动态规划

天津大学

2023 年 7 月 12 日

目录

目录

线性动态规划问题

Description

具有「线性」阶段划分的动态规划方法统称为线性动态规划（简称为「线性 DP」）

例题

- 最长递增子序列 LIS（Longest Increasing Subsequence）
- 最长公共子序列 LCS（Longest Common Subsequence）
- 字符串编辑距离（Levenshtein 距离）
- 求和问题

线性动态规划问题

LIS 问题

给定一个长度为 N 的数列 (w[N])，求数值严格单调递增的子序列的长度最长是多少。

输入格式

第一行包含整数 N。第二行包含 N 个整数，表示完整序列。

输出格式

输出一个整数，表示最大长度。数据范围

$1 \leq N \leq 1000$ $10^9 \leq w[i] \leq 10^9$

3 1 2 1 8 5 6

3 1 2 1 8 5 6

$(a[j] < a[i])f[i] = \max(f[i], f[j]+1)$

$O(n^2)$ 动态规划

```cpp
#include <iostream>
using namespace std;
const int N = 1010;
int n;
int w[N], f[N];
int main() {
    cin >> n;
    for (int i = 0; i < n; i++) cin >> w[i];
    int mx = 1;  // 找出所计算的f[i]之中的最大值，边算边找
    for (int i = 0; i < n; i++) {
        f[i] = 1; // 设f[i]默认为1，即自己
        for (int j = 0; j < i; j++) {
            if (w[i] > w[j]) f[i] = max(f[i], f[j] + 1);
        }
        mx = max(mx, f[i]);
    }
    cout << mx << endl;
    return 0;
}
}
```

原始数列 : 3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个: 1 1 0 0 0 0 0

原始数列 : 3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个: 1 1 0 0 0 0 0
第 3 位数前 1 个: 1 1 1 0 0 0 0

原始数列 : 3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个: 1 1 0 0 0 0 0
第 3 位数前 1 个: 1 1 1 0 0 0 0
第 3 位数前 2 个: 1 1 2 0 0 0 0

原始数列 : 3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个: 1 1 0 0 0 0 0
第 3 位数前 1 个: 1 1 1 0 0 0 0
第 3 位数前 2 个: 1 1 2 0 0 0 0
第 4 位数前 1 个: 1 1 2 1 0 0 0

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0
……
第 7 位数前 1 个：1 1 2 1 3 3 2

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0
……
第 7 位数前 1 个：1 1 2 1 3 3 2
第 7 位数前 2 个：1 1 2 1 3 3 2

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0
……
第 7 位数前 1 个：1 1 2 1 3 3 2
第 7 位数前 2 个：1 1 2 1 3 3 2
第 7 位数前 3 个：1 1 2 1 3 3 3

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0
……
第 7 位数前 1 个：1 1 2 1 3 3 2
第 7 位数前 2 个：1 1 2 1 3 3 2
第 7 位数前 3 个：1 1 2 1 3 3 3
第 7 位数前 4 个：1 1 2 1 3 3 3

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0
……
第 7 位数前 1 个：1 1 2 1 3 3 2
第 7 位数前 2 个：1 1 2 1 3 3 2
第 7 位数前 3 个：1 1 2 1 3 3 3
第 7 位数前 4 个：1 1 2 1 3 3 3
第 7 位数前 5 个：1 1 2 1 3 3 3

原始数列 : 3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个: 1 1 0 0 0 0 0
第 3 位数前 1 个: 1 1 1 0 0 0 0
第 3 位数前 2 个: 1 1 2 0 0 0 0
第 4 位数前 1 个: 1 1 2 1 0 0 0
第 4 位数前 2 个: 1 1 2 1 0 0 0
第 4 位数前 3 个: 1 1 2 1 0 0 0
.....
第 7 位数前 1 个: 1 1 2 1 3 3 2
第 7 位数前 2 个: 1 1 2 1 3 3 2
第 7 位数前 3 个: 1 1 2 1 3 3 3
第 7 位数前 4 个: 1 1 2 1 3 3 3
第 7 位数前 5 个: 1 1 2 1 3 3 3
第 7 位数前 6 个: 1 1 2 1 3 3 4

原始数列　　：3 1 2 1 8 5 6(w[N])
第 2 位数前 1 个：1 1 0 0 0 0 0
第 3 位数前 1 个：1 1 1 0 0 0 0
第 3 位数前 2 个：1 1 2 0 0 0 0
第 4 位数前 1 个：1 1 2 1 0 0 0
第 4 位数前 2 个：1 1 2 1 0 0 0
第 4 位数前 3 个：1 1 2 1 0 0 0
……
第 7 位数前 1 个：1 1 2 1 3 3 2
第 7 位数前 2 个：1 1 2 1 3 3 2
第 7 位数前 3 个：1 1 2 1 3 3 3
第 7 位数前 4 个：1 1 2 1 3 3 3
第 7 位数前 5 个：1 1 2 1 3 3 3
第 7 位数前 6 个：1 1 2 1 3 3 4
答案（f[N]）:4

Functions again

最大连续子段和

给定有 n 个元素的数组 a 以及公式 $f(l,r)=\Sigma_{i=l}^{r-1} \mid a[i]-a[i+1] \mid \cdot(-1)^{i-l}$。求在 $1 \leq l<r \leq n$ 的情况下，最大的 $f(l,r)$。

O(n) 动态规划

```
1     dp[1] = 0;
2     long long mx = 0, mn = 0, ans = -0x3f3f3f3f;
3     for (int i = 2; i <= n; ++i)
4     {
5         long long  temp = abs(a[i] - a[i - 1]);//奇负偶正
6         if (i % 2 == 1) temp = -temp;
7         dp[i] = dp[i - 1] + temp;
8
9         ans = max(dp[i] - mn, ans);//大正
10        ans = max(-(dp[i] - mx), ans);//abs(大负)
11        ans = max(dp[i], ans);//
12
13    if (i % 2 == 1) mn = min(mn, dp[i]);//负, 亏, 维护最小
14    else    mx = max(mx, dp[i]);//正, 赚, 维护最大
15    }
```

多维动态规划问题

Description

多维动态规划是在线性动态规划的基础上扩展状态维数，再进行状态转移的设计。难点在于所设计的状态的维数及将状态设置好后，对状态的转移。

给定一个如下图所示的数字三角形，从顶部出发，在每一结点可以选择移动至其左下方的结点或移动至其右下方的结点，一直走到底层，要求找出一条路径，使路径上的数字的和最大。

```
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5
```

输入格式

第一行包含整数 n，表示数字三角形的层数。

接下来 n 行，每行包含若干整数，其中第 i 行表示数字三角形第 i 层包含的整数。

输出格式

输出一个整数，表示最大的路径数字和。

数据范围

$1 \leq n \leq 500$,
$-10000 \leq$ 三角形中的整数 ≤ 10000

$O(n^2)$ 动态规划

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[505][505];
int dp[505][505];
int n;
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=i;j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    for(int i=1;i<=n;i++)dp[n][i]=a[n][i];//初始化最后一行
    for(int i=n-1;i>=1;i--)//倒序，从倒数第二层开始往上走
    {
        for(int j=1;j<=i;j++)
        {
            dp[i][j]=max(dp[i+1][j],dp[i+1][j+1])+a[i][j];
        }
    }
    cout<<dp[1][1]<<endl;
}
```

目录

01 背包

最基本的背包问题就是 01 背包问题（01 knapsack problem）：一共有 N 件物品，第 i（i 从 1 开始）件物品的重量为 w[i]，价值为 v[i]。在总重量不超过背包承载上限 W 的情况下，能够装入背包的最大价值是多少？

4 5
1 2
2 4
3 4
4 5

$$dp[i][j] = \begin{cases} dp[i-1][j], & j < w[i] \\ \max(dp[i-1][j],\ dp[i-1][j-w[i]] + v[i]), & j \geq w[i] \end{cases}$$

图：01 背包递推公式

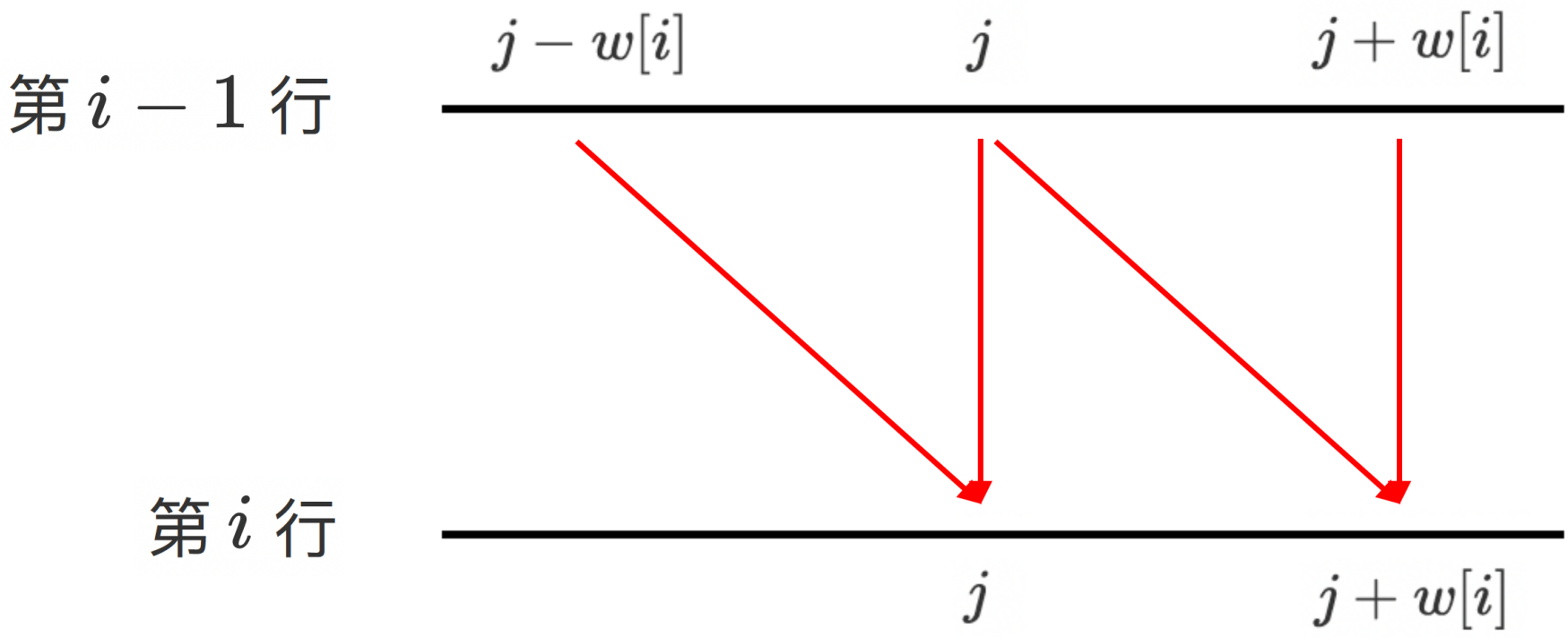
$O(n^2)$ 动态规划

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,m;
int dp[1005][1005];
int w[1005],v[1005];
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {   scanf("%d%d", &w[i], &v[i]);
        for(int j=1;j<=m;j++)
        {
            if(j<w[i])dp[i][j]=dp[i-1][j];
            else dp[i][j]=max(dp[i-1][j-w[i]]+v[i],dp[i
                -1][j]);
        }
    }
    cout<<dp[n][m];
}
```

```
2 0 0 0 0  2 2 0 0 0  2 2 2 0 0  2 2 2 2 0  2 2 2 2 2
0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0
0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0
0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0

2 2 2 2 2  2 2 2 2 2  2 2 2 2 2  2 2 2 2 2  2 2 2 2 2
2 0 0 0 0  2 4 0 0 0  2 4 6 0 0  2 4 6 6 0  2 4 6 6 6
0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0
0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0

2 2 2 2 2  2 2 2 2 2  2 2 2 2 2  2 2 2 2 2  2 2 2 2 2
2 4 6 6 6  2 4 6 6 6  2 4 6 6 6  2 4 6 6 6  2 4 6 6 6
2 0 0 0 0  2 4 0 0 0  2 4 6 0 0  2 4 6 6 0  2 4 6 6 8
0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 0 0 0 0

2 2 2 2 2  2 2 2 2 2  2 2 2 2 2  2 2 2 2 2  2 2 2 2 2
2 4 6 6 6  2 4 6 6 6  2 4 6 6 6  2 4 6 6 6  2 4 6 6 6
2 4 6 6 8  2 4 6 6 8  2 4 6 6 8  2 4 6 6 8  2 4 6 6 8
2 0 0 0 0  2 4 0 0 0  2 4 6 0 0  2 4 6 6 0  2 4 6 6 8
```

图: 01 背包 dp 数组

考虑优化问题：需要这么多空间吗



图：dp 数组中，第 i 行只使用第 i-1 行

正序遍历递推关系式比较

$$dp[i][j] = \begin{cases} dp[i-1][j], & j < w[i] \\ \max(dp[i-1][j],\ dp[i-1][j-w[i]] + v[i]), & j \geq w[i] \end{cases}$$

$$dp[i][j] = \begin{cases} dp[i-1][j], & j < w[i] \\ \max(dp[i-1][j],\ dp[i][j-w[i]] + v[i]), & j \geq w[i] \end{cases}$$

图: 上图为二维 dp 数组递推式，下图为一维 dp 数组压缩前递推式

$O(n^2)$ 动态规划

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
int dp[1005];
int n,m,w,v;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {  scanf("%d%d",&w,&v);
        for(int j=m;j>=1;j--)
        {
            if(w<=j) dp[j]=max(dp[j-w]+v,dp[j]);
        }
    }
    cout<<dp[m];
}
```

完全背包

有 N 种物品和一个容量是 V 的背包，每种物品都有无限件可用。第 i 种物品的体积是 vi，价值是 wi。求解将哪些物品装入背包，可使这些物品的总体积不超过背包容量，且总价值最大。输出最大价值。

$O(n^3)$ 动态规划

```
//全暴力
int dp[1005][1005];
int n,m,w,v,s;
int main()
{ cin>>n>>m;
    for (int i = 1; i <= n; i ++ )
    { cin>>w>>v>>s;
        for (int j = 1; j <=m; j ++ )
        {
            int t=min(s,j/w);
            for(int k=0;k<=t;k++)
    { dp[i][j]=max(dp[i-1][j-k*w[i]]+k*v[i],dp[i][j]); }
        }
    }
    cout<<dp[n][m];
}
```

$O(n^3)$ 动态规划

```
1  //全暴力
2  int dp[1005][1005];
3  int n,m,w,v,s;
4  int main()
5  { cin>>n>>m;
6      for (int i = 1; i <= n; i ++ )
7      { cin>>w>>v>>s;
8          for (int j = 1; j <=m; j ++ )
9          {
10             int t=min(s,j/w);
11             for(int k=0;k<=t;k++)
12     { dp[i][j]=max(dp[i-1][j-k*w[i]]+k*v[i],dp[i][j]); }
13         }
14     }
15     cout<<dp[n][m];
16 }
```

怎么优化呢

$O(n^2)$ 动态规划

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
int p[1005];
int n,m,w,v;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {
        scanf("%d%d", &w, &v);
        for(int j=1;j<=m;j++)
        {
            if(j>=w)p[j]=max(p[j-w]+v,p[j]);
        }
    }
    cout<<p[m]<<endl;
}
```

多重背包

有 N 种物品和一个容量是 V 的背包。第 i 种物品最多有 si 件，每件体积是 wi，价值是 vi。求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。输出最大价值。

考虑前文中使用到的完全背包的暴力情况：

```
//全暴力
int dp[1005][1005];
int n,m,w,v,s;
int main()
{ cin>>n>>m;
    for (int i = 1; i <= n; i ++ )
    { cin>>w>>v>>s;
        for (int j = 1; j <=m; j ++ )
        {
            int t=min(s,j/w);//修改此处即可
            for(int k=0;k<=t;k++)
    { dp[i][j]=max(dp[i-1][j-k*w[i]]+k*v[i],dp[i][j]); }
        }
    }
    cout<<dp[n][m];
}
```

还是会爆 tle，考虑如何优化时间复杂度？

二进制优化：1895=sum((1 2 4 8 16 32 64 128 256 512 1024) 872)

下面用数学语言来描述上面的例子。对于任意的正整数 s，我们都可以找到 $\lfloor \log_2 s \rfloor + 1 \triangleq k$ 个正整数 $a_1, \cdots, a_k$，使得 $\forall n \in [0, s]$，都有

$$n = v^{\mathrm{T}} a, \quad a = (a_1, \cdots, a_k)^{\mathrm{T}}, \quad a_i = \begin{cases} 2^{i-1}, & 1 \le i \le k-1 \\ s - 2^{k-1} + 1 \, (\in [1, 2^{k-1}]), & i = k \end{cases}$$

其中 $v = (v_1, \cdots, v_k)^{\mathrm{T}}$，且其分量非 0 即 1。

图：二进制优化

使用二进制优化，将多重背包问题转化为“大”01 背包问题

```
1  //全暴力
2  int dp[N];
3  int w[N],v[N];
4  int n,m,a,b,s;
5  int main()
6  {
7      cin>>n>>m;
8      int cnt=0;
9      //二进制处理
10     for(int i=1;i<=n;i++)
11     {
12         for(int j=m;j>=w[i];j--)
13         {
14             dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
15         }
16     }
17     cout<<dp[m]<<endl;
18 }
```

二进制处理方法：

```
1     for(int i=1;i<=n;i++)
2     {
3         scanf("%d%d%d",&a,&b,&s);
4         int k=1;
5         while(s>k)
6         {     cnt++;
7             w[cnt]=a*k;
8             v[cnt]=b*k;
9             s-=k;
10            k*=2;
11        }
12        if(s>0)
13        {    cnt++;
14            w[cnt]=a*s;
15            v[cnt]=b*s;
16        }
17    }
18    n=cnt;
```

有 N 组物品和一个容量是 V 的背包。每组物品有若干个，同一组内的物品最多只能选一个。每件物品的体积是 vij，价值是 wij，其中 i 是组号，j 是组内编号。求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。
输出最大价值。
输入格式第一行有两个整数 N，V ，用空格隔开，分别表示物品组数和背包容量。
接下来有 N 组数据:
每组数据第一行有一个整数 Si ，表示第 i 个物品组的物品数量；每组数据接下来有 Si 行，每行有两个整数 vij,wij ，用空格隔开，分别表示第 i 个物品组的第 j 个物品的体积和价值;

```cpp
#include<bits/stdc++.h>
using namespace std;
int dp[105];
int w[105],v[105];
int n,m,s;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    {   cin>>s;
        for(int j=1;j<=s;j++)
            cin>>w[j]>>v[j];
        for(int j=m;j>=0;j--)
        {
            for(int k=1;k<=s;k++)
            {
                if(j>=w[k])dp[j]=max(dp[j],dp[j-w[k]]+v[k
                    ]);
            }
        }
    }
    cout<<dp[m]<<endl;
}
```

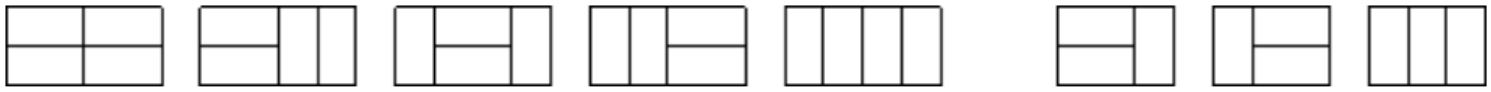
目录

蒙德里安的梦想

求把 $N \times M$ 的棋盘分割成若干个 1×2 的长方形，有多少种方案。

例如当 $N = 2$，$M = 4$ 时，共有 5 种方案。当 $N = 2$，$M = 3$ 时，共有 3 种方案。

如下图所示：



输入格式

输入包含多组测试用例。

每组测试用例占一行，包含两个整数 N 和 M。

当输入用例 $N = 0$，$M = 0$ 时，表示输入终止，且该用例无需处理。

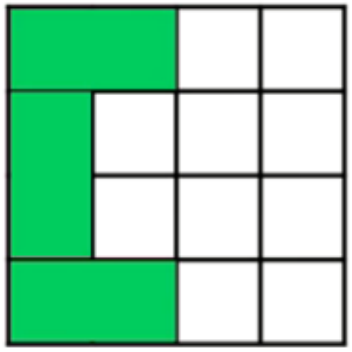
输出格式

每个测试用例输出一个结果，每个结果占一行。

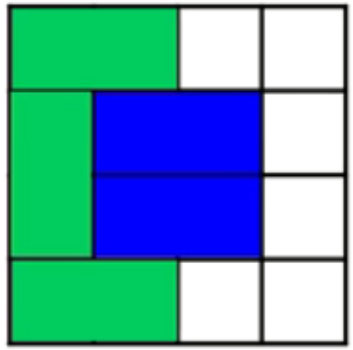
数据范围

$1 \leq N, M \leq 11$

我们考虑按列摆放，某列的各行用0或1表示摆放状态。
如果某行是1，表示横放，并且向下一列伸出；
如果某行是0，表示竖放，或者由前一列伸出。



第1列：1001



第2列：0110

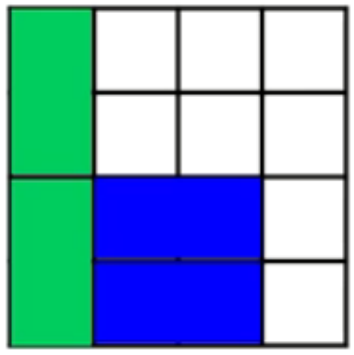
1. 状态表示：f[i , j]表示摆放第i列，状态为j时的方案数。

状态转移：f[i-1 , k] → f[i , j]

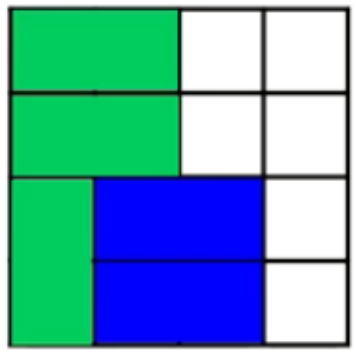
2. 状态计算：f[i , j] = ∑ f[i-1 , k]

3. 初值：f[0, 0]=1，其他为0

4. 目标：f[m, 0]



0000 → 0011



1100 → 0011

```
for(int i=0;i<1<<n;i++)//二进制状态表示
    {
        st[i]=1;
        int cnt=0;//记录连续0的个数
        for(int j=0;j<n;j++)
        {
            if(i>>j &1)//右移1~n位
            {
                if(cnt&1)//判断奇偶性，若为1则判断
                {
                    st[i]=0;
                    break;
                }
            }
            else cnt++;
        }
        if(cnt&1)//全部操作后，检查高位0的个数
                {
                    st[i]=0;
                }
    }
```

```
memset(dp, 0, sizeof dp);
dp[0][0]=1;
for(int i=1;i<=m;i++)//列
{
    for(int j=0;j< 1<<n;j++)//当前列状态
    {
        for(int k=0;k< 1<<n;k++)前一列状态
        {
          if(st[j|k]&&(j&k)==0)dp[i][j]+=dp[i-1][k];
        }
    }
}
        cout<<dp[m][0]<<endl;
```

小国王

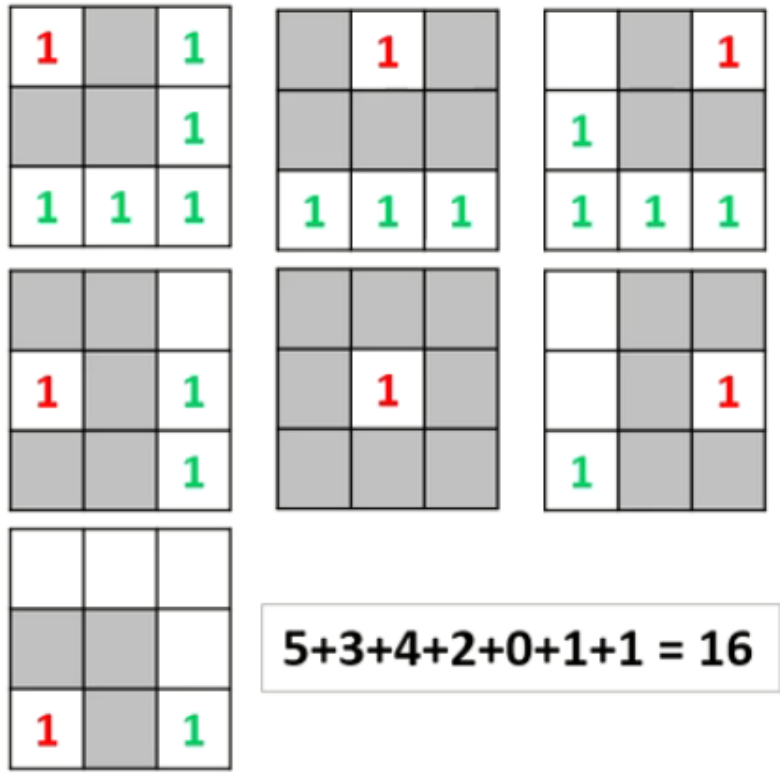
小国王 状压DP

【题目】
在 n×n 的棋盘上放 k 个国王，国王可攻击相邻的8个格子，求使他们无法互相攻击的方案总数。

【输入格式】
共一行，包含两个整数 n 和 k。
$1 \le n \le 10$ ， $0 \le k \le n^2$

【输出格式】
共一行，表示方案总数，若不能够放置则输出0。

【输入样例】
3 2
【输出样例】
16



5+3+4+2+0+1+1 = 16

[1 0 0] → [4]
[1 0 1] → [5]

状态压缩存储：
用二进制表示状态，

小国王 状压DP

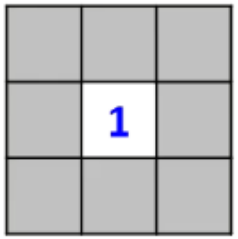
每行的所有状态：000, 001, 010, 011, 100, 101, 110, 111

每行的合法状态：000, 001, 010, 100, 101

1. 行内合法：如果 !(i & i>>1) 为真，则 i 合法

i=5: 1 0 1
0 1 0

i=6: 1 1 0
0 1 1



b:	×	×	×
a:	0	1	0

2. 行间兼容：如果 !(a&b) && !(a&b>>1) && !(a&b<<1)为真，则 a,b 兼容

b：[0 0 0] , [0 0 1] , [0 1 0] , [1 0 0] , [1 0 1]

a：[0 0 0] , [0 0 1] , [0 1 0] , [1 0 0] , [1 0 1]

3. 状态表示：f[i , j , a]表示前i行已放了j个国王，第i行的第a个状态的方案数

4. 状态计算：f[i , j , a] = ∑ f[i-1 , j-c[a] , b]

5. 总方案数：ans = ∑ f[n , k , a]

目录

石子合并

设有 N 堆石子排成一排，其编号为 $1, 2, 3, \ldots, N$。

每堆石子有一定的质量，可以用一个整数来描述，现在要将这 N 堆石子合并成为一堆。

每次只能合并相邻的两堆，合并的代价为这两堆石子的质量之和，合并后与这两堆石子相邻的石子将和新堆相邻，合并时由于选择的顺序不同，合并的总代价也不相同。

例如有 4 堆石子分别为 `1 3 5 2`，我们可以先合并 1、2 堆，代价为 4，得到 `4 5 2`，又合并 1、2 堆，代价为 9，得到 `9 2`，再合并得到 11，总代价为 $4 + 9 + 11 = 24$；

如果第二步是先合并 2、3 堆，则代价为 7，得到 `4 7`，最后一次合并代价为 11，总代价为 $4 + 7 + 11 = 22$。

问题是：找出一种合理的方法，使总的代价最小，输出最小代价。

关键点：最后一次合并一定是左边连续的一部分和右边连续的一部分进行合并

状态表示：$f[i][j]$ 表示将 i 到 j 这一段石子合并成一堆的方案的集合，属性 Min

状态计算：

（1）$i < j$ 时，$f[i][j] = \min\limits_{i \le k \le j-1} f[i][k] + f[k+1][j] + s[j] - s[i-1]$

（2）$i = j$ 时，$f[i][i] = 0$（合并一堆石子代价为 0）

问题答案：$f[1][n]$

```
const int N = 310;int s[N];int n;int dp[N][N];
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)cin>>s[i],s[i]+=s[i-1];
    for(int len=2;len<=n;len++)//枚举长度
    {
        for(int i=1;i+len-1<=n;i++)//枚举左端点
        {
            int j=i+len-1;//知左知长，得右
            dp[i][j]=0x3f3f3f3f;
            for(int k=i;k<j;k++)//划分点
            {
dp[i][j]=min(dp[i][j],dp[i][k]+dp[k+1][j]+s[j]-s[i-1]);
            }
        }
    }
    cout<<dp[1][n]<<endl;
}
```

目录

树的重心

树的重心 树形DP

【题目】

给定一颗树，树中包含n个结点（编号1~n）和n-1条无向边。

请你找到树的重心，并输出将重心删除后，剩余各个连通块中点数的最大值。

重心定义：重心是指树中的一个结点，如果将这个点删除后，剩余各个连通块中点数的最大值最小，那么这个结点被称为树的重心。

【输入格式】

第一行包含整数n，表示树的结点数。$1 \le n \le 10^5$

接下来n-1行，每行包含两个整数a和b，表示点a和点b之间存在一条边。

【输出格式】

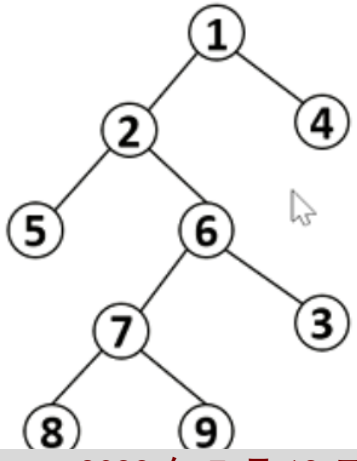
输出一个整数m，表示将重心删除后，剩余各个连通块中点数的最大值。

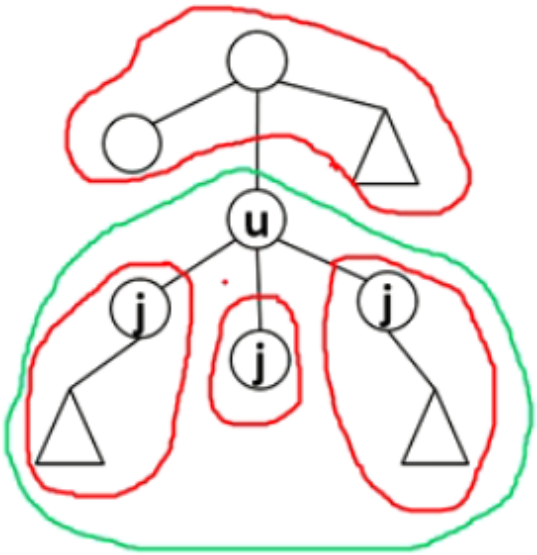
【输入样例】

```
9
1 4
1 2
2 6
2 5
6 3
6 7
7 9
7 8
```

【输出样例】

```
4
```





- 任取一点u，若以u为重心，则分为两类：一类是u的子树，一类是u上面的部分。
- 需要算出u的最大子树的结点数和u上面的部分的结点数，然后取二者的最大值即可。

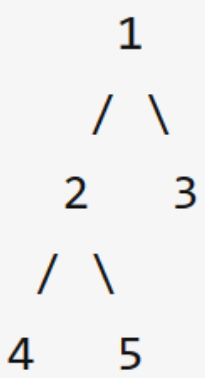
- **size:** 记录u的最大子树的结点数；
- **sum:** 记录以u为根的子树的结点数；
- **n-sum:** u上面的部分的结点数；
- **ans = max(size, n-sum)**。

- ✓ **size、sum**可以在向下遍历返回时统计和累加；
- ✓ **vis[u]:** 标记u这个点被搜过。
- ✓ 因为u是任取的一点，所以在遍历每个点时都会得到一个ans，取最小值即可。

二叉树的直径

给定一棵二叉树，你需要计算它的直径长度。一棵二叉树的直径长度是任意两个结点路径长度中的最大值。这条路径可能穿过也可能不穿过根结点。

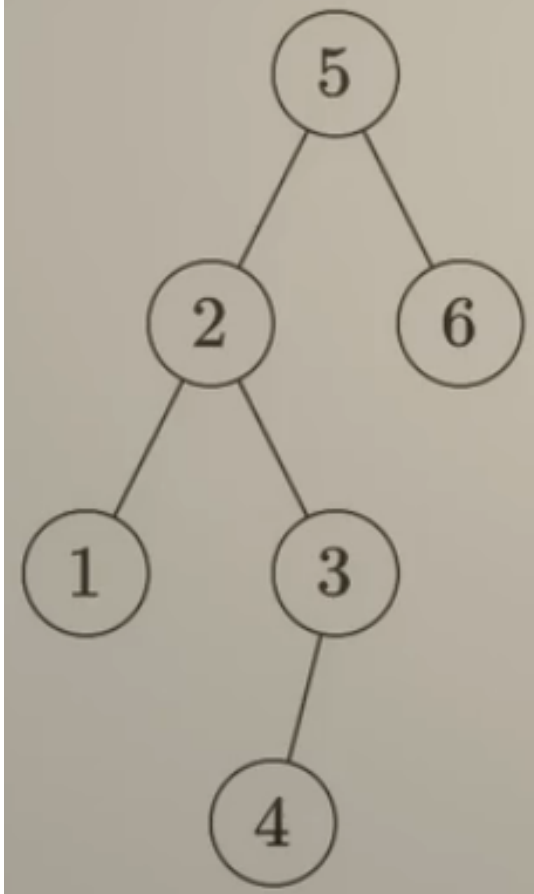
示例 :
给定二叉树

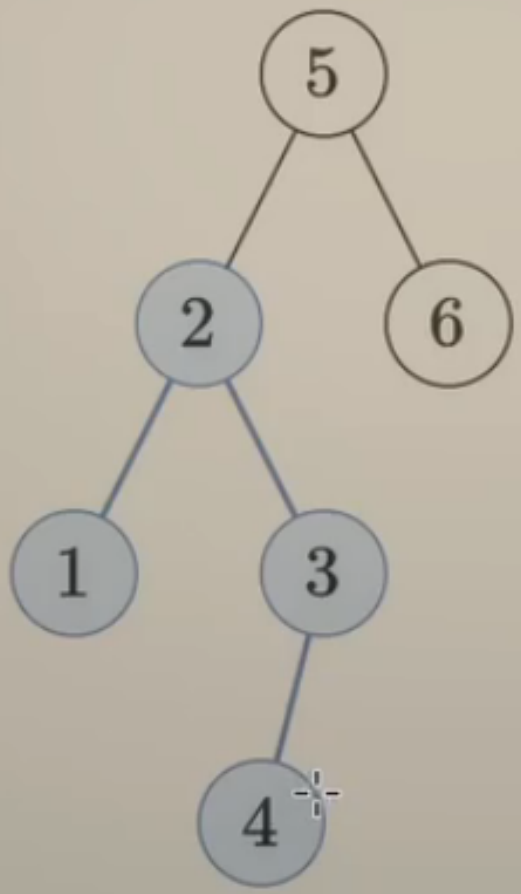


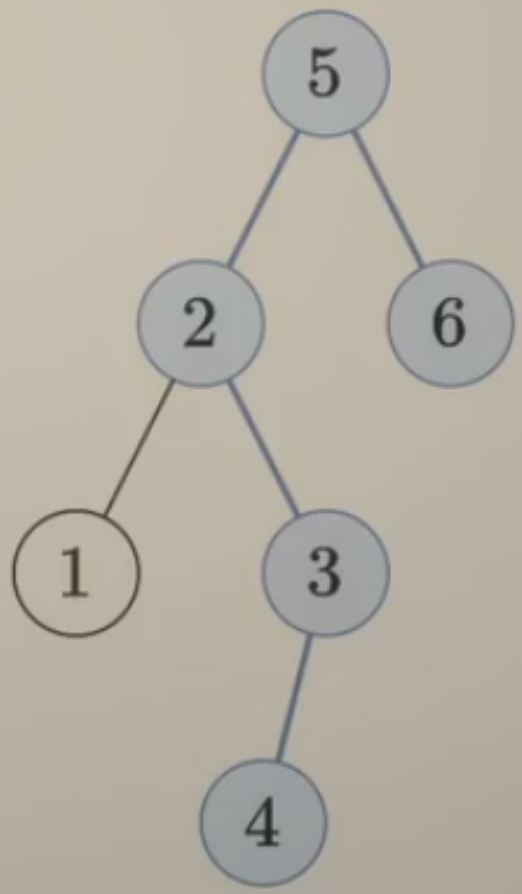
返回 3, 它的长度是路径 [4,2,1,3] 或者 [5,2,1,3]。
注意：两结点之间的路径长度是以它们之间边的数目表示。

543. 二叉树的直径

换个角度看直径：
从一个叶子出发向上，在某个节点「拐弯」，向下到达另一个叶子。
得到了由两条链拼起来的路径。（也可能只有一条链）







```java
int dfs(TreeNode node){
        if(node != null){
            // 当前节点不为空
            int maxL = dfs(node.left);
            int maxR = dfs(node.right);
            // 当前节点的路径最大值等于左子树最大边数加上右子树最大边数
            int maxCur = maxL + maxR;
            res = maxCur > res ? maxCur : res;
            // 返回左右子树的最大边数为该点为左右子树的最大边数
            return Math.max(maxL, maxR) + 1;
        }
        // 当前节点为空返回0
        return 0;
    }
```

目录

线性动态规划问题

LIS 问题

给定一个长度为 N 的数列 (w[N])，求数值严格单调递增的子序列的长度最长是多少。

输入格式

第一行包含整数 N。第二行包含 N 个整数，表示完整序列。

输出格式

输出一个整数，表示最大长度。数据范围

$1 \leq N \leq 100000$ $10^9 \leq w[i] \leq 10^9$

```cpp
#include<bits/stdc++.h>
using namespace std;
int n;vector<int>a;int cnt=0;int k;
int main() {
    cin >> n;cin>>k;
   a.push_back(k);
   for(int i=2;i<=n;i++)
   {   cin>>k;
       if(k>a.back())
       {
           a.push_back(k);
       }
       else
       {
           *lower_bound(a.begin(),a.end(),k)=k;
       }
   }
   cout<<a.size()<<endl;
}
```

区间 dp

约翰的奶牛们从小娇生惯养，她们无法容忍牛棚里的任何脏东西。约翰发现，如果要使这群有洁癖的奶牛满意，他不得不雇佣她们中的一些来清扫牛棚，约翰的奶牛中有 $N(1 \leq N \leq 10000)$ 头愿意通过清扫牛棚来挣一些零花钱。

由于在某个时段中奶牛们会在牛棚里随时随地地乱扔垃圾，自然地，她们要求在这段时间里，无论什么时候至少要有一头奶牛正在打扫。需要打扫的时段从某一天的第 M 秒开始，到第 E 秒结束 $(0 \leq M \leq E \leq 86399)$。注意这里的秒是指时间段而不是时间点，也就是说，每天需要打扫的总时间是 $E - M + 1$ 秒。

约翰已经从每头牛那里得到了她们愿意接受的工作计划：对于某一头牛，她每天都愿意在第 $T_1 \ldots T_2$ 秒的时间段内工作 $(M \leq T_1 \leq T_2 \leq E)$，所要求的报酬是 S 美元 $(0 \leq S \leq 500000)$。与需打扫时段的描述一样，如果一头奶牛愿意工作的时段是每天的第 $10 \ldots 20$ 秒，那她总共工作的时间是 11 秒，而不是 10 秒。

约翰一旦决定雇佣某一头奶牛，就必须付给她全额的工资，而不能只让她工作一段时间，然后再按这段时间在她愿意工作的总时间中所占的百分比来决定她的工资。现在请你帮约翰决定该雇佣哪些奶牛以保持牛棚的清洁，当然，在能让奶牛们满意的前提下，约翰希望使总花费尽量小。

输入格式

第 1 行：3 个正整数 N, M, E。

第 2 到 $N+1$ 行：第 $i+1$ 行给出了编号为 i 的奶牛的工作计划，即 3 个正整数 T_1, T_2, S。

目录

最低通行费

一个商人穿过一个 $N \times N$ 的正方形的网格，去参加一个非常重要的商务活动。

他要从网格的左上角进，右下角出。

每穿越中间 1 个小方格，都要花费 1 个单位时间。

商人必须在 $(2N-1)$ 个单位时间穿越出去。

而在经过中间的每个小方格时，都需要缴纳一定的费用。

这个商人期望在规定时间内用最少费用穿越出去。

请问至少需要多少费用?

注意：不能对角穿越各个小方格（即，只能向上下左右四个方向移动且不能离开网格）。

输入格式

第一行是一个整数，表示正方形的宽度 N。

后面 N 行，每行 N 个不大于 100 的正整数，为网格上每个小方格的费用。

魔法百合

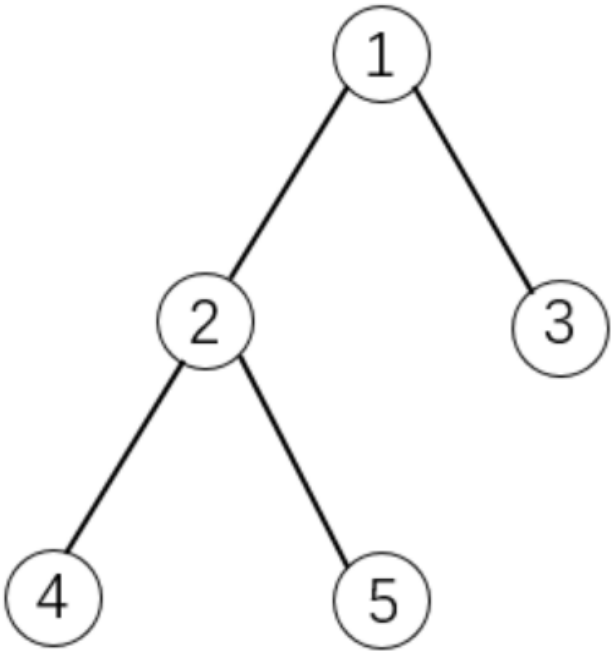
森林里有一口很深的魔法井，井中有 L 朵百合花。你带着一个大空篮子和足够多的硬币来到了井边。这个井有魔力，向里面投入硬币可以发生神奇的事情：
如果你向井里一次性投入 1 个硬币，井就会将一朵百合花扔进你的篮子里。如果你向井里一次性投入 4 个硬币，井就会统计并记录到目前为止，已经扔进你的篮子里的百合花的数量如果你向井里一次性投入 2 个硬币，井就会将等同于上次记录数量的百合花扔进你的篮子里。有一点需要特别注意，如果你向井里一次性投入 1 个或 2 个硬币后，井中已经没有足够的百合花扔给你了，那么井就不会发动任何魔法，也不会扔给你任何百合花（钱白花了）。请你计算，为了将所有百合花都收入篮中，所需要花费的最少硬币数量。

树形 dp

有 N 个物品和一个容量是 V 的背包。

物品之间具有依赖关系，且依赖关系组成一棵树的形状。如果选择一个物品，则必须选择它的父节点。

如下图所示：



如果选择物品5，则必须选择物品1和2。这是因为2是5的父节点，1是2的父节点。

每件物品的编号是 i，体积是 v_i，价值是 w_i，依赖的父节点编号是 p_i。物品的下标范围是 $1 \ldots N$。

求解将哪些物品装入背包，可使物品总体积不超过背包容量，且总价值最大。

先比较一下以往 **线性背包DP** 的 **状态转移**，第 i 件 **物品** 只会依赖第 $i-1$ 件 **物品** 的状态

如果本题我们也采用该种 **状态依赖关系** 的话，对于节点 i，我们需要枚举他所有子节点的组合 2^k 种可能

再枚举 **体积**，**最坏时间复杂度** 可能会达到 $O(N \times 2^N \times V)$（所有子节点都依赖根节点）
最终毫无疑问会 **TLE**

因此我们需要换一种思考方式，那就是枚举每个 **状态 分给各个子节点** 的 **体积**

这样 **时间复杂度** 就是 $O(N \times V \times V)$

修草奶牛

在一年前赢得了小镇的最佳草坪比赛后，FJ 变得很懒，再也没有修剪过草坪。现在，新一轮的最佳草坪比赛又开始了，FJ 希望能够再次夺冠。然而，FJ 的草坪非常脏乱，因此，FJ 只能够让他的奶牛来完成这项工作。FJ 有 N 只排成一排的奶牛，编号为 1 到 N。每只奶牛的效率是不同的，奶牛 i 的效率为 Ei 。靠近的奶牛们很熟悉，如果 FJ 安排超过 K 只连续的奶牛，那么这些奶牛就会罢工去开派对。因此，现在 FJ 需要你的帮助，计算 FJ 可以得到的最大效率，并且该方案中没有连续的超过 K 只奶牛。

目录

Thank you! Any Question?