DP（Dynamic Programming）

- 最大上升子序列LIS（Longest Increasing Subsequence）
- 最长公共子串LCS（Longest CommonSubstring）
- 最长回文子序列
- 补充知识：偏序集
- 最大子矩阵问题
  - 枚举边界
  - 悬线法
  - 入门题
- 区间dp
  - 2020icpc昆明C题（序列自动机优化）
  - 插一嘴
- 树上DP
  - 求树上最大匹配及其方案数
- 状压DP
  - 常见位操作
  - 入门题
- 数位DP
  - 常用优化
  - 有前导零的数位DP
  - 2020上海站C题Sum of Log 链接
  - 维护平方和
- SOSdp
- 换根DP
- 斜率DP

DP（Dynamic Programming）

最大上升子序列LIS（Longest Increasing Subsequence）

> (1) O(n^2)

```
#include<iostream>
#include<algorithm>
#define ll long long
using namespace std;
const ll maxn=100;
ll lis(ll n){
  ll list[maxn];
  ll cnt[maxn];
  for(ll i=0;i<n;i++)
  {
    cin>>list[i];
  }
  for(ll i=0;i<n;i++)
  {
    cnt[i]=1;
    for(ll j=0;j<i;j++)
    {
      if(list[i]>list[j]&&cnt[j]+1>cnt[i])
        cnt[i]=cnt[j]+1;
    }
  }
  ll maxlen=0;
  for(ll i=0;i<n;i++)
  {
    maxlen=max(maxlen,cnt[i]);
  }
  return maxlen;
}
```

> (2) O $(n\log_2 n)$

```
#include<iostream>
#include<algorithm>
#include<cstdio>
using namespace std;
#define ll long long
const int maxn=1e5+10;
int dp[maxn],num[maxn];
int main()
{
  int n;
```

```
  cin>>n;
  for(int i=1;i<=n;i++)
  {
    cin>>num[i];
  }
  int len=1;
  dp[1]=num[1];
  for(int i=2;i<=n;i++)
    {
      if(num[i]<=dp[len])
        {
          int pos=lower_bound(dp+1, dp+len, num[i])-dp;
              //二分查找大于等于list[i]的第一个值的位置
          dp[pos]=num[i]; //代替掉第一个大于它的元素
        }
      else {
        dp[++len]=num[i];
      }
    }
  cout<<len;

}
```

其中dp数组不能存储LCS的内容，只用于替换LCS中的内容，它只是存储的对应长度LIS的最小末尾。

如输入：

```
7
1 2 4 3 2 5 0
```

dp为：

```
0 2 3 5 0 0 0
```

最后`0`更新了`dp[0]`的`1`.

最长公共子串LCS（Longest CommonSubstring）

> $O(n^2)$

Xi=＜x1，…，xi＞即X序列的前i个字符 (1≤i≤m)（前缀）

Yj=＜y1，…，yj＞即Y序列的前j个字符 (1≤j≤n)（前缀）

假定Z=＜z1，…，zk＞∈LCS(X , Y)。

若xm=yn（最后一个字符相同），则不难用反证法证明：该字符必是X与Y的任一最长公共子序列Z（设长度为k）的最后一个字符，即有zk = xm = yn 且显然有Zk-1∈LCS(Xm-1 , Yn-1)即Z的前缀Zk-1是Xm-1与Yn-1的最长公共子序列。此时，问题化归成求Xm-1与Yn-1的LCS（LCS(X , Y)的长度等于LCS(Xm-1 , Yn-1)的长度加1）。

若xm≠yn，则亦不难用反证法证明：要么Z∈LCS(Xm-1, Y)，要么Z∈LCS(X , Yn-1)。由于zk≠xm与zk≠yn其中至少有一个必成立，若zk≠xm则有Z∈LCS(Xm-1 , Y)，类似的，若zk≠yn 则有Z∈LCS(X , Yn-1)。此时，问题化归成求Xm-1与Y的LCS及X与Yn-1的LCS。LCS(X , Y)的长度为：max{LCS(Xm-1 , Y)的长度, LCS(X , Yn-1)的长度}。

```
#include<iostream>
#include<cstring>
#include<string>
#define ll long long
using namespace std;
const int maxn=2010;
int dp[maxn][maxn];
ll LCS(string & s1,string & s2)
{
    ll len1=s1.length();
    ll len2=s2.length();
    for(ll i=1;i<=len1;i++)
    {
        for(ll j=1;j<=len2;j++)
        {
            if(s1[i-1]==s2[j-1])
                dp[i][j]=dp[i-1][j-1]+1;
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    }
    return dp[len1][len2];
}
int main()
{
    string s1,s2;
    while(cin>>s1>>s2)
    {
        memset(dp,0,sizeof(dp));
        ll l1=s1.length();
        ll l2=s2.length();
        cout<<l1+l2-2*LCS(s1, s2)<<endl;
    }
```

```
}
```

> O(nlogn)

题目描述

给出 1,2,...,n 的两个排列 P1 和 P2，求它们的最长公共子序列。

输入格式

第一行是一个数 n。
接下来两行，每行为 n 个数，为自然数 1,2,...,n的一个排列。

输出格式

一个数，即最长公共子序列的长度。

输入样例

```
5
3 2 1 4 5
1 2 3 4 5
```

输出样例

```
3
```

[思路]

```
将LCS转化为LIS问题
首先，我们可以想到，最长公共子序列，就是两段所含数字完全一样，
并且数字的顺序也是完全一样的序列。
而顺序，我们可以想到类似哈希的思想，考虑建立一个类似map的关系数组f[ai]=i，
那么我们找到的序列只要是上升的，顺序就是一样的，然后考虑数字完全一样，
由于我们已经有了一个f[ai]=i,所以我们把对应的bi改成f[bi]，就可以确保数字相等了呀!

假设有两个序列 s1[ 1~6 ] = { a, b, c , a, d, c },
 s2[ 1~7 ] = { c, a, b, e, d, a, b }。

记录s1中每个元素在s2中出现的位置，再将位置按降序排列，则上面的例子可表示为：

loc( a) = { 6, 2 }, loc( b ) = { 7, 3 }, loc( c ) = { 1 }, loc( d ) = { 5 }。

（坐标一定要倒着插）

将s1中每个元素的位置按s1中元素的顺序排列成一个序列s3 = { 6, 2, 7, 3, 1, 6, 2, 5, 1 }。

在对s3求LIS得到的值即为求LCS的答案。
```

代码

```
#include<iostream>
#include<cstdio>
#include<string>
#include<vector>
#include<map>
#include<cstring>
#include<algorithm>
using namespace std;
#define int long long
#define IOFast() ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
map<char, vector<int> > m;
string s1,s2;
vector<int> sequence,dp;
void init(){
    dp.clear();
    sequence.clear();
    m.clear();
  int len1=s1.length();
  int len2=s2.length();
  for(int i=0;i<len1;i++){
    m[s1[i]].push_back(i);
  }
  for(int i=0;i<len2;i++){
    int len=m[s2[i]].size();
    for(int j=len-1;j>=0;j--)
      sequence.push_back(m[s2[i]][j]);
  }
}
void solve(){
  if(sequence.empty())
  {
    cout<<"0\n";
    return;
  }
  dp.push_back(sequence[0]);
  int top=0;
  int len=sequence.size();
  for(int i=1;i<len;i++)
  {
    if(sequence[i]>dp[top])
    {dp.push_back(sequence[i]);top++;continue;}
    int pos=lower_bound(dp.begin(),dp.end(),sequence[i])-dp.begin();
    dp[pos]=sequence[i];
  }
  cout<<top+1<<'\n';
}
signed main(){
```

```
  while(cin>>s1>>s2){
    init();
    solve();
  }
}
```

最长回文子序列

> 方法一：

原串倒叙和自己做LCS

> 方法二：

另dp[i][j]为区间内最长回文子串长度，若a[i]=a[j]，则dp[i][j]=max(dp[i][j],dp[i+1][j-1])，否则为max(dp[i+1][j],dp[i][j-1])，然后区间dp

补充知识：偏序集

设R为非空集合A上的关系，如果R是自反的、反对称的和可传递的，则称R为A上的偏序关系，简称偏序，通常记作≦。一个集合A与A上的偏序关系R一起叫作偏序集，记作<A,R>或<A, ≦>。其中（自反性）对任一，则x≦x；（反对称性）如果x≦y,且y≦x,则x=y；（传递性）如果x≦y,且y≦c ,则x≦c。

（1）自反性：a≤a，∀a∈P；

（2）反对称性：∀a，b∈P，若a≤b且b≤a，则a=b；

（3）传递性：∀a，b，c∈P，若a≤b且b≤c，则a≤c；

Dilworth定理：　偏序集能划分成的最少的全序集的个数与最大反链的元素个数相等。

Dilworth定理的对偶定理：对于一个偏序集，其最少反链划分数等于其最长链的长度。

对于导弹拦截问题：

也就是说把一个数列划分成最少的最长上升子序列的数目就等于这个数列的最长下降子序列的长度。

CSDN

最大子矩阵问题

【最大子矩阵问题】

在一个给定的矩形中有一些障碍点，找出内部不包含障碍点的、轮廓与整个矩形平行或重合的最大子矩形。

【定义子矩形】

有效子矩形：内部不包含障碍点的、轮廓与整个矩形平行或重合的子矩形。

极大子矩形：每条边都不能向外扩展的有效子矩形。

最大子矩形：所有有效子矩形中最大的一个（或多个）。

【极大化思想】

在一个有障碍点的矩形中最大子矩形一定是极大子矩形。

设计算法的思路：枚举所有的极大子矩形，找到最大子矩形。

设NM分别为整个矩形的长和宽，S为内部的障碍点数。

枚举边界

由于极大子矩形的每一条边都不能向外扩展，那么极大子矩阵的每条边要么覆盖了障碍点，要么与整个矩形的边界重合基本算法：枚举上下左右四个边界，然后判断组成的矩形是否是有效子矩形。

复杂度：O（S^5） 可以改进的地方：产生了大量的无效子矩形。

初步改进的算法：枚举左右边界，然后对处在边界内的点排序，每两个相邻的点和左右边界组成一个矩形。

复杂度：O（S^3） 可以改进的地方：枚举了部分不是极大子矩形的情况。

综上，设计算法的方向：

1、保证每一个枚举的矩形都是有效的。

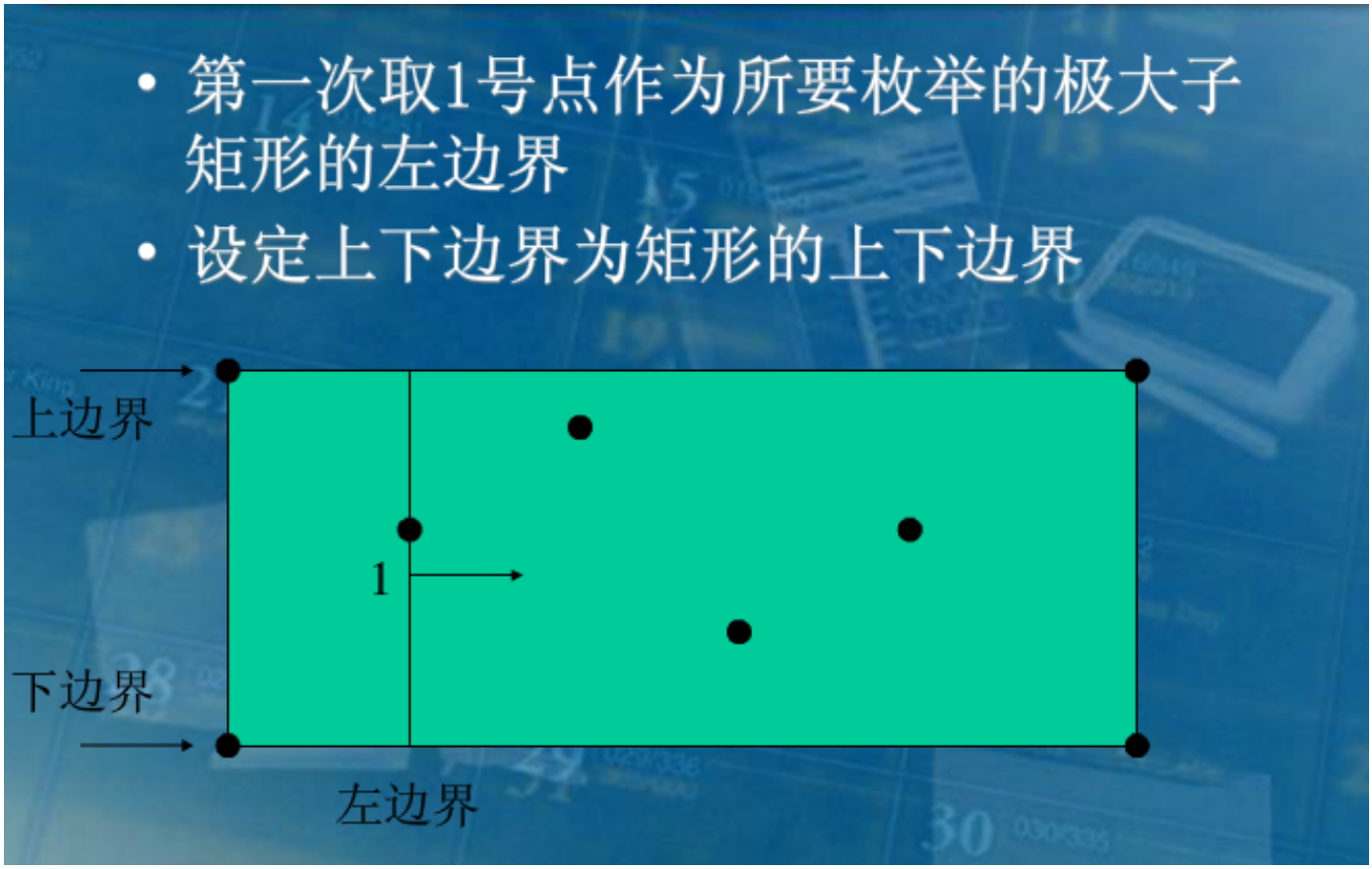
2、保证每一个枚举的矩形都是极大的。

算法的过程：

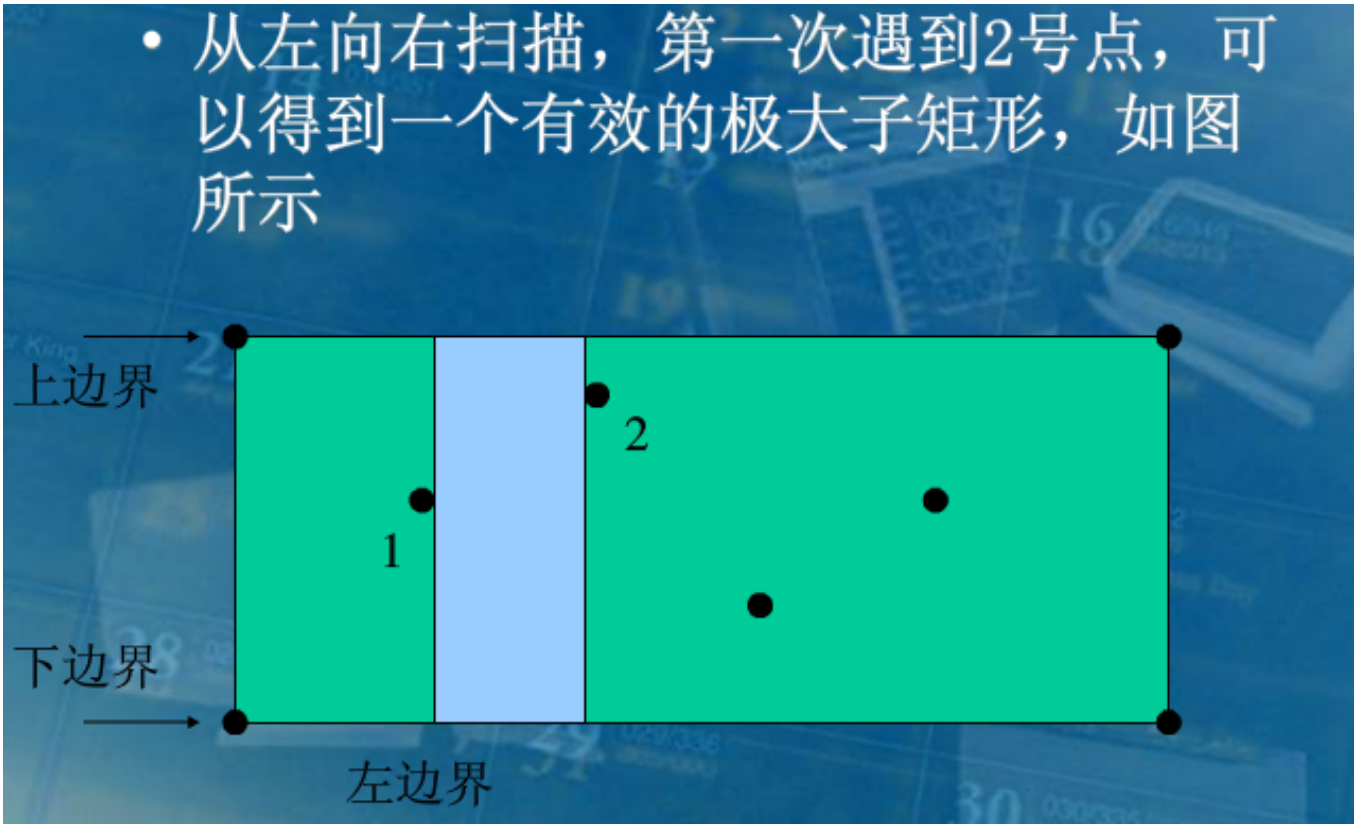
枚举极大子矩形的左边界——>根据确定的左边界，找出相关的极大子矩形——>检查和处理遗漏的情况

(1)按照横坐标从小到大的顺序将所有的点编号为1，2，3...

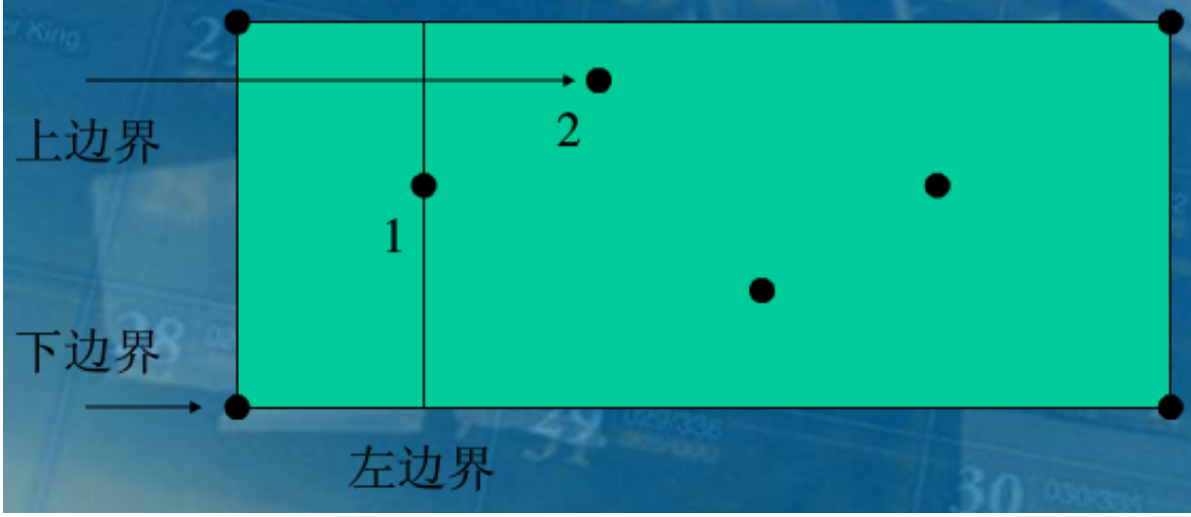
(2)首先选取1号点作为要枚举的极大子矩形的左边界，设定上下边界为矩形的上下边界



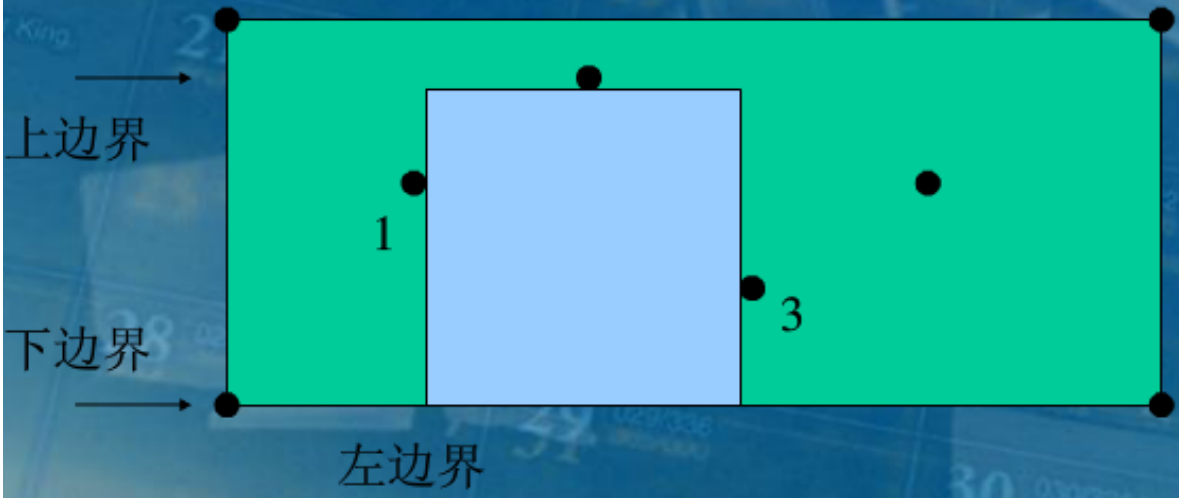
(3)从左到右扫描，第一次到2号点，确定一个极大子矩形，修改上下边界；第二次找到3号点，以此类推。

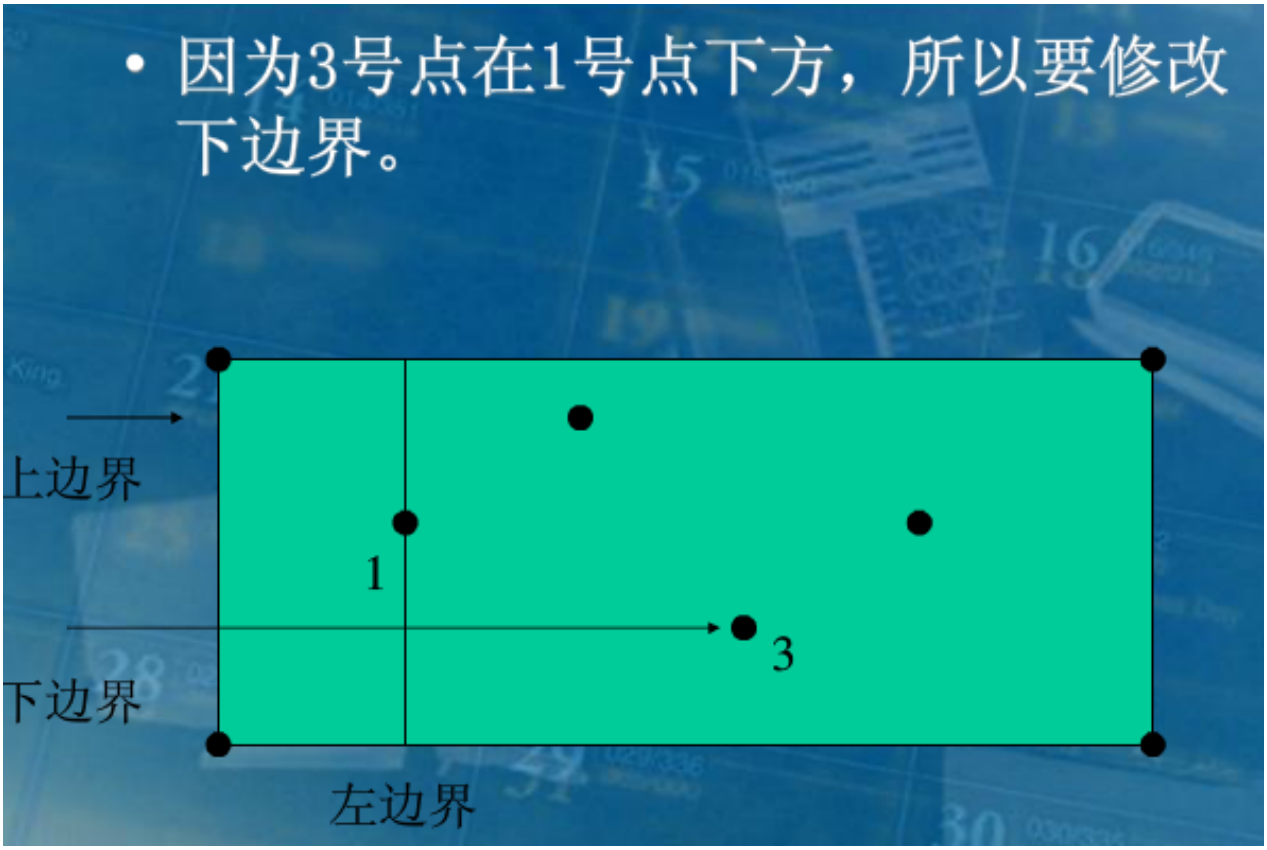


- 因为左边界覆盖1号点且右边界在2号点右边的有效子矩形都不能包含2号点，所以需要修改上下边界
- 2号点在1号点上方，因此要修改上边界

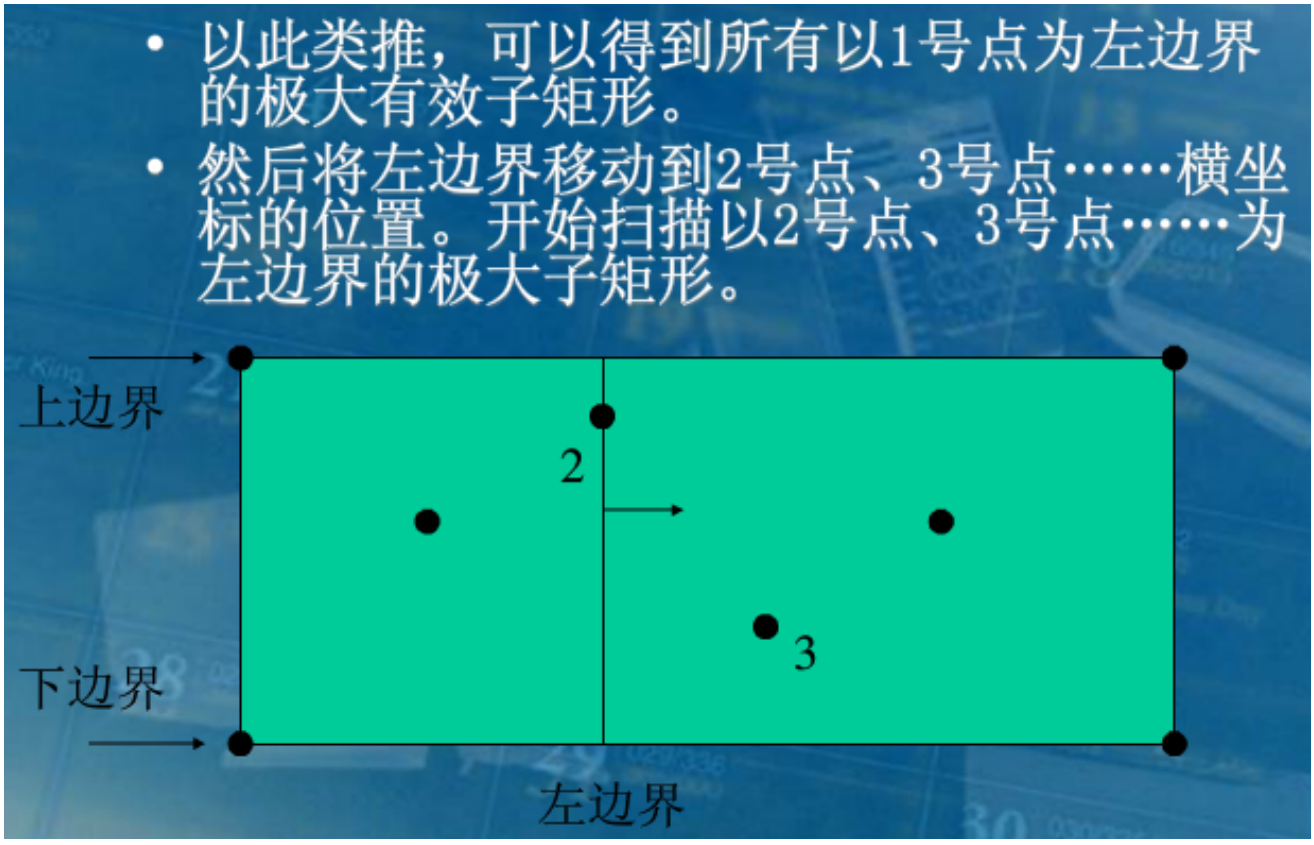


- 继续扫描到3号点，又得到一个极大有效子矩形，如图所示





(4)将左边界移动到2号点，3号点，，，以同样的方法枚举



遗漏的情况：

1、矩形的左边界与整个矩形的左边界重合。解决方法：用类似的方法从左到右扫一遍

2、矩形的左边界与整个矩形的左边界重合，且矩形的右边界与整个矩形的右边界重合。解决方法：预处理时增加特殊判断。

优点：利用的极大化思想，复杂度可以接受，编程实现简单。

缺点：使用有一定的局限性，不适合障碍点较密集的情况。

悬线法

时间复杂度O(NM) 空间复杂度O(NM)

> 用途

针对求给定矩阵中满足某条件的极大矩阵，比如“面积最大的长方形、正方形”、“周长最长的矩形等等”。

> 定义

有效竖线：除了两个端点外，不覆盖任何一个障碍点的竖直线段。

悬线：上端覆盖了一个障碍点或者到达整个矩形上边界的有效线段。

每个悬线都与它底部的点一一对应，矩形中的每一个点（矩形顶部的点除外）都对应了一个悬线。悬线的个数=(N-1)*M.

如果把一个极大子矩形按照横坐标的不同切割成多个与y轴平行的线段，那么其中至少有一个悬线。

如果把一个悬线向左右两个方向尽可能的移动，那么就得到了一个矩形，我们称它为悬线对应的矩形。

悬线对应的矩形不一定是极大子矩形，因为下边界可能还可以向下扩展。

> 设计算法

原理：所有悬线对应矩形的集合一定包含了极大子矩形的集合。

通过枚举所有的悬线，找出所有的极大子矩形。

算法规模：

悬线个数＝(N－1)×M

极大子矩形个数≤悬线个数

> 具体方法

设 H[i,j]为点(i,j)对应的悬线的长度。

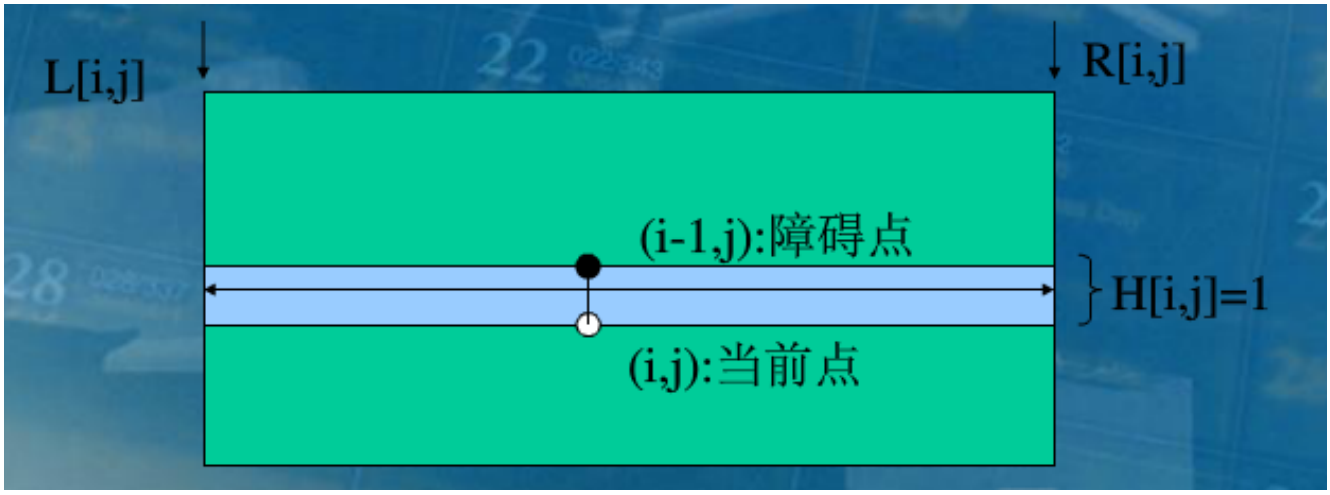
L[i,j]为点(i,j)对应的悬线向左最多能够移动到的位置。

R[i,j]为点(i,j)对应的悬线向右最多能够移动到的位置。

考虑点(i,j)对应的悬线与点(i-1,j)对应的悬线的关系（递推思想）：

如果(i－1,j)为障碍点，那么(i,j)对应的悬线长度1，左右能移动到的位置是整个矩形的左右边界。

即 H[i,j]＝1， L[i,j]=0，R[i,j]=m

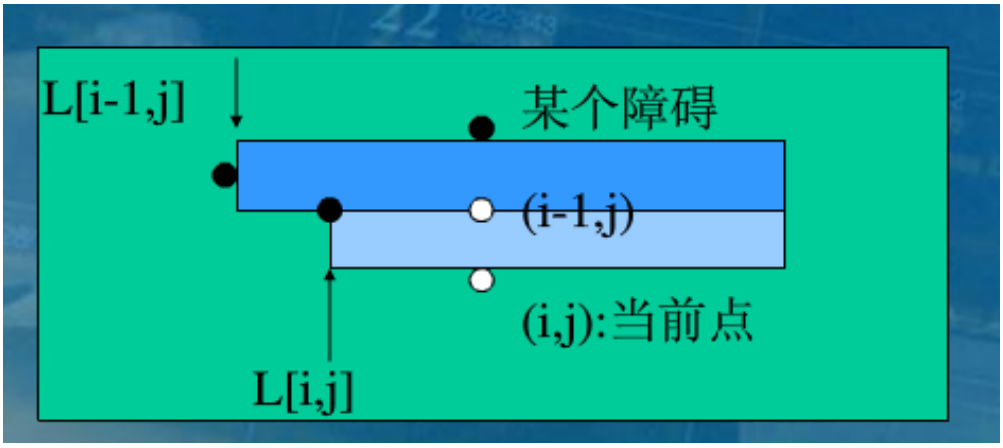


如果(i－1,j)不是障碍点，那么(i,j)对应的悬线长度为(i-1,j)对应的悬线长度＋1。

H[i,j]＝H[i-1,j]+1

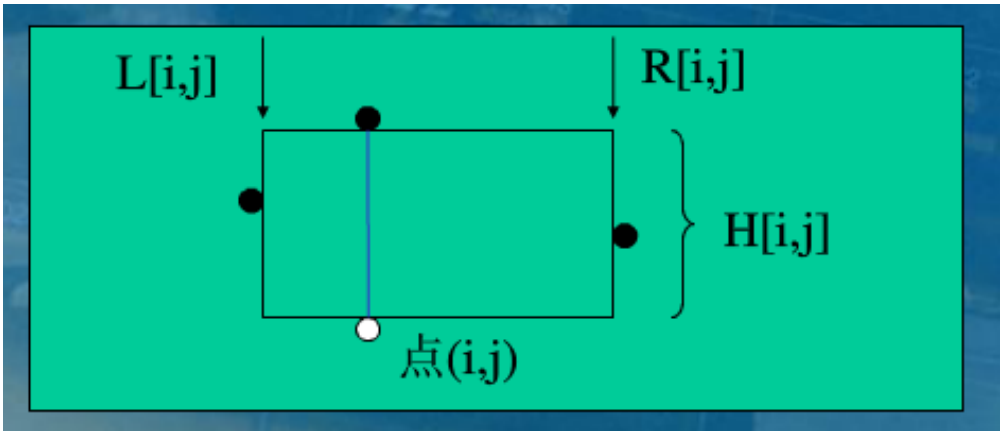
如果(i－1,j)不是障碍点，那么，如图所示，(i,j)对应的悬线左右能移动的位置要在(i-1,j)的基础上变化。

L[i-1,j]=max(L[i,j],(i-1,j)左边第一个障碍点的位置)



同理，也可以得到R[i,j]的递推式

R[i,j]=min(R[i-1,j], (i-1,j)右边第一个障碍点的位置)



优点： 复杂度与障碍点个数没有直接关系。

缺点：障碍点少时处理较复杂，不如算法1。

入门题

[棋盘制作]

大意：求黑白交错的最大正方形和长方形

```
#include<iostream>
#include<cstdio>
#include<string>
#include<vector>
#include<cmath>
#include<stack>
#include<queue>
#include<map>
#include<unordered_map>
#include<set>
#include<cstring>
#include<algorithm>
#include<climits>
#include<numeric>
#define int long long
#define pii pair<int,int>
#define IOFast() ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
using namespace std;
const int maxn=2e3+10;
const int inf=9999999999999999ll;
int n,m;
int mp[maxn][maxn];
int up[maxn][maxn];
int left_[maxn][maxn];
int right_[maxn][maxn];
int ans1,ans2;
void init(){
  cin>>n>>m;
  for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
      {
        cin>>mp[i][j];
        if((i+j)&1){
          mp[i][j]=!mp[i][j];
        }
      }
}
void calc(int x){
  int lz,rz;
  memset(up,0,sizeof up);
  memset(left_,0x3f,sizeof left_);
  memset(right_,0x3f,sizeof right_);
```

```
  for(int i=1;i<=n;i++){
    lz=0;
    for(int j=1;j<=m;j++){
      if(mp[i][j]==x)
        lz=j;
      else
      {
        up[i][j]=up[i-1][j]+1;
        left_[i][j]=min(left_[i-1][j],j-lz);
      }
    }
  }
  for(int i=1;i<=n;i++){
    rz=m+1;
    for(int j=m;j>=1;j--){
      if(mp[i][j]==x)
          rz=j;
      else
      {
        right_[i][j]=min(right_[i-1][j],rz-j);
      }
    }
  }
  for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
    {
      ans1=max(ans1,up[i][j]*(right_[i][j]+left_[i][j]-1ll));
      ans2=max(ans2,min(up[i][j],right_[i][j]+left_[i][j]-1ll)*min(up[i][j],right_[i]
[j]+left_[i][j]-1ll));
    }
  return;
}
void solve(){
  calc(0);
  calc(1);
  cout<<ans2<<endl<<ans1<<endl;
}
signed main(){
  IOFast();
  init();
  solve();
}
```

区间dp

伪代码

```
int dp[i][j]; 区间i到j的最佳答案
for （枚举区间长度）
  for(枚举左端点)
    for(枚举中间分割端点)
```

2020icpc昆明C题（序列自动机优化）

Bob lives in a chaotic country with n cities in a row, numbered from 1 to n. These cities are owned by different lords, and the i-th cities currently belongs to the ai-th lord. To simply problems, we assume there are n lords in the country, and they are also numbered from 1 to n. Some lords may take control of multiple cities, while some new-born lords have not got any cities yet.

Obviously, the greedy lords are not satisfied with the number of territories they have, so the country is constantly at war. Bob wants to change that, by making all the cities belong to the same lord!

Bob can perform some magical operations to support his grand plan. With the help of each magic, Bob can do the following:

Choose some cities with consecutive indices such that they belong to the same lord, and assign them to any other lord.

As magics are really tiring, Bob wants to know the minimum number of such operations he needs to use to make all the cities belong to one lord.

The following picture shows an example where n=6. Different shapes are used to represent cities belonging to different lords. As shown in the picture, the minimum number of magic operations used is 2.

(currently no Lord will have more than 15 cities, which means no one a i will appear more than 15 times in this line.)

```
#include<iostream>
#include<cstdio>
#include<map>
#include<cstring>
using namespace std;
const int maxn=5e3+10;
const int inf=99999999;
int pre[maxn];
int dp[maxn][maxn];
int a[maxn];
int n;
int top=1;
int pos[maxn];


int dfs(int l,int r)
{
    if(l>=r) return 0;
    if(dp[l][r]!=inf) return dp[l][r];
    int ans=r-l;
```

```
    ans=min(ans,dfs(l+1,r)+1);
    ans=min(ans,dfs(l,r-1)+1);
    if(a[l]==a[r])
        {
            ans=min(ans,dfs(l+1,r-1)+1);
        }
    for(int k=pre[r];k>l;k=pre[k])
        {
            ans=min(ans,dfs(l,k-1)+dfs(k,r)+1);
            if(a[l]==a[r])
                {
                    ans=min(ans,dfs(l,k-1)+dfs(k+1,r));
                }
        }
    return dp[l][r]=ans;
}
void solve()
{
    int cnt=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++){
        cin>>a[i];
        if(a[i]!=a[i-1]){
            pre[i]=pos[a[i]];
            pos[a[i]]=i;
        }
        else
        {
            i--;
            n--;
        }
    }
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            dp[i][j]=inf;
    int ans=dfs(1,n);
    cout<<ans<<endl;
}

int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        solve();
        memset(pos, 0, sizeof(pos));
        memset(pre, 0, sizeof(pre));
    }
```

```
}
```

插一嘴

> 合并石子是经典的区间DP,下方为求环型摆放的石头，在合并时的最大、最小值求法，弱化版题链接

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
const int maxn=1e2+10;
const int inf=99999999;
int a[2*maxn],dp1[2*maxn][2*maxn],dp2[2*maxn][2*maxn],sum[2*maxn];
int main()
{
    int N;
    cin>>N;
    for(int i=1;i<=N;i++)
    {
        scanf("%d",&a[i]);
        a[i+N]=a[i];
        sum[i]=sum[i-1]+a[i];
    }
    for(int i=N+1;i<=2*N;i++)
    {
        sum[i]=sum[i-1]+a[i];
    }
    for(int len=1;len<N;len++)
    {
        for(int i=1,j=len+i;(i<=2*N)&&(j<=2*N);i++,j=i+len)
        {
            dp2[i][j]=inf;
            for(int k=i;k<j;k++)
            {
               dp1[i][j]=max(dp1[i][j],dp1[i][k]+dp1[k+1][j]+sum[j]-sum[i-1]);
               dp2[i][j]=min(dp2[i][j],dp2[i][k]+dp2[k+1][j]+sum[j]-sum[i-1]);
            }
        }
    }
    int ans1=0,ans2=inf;
    for(int i=1;i<=N;i++)
    {
        ans1=max(ans1,dp1[i][i+N-1]);
        ans2=min(ans2,dp2[i][i+N-1]);
    }
    printf("%d\n%d",ans2,ans1);
}
```

> 但当石头数量多到一定程度后，$O(n^3)$显然过不了，但有一种O(n)冲过去的算法

> GarsiaWachs算法
>
> 这个算法可以用来解决石子合并问题(大数据版)
>
> 对于一堆石子（链式）：
>
> 1.从最左边开始向右走直到 stone[k] <= stone[k + 2]
>
> 然后合并 stone[k + 1] += stone[k];
>
> 2.从k 往前走 直到 stone[j] > stone[k] + stone[k + 1]　然后将新合并的数放在 j 后面
>
> 如果不存在这样的 j ，就放在最前面
>
> 3.如果我们找不到这样的 k , 就合并最后的两个数即 stone[n] 和 stone[n - 1]
>
> （显然这两个数是最小的)
>
> 4.重复第一步

图例

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

	下标:	1	2	3	4	5	6	
	石子:	18	5	14	1	4	8	
1. 查找 发现 k = 4，则合并		18	5	14	_	5	8	d = 1 + 4 = 5　ans = ans + d = 5
2.移位覆盖第四位			18	5	14	5	8	left = 2
3.从第4位向左查找-》j = 4 不改变			18	5	14	5	8	
4.继续查找-》 k = 3 合并			18	_	19	5	8	d = 5 + 14 = 19　ans = ans + d = 24
5.移位覆盖第三位				18	19	5	8	left = 3
6.从第三位查找 18<19 交换				19	18	5	8	
7.继续查找-》 k 不存在 合并最后两位				19	18	13	_	d = 5 + 8 = 13　ans = ans + d = 37
8.继续查找-》 k 不存在 合并最后两位				19	31	_		d = 18 + 13 = 31　ans = ans + d = 68

9.只剩两堆，ans = ans + 19 + 31 = 118 即为答案

代码

```
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;
const int N = 4e4 + 10;
int n;
int stone[N];
int main()
{
    cin >> n;
    for(int i = 1; i <= n; i++) scanf("%d", &stone[i]);
```

```
    //left 代表最左边的下标，  ans 记录答案
    int ans = 0, left = 1;
    //当至少存在三堆石子时，  即 left n-1 n
    while(left < n - 1)
    {
        int k;
        for(k = left; k < n - 1; k++)
        {
            if(stone[k] <= stone[k+2])
            {
                stone[k+1] += stone[k]; //合并
                ans += stone[k+1];      //加到答案里
                // k 左边的往右移一位 将 k 覆盖掉
                for(int j = k; j > left; j--)   stone[j] = stone[j - 1];
                left ++;    //由于移位所以最左边要加 1

                //向左查找 也就是第二步
                int j = k + 1;
                while(stone[j] > stone[j-1] && j > left)
                {
                    //交换，将新合并石子的前移
                    swap(stone[j], stone[j-1]);
                    j --;
                }
                break;
            }
        }
        //不存在这样的 k
        if(k == n - 1)
        {
            //将最右边的石子加到倒数第二堆，
            stone[n - 1] += stone[n];
            //加上目前最右边的石子，最右边的石子已经不存在了 所以 --n
            ans += stone[--n];
        }
    }
    //加上还剩下的两堆石子
    ans += stone[n] + stone[n-1];
    cout << ans << endl;
    return 0;
}
```

> 这个问题好像还叫“最优树”

树上DP

> 树上DP的dp数组中，第一维通常表示以root为根的子树，第二维通常表示该子树中根节点是否有做某个操作或 该子树中某个状态的数值

> 遍历树的方法需要根据存树的方式而定，但基本都是dfs遍历

例题1链接

题目描述

有一棵苹果树，如果树枝有分叉，一定是分2叉（就是说没有只有1个儿子的结点）

这棵树共有N个结点（叶子点或者树枝分叉点），编号为1-N,树根编号一定是1。

我们用一根树枝两端连接的结点的编号来描述一根树枝的位置。下面是一颗有4个树枝的树

```
2   5
 \ /
  3   4
   \ /
    1
```

现在这颗树枝条太多了，需要剪枝。但是一些树枝上长有苹果。

给定需要保留的树枝数量，求出最多能留住多少苹果。

输入格式

第1行2个数，N和Q(1<=Q<= N,1<N<=100)。

N表示树的结点数，Q表示要保留的树枝数量。接下来N-1行描述树枝的信息。

每行3个整数，前两个是它连接的结点的编号。第3个数是这根树枝上苹果的数量。

每根树枝上的苹果不超过30000个。

输出格式

一个数，最多能留住的苹果的数量。

```
#include <iostream>
#include<vector>
#include<cstring>
#include<fstream>
const int maxn=6e3+10;
using namespace std;
#define ll long long
int tot;
int n,q;
int head[maxn];

struct edge{
```

```
  int w,to,next;
}e[maxn<<1];
//链式前向星存储
inline void add(int u,int v,int w)
{
  e[tot]={w,v,head[u]};
  head[u]=tot++;
}
int sz[maxn];
int dp[maxn][maxn];
void dfs(int st,int fa)
{
  for(int i=head[st];~i;i=e[i].next)
  {
    int v=e[i].to;
    if(fa==v) continue;
    dfs(v,st);
    sz[st]+=sz[v]+1;
    for(int j=min(sz[st],q);j>=1;j--)
    {
      for(int k=min(sz[v],j-1);k>=0;k--)
      {
        dp[st][j]=max(dp[st][j],dp[st][j-k-1]+dp[v][k]+e[i].w);
      }
    }
  }
}
int main() {

  cin>>n>>q;
  memset(head, -1, sizeof(head));
  for(int i=1;i<n;i++)
  {
    int l,r,cost;
    cin>>l>>r>>cost;
    add(l,r,cost);
    add(r,l,cost);
  }
  dfs(1,-1);
  cout<<dp[1][q];
}
```

例题2链接

你的任务是要对一棵二叉树的节点进行染色。每个节点可以被染成红色、绿色或蓝色。并且，一个节点与其子节点的颜色必须不同，如果该节点有两个子节点，那么这两个子节点的颜色也必须不同。给定一颗二叉树的二叉树序列，请求出这棵树中最多和最少有多少个点能够被染成绿色。

输入格式

输入只有一行一个字符串s，表示二叉树序列。

输出格式

输出只有一行，包含两个数，依次表示最多和最少有多少个点能够被染成绿色。

```
#include<iostream>
#include<cstdio>
#include<string>
#include<cstring>
using namespace std;
const int maxn=2e6+10;
int leftchild[maxn],rightchild[maxn];
char s[maxn];     //1开始存图
int tot;
int dp[maxn][2];    //0表示不用绿色，1表示用绿色
void dfs(int root)
{
  tot++;
  if(s[root]=='\0') return;
  if(s[root]=='1')
  {
    leftchild[root]=root+1;
    dfs(root+1);
  }
  if(s[root]=='2')
  {
    leftchild[root]=root+1;
    dfs(root+1);
    rightchild[root]=tot+1;    //右子树根节点为当前总访问点的下一个
    dfs(tot+1);
  }
}
int main()
{
  scanf("%s",s+1);
  int len=strlen(s+1);
  dfs(1);
  for(int i=len;i>=1;i--)
  {
      dp[i][1]=dp[leftchild[i]][0]+dp[rightchild[i]][0]+1;
      dp[i][0]=max(dp[leftchild[i]][1]+dp[rightchild[i]][0],
      dp[leftchild[i]][0]+dp[rightchild[i]][1]);
  }
  cout<<max(dp[1][0],dp[1][1]);
  for(int i=len;i>=1;i--)
    {
        dp[i][1]=dp[leftchild[i]][0]+dp[rightchild[i]][0]+1;
        dp[i][0]=min(dp[leftchild[i]][1]+dp[rightchild[i]][0],
        dp[leftchild[i]][0]+dp[rightchild[i]][1]);
```

```
    }
  cout<<' '<<min(dp[1][0],dp[1][1]);
}
```

求树上最大匹配及其方案数

dp[0][u]表示u子树不使用u点时的最大匹配，dp[1][u]表示u子树使用u时的最大匹配

$$dp[0][u]=\sum max(dp[0][v],dp[1][v])$$
$$dp[1][u]=max(dp[0][u]-max(dp[0][v],dp[1][v])+dp[0][v]+1)$$

求dp[1][u]之前需要先算出dp[0][u]，然后在消除v的影响之后，再补上dp[0][v]+1

```
int n;
vc < vi > G;
const int N = 1e3 + 20;
int dp[2][N];
int plan[2][N];
int h[N];

void input(){
  cin >> n;
  G.resize(n + 1);
  forn(i, n){
    int from;
    cin>>from;
    int cnt;
    cin >> cnt;
    forn(j, cnt){
      int to;
      cin >> to;
      G[from].push_back(to);
      G[to].push_back(from);
    }
  }
}

void dfs(int u, int fa){
  plan[0][u]=1;
  dp[0][u]=dp[1][u]=0;
  for(int i = 0; i < G[u].size();  ++ i){
    int v = G[u][i];
    if(v == fa) continue;
    dfs(v,u);
    dp[0][u] += max(dp[1][v], dp[0][v]);
    plan[0][u]=plan[0][u]*h[v];
  }
  for(int i = 0; i < G[u].size();  ++ i){
    int v = G[u][i];
```

```
    if(v == fa) continue;
    dp[1][u] = max(dp[1][u],dp[0][u] - max(dp[0][v], dp[1][v]) + dp[0][v] + 1);
  }
  for(int i = 0; i < G[u].size();  ++ i){
    int v = G[u][i];
    if(v == fa) continue;
    if(h[v]!=0){
      int now = dp[0][u] - max(dp[0][v], dp[1][v]) + dp[0][v] + 1;
      if(now==dp[1][u]){
        plan[1][u] += plan[0][u] / h[v] * plan[0][v];
      }
    }
  }
  if(dp[1][u]==dp[0][u]){
    h[u]=plan[0][u]+plan[1][u];
  }else if(dp[1][u]>dp[0][u])
    h[u]=plan[1][u];
  else h[u]=plan[0][u];
}

signed main(){
  IOFast();
  input();
  dfs(1, -1);
  cout << max(dp[1][1], dp[0][1]) << '\n';
  cout << (dp[1][1] == dp[0][1]?(plan[0][1] + plan[1][1]):(dp[1][1]>dp[0][1]?plan[1]
[1]:plan[0][1]));
}
```

状压DP

> 状压dp过程中需要枚举每一种排列情况来进行dp，代码里通常会套一堆for循环

> dp数组通常有一维存代表组合情况的二进制数，如果是地图，通常还会有一维存行数或列数

> 行数和列数中较大者存在dp数组中，并在dp过程中枚举；较小者枚举排列情况，情况数是1<<m

常见位操作

> 1.取出x的第i位:

```
int main()
{
    y = (x>>(i-1))&1;
    return 0;
}
```

> 2.将x第i位取反:

```
int main()
{
    x ^= (1<<(i-1));
    return 0;
}
```

> 3.将x第i位变为1：

```
int main()
{
    x |= (1<<(i-1));
    return 0;
}
```

> 4.将x第i位变为0：

```
int main()
{
    x &= (~(1<<(i-1)));
    return 0;
}
```

> 5.将x最靠右的1去掉：

```
int main()
{
    x = x&(x-1);
    return 0;
}
```

> 6.取出x最靠右的1:

```
int main()
{
    y = x&(-x);
    return 0;
}
```

> 7.判断是否有两个连续的一:

```
int main()
{
    if(x&(x<<1)) cout<<"YES";
    return 0;
}
```

> 8.枚举子集:

```
int main()
{
  //输出使用sta的二进制数中的1，能够组成的所有数
  //或者说是sta的二进制数中去掉任意个1的所有情况，除了0
  //比如1100输出1100、1000、0100
    for( int x = sta ; x ;  x = ( ( x - 1 )&sta) )
      cout<<x;
    return 0;
}
```

> 最重要的：统计一的个数

```
int table[16] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
int popcount(unsigned int x) { // 返回x的二进制中1的个数
    int ret = 0;
    for(int i = 0; i < 8; i++) ret += table[x & 15], x >>= 4;
    return ret;
}
或
int count_one(int x)//x的二进制中第0到n-1位中1的个数
{
    int cnt = 0;
    for(int i = 0; i <= n - 1; ++i)   //n为最高位
```

```
        if(x & (1 << i))//注意做完按位与后的结果不是1，如可能为000010000，所以不要写成x & (1 <<
i) == 1
            cnt++;
    return cnt;
}
int bsrun(int n){
    int tmp=n - ((n>>1) &033333333333)-((n>>2) &011111111111);
    return((tmp+(tmp>>3)) &030707070707) %63;
}
```

> 异或运算的性质

```
x xor y xor y = x
x xor y = k -> x xor k = y / y = x xor k
```

入门题

在N×N的棋盘里面放K个国王，使他们互不攻击，共有多少种摆放方案。国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共8个格子。

```
#include<cstdio>
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
#define ll long long
const int maxn=515;
ll dp[15][maxn][maxn];    //不能滚动数组优化
int n,k,cnt;
ll ans=0;
int num[maxn],s[maxn];
int table[16] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
inline int popcount(unsigned int x) {
    int ret = 0;
    for(int i = 0; i < 8; i++) ret += table[x & 15], x >>= 4;
    return ret;
}
inline void init()
{
  cnt=0;
  for(int i=0;i<(1<<n);i++)
  {
    if(i&(i<<1))  continue;
        int sum=0;
        for(int j=0;j<n;++j)
            if(i&(1<<j))++sum;
```

```
        num[++cnt]=sum;
    //num[++cnt]=popcount(i);
    s[cnt]=i;
  }
  return;
}
inline void solve()
{
//  for(int i=1;i<=cnt;i++)
//  {
//    dp[1][s[i]][num[i]]=1;
//  }
    dp[0][1][0]=1;
  for(int i=1;i<=n;i++)        //枚举行
    for(int j=1;j<=cnt;j++)      //枚举有效国王排列情况
    //考虑正确性：因为外层循环枚举状态是从小到大枚举，
    //所以保证当前状态某一位少1(即当前使用硬币数减一)的状态已经被转移过了
      for(int t=k;t>=0;t--)    //枚举国王数，从0到k也可以
      {
        if(t>=num[j])
        {
          for(int J=1;J<=cnt;J++)
          {
            if(!(s[j]&s[J])&&!(s[j]&(s[J]<<1))&&!(s[j]&(s[J]>>1)))
            //从上往下更新没有相邻的
              dp[i][j][t]+=dp[i-1][J][t-num[j]];
          }
        }
      }
  for(int i=1;i<=cnt;i++)
  {
    ans+=dp[n][i][k];
  }
  cout<<ans;
  return;
}
int main()
{
  cin>>n>>k;
  init();
  solve();
}
```

> 此题中只需考虑上下左右一层，若要考虑上两层没有连着的情况，则dp中需要多开一维保存上一层的情况

数位DP

> 通常第一维存数位，第二位存状态

```
int main()
{
    long long le,ri;
    while(~scanf("%lld%lld",&le,&ri))
        printf("%lld\n",solve(ri)-solve(le-1));
}
```

> 模板

```
typedef long long ll;
int a[20];
ll dp[20][state];//不同题目状态不同
ll dfs(int pos,/*state变量*/,bool lead/*前导零*/,bool limit/*数位上界变量*/)
//不是每个题都要判断前导零，当答案与二进制中0的数量有关时，就需要前导零参数
{
    //递归边界，既然是按位枚举，最低位是0，那么pos==0说明这个数我枚举完了,是符合条件的
    if(pos==0) return 1;/*这里一般返回1，表示你枚举的这个数是合法的，
    那么这里就需要你在枚举时必须每一位都要满足题目条件，也就是说当前枚举到pos位，
    一定要保证前面已经枚举的数位是合法的。不过具体题目不同或者写法不同的话不一定要返回1 */
    //第二个就是记忆化(在此前可能不同题目还能有一些剪枝)
    if(!limit && !lead && dp[pos][state]!=-1) return dp[pos][state];
    /*常规写法都是在没有限制的条件记忆化，这里与下面记录状态是对应的*/
    int up=limit?a[pos]:9;//根据limit判断枚举的上界up;这个的例子前面用213讲过了
    ll ans=0;
    //开始计数
    for(int i=0;i<=up;i++)//枚举，然后把不同情况的个数加到ans就可以了
    {
        if() ...
        else if()...
        ans+=dfs(pos-1,/*状态转移*/,lead && i==0,limit && i==a[pos])
         //最后两个变量传参都是这样写的
        /*这里还算比较灵活，不过做几个题就觉得这里也是套路了
        大概就是说，我当前数位枚举的数是i，然后根据题目的约束条件分类讨论
        去计算不同情况下的个数，还有要根据state变量来保证i的合法性，比如题目
        要求数位上不能有62连续出现,那么就是state就是要保存前一位pre,然后分类，
        前一位如果是6那么这意味就不能是2，这里一定要保存枚举的这个数是合法*/
    }
    //计算完，记录状态
    if(!limit && !lead) dp[pos][state]=ans;
    /*这里对应上面的记忆化，在一定条件下时记录，保证一致性，
    当然如果约束条件不需要考虑lead，这里就是lead就完全不用考虑了*/
    return ans;
}
ll solve(ll x)
```

```
{
    int pos=0;
    while(x)//把数位都分解出来
    {
        a[++pos]=x%10;//编号为[1,pos)
        x/=10;
    }
    return dfs(pos-1/*从最高位开始枚举*/,/*一系列状态 */,true,true);
    //刚开始最高位都是有限制并且有前导零的，显然比最高位还要高的一位视为0嘛
}
int main()
{
    ll le,ri;
    while(~scanf("%lld%lld",&le,&ri))
    {
        //初始化dp数组为-1,这里还有更加优美的优化,后面讲
        printf("%lld\n",solve(ri)-solve(le-1));
    }
}
```

要考虑0是否在solve中被重复计算多次，和值相关的数位dp，如果数据范围不大，可以在dp数组中开出一维存

常用优化

> 第一:memset(dp,-1,sizeof dp);放在多组数据外面。

这一点是一个数位特点，使用的条件是：约束条件是每个数自身的属性，而与输入无关。

不过说几个我遇到的简单的约束：

1.求数位和是10的倍数的个数,这里简化为数位sum%10这个状态，即dp[pos][sum]这里10 是与多组无关的，所以可以memset优化，不过注意如果题目的模是输入的话那就不能这样了。

2.求二进制1的数量与0的数量相等的个数，这个也是数自身的属性。

下面介绍的方法就是要行memset优化，把不满足前提的通过修改，然后优化。
介绍之前,先说一种较为笨拙的修改，那就是增加状态，前面讲limit的地方说增加一维dp[pos][state][limit]，能把不同情况下状态分别记录(不过这个不能memset放外面)。基于这个思想，我们考虑：约束为数位是p的倍数的个数，其中p数输入的，这和上面sum%10类似，但是dp[pos][sum]显然已经不行了，每次p可能都不一样，为了强行把memset提到外面加状态dp[pos][sum][p]，对于每个不同p分别保存对应的状态。这里前提就比较简单了，你dp数组必须合法，p太大就G_G了。所以对于与输入有关的约束都可以强行增加状态(这并不代表能ac，如果题目数据少的话就随便你乱搞了)

> 第二：相减

题目给了个f(x)的定义：F(x) = An * 2^{n-1} + An-1 * 2^{n-2}+ ... + A2 * 2 + A1 * 1，Ai是十进制数位，然后给出a,b求区间[0,b]内满足f(i)<=f(a)的i的个数。

常规想：这个f(x)计算就和数位计算是一样的，就是加了权值，所以dp[pos][sum]，这状态是基本的。a是题目给定的，f(a)是变化的不过f(a)最大好像是4600的样子。如果要memset优化就要加一维存f(a)的不同取值，那就是dp[10][4600][4600]，这显然不合法。
这个时候就要用减法了，dp[pos][sum]，sum不是存当前枚举的数的前缀和(加权的)，而是枚举到当前pos位，后

面还需要凑sum的权值和的个数。

意思大概就是第二维表示与上限值的差值，而这个差值与累加的最初值是多少无关，仅与这个差值本身有关。

> 普通加法，需要每次询问都memset

```
int dfs(int pos,int sum,bool limit)
{
    if(pos==0)    return 1;
    if(!limit&&~dp[pos][sum]) return dp[pos][sum];
    int up=limit?digit[pos]:9;
    int temp=0;
    for(int i=0;i<=up;i++)
    {
        if(sum+i*(1<<(pos-1))>need)
            break;
        temp+=dfs(pos-1,sum+i*(1<<(pos-1)),limit&&i==digit[pos]);
    }
    if(!limit)    dp[pos][sum]=temp;
    return temp;
}
int solve(int x){
    int pos=1;
    while(x)
    {
        digit[pos++]=x%10;
        x/=10;
    }
    return dfs(pos-1,0,true);
}
```

> 减法优化

```
int dfs(int pos,int dif,bool limit)
{
  if(pos==0)  return 1;
  if(!limit&&~dp[pos][dif]) return dp[pos][dif];
  int up=limit?digit[pos]:9;
  int temp=0;
  for(int i=0;i<=up;i++)
  {
    if(i*(1<<(pos-1))>dif)
      break;
    temp+=dfs(pos-1,dif-i*(1<<(pos-1)),limit&&i==digit[pos]);
  }
  if(!limit)  dp[pos][dif]=temp;
  return temp;
}
int solve(int x){
```

```
  int pos=1;
  while(x)
  {
    digit[pos++]=x%10;
    x/=10;
  }
  return dfs(pos-1,need,true);
}
```

有前导零的数位DP

例题 POJ 3252

这题的约束就是一个数的二进制中0的数量要不能少于1的数量，通过上一题，这题状态就很简单了，dp[pos][num],到当前数位pos,0的数量减去1的数量不少于num的方案数，一个简单的问题，中间某个pos位上num可能为负数(这不一定是非法的，因为我还没枚举完嘛，只要最终的num>=0才能判合法，中途某个pos就不一定了)，这里比较好处理，Hash嘛，最小就-32吧(好像),直接加上32，把32当0用。这题主要是要想讲一下lead的用法，显然我要统计0的数量，前导零是有影响的。至于!lead&&!limit才能dp，都是类似的，自己慢慢体会吧。

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<bitset>
#include<fstream>
using namespace std;
#define ll long long
int digit[100];
int dp[100][100];
ll dfs(int pos,int sta,bool lead,bool limit)
{
  if(pos==-1)
    return sta>=50;
  if(!limit && !lead && dp[pos][sta]!=-1) return dp[pos][sta];
  int up=limit?digit[pos]:1;
  int ans=0;
  for(int i=0;i<=up;i++)
  {
    if(lead && i==0)
      ans+=dfs(pos-1,sta,lead,limit && i==digit[pos]);
    else ans+=
    dfs(pos-1,sta+(i==0?1:-1),lead && i== 0 ,limit && i==digit[pos]);
  }
  if(!limit&&!lead) dp[pos][sta]=ans;
  return ans;
}
ll solve(int x)
{
  int pos=0;
```

```
  while(x)
  {
    digit[pos++]=x&1;
    x>>=1;
  }
  return dfs(pos-1,50,true,true);
}
int main()
{
  ll st,ed;
  memset(dp, -1, sizeof(dp));
  while(cin>>st>>ed)
  cout<<solve(ed)-solve(st-1)<<'\n';
}
```

链接

ps:三个心得:
1.memset比朴素循环快

2.STL的max比自己define的max要快

3.关于数位dp是否该省limit状态：(下面是自己意淫,可能会有很谜语人)

以往我们在一个维度上的dp的时候，有两种写法。一种写法就是不将limit这一位放入状态。然后在记忆化的时候加上 ! l i m i t !limit !limit的条件.

乍一看感觉这样的做法是会导致不完全的记忆化。导致重复计算.但是在一维的情况下, 在某一位 且 顶到上界 的情况是唯一的.所以不记忆化也是可以的.

但是在两个维度上的数位dp的时候(如这个问题),我们一定要把两个limit的状态都放入状态的.不然就会导致不完全的记忆化,产生重复计算.

为什么,因为 一个顶到了,一个没顶到 这样的状态我没记忆化.但是这样的情况显然不是唯一的啊...所以导致比赛的时候无限TLE...

所以的出来的结论是:无论什么情况下,将所有状态都放入dp数组是最保险的行为.

2020上海站C题Sum of Log 链接

$$\sum_{i=0}^{x}\sum_{j=[i=0]}^{y}[i\&j=0]\lfloor log_2(i+j)+1\rfloor$$

题解:很容易想到枚举i+j的位数，然后求再范围内的[i&j==0]的(i,j)对。

考虑用数位dp。记 f[k][l1][l2] 表示0-k为i&j=0的个数。用数位dp计算就行。

```
#include<bits/stdc++.h>
using namespace std;
using ll=long long;
#define int ll
#define ull unsigned long long
#define pii pair<int,int>
#define vc vector
#define vi vector<int>
#define db double
#define PI acos(-1.0)
/* #define ls u<<1 */
/* #define rs u<<1|1 */
#define mk make_pair
#define fi first
#define se second
#define forn(i, n) for (int i = 1; i <= n; ++i)
#define forr(i, n) for (int i = n; i >= 1; --i)
#define IOFast() ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#ifndef ONLINE_JUDGE
#define dbg(x...) do { cout << "\033[33;1m " << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#endif

#ifndef ONLINE_JUDGE
#define fileopen() do{ freopen("in", "r", stdin); freopen("out", "w", stdout);  } while
(0)
#else
#define fileopen()
#endif

int a[40], b[40];
int dp[40][2][2];
int len1, len2;
int x, y;
const int mod = 1e9 + 7 ;

void init(){
  for(int i = 31; i >= 0;  -- i){
    a[i] = (x >> i) & 1;
```

```
    b[i] = (y >> i) & 1;
  }
}

int dfs(int pos, int limit1, int limit2){
  if(pos == -1) return 1;
  if(~dp[pos][limit1][limit2]) return dp[pos][limit1][limit2];
  int up1 = limit1?a[pos]:1;
  int up2 = limit2?b[pos]:1;
  int ans = 0;
  for(int i = 0; i <= up1;  ++ i){
    for(int j = 0; j <= up2;  ++ j){
      if(i == 1 && j == 1) continue;
      ans += dfs(pos - 1, limit1 && i == up1, limit2 && j == up2);
      ans %= mod;
    }
  }
  return dp[pos][limit1][limit2] = ans;
}
void solve(){
  int ans = 0;
  int len1 = (int)log2(x), len2 = (int)log2(y);
  for(int i = 0; i <= len1;  ++ i)  ans += dfs(i - 1, i == len1, i > len2) * (i + 1) %
mod, ans %= mod;
  for(int i = 0; i <= len2;  ++ i)  ans += dfs(i - 1, i > len1, i == len2) * (i + 1) %
mod, ans %= mod;
  cout << ans << '\n';
}

signed main(){
  IOFast();
  int t;
  cin >> t;
  while(t -- ){
    cin >> x >> y;
    memset(dp,  - 1, sizeof dp);
    init();
    solve();
  }
}
```

维护平方和

```
#include<bits/stdc++.h>
using namespace std;
using ll=long long;
#define int ll
#define ull unsigned long long
```

```
#define pii pair<int,int>
#define vc vector
#define vi vector<int>
#define db double
#define PI acos(-1.0)
/* #define ls u<<1 */
/* #define rs u<<1|1 */
#define mk make_pair
#define fi first
#define se second
#define forn(i, n) for (int i = 1; i <= n; ++i)
#define forr(i, n) for (int i = n; i >= 1; --i)
#define IOFast() ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#ifndef ONLINE_JUDGE
#define dbg(x...) do { cout << "\033[33;1m " << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#endif

#ifndef ONLINE_JUDGE
#define fileopen() do{ freopen("in", "r", stdin); freopen("out", "w", stdout);  } while
(0)
#else
#define fileopen()
#endif


const int mod = 1e9 + 7;
int tot;
int bit[100];

struct node{
  int num, sum, sqsum;
};

node dp[100][8][8];
int pw[100];

node dfs(int pos, int sum, int num, int limit){
  if(pos == 0){
    node now;
    now.num = (num != 0 && sum != 0);
    now.sum = now.sqsum = 0;
    return now;
```

```
  }
  if(!limit && dp[pos][sum][num].num != -1) return dp[pos][sum][num];
  node ans;
  ans.num = ans.sum = ans.sqsum = 0;
  int up = limit?bit[pos]:9;
  for(int i = 0; i <= up;  ++ i){
    if(i == 7)   continue;
    node now = dfs(pos - 1,(sum + i) % 7, (10 * num + i) % 7, limit && i == up);
    ans.num += now.num;
    ans.num %= mod;
    ans.sum += (now.sum + i * pw[pos - 1] % mod * now.num % mod) % mod;
    ans.sum %= mod;
    ans.sqsum += (now.sqsum + 2 * (pw[pos - 1] * i % mod) % mod * now.sum % mod) % mod;
    ans.sqsum %= mod;
    ans.sqsum += (now.num * (pw[pos - 1] * i % mod) % mod * (pw[pos - 1] * i % mod) %
mod);
    ans.sqsum %= mod;
   // (ago+k1)^2+(ago+k2)^2+(ago+k3)^2
    //=3*ago^2+2*ago*(k1+k2+k3)+k2^2+k2^2+k3^2
    //=Count1*ago^2+2*ago*sum1+sum2;
  }
  if(!limit) dp[pos][sum][num] = ans;
  return ans;
}


int solve(int x){
  tot = 0;
  while(x){
    bit[ ++ tot] = x % 10;
    x /= 10;
  }
  return dfs(tot, 0, 0, 1).sqsum;
}
signed main(){
  IOFast();
  int t;
  cin >> t;
  pw[0] = 1;
  for(int i = 1; i < 100;  ++ i){
    pw[i] = (pw[i - 1] * 10) % mod;
  }
  memset(dp,  - 1, sizeof dp);
  while(t -- ){
    int l, r;
    cin >> l >> r;
    cout << (solve(r) - solve(l - 1) + mod) % mod << '\n';
  }
}
```

SOSdp

> O(n^3)

```
for(int s=m;s;s=(s-1)&m)
  ...you can use s ...
```

> O(n*2^n)

```
for(int j=0;j<n;++j)
  for(int mask=0;mask<(1<<n);++mask)
    if(mask&(1<<j)) f[mask]+=f[mask^(1<<j)]
```

mask > mask^(1<<j)

以上为维护高维前缀和（所有子集的和）

```
for(int j=0;j<n;++j)
  for(int mask=0;mask<(1<<n);++mask)
    if(mask&(1<<j)) f[mask^(1<<j)]+=f[mask]
```

以上为维护高维后缀和（所有超集的和）

- 枚举总位数为n，1个数为k的所有子集

```
template<typename T>
void subset(int k, int n, T&& f) {
    int t = (1 << k) - 1;
    while (t < 1 << n)
    {
        f(t);
        int x = t & -t, y = t + x;
        t = ((t & ~y) / x >> 1) | y;
    }
}
```

换根DP

```
int n;
const int maxn = 2e5 + 10;
const int inf = 0x3f3f3f3f;
vc < vi >  G;
int dp[2][maxn];
int ans[maxn];
```

```
void input(){
  cin >> n;
  G.resize(n + 1);
  forn(i, n - 1){
    int u, v;
    cin >> u >> v;
    G[u].pb(v);
    G[v].pb(u);
  }
}

void dfs1(int u, int fa){
  for(auto &v:G[u]){
    if(v == fa) continue;
    dfs1(v, u);
    dp[0][u] += max(dp[1][v], dp[0][v]);
    dp[1][u] = max(dp[1][u], -max(dp[1][v], dp[0][v]) + dp[0][v] + 1);
  }
  dp[1][u] += dp[0][u];
}

void dfs2(int u, int fa){
  int mx1 = 0, val1 = -inf, mx2 = 0, val2 = -inf;
  for(auto &v:G[u]){
    int val = dp[0][v] + 1 - max(dp[0][v], dp[1][v]);
    if(val >= val1){
      if(mx1){
        val2 = val1;
        mx2 = mx1;
      }
      val1 = val, mx1 = v;
    }else if(val >= val2){
      val2 = val;
      mx2 = v;
    }
  }


  dp[1][u] = dp[0][u] + val1;
  ans[u] = dp[0][u];

  for(auto & v:G[u]){
   // assume that dp[u] has the correct answer of the subtree u
   // eliminate the influence of the subtree v from subtree u
   // make up for the influence of the father from the last level
    if(v == fa) continue;
    int u0 = dp[0][u], u1 = dp[1][u], v0 = dp[0][v], v1 = dp[1][v];

    dp[0][u] -= max(dp[0][v], dp[1][v]);
```

```
    if(v == mx1) dp[1][u] = dp[0][u] + val2;
    else dp[1][u] = dp[0][u] + val1;

   // at this time the influence of subtree v has been eliminate from dp[0][u] and
dp[1][u]
    dp[0][v] += max(dp[0][u], dp[1][u]);

    dfs2(v, u);

     // return back to calculate the next subtree
    dp[0][u] = u0, dp[1][u] = u1, dp[0][v] = v0, dp[1][v] = v1;
  }
}

signed main(){
  IOFast();
  input();
  dfs1(1,  -1);
  int res = 0;
  int match = max(dp[1][1], dp[0][1]);
  dfs2(1,  -1);
  forn(i, n){
    if(ans[i] == match) res ++ ;
  }
  cout << res << '\n';
}
```

斜率DP

```
#include<bits/stdc++.h>
using namespace std;
using ll=long long;
#define int ll
#define ull unsigned long long
#define pii pair<int,int>
#define vc vector
#define vi vector<int>
#define db double
#define PI acos(-1.0)
/* #define ls u<<1 */
/* #define rs u<<1|1 */
#define mk make_pair
#define fi first
#define se second
#define forn(i, n) for (int i = 1; i <= n; ++i)
#define forr(i, n) for (int i = n; i >= 1; --i)
```

```
#define IOFast() ios::sync_with_stdio(0),cin.tie(0),cout.tie(0)
#ifndef ONLINE_JUDGE
#define dbg(x...) do { cout << "\033[33;1m " << #x << " -> "; err(x); } while (0)
void err() { cout << "\033[39;0m" << endl; }
template<template<typename...> class T, typename t, typename... A>
void err(T<t> a, A... x) { for (auto v: a) cout << v << ' '; err(x...); }
template<typename T, typename... A>
void err(T a, A... x) { cout << a << ' '; err(x...); }
#else
#define dbg(...)
#endif

#ifndef ONLINE_JUDGE
#define fileopen() do{ freopen("in", "r", stdin); freopen("out", "w", stdout);  } while
(0)
#else
#define fileopen()
#endif

const int maxn = 5e5 + 10;

int sum[maxn];
int q[maxn];
int dp[maxn];
int head, tail, n, m;

int getup(int i, int j){
  return  (dp[i] + sum[i] * sum[i])  - (dp[j] + sum[j] * sum[j]);
}

int getdown(int i, int j){
  return 2 * (sum[i] - sum[j]);
}

int getdp(int i, int j){
  return (dp[j] + m + (sum[i] - sum[j]) * (sum[i] - sum[j]));
}

void solve(){
  forn(i, n){
    cin >> sum[i];
  }
  dp[0] = sum[0] = 0;
  forn(i, n){
    sum[i] = sum[i - 1] + sum[i];
  }
  head = tail = 0;
  q[ ++ tail] = 0;
  forn(i, n){
```

```
    while(head + 1 < tail && getup(q[head + 1], q[head]) <= sum[i] * getdown(q[head +
1], q[head]))
      head ++ ;
    dp[i] = getdp(i, q[head]);
    while(head + 1 < tail && getup(i, q[tail - 1]) * getdown(q[tail - 1], q[tail - 2])
<= getup(q[tail - 1], q[tail - 2]) * getdown(i, q[tail - 1]))
      tail -- ;
    q[tail ++ ] = i;
  }
  cout << dp[n] << '\n';
}

signed main(){
  IOFast();
  while(cin >> n >> m){
    solve();
  }
}
```