][j]-=f[k][t]*f[i][j-t]$

所以，我们把这12个点先去掉对应的孩子，然后做修改，改完以后再加回来就行了

接下来，怎么处理询问呢？

注意到，询问x时，也最多只需要考虑x往上走12个点，我们把这些点从上往下分别叫做1,2,3,4,...,12

那么，我们的目标是让x成为树根。此时，我们就可以写一个换根DP：把根从1换到2，再从2换到3，最后一直换到x。

那怎么换根呢？还是利用刚刚的DP删孩子的方法，把1删掉一个孩子2，然后给2添加一个孩子1，就使得根从1换到了2。

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<cmath>
```

```
#include<cstdlib>
#include<ctime>
#include<queue>
#include<set>
#include<map>
#include<stack>
#include<bitset>
#include<assert.h>
using namespace std;
typedef long long LL;
const int N=1e5+100,M=12;
int gi() {
    int w=0;bool q=1;char c=getchar();
    while ((c<'0'||c>'9') && c!='-') c=getchar();
    if (c=='-') q=0,c=getchar();
    while (c>='0'&&c <= '9') w=w*10+c-'0',c=getchar();
    return q? w:-w;
}
const int mod=1e9+7;
int f[N][M+1];
int fa[N],st[N];
vector<int>e[N];
bool o[N];
void dp(int a,int b) {
    for (int i=M;i;i--)
        for (int j=1;j<i;j++)
            f[a][i]=(f[a][i]+1LL*f[a][j]*f[b][i-j])%mod;
}
 void Idp(int a,int b) {
    for (int i=1;i<=M;i++)
        for (int j=1;j<i;j++)
            f[a][i]=(f[a][i]-1LL*f[a][j]*f[b][i-j])%mod;
}
int cnt;
void dfs(int k) {
    ++cnt;
    for (int t:e[k])
        if (t!=fa[k]) {
            fa[t]=k;
            dfs(t);
            dp(k,t);
        }
}
int main()
{
    int n=gi(),m=gi(),i,k,s,t,top;

    for (i=1;i<n;i++) {
        s=gi(),t=gi();
        e[s].push_back(t);
        e[t].push_back(s);
    }
    for (i=1;i<=n;i++)
        f[i][1]=1;
    dfs(1);
    assert(cnt==n);
    while (m--)
        if (gi()) {
```

```
            k=gi(),s=gi();
            for (i=1;i<=M;i++) f[0][i]=0;
            for (t=k,top=0;t;t=fa[t]) {
                st[++top]=t;
                if (top==M||o[t])
                    break;
            }
            while (t=st[top--]) {
                for (i=1;i<=M;i++) f[n+1][i]=f[0][i],f[0][i]=f[t][i];
                dp(0,n+1);
                if (top) Idp(0,st[top]);
            }
            printf("%d\n",(f[0][s]+mod)%mod);
        } else {
            s=gi(),t=gi();
            if (fa[t]==s) s=t;

            st[top=1]=s;
            for (k=fa[s];top<M&&k;k=fa[k]) {
                st[++top]=k;
                if (o[k]) break;
            }
            for (i=top;i>2;i--)
                Idp(st[i],st[i-1]);
            if (o[s])
                dp(st[2],st[1]);
            else
                Idp(st[2],st[1]);
            for (i=3;i<=top;i++)
                dp(st[i],st[i-1]);
            o[s]^=1;
        }
    return 0;
}
```