动态规划

胡船长

初航我带你，远航靠自己

一、递推问题

1. 如何求解递推问题
2. 容斥原理的基本思想
3. 随堂练习1：HZOJ-40-爬楼梯
4. 随堂练习2：HZOJ-41-墙壁涂色

二、递推-课后实战题

1. 数的划分-P1025
2. 数的计算-P1028
3. 神经网络-P1038
4. 栈-P1044
5. 循环-P1050
6. 传球游戏-P1057
7. Hanoi 双塔问题-P1096

三、动态规划

1. 如何求解动态规划问题
2. 全面剖析：数字三角形问题
3. DP 经典问题入门 8 讲

四、动态规划的优化

1. 优化①：去除冗余状态
2. 优化②：状态重定义
3. 优化③：优化转移过程
4. 优化④：斜率优化

五、动态规划-课后实战题

1. 低价购买-P1108
2. 杂务-P1113
3. 最大子段和-P1115
4. NASA 的食物计划-P1507
5. 魔族密码-P1481
6. 找啊找啊找 GF-P1509
7. 书本整理-P1103
8. 删数-P2426
9. 垃圾陷阱-P1156
10. 最大正方形 II -P1681
11. 导弹拦截-P1158
12. 三元上升子序列-P1637
13. 多人背包-P1858
14. 仓库建设-P2120

一. 递推问题

1-1. 如何求解递推问题

兔子繁殖问题

月份	一月	二月	三月	四月	五月	六月
总对数	1	2	3	5	8	13
新增对数	0	1	**1**	**2**	**3**	**5**

一月：1　对兔子
二月：2　对兔子
三月：3　对兔子（一月及以前出生的兔子生了小兔子）
四月：5　对兔子（二月及以前出生的兔子生了小兔子）
五月：8　对兔子（三月及以前出生的兔子生了小兔子）
六月：13　对兔子（四月及以前出生的兔子生了小兔子）
……

兔子繁殖问题

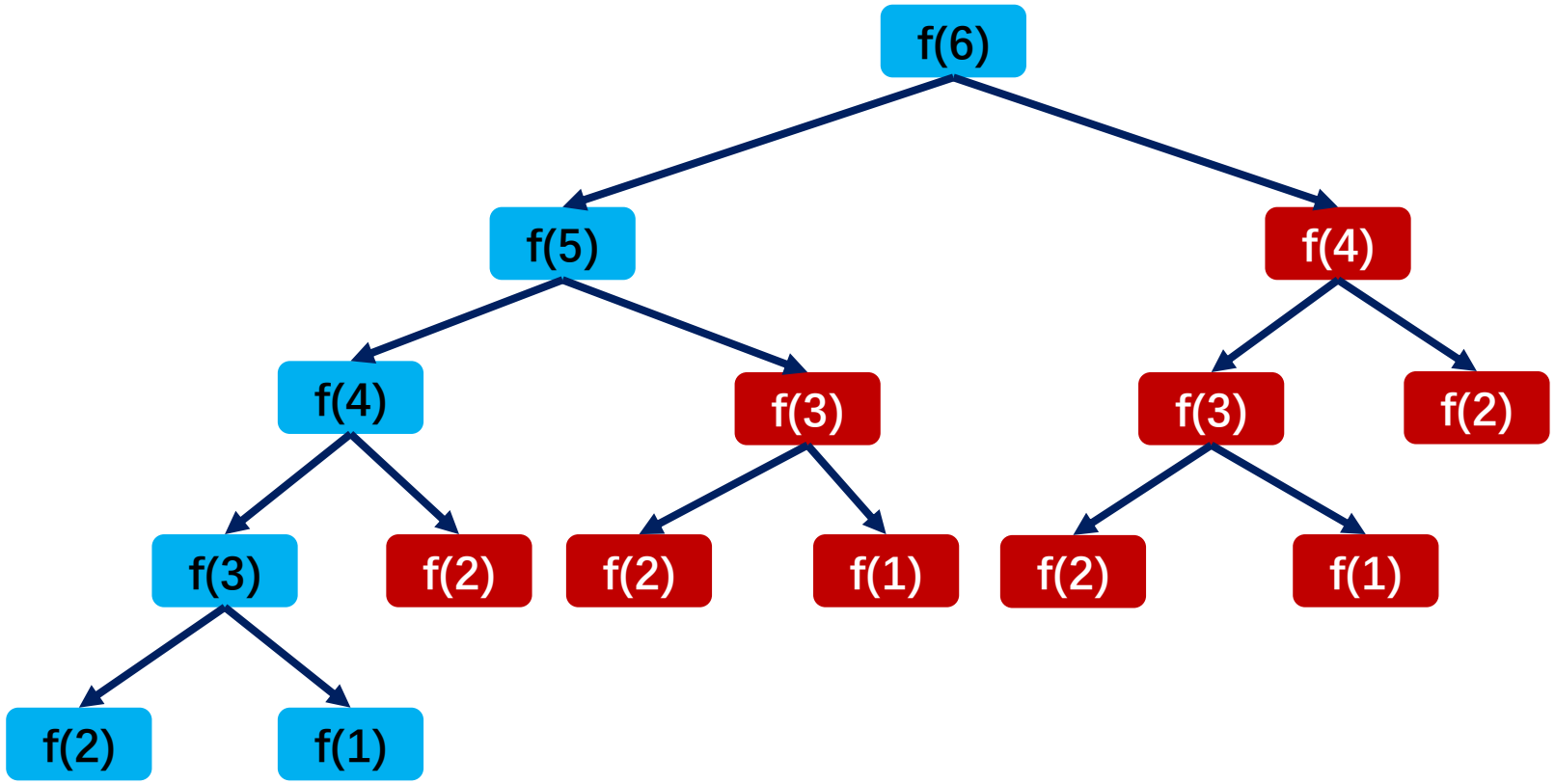
$$f(n) = \begin{cases} 1\ (n = 1) \\ 2\ (n = 2) \\ f(n-1) + f(n-2) \end{cases}$$

```
/*************************************************
    > File Name: 1.cpp
    > Author: hug
    > Mail:   hug@haizeix.com
    > Created Time: 四  7/27 14:58:33 2017
 *************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;

int f(int n) {
    switch (n) {
        case 1: return 1;
        case 2: return 2;
        default : return f(n - 1) + f(n - 2);
    }
    return 0;
}

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        printf("%d\n", f(n));
    }
    return 0;
}
```

兔子繁殖问题

思考：当 n = 40 时，用上述程序求解会出现什么问题？n = 60呢？

兔子繁殖问题



兔子繁殖问题

思考： 当 n = 40 时，用上述程序求解会出现什么问题？n = 60呢？

答：会出现两个问题

一：程序运行效率问题

二：程序计算结果超过整型表示范围，结果溢出出错

怎么解决上述两个问题呢？

兔子繁殖问题

思考： 当 n = 40 时，用上述程序求解会出现什么问题？n = 60呢？

答：会出现两个问题

一：程序运行效率问题 （递归过程加记忆化 或 改成逆向递推求解）

二：程序计算结果超过整型表示范围，结果溢出出错（改成大整数求解）

兔子繁殖问题

```cpp
/*************************************************
    > File Name: 1.cpp
    > Author: hug
    > Mail:   hug@haizeix.com
    > Created Time: 四  7/27 14:58:33 2017
 *************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;

const int MAX_N = 1000;
int f_result[MAX_N] = {0};

int f(int n) {
    if (f_result[n]) return f_result[n];
    f_result[n] = f(n - 1) + f(n - 2);
    return f_result[n];
}

int main() {
    f_result[1] = 1;
    f_result[2] = 2;
    int n;
    while (scanf("%d", &n) != EOF) {
        printf("%d\n", f(n));
    }
    return 0;
}
```

```cpp
/*************************************************
    > File Name: 1.cpp
    > Author: hug
    > Mail:   hug@haizeix.com
    > Created Time: 四  7/27 14:58:33 2017
 *************************************************/

#include <iostream>
#include <cstdio>
#include <cstdlib>
using namespace std;

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {
        int fn_1 = 0, fn_2 = 0, fn = 1;
        for (int i = 0; i < n; i++) {
            fn_1 = fn_2;
            fn_2 = fn;
            fn = fn_1 + fn_2;
        }
        printf("%d\n", fn);
    }
    return 0;
}
```

如何求解递推问题

1、确定递推状态

例如：f(n)代表第 n 个月兔子的总对数

2、确定递推公式

例如：$$f(n)=\begin{cases}1\ (n=1)\\ 2\ (n=2)\\ f(n-1)+f(n-2)\end{cases}$$

3、程序实现

如之前所有程序

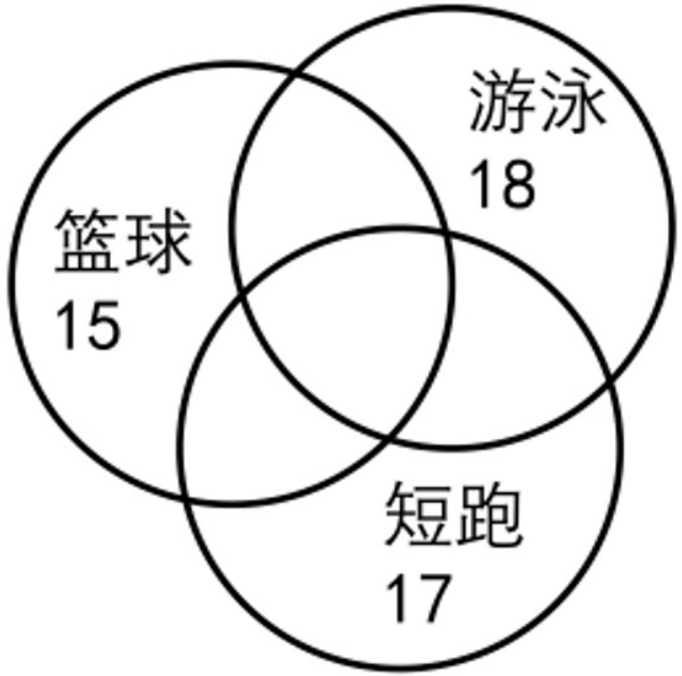
1-2. 容斥原理的基本思想

容 斥 原 理

在计数时，必须注意没有重复，没有遗漏。为了使重叠部分不被重复计算，人们研究出一种新的计数方法，这种方法的基本思想是：先不考虑重叠的情况，把包含于某内容中的所有对象的数目先计算出来，然后再把计数时重复计算的数目排斥出去，使得计算的结果既无遗漏又无重复，这种计数的方法称为容斥原理。

容斥原理

统计整体的时候，把能想到的都加进来，多了就减掉，少了就再加。



钱 币 问 题

【编写程序】现在给你1、2、5、10、20、50、100、200元若干张，问你想要凑足N元钱，一共有多少种不同的方法？

注意(1,1,2)和(1,2,1)属于同一种方法。

钱币问题:容斥关系

1、确定递推状态

f[i][j]代表，用前 i 种钱币，凑足 j 元钱的方法总数

钱 币 问 题

1、确定递推状态

f[i][j]代表，用前 i 种钱币，凑足 j 元钱的方法总数

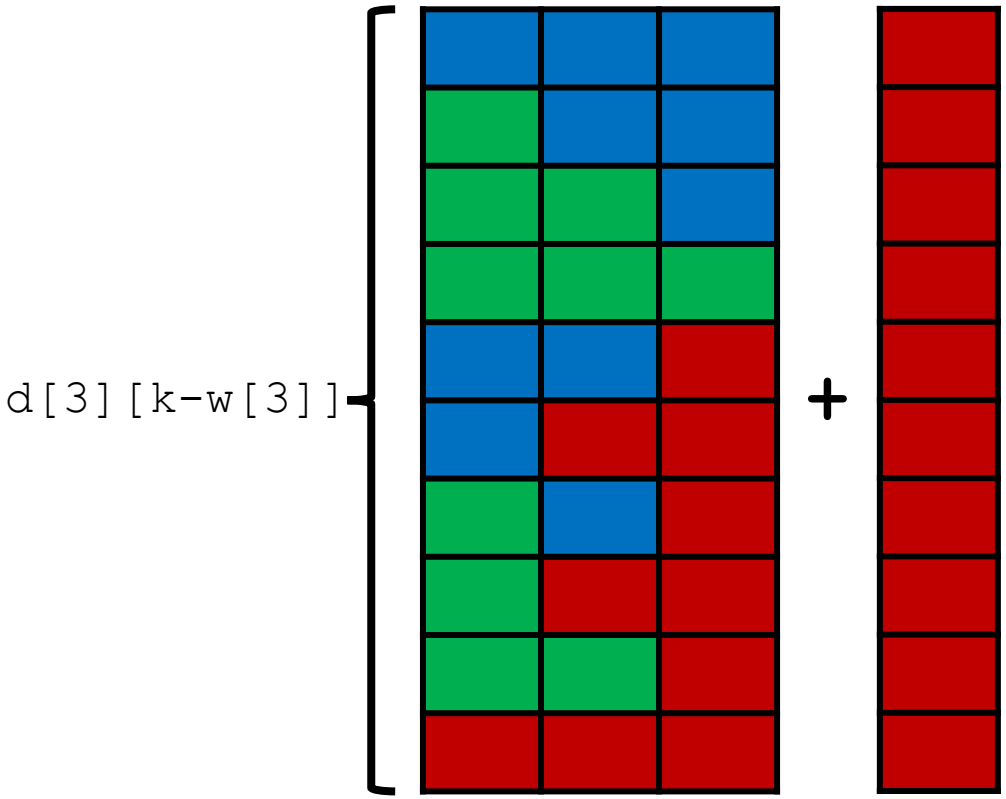
2、确定递推公式

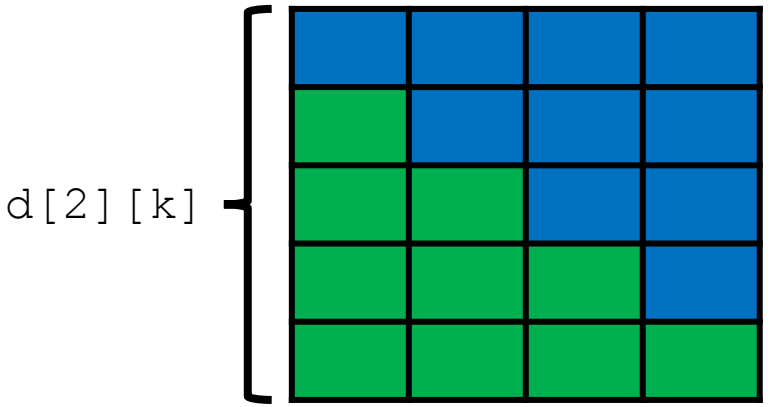
f[i][j] = f[i - 1][j] + f[i][j - w[i]]

其中 w[i]为第 i 种钱币的面值

钱 币 问 题

求解 d[3][k]

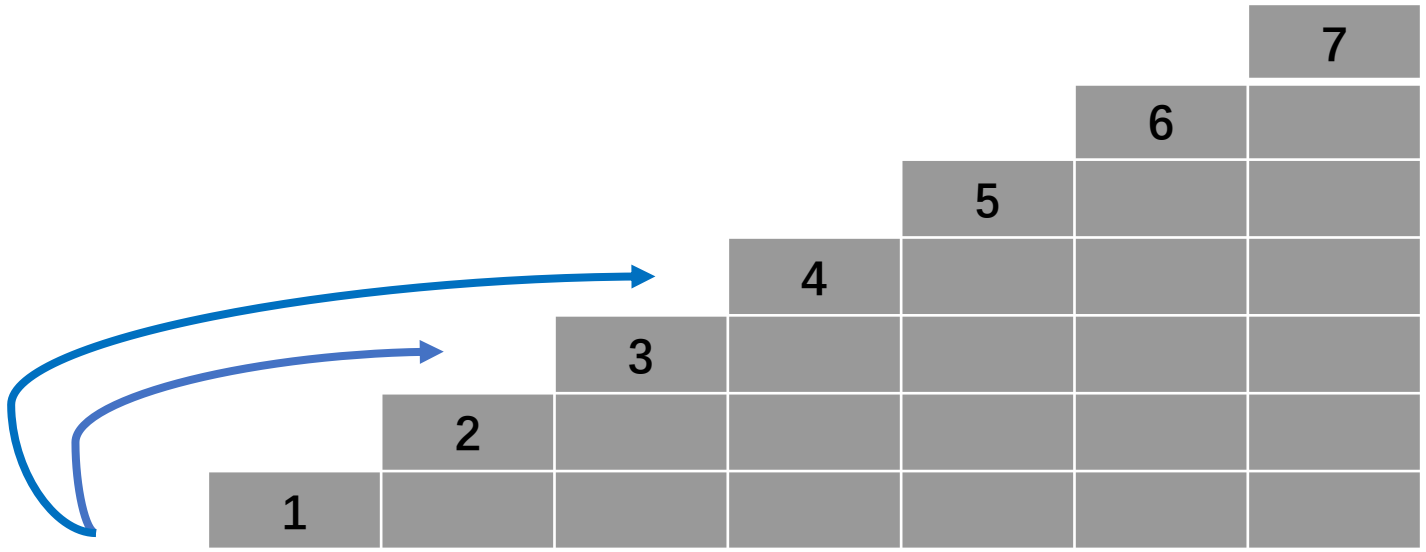




1-3/4. 随堂练习

HZOJ-40：爬楼梯

【编写程序】一个人每次只能走2节楼梯或者3节楼梯，问走到第N节楼梯一共有多少种方法。

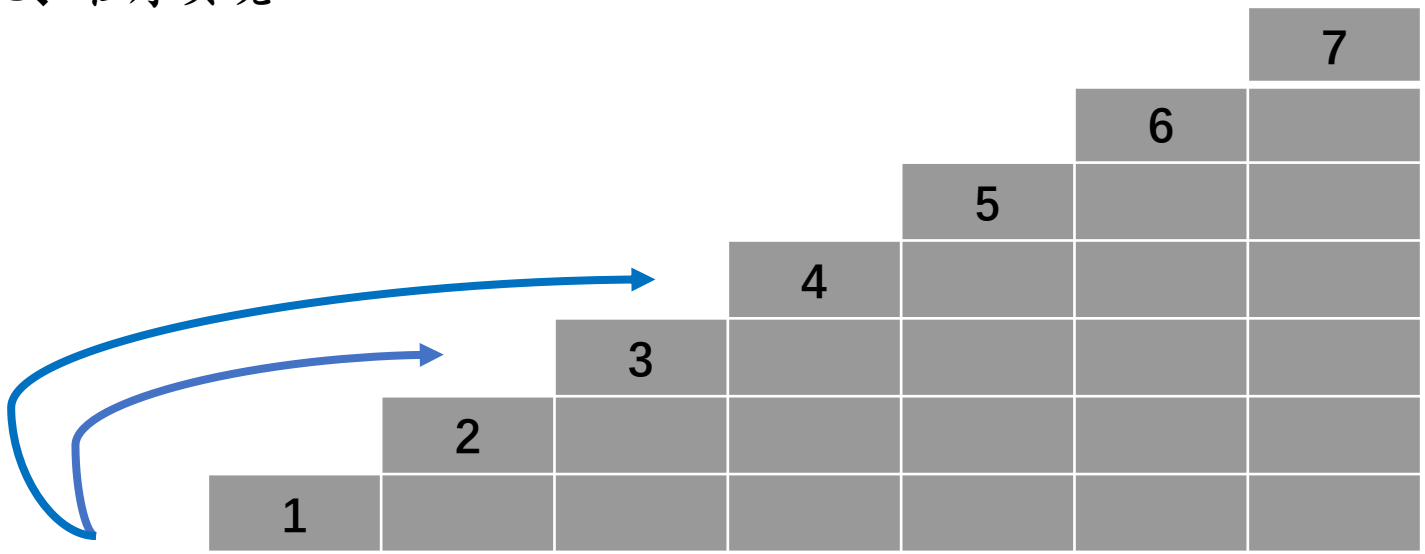


HZOJ-40：爬楼梯

1、确定递推状态

2、确定递推公式

3、程序实现



HZOJ-40：爬楼梯

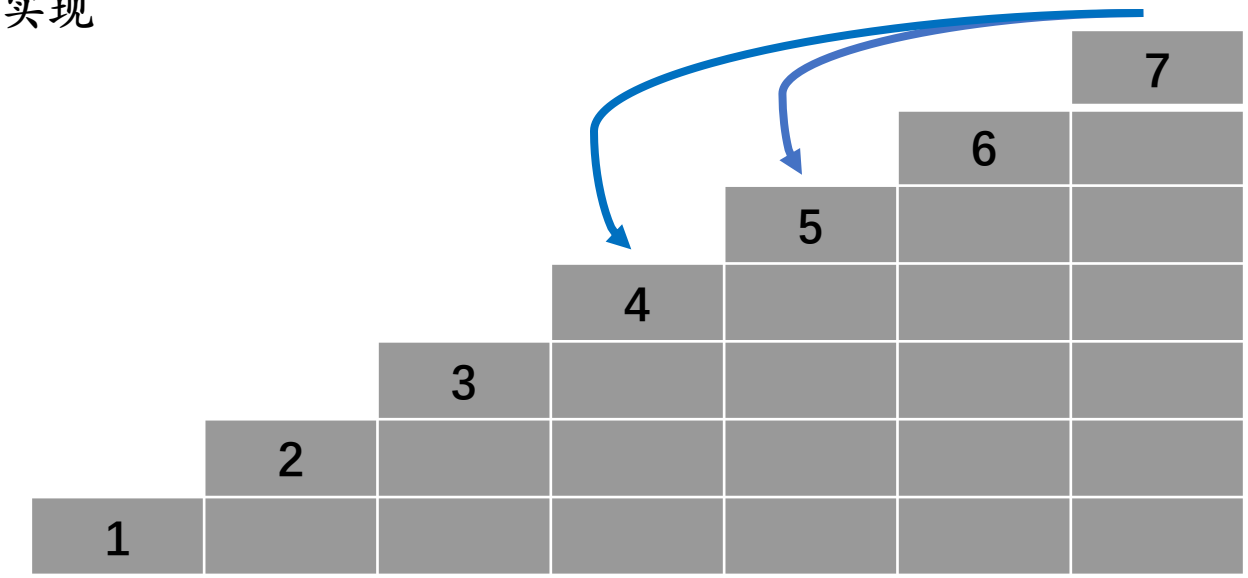
1、确定递推状态

f(n)代表走到第 n 节台阶的方法总数

2、确定递推公式

```
f(n) = f(n - 2) + f(n - 3)
```

3、程序实现



HZOJ-41：墙壁涂色

【编写程序】给一个环形的墙壁涂颜色，颜色一共有三种，分别是红、黄、蓝，墙壁被竖直地划分成wallsize个部分，相邻的部分颜色不能相同。请你写出函数paintWallCounts计算出一共有多少种给房间上色的方案。

例如，当 wallsize = 5 时，下面就是一种合法方案。

由于墙壁是环形，下面的方案就不合法。

HZOJ-41：墙壁涂色

1、确定递推状态

2、确定递推公式

例如，当 wallsize = 5 时，下面就是一种合法方案。

由于墙壁是环形，下面的方案就不合法。

HZOJ-41：墙壁涂色

1、确定递推状态

f[n][i][j] 代表前 n 块墙壁，在不考虑头尾成环的前提下，第1块涂颜色 i，第 n 块图颜色 j 的方法总数

2、确定递推公式

$$f[n][i][j] = \sum_{k=0}^{2} f[n-1][i][k] \ (k \neq j)$$

例如，当 wallsize = 5 时，下面就是一种合法方案。

由于墙壁是环形，下面的方案就不合法。

HZOJ-41：墙壁涂色

3、递推式的初值条件

4、如何利用 f 求解最终答案?

HZOJ-41：墙壁涂色

3、递推式的初值条件

```
f[1] = {
    1, 0, 0,
    0, 1, 0,
    0, 0, 1
}
```

```
f[2] = {
    0, 1, 1,
    1, 0, 1,
    1, 1, 0
}
```

4、如何利用 f 求解最终答案?

$$ans = \sum_{i=0}^{2}\sum_{j=0}^{2} f[wallsize][i][j] \ (i \neq j)$$

二、递推-课后实战题

1. 数的划分-P1025
2. 数的计算-P1028
3. 神经网络-P1038
4. 栈-P1044
5. 循环-P1050
6. 传球游戏-P1057
7. Hanoi 双塔问题-P1096

三. 动态规划

三、动态规划

1. 如何求解动态规划问题
2. 全面剖析：数字三角形问题
3. DP 经典问题入门 8 讲

3-1. 如何求解动态规划问题

如何求解动归问题

1、确定动归状态

例如：f(i,j) 代表从底边走到（i,j）点所能获得的最大值

2、确定状态转移方程

例如：$f(i,j) = max\begin{Bmatrix} f(i+1,j) \\ f(i+1,j+1) \end{Bmatrix} + val(i,j)$

3、正确性证明：求助于数学归纳法

4、程序实现

通过问题要对哪些『概念』产生感觉？

阶段

把所给求解问题的过程恰当地分成若干个相互联系的阶段，以便于求解，过程不同，阶段数就可能不同。描述阶段的变量称为阶段变量。

状态

一个阶段中，包含了若干个状态，状态与状态之间存在某种递推关系，这种递推关系一般满足『最优子结构』和『重叠子问题』两个性质。

无后效性

如果给定某一阶段的状态，则在这一阶段以后过程的发展不受这阶段以前各段状态的影响，所有各阶段都确定时，整个过程也就确定了。

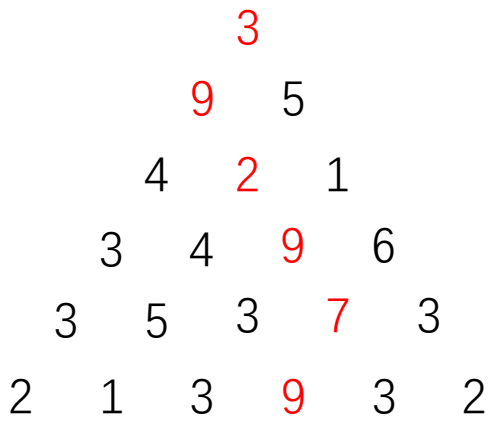
决策

一个阶段的状态给定以后，从该状态演变到下一阶段某个状态的一种选择称为决策。在许多问题中，决策可以表示为一个数或一组数。

3-2. 全面剖析：数字三角形问题

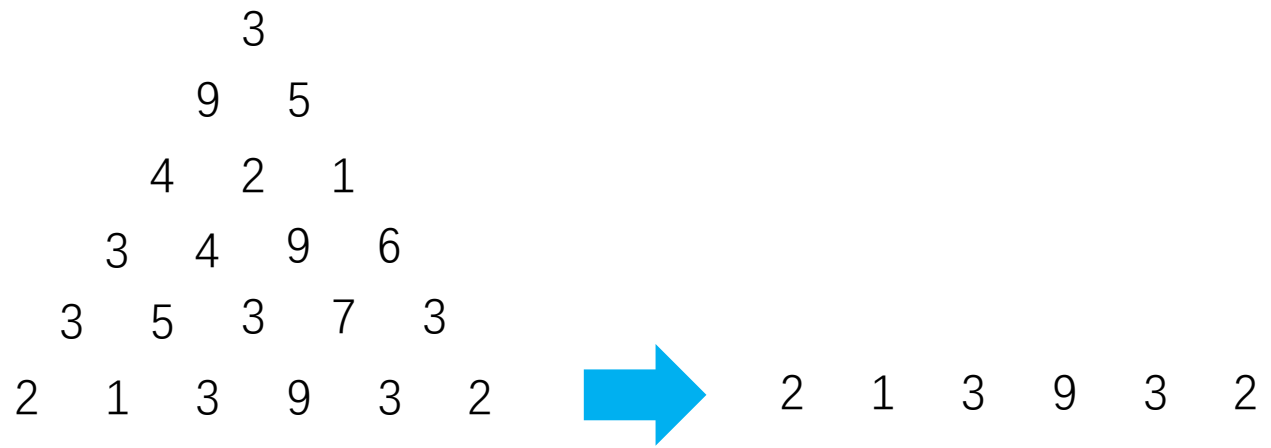
HZOJ-43：数字三角形

【编写程序】有一个由数字组成的三角形，站在上一层的某个点，只能到达其下方左右的两个点。现在请找到一条从上到下的路径，使得路径上所有数字相加之和最大。



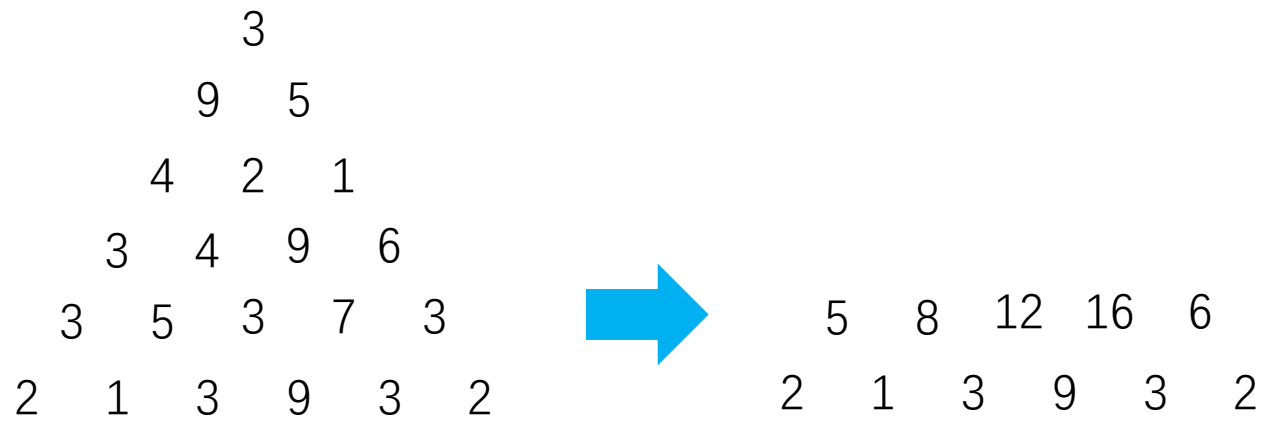
HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



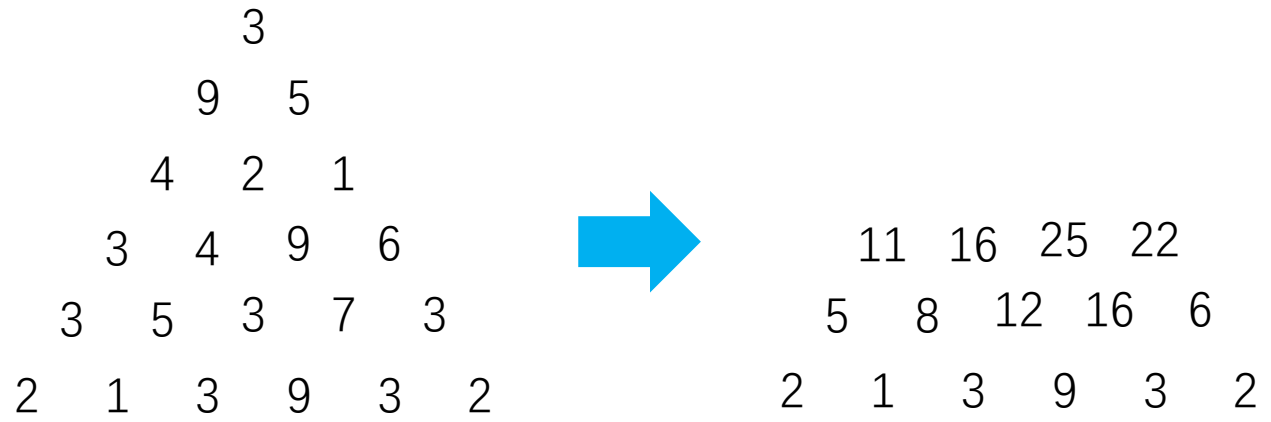
HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



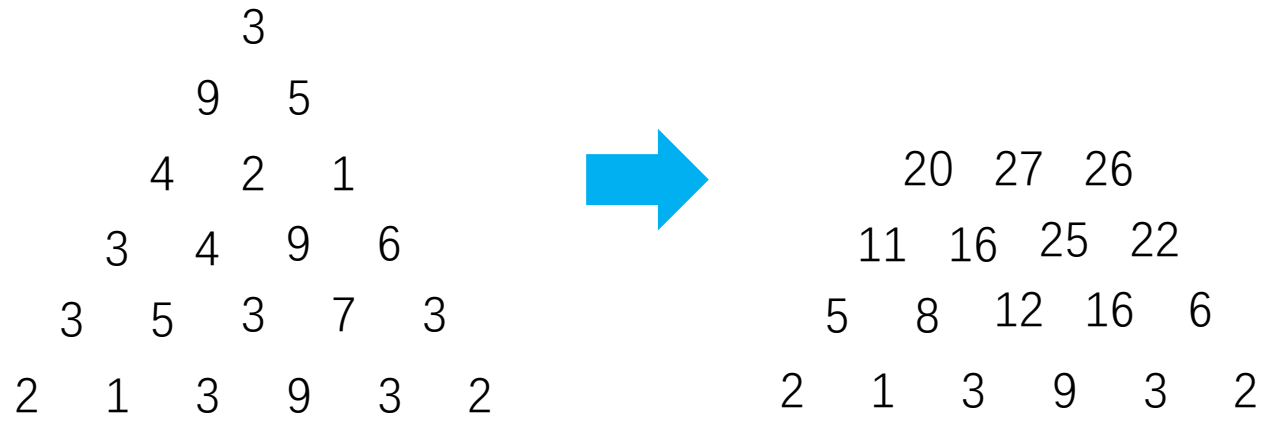
HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



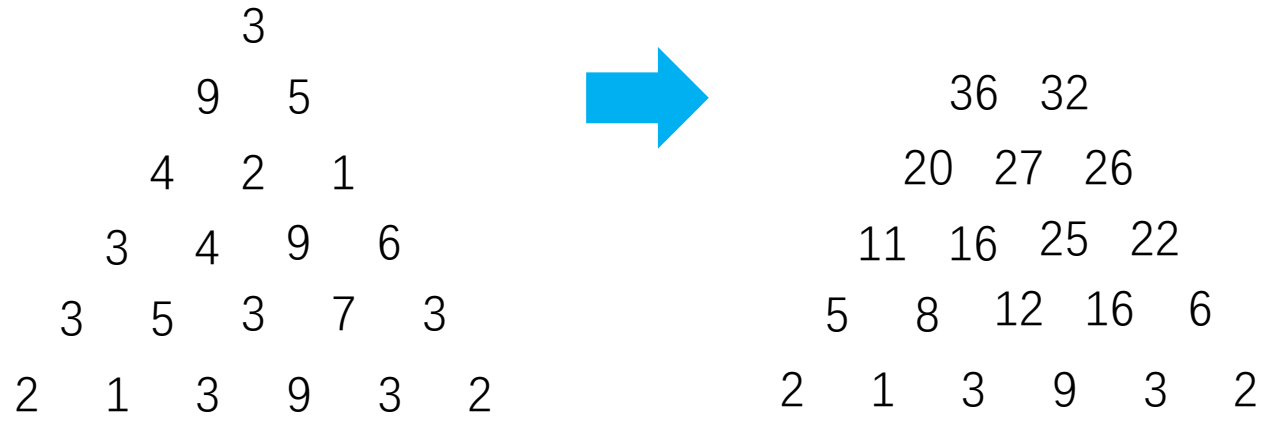
HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



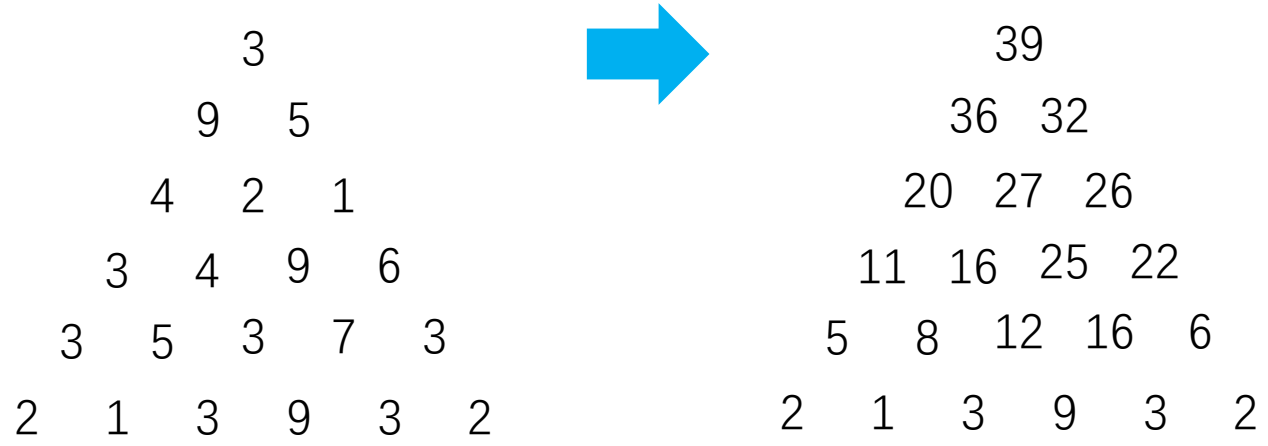
HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



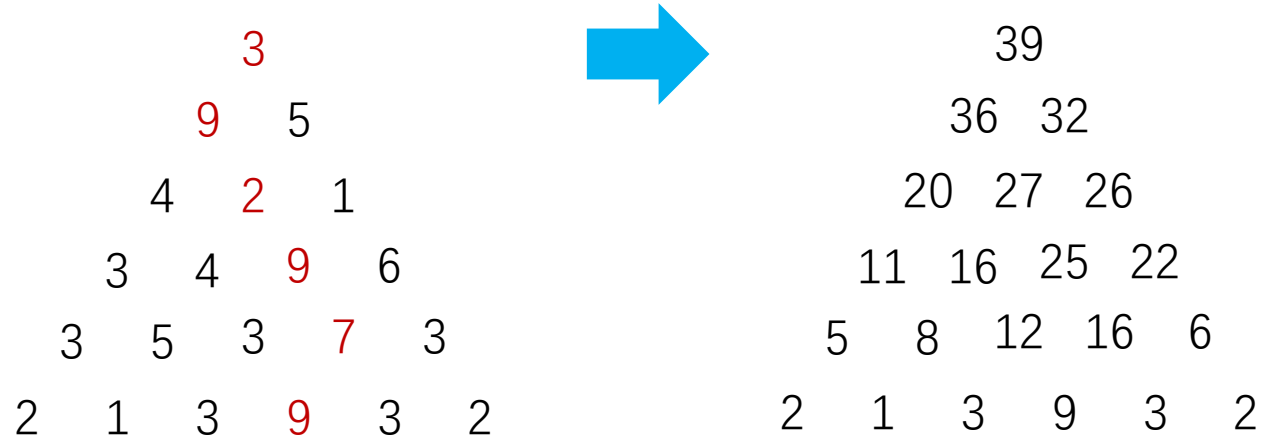
HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。



HZOJ-43：数字三角形

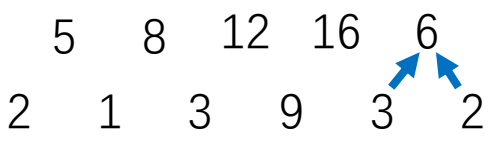
1、确定递推状态：

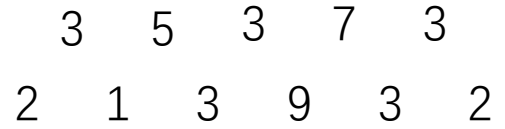
f(i,j) 代表从底边走到 (i,j) 点所能获得的最大值

2、确定递推公式

f(i,j) = max[f(i+1,j), f(i+1,j+1)] + val(i, j)

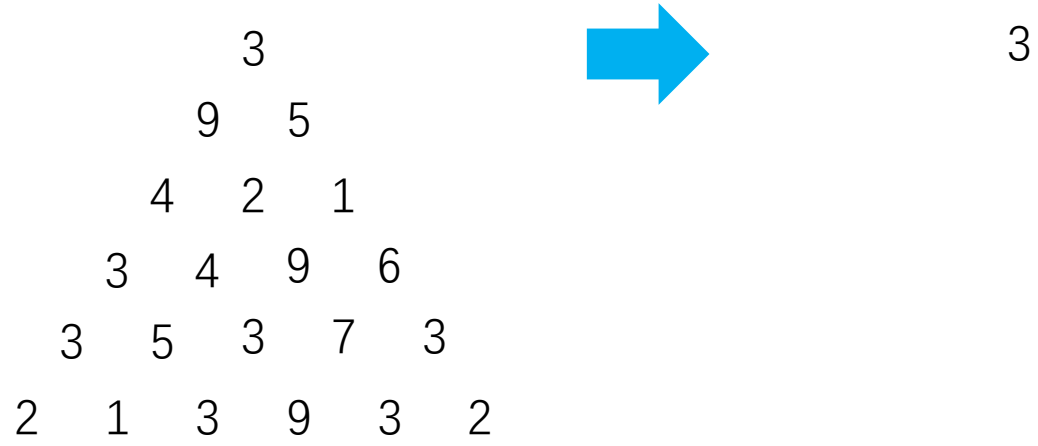
3、递推公式解读





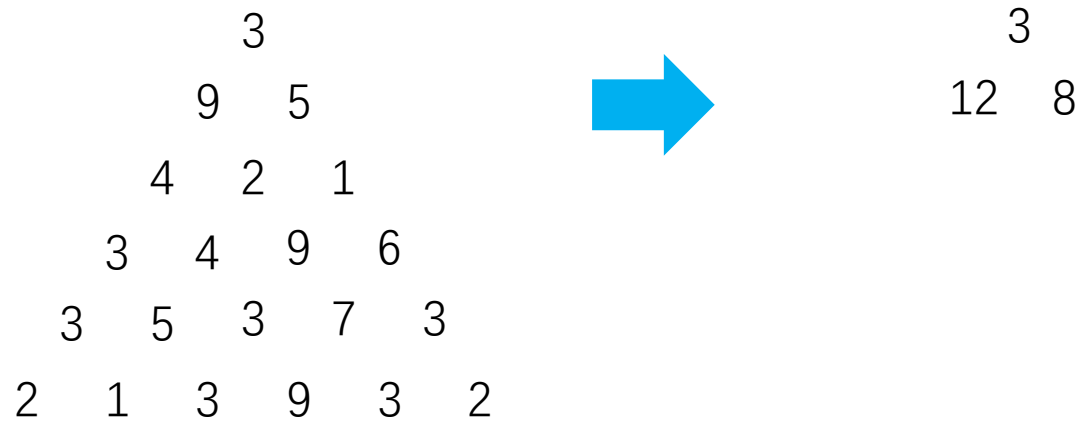
HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



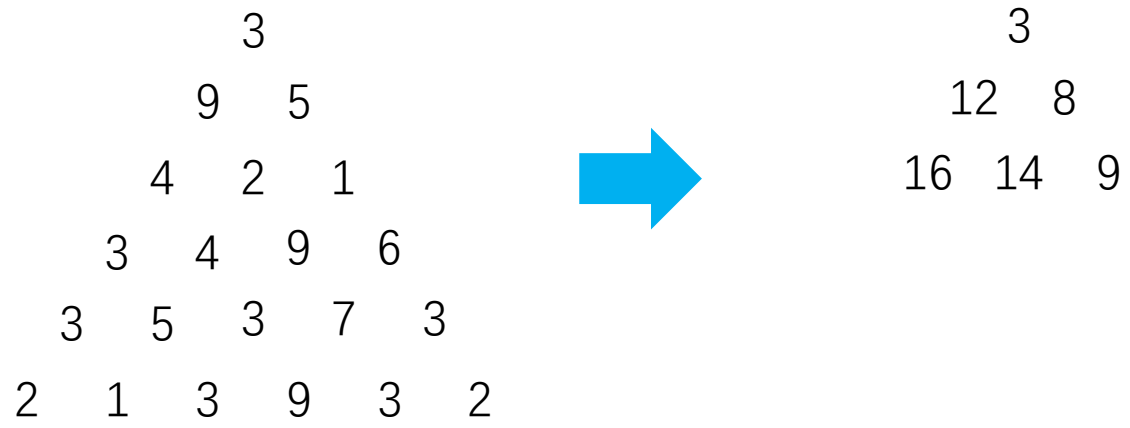
HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



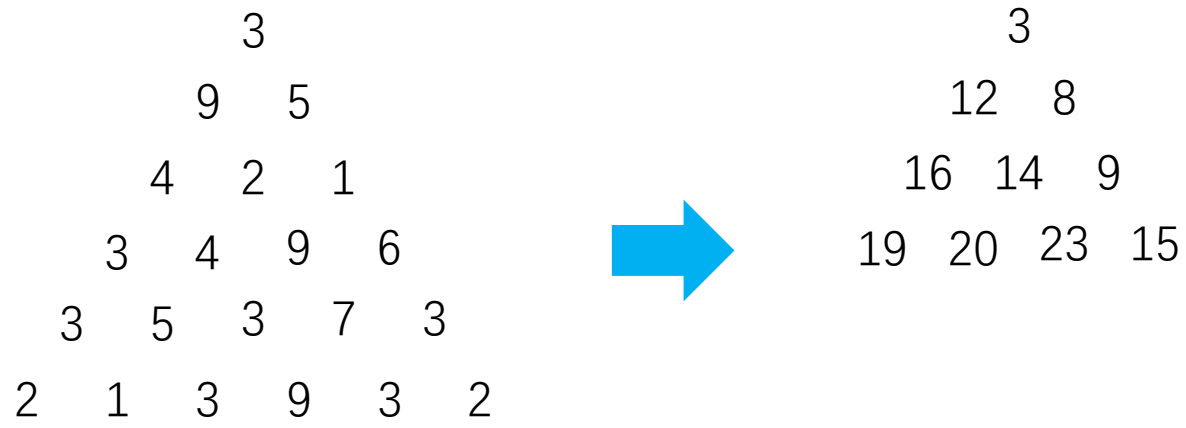
HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



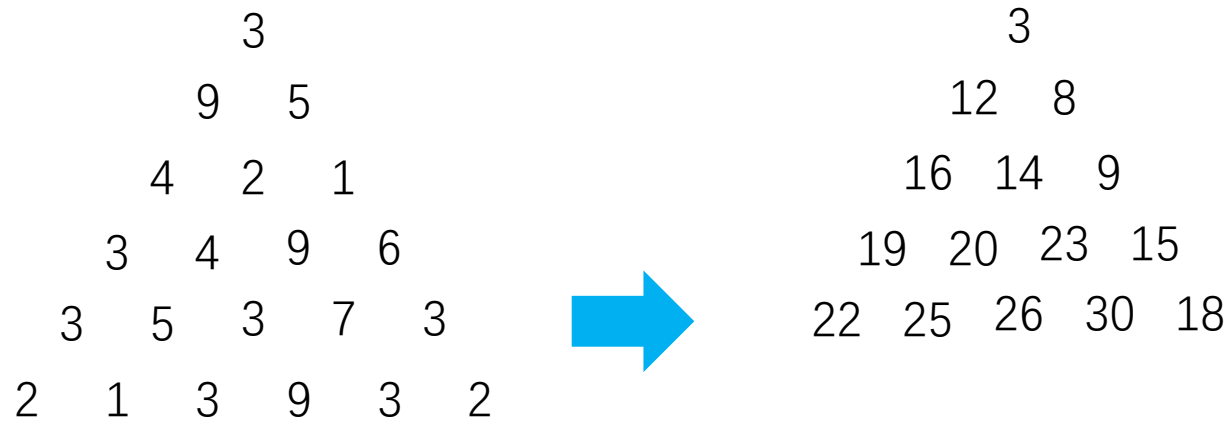
HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



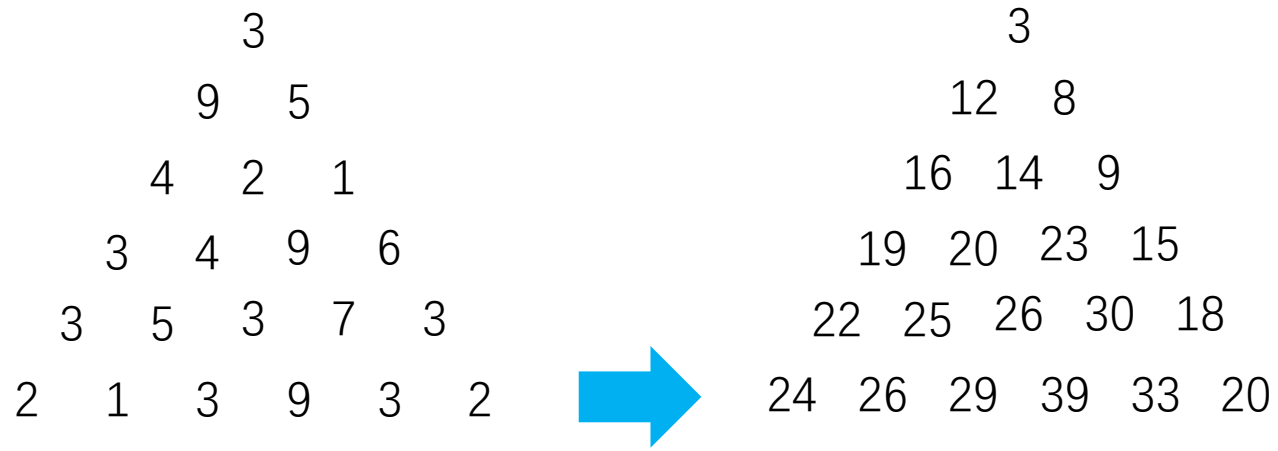
HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



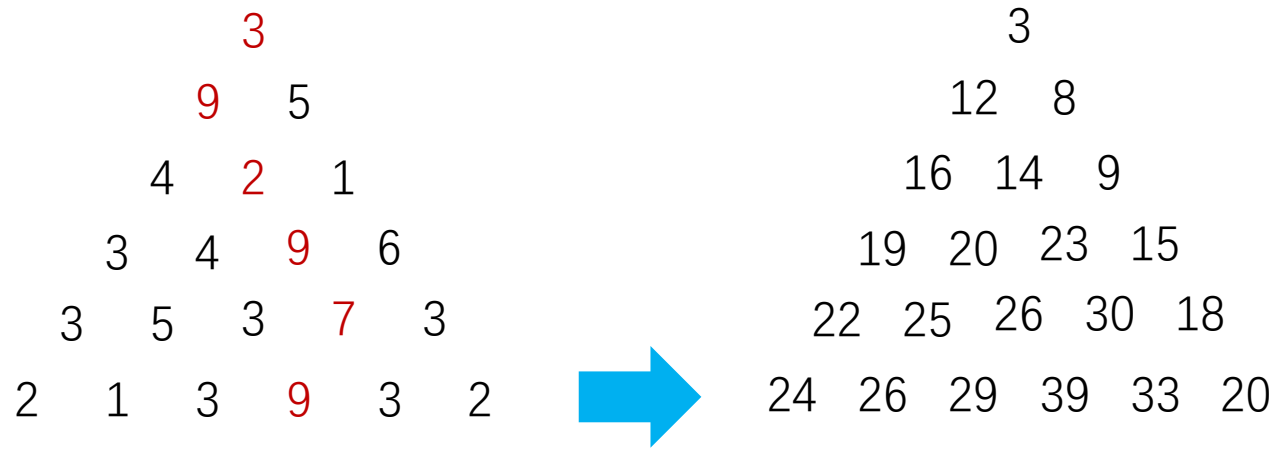
HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



HZOJ-43：数字三角形

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。



HZOJ-43：数字三角形

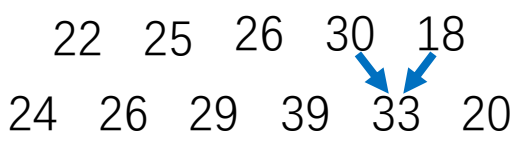
1、确定递推状态：

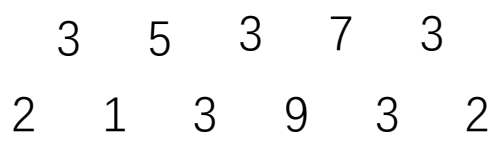
f(i,j) 代表从顶点走到 (i,j) 点所能获得的最大值

2、确定递推公式

f(i,j) = max[f(i-1,j), f(i-1,j-1)] + val(i, j)

3、递推公式解读



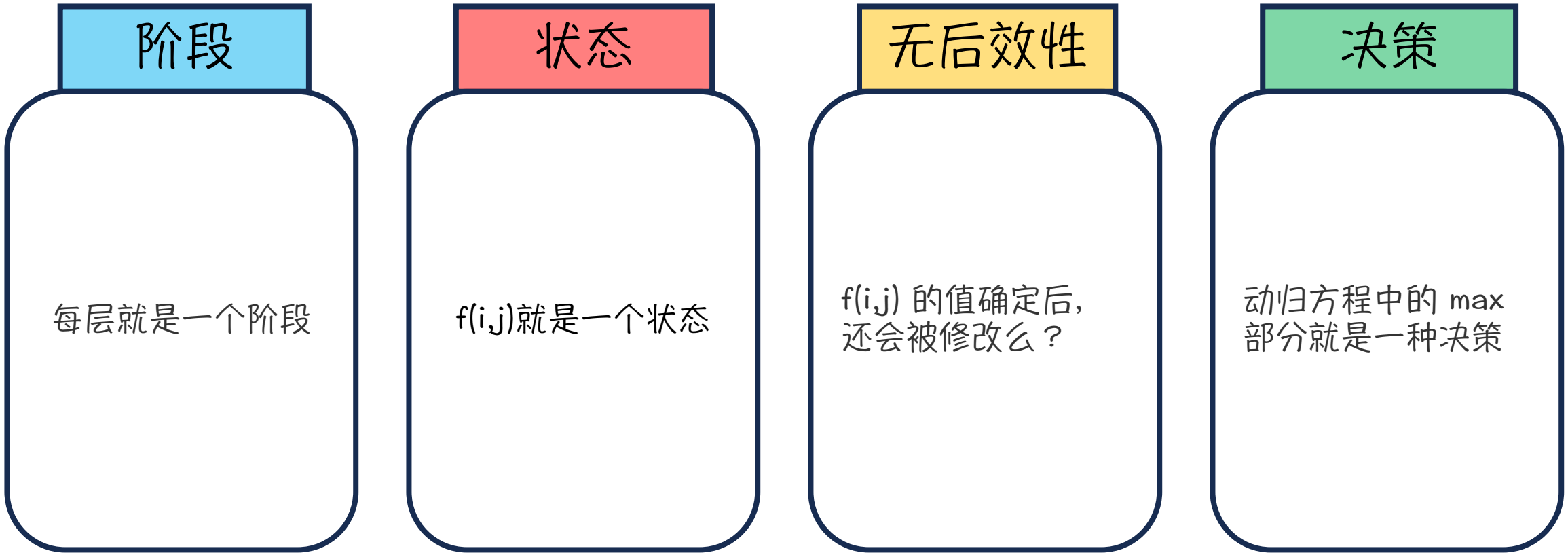


HZOJ-43：数字三角形

思考-1：如果由下向上走的话，很容易获得最后一行站在每个点上能够获得的最大值，由此得到倒数第二行的每个点的最大值，依次递推。

思考-2：如果由上向下走的话，很容易获得第一行站在每个点上能够获得的最大值，由此得到第二行的每个点的最大值，依次递推。

通过问题要对哪些『概念』产生感觉？



如何求解动归问题

1、确定动归状态

例如：f(i,j) 代表从底边走到（i,j）点所能获得的最大值

2、确定状态转移方程

例如：$f(i,j) = max\begin{Bmatrix} f(i+1,j) \\ f(i+1,j+1) \end{Bmatrix} + val(i,j)$

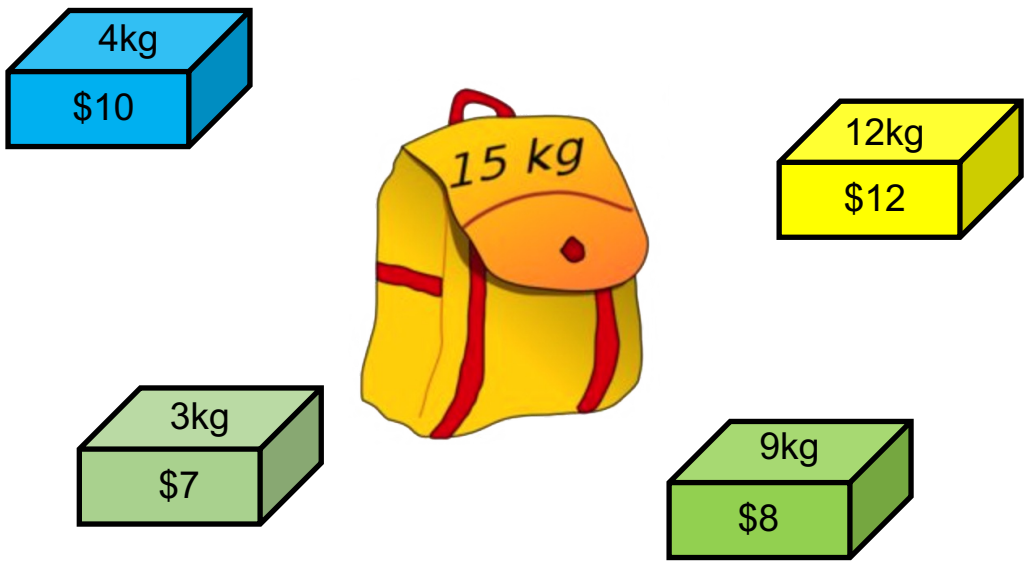
3、正确性证明：求助于数学归纳法

4、程序实现

3-3. DP经典问题

HZOJ-47：0/1背包

【编写程序】给有一个能承重 V 的背包，和 n 件物品，我们用重量和价值的二元组来表示一个物品，第 i 件物品表示为(v_i, w_i)，问在背包不超重的情况下，得到物品的最大价值是多少?



HZOJ-47：0/1背包

1、确定动归状态

dp(i,j)代表前 i 件物品，背包承重为 j 时所获得的最大价值

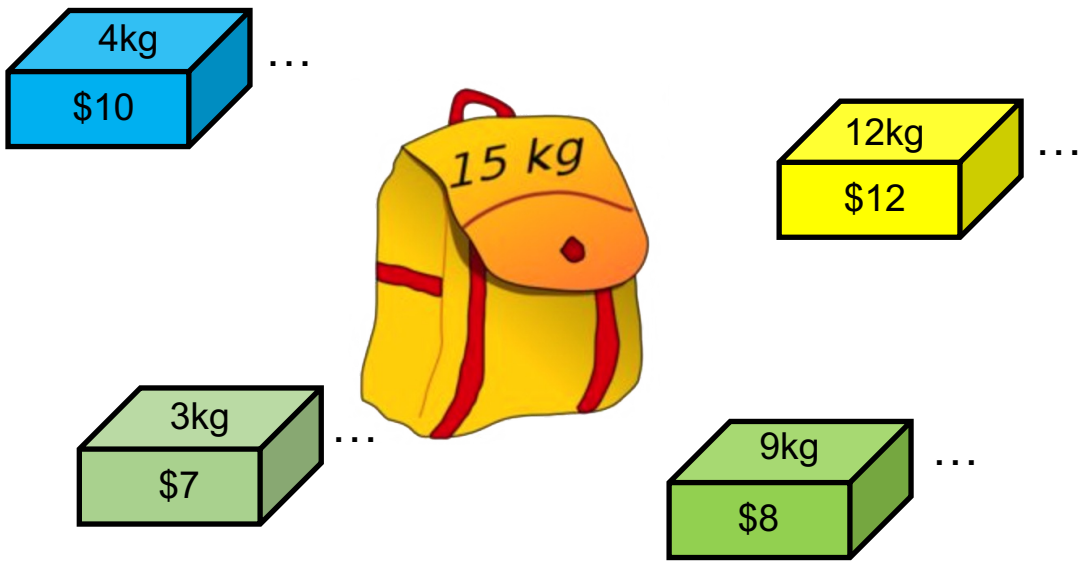
2、确定状态转移方程

$$dp(i,j) = max\begin{cases} dp[i-1][j] & |\text{不选第 i 件物品} \\ dp[i-1]\left[j-v[i]\right]+w[i] & |\text{选第 i 件物品} \end{cases}$$

3、分析容斥

HZOJ-48：完全背包

【编写程序】给有一个能承重 V 的背包，和 n 种物品，每种数量任意多，我们用重量和价值的二元组来表示一个物品，第 i 种物品表示为 (v_i, w_i)，问在背包不超重的情况下，得到物品的最大价值是多少?



HZOJ-48：完全背包

1、确定动归状态

dp(i,j)代表前 i 件物品，背包承重为 j 时所获得的最大价值

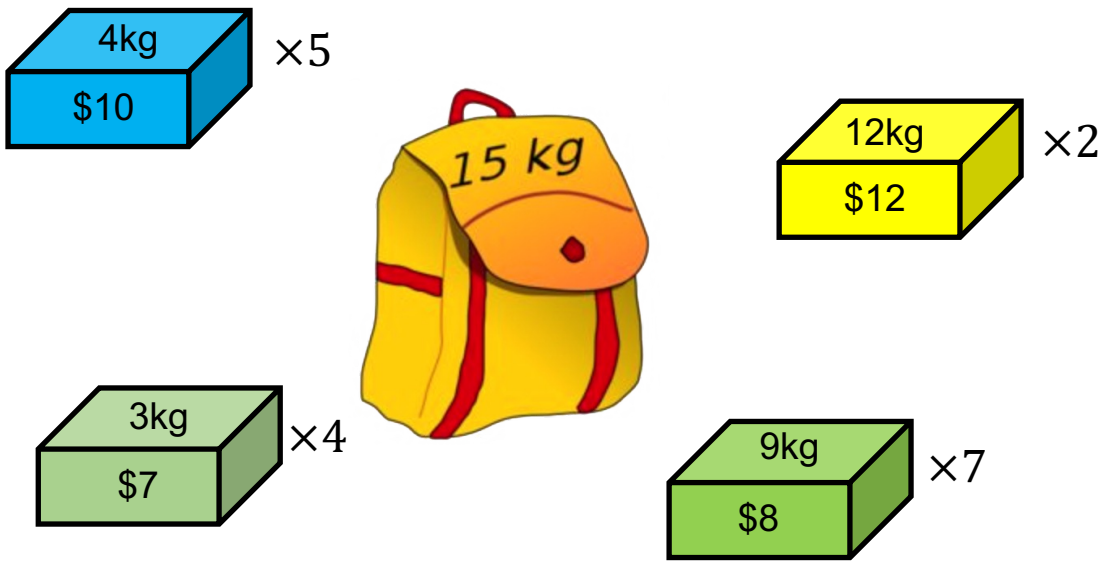
2、确定状态转移方程

$$dp(i,j) = max\begin{cases} dp[i-1][j] & | \text{不选第 i 件物品} \\ dp[i]\left[j-v[i]\right]+w[i] & | \text{选第 i 件物品} \end{cases}$$

3、分析容斥

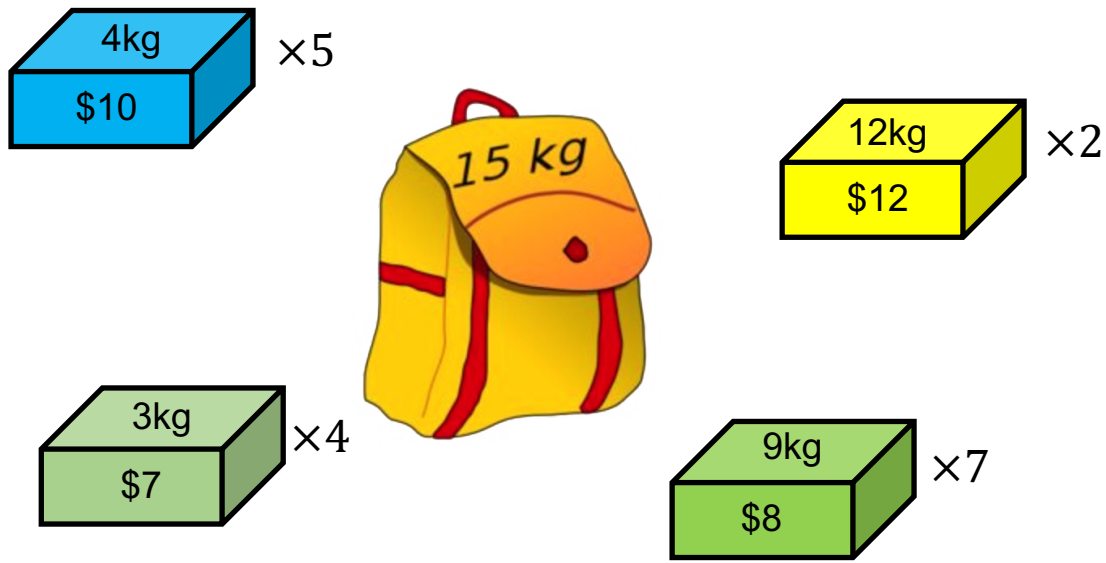
HZOJ-49：多重背包

【编写程序】给有一个能承重 V 的背包，和 n 种物品，每种物品的数量有限多，我们用重量、价值和数量的三元组来表示一种物品，第 i 种物品表示为(v_i, w_i, s_i)，问在背包不超重的情况下，得到物品的最大价值是多少？



HZOJ-49：多重背包

【思考】如下图，可否当成是包含18件物品的0/1背包做?



HZOJ-49：多重背包

1、确定动归状态

dp(i,j)代表前 i 件物品，背包承重为 j 时所获得的最大价值

2、确定状态转移方程

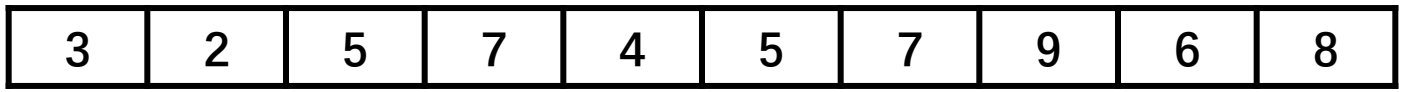
$$dp(i,j) = max\{dp[i-1][j-k*v[i]] + k*w[i]\}$$

第 i 种物品选择 k 件，$k \in [0, s_i]$

3、分析容斥

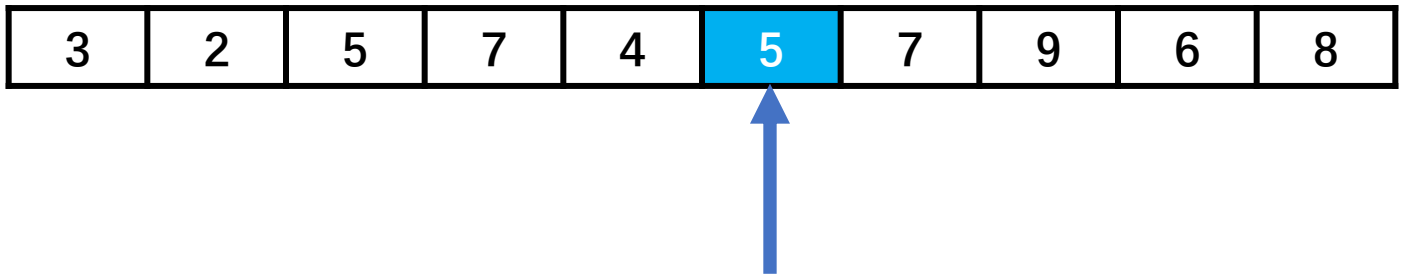
HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度



HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度



HZOJ-44：最长上升子序列

1、确定动归状态

例如：dp(i) 代表以 i 位为结尾的上升子序列最长长度

2、确定状态转移方程

例如：$dp(i) = max\{dp(j)\} + 1 \mid j \in [1,\ i\ - 1]$

3、分析容斥

HZOJ-45：最长公共子序列

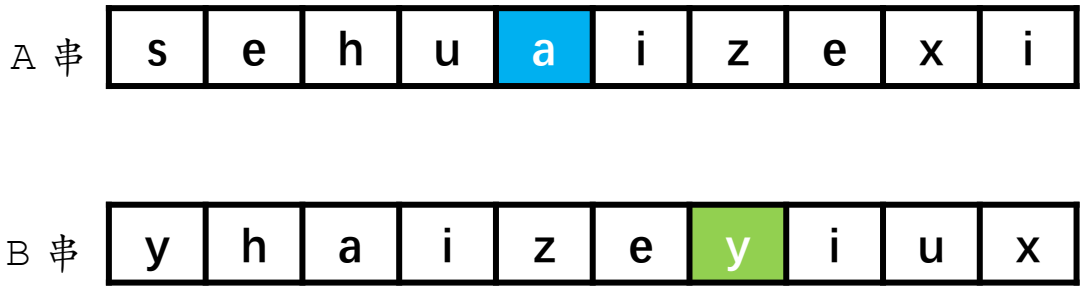
【编写程序】给出两个字符串，求其两个的最长公共子序列。

s	e	h	u	a	i	z	e	x	i

y	h	a	i	z	e	y	i	u	x

HZOJ-45：最长公共子序列

【编写程序】给出两个字符串，求其两个的最长公共子序列。



考虑：

A串长度为 i 位

B串长度为 j 位

的最长公共子序列

HZOJ-45：最长公共子序列

1、确定动归状态

dp(i,j) 代表

A串长度为 i 位，B串长度为 j 位的最长公共子序列长度

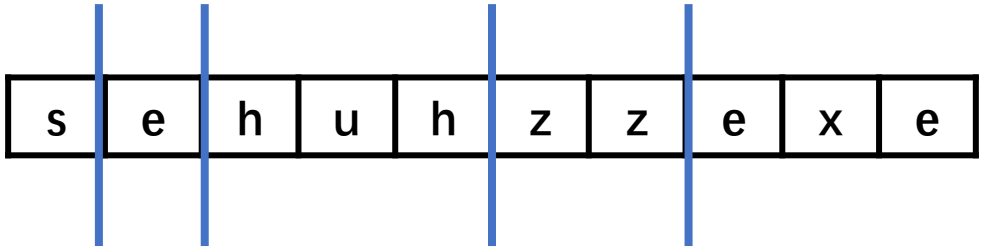
2、确定状态转移方程

$$dp(i,j) = max\begin{cases} dp(i,j-1) \\ dp(i-1,j) \\ dp(i-1,j-1) + (A[i] == B[j]) \end{cases}$$

3、分析容斥

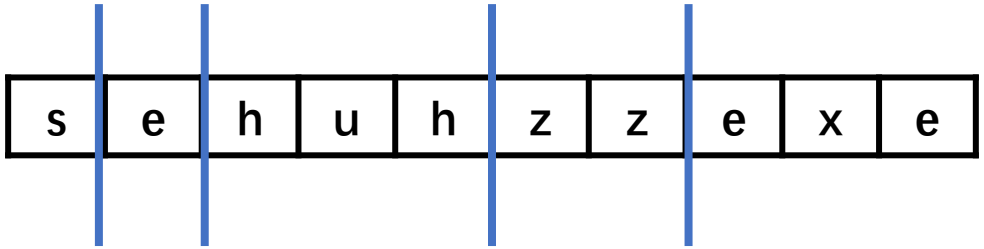
HZOJ-46：切割回文

【编写程序】给出一个字符串S，问对字符串 S 最少切几刀，使得分成的每一部分都是一个回文串（注意：单一字符，是回文串）。



HZOJ-46：切割回文

【编写程序】给出一个字符串S，问对字符串 S 最少切几刀，使得分成的每一部分都是一个回文串（注意：单一字符，是回文串）。



考虑：字符串短的时候，是否好解决?

HZOJ-46：切割回文

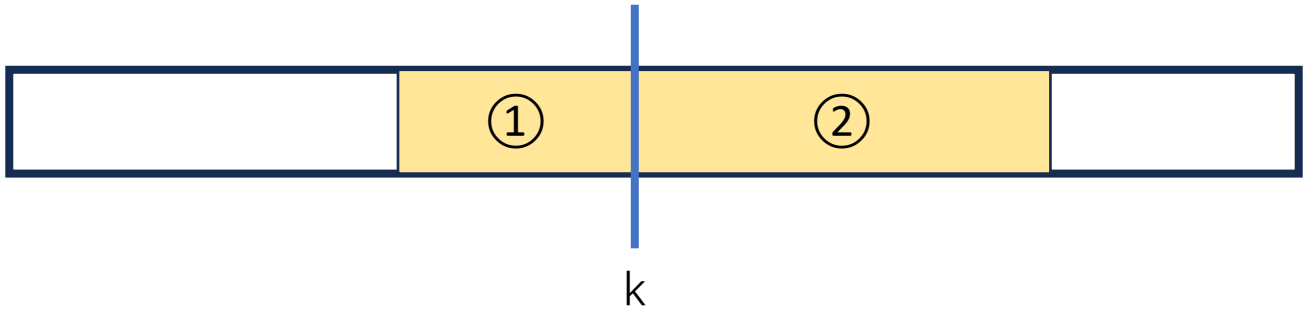
1、确定动归状态

dp(i,j) 代表从 i 到 j 最少切多少刀

2、确定状态转移方程

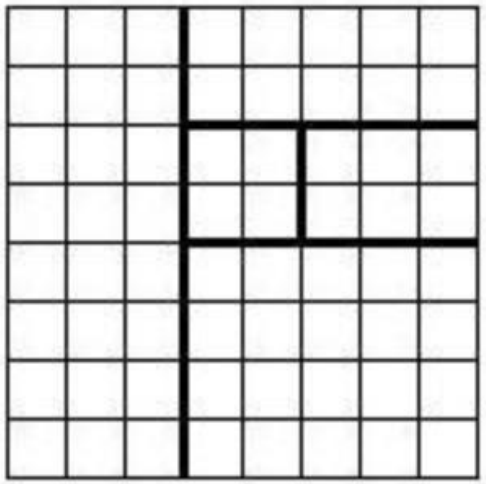
$$dp(i,j) = min\begin{Bmatrix} dp(i,k) + dp(k+1,j) + 1 | \, k \in [i,j) \\ 0 \quad | \, S[i] = S[j] \text{ 且 } dp(i+1,j-1) = 0 \end{Bmatrix}$$

3、分析转移方式

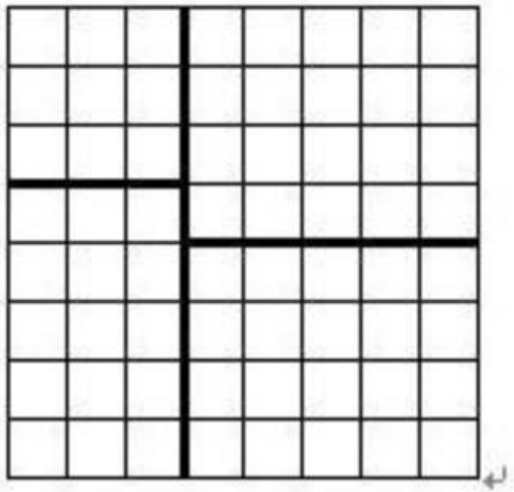


HZOJ-360：棋盘分割

【编写程序】给一个棋盘，切分成 n 块棋盘以后，求最小的平方和值。



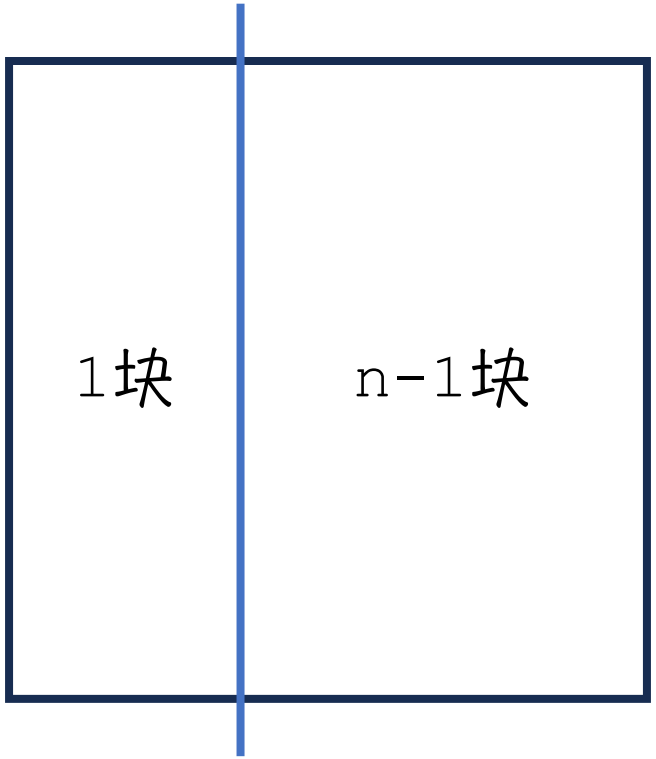
允许的分割方案



不允许的分割方案

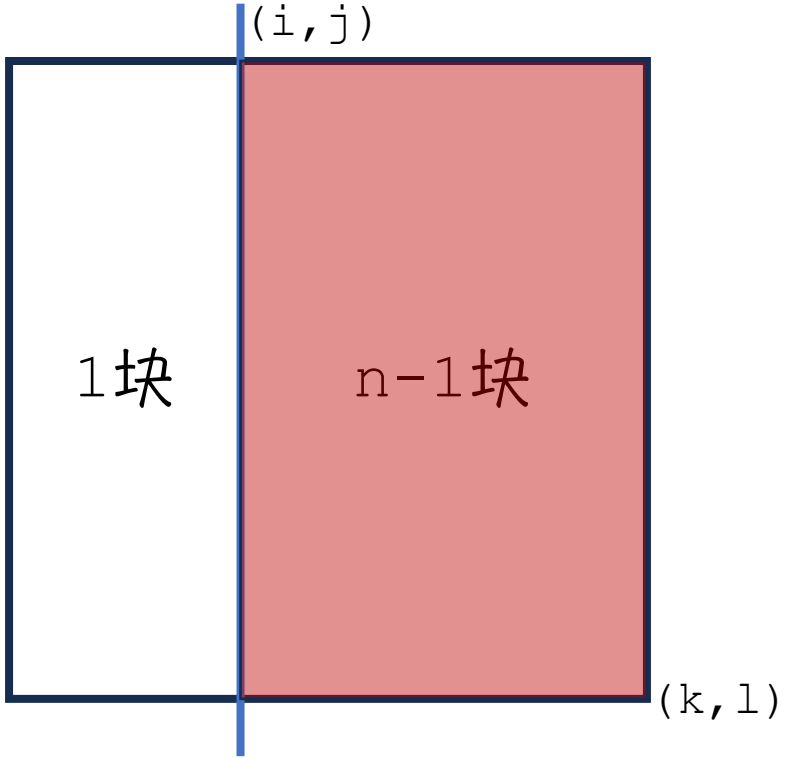
HZOJ-360：棋盘分割

【编写程序】给一个棋盘，切分成 n 块棋盘以后，求最小的平方和值。



HZOJ-360：棋盘分割

【编写程序】给一个棋盘，切分成 n 块棋盘以后，求最小的平方和值。



HZOJ-360：棋盘分割

1、确定动归状态

dp(n,i,j,k,l) 代表从 (i,j)->(k,l)的矩形，切分 n 块的最小平方和值。

2、确定状态转移方程

$$dp(n,i,j,k,l) = min\begin{Bmatrix} dp(n,i,j,c,l) + dp(n-1,c+1,j,k,l) \mid c \in [i,k) \\ dp(n-1,i,j,c,l) + dp(1,c+1,j,k,l) \mid c \in [i,k) \\ dp(1,i,j,k,c) + dp(n-1,i,c+1,k,l) \mid c \in [j,l) \\ dp(n-1,i,j,k,c) + dp(1,i,c+1,k,l) \mid c \in [j,l) \end{Bmatrix}$$

3、分析转移方式

四. 动态规划的优化

四、动态规划的优化

1. 优化①：去除冗余状态
2. 优化②：状态重定义
3. 优化③：优化转移过程
4. 优化④：斜率优化

4-1. 去除冗余状态

P1541：乌龟棋

1、确定动归状态

2、确定状态转移方程

HZOJ-47：0/1背包

1、确定动归状态

dp(i,j)代表前 i 件物品，背包承重为 j 时所获得的最大价值

2、确定状态转移方程

$$dp(i,j) = max\begin{cases} dp[i-1][j] & |\text{不选第 i 件物品} \\ dp[i-1]\left[j-v[i]\right]+w[i] & |\text{选第 i 件物品} \end{cases}$$

4-2. 状态重定义

HZOJ-41：墙壁涂色

1、确定递推状态

f[n][i][j] 代表前 n 块墙壁，在不考虑头尾成环的前提下，第1块涂颜色 i，第 n 块图颜色 j 的方法总数

2、确定递推公式

$$f[n][i][j] = \sum_{k=0}^{2} f[n-1][i][k] \ (k \neq j)$$

例如，当 wallsize = 5 时，下面就是一种合法方案。

由于墙壁是环形，下面的方案就不合法。

HZOJ-41：墙壁涂色

1、确定递推状态

f[n][j] 代表前 n 块墙壁，在不考虑头尾成环的前提下，第1块涂颜色 0，第 n 块图颜色 j 的方法总数

2、确定递推公式

$$f[n][j] = \sum_{k=0}^{2} f[n-1][k] \ (k \neq j)$$

3、最终答案

$$ans = 3 \times (f[n][1] + f[n][2])$$

HZOJ-41：墙壁涂色

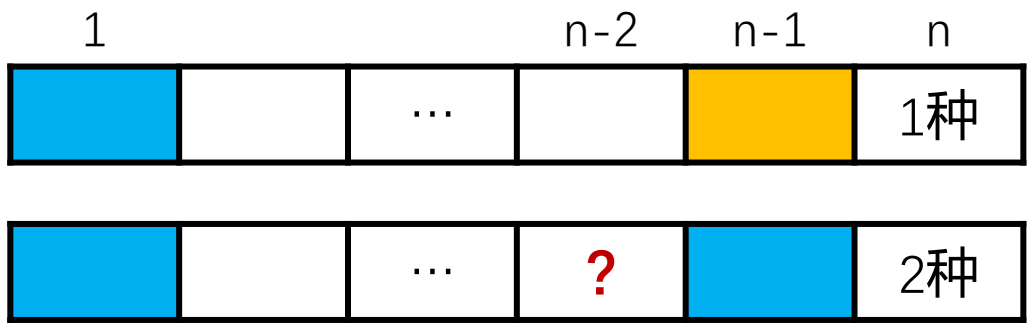
1、确定递推状态

f[n] 代表前 n 块墙壁，首尾颜色不同的方法总数

2、确定递推公式

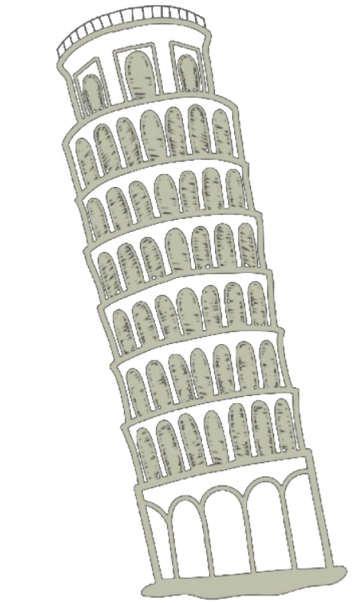
$$f[n] = f[n-1] + 2 \times f[n-2]$$

4、公式的理解

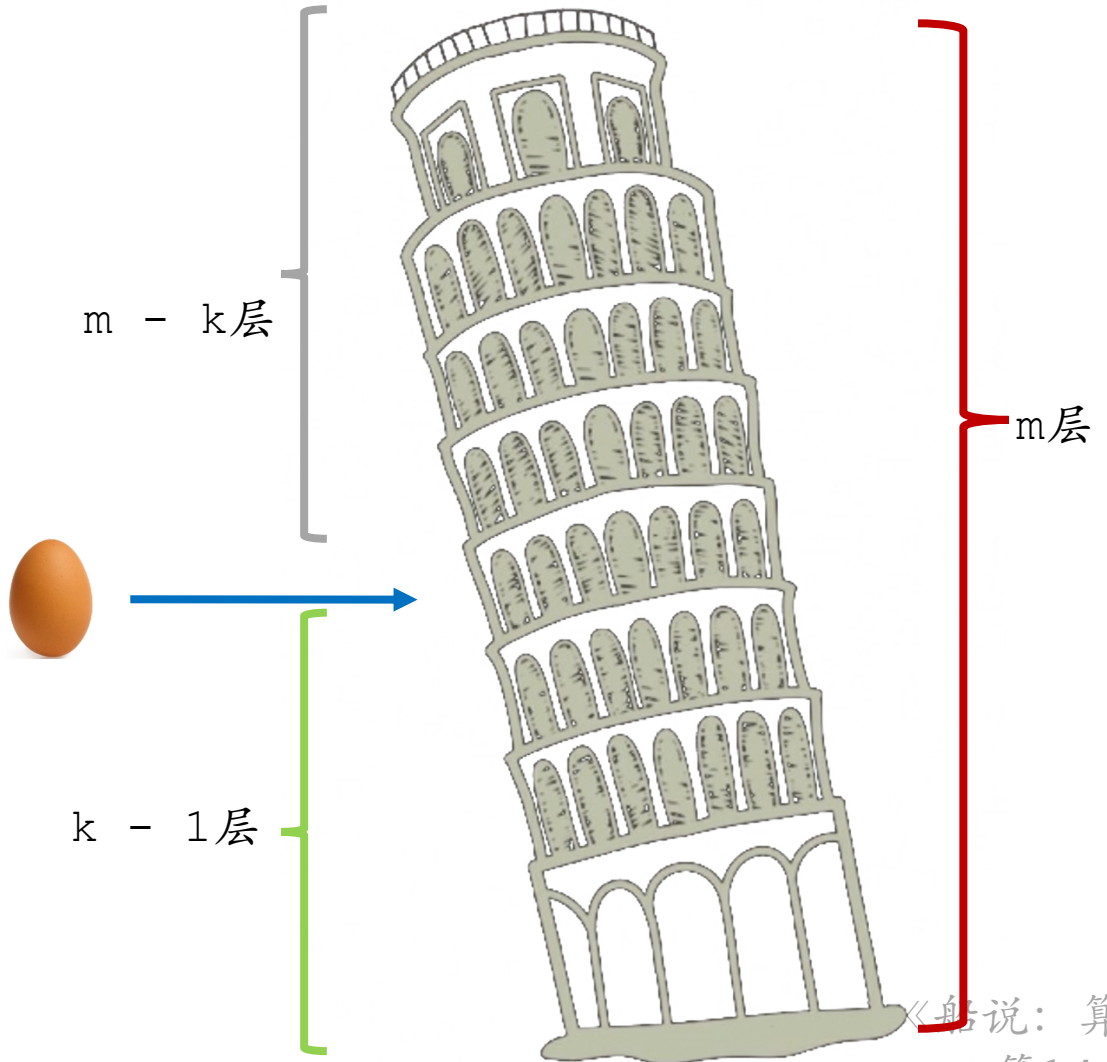


HZOJ-50：扔鸡蛋

【编写程序】定义鸡蛋的硬度为 k，则代表鸡蛋最高从 k 楼扔下来不会碎掉，现在给你 n 个硬度相同的鸡蛋，楼高为 m，问最多最少测多少次，可以测出鸡蛋的硬度。



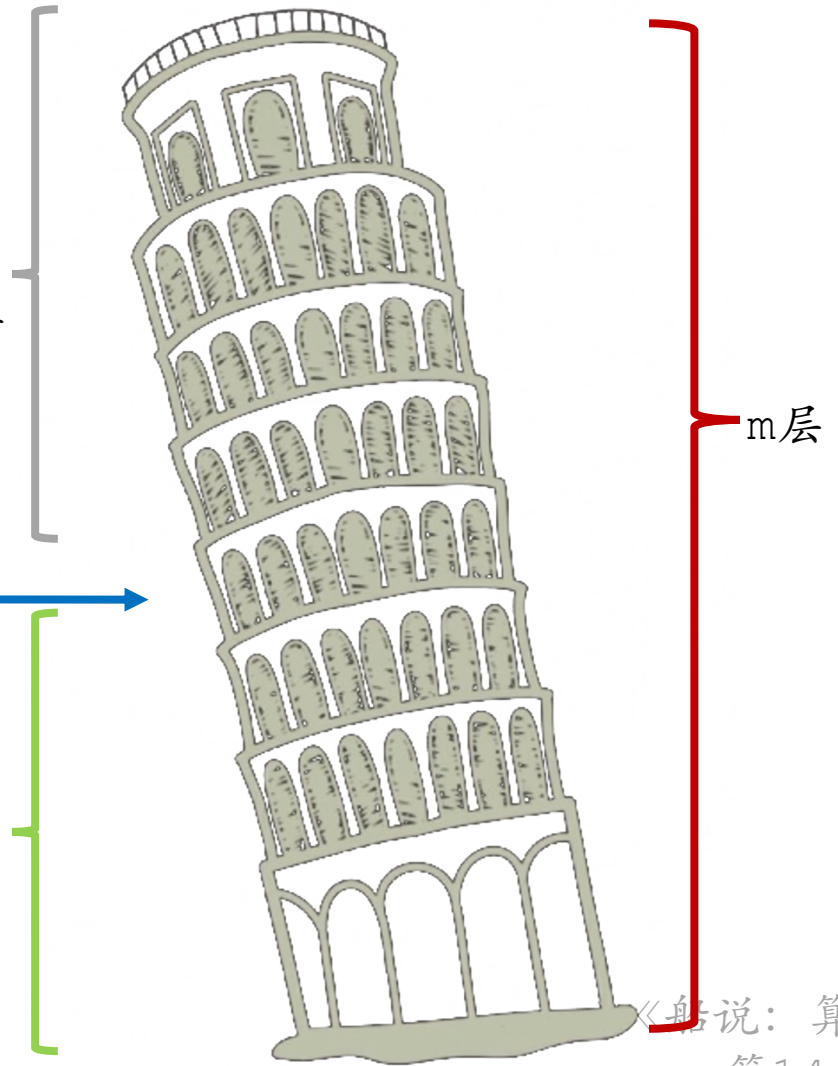
HZOJ-50：扔鸡蛋



HZOJ-50：扔鸡蛋

没碎，用 n 个鸡蛋，测 m-k 层

碎了，用n-1个鸡蛋测 k-1 层



HZOJ-50：扔鸡蛋

1、确定动归状态

dp(i,j)代表前 i 个鸡蛋测 j 层楼的最多最少次数

2、确定状态转移方程

$$dp(i,j) = min\{\, max(dp[i][j-k], dp[i-1][k-1])\} + 1$$

第 1 次扔鸡蛋，选择 k 层，$k \in [1,j]$

3、时空复杂度分析

状态数量是n*m的，转移是O(n)的，总时间复杂度O(nm^2)

空间复杂度O(nm)的

HZOJ-50：扔鸡蛋

思考：如果m特别大，怎么办？可以想象成10000000000这么大。

1、确定动归状态

dp(i,j)代表前n个鸡蛋测j层楼的最多最少次数

2、时空复杂度分析

状态数量是n*m的，转移是O(n)的，总时间复杂度$O(n^2m)$

空间复杂度O(nm)的

HZOJ-50：扔鸡蛋

思路引导：

如 dp[i][j]=k，观察发现 j 和 k 正相关，j 的范围大，k 的范围小，试着将 j 和 k 互换一下试试。

HZOJ-50：扔鸡蛋

1、确定动归状态

dp(i,k)代表 i 个鸡蛋测 k 次，最多能测多少层楼

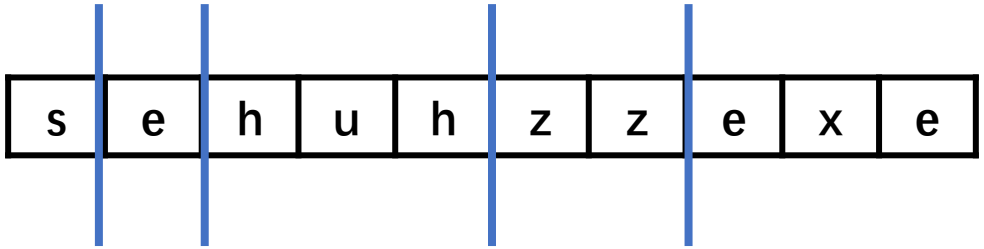
2、确定状态转移方程

$$dp(i,\mathrm{k}) = dp[i][k-1] + dp[i-1][k-1] + 1$$

4-3. 优化转移过程

HZOJ-46：切割回文

【编写程序】给出一个字符串S，问对字符串 S 最少切几刀，使得分成的每一部分都是一个回文串（注意：单一字符，是回文串）。



HZOJ-46：切割回文

1、确定动归状态

dp(i,j) 代表从 i 到 j 最少切多少刀

2、确定状态转移方程

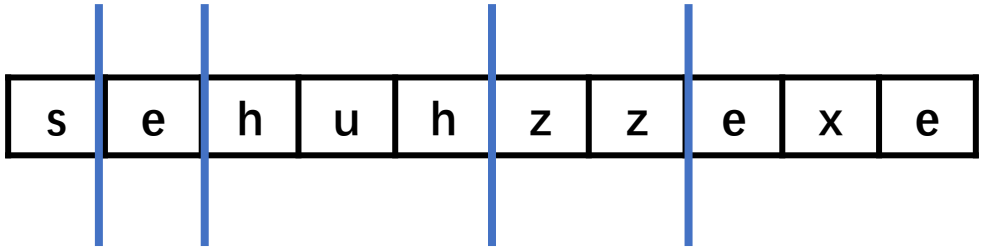
$$dp(i,j) = min\begin{Bmatrix} dp(i,k) + dp(k+1,j) + 1 | \, k \in [i,j) \\ 0 \quad | \, S[i] = S[j] \text{ 且 } dp(i+1,j-1) = 0 \end{Bmatrix}$$

3、分析时间复杂度

枚举区间是 N^2 的，区间内枚举切割的位置是 N 的，综合来说，时间复杂度是 O(N^3) 的。

HZOJ-46：切割回文

思考：观察下图，沿着区间 DP 的思想，固定区间起始位置，改成序列 DP 的思考方式，以每个位置作为结尾，只需要关心最后一刀切在哪里。



HZOJ-46：切割回文

1、确定动归状态

dp(i) 代表从1到 i 最少切多少刀

2、确定状态转移方程

$$dp(i) = min\{dp[k] + 1\} \mid \mathrm{S[k+1..i]\ is\ a\ palindrome}$$

3、填表格

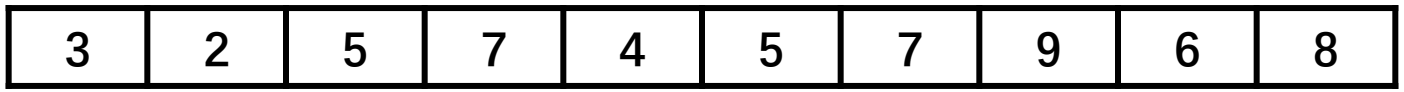
h	e	x	e	u	e	x	e	h	u	e	x	e	l	g	d

4、分析时间复杂度

初始化回文串信息 N^2，求解 dp 数组 N^2，总的时间复杂度 O(N^2)

HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度



HZOJ-44：最长上升子序列

1、确定动归状态

dp(i) 代表以 i 位为结尾的上升子序列最长长度

2、确定状态转移方程

$$dp(i) = max\{dp(j)\} + 1 \mid j \in [1,\ i\ - 1]$$

3、时间复杂度分析

求解 dp(i) 需要遍历 i - 1 次，所以总体时间复杂度为 O(n^2)

HZOJ-44：最长上升子序列

1、确定动归状态

dp(i) 代表以 i 位为结尾的上升子序列最长长度

len(j) 是一个动态更新的数组，记录长度为 j 序列的末尾最小值

2、确定状态转移方程

$$dp(i) = BinarySearch(val[i]) + 1$$

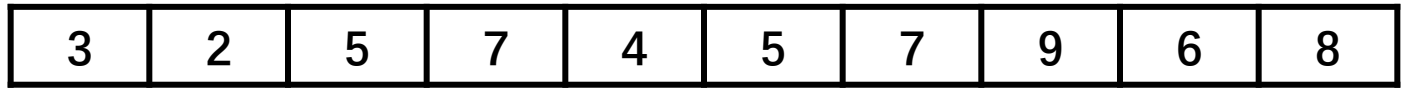
$$len(dp(i)) = val[i]$$

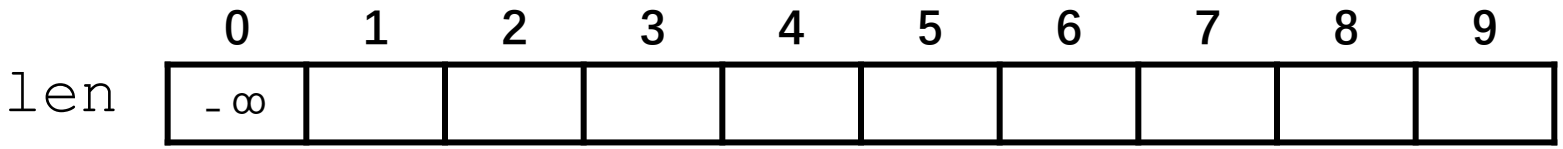
3、时间复杂度分析

求解 dp(i) 需要遍历 logn 次操作，所以总体时间复杂度O(nlogn)

HZOJ-44：最长上升子序列

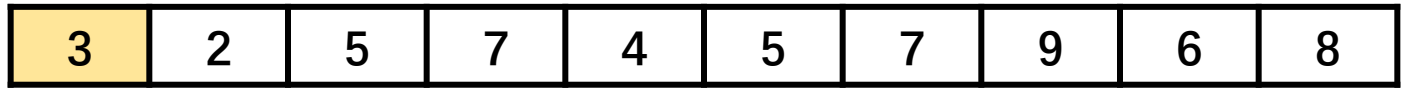
【编写程序】有一个数字序列，求其中最长上升子序列的长度

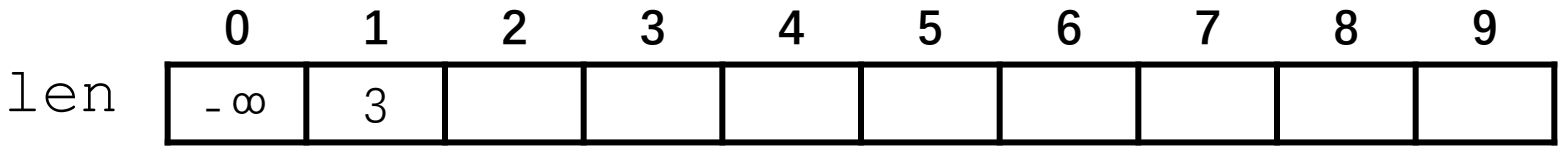




HZOJ-44：最长上升子序列

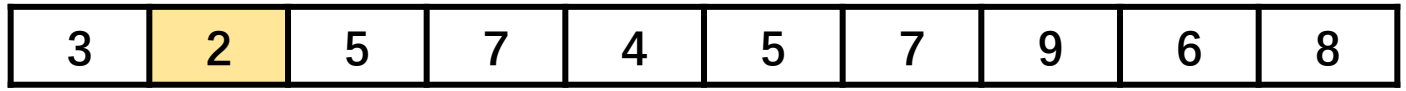
【编写程序】有一个数字序列，求其中最长上升子序列的长度

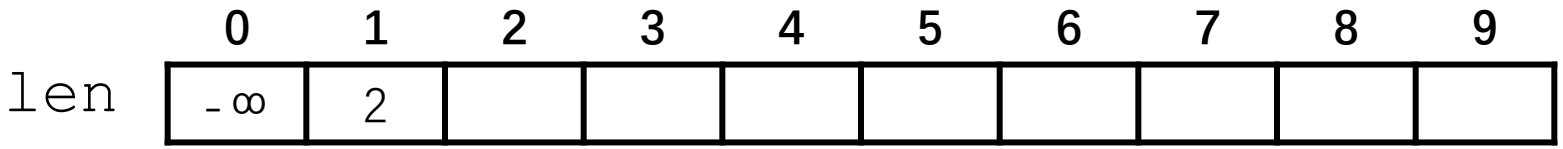




HZOJ-44：最长上升子序列

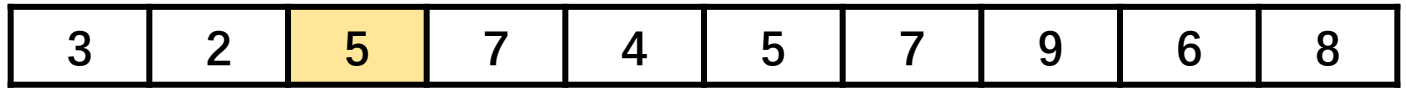
【编写程序】有一个数字序列，求其中最长上升子序列的长度

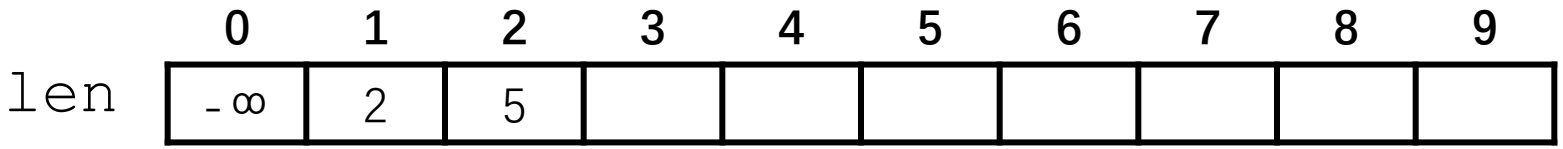




HZOJ-44：最长上升子序列

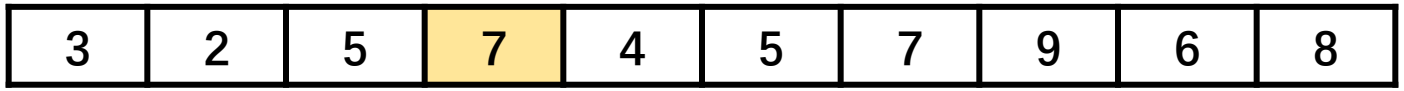
【编写程序】有一个数字序列，求其中最长上升子序列的长度

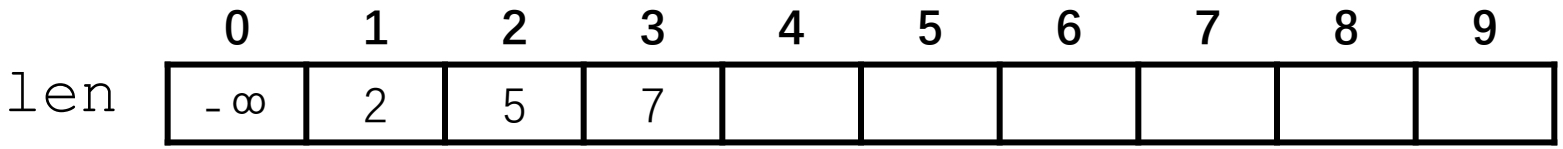




HZOJ-44：最长上升子序列

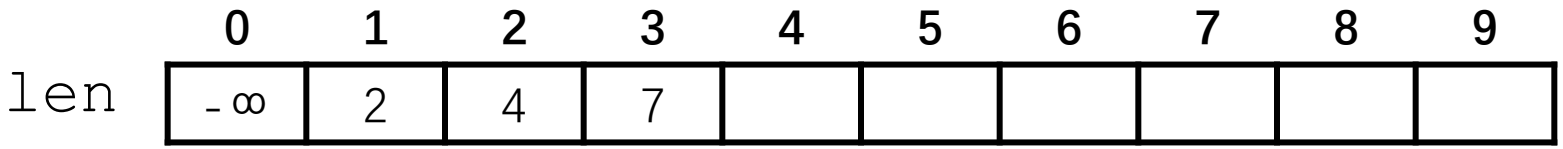
【编写程序】有一个数字序列，求其中最长上升子序列的长度





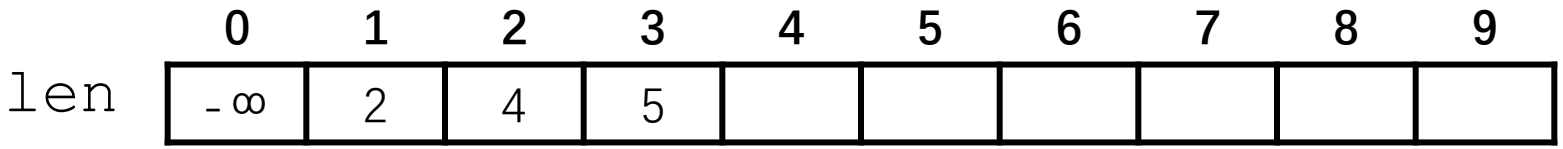
HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度



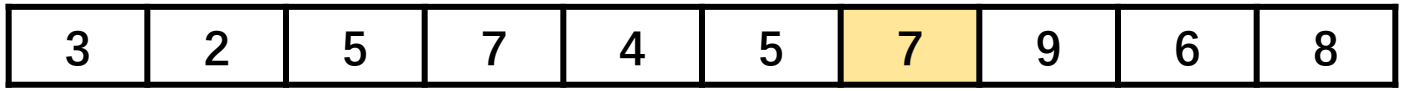
HZOJ-44：最长上升子序列

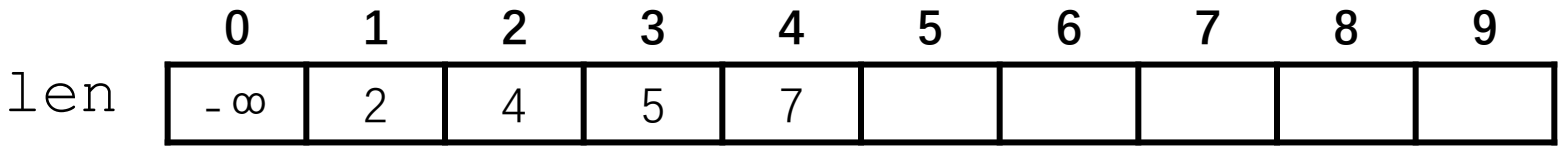
【编写程序】有一个数字序列，求其中最长上升子序列的长度



HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度





HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度

3	2	5	7	4	5	7	9	6	8

	0	1	2	3	4	5	6	7	8	9
len	-∞	2	4	5	7	9				

HZOJ-44：最长上升子序列

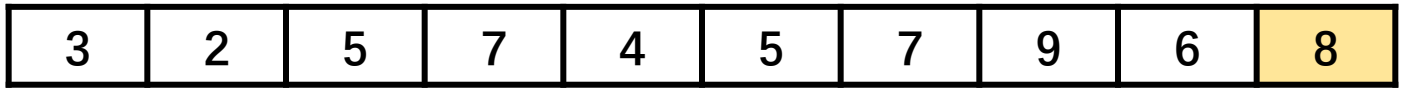
【编写程序】有一个数字序列，求其中最长上升子序列的长度

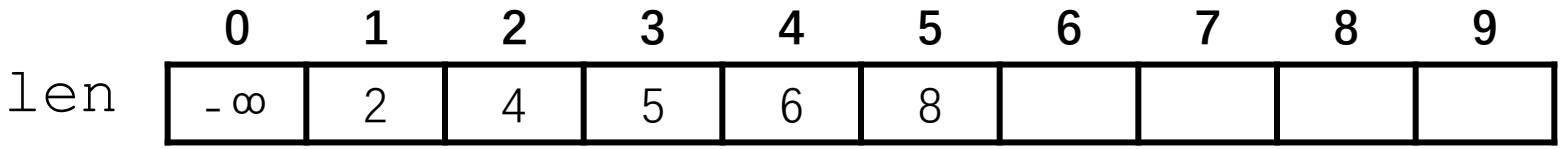
3	2	5	7	4	5	7	9	6	8

	0	1	2	3	4	5	6	7	8	9
len	-∞	2	4	5	6	9				

HZOJ-44：最长上升子序列

【编写程序】有一个数字序列，求其中最长上升子序列的长度



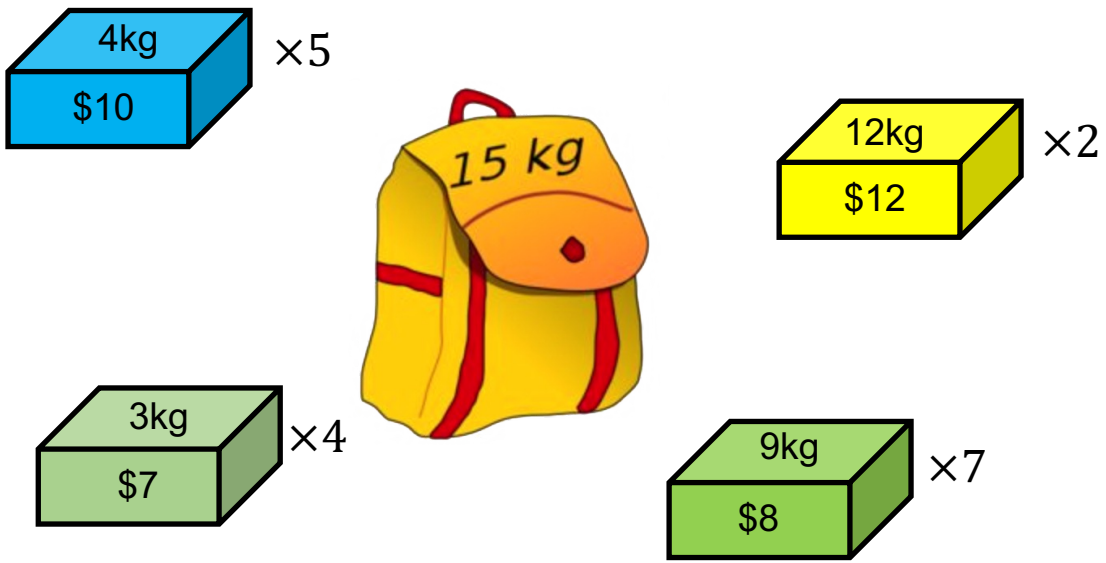


HZOJ-44：最长上升子序列

证明： len 数组的单调性

HZOJ-49：多重背包

【编写程序】给有一个能承重 V 的背包，和 n 种物品，每种物品的数量有限多，我们用重量、价值和数量的三元组来表示一种物品，第 i 种物品表示为(v_i, w_i, s_i)，问在背包不超重的情况下，得到物品的最大价值是多少？



HZOJ-49：多重背包

1、确定动归状态

dp(i,j)代表前 i 件物品，背包承重为 j 时所获得的最大价值

2、确定状态转移方程

$$dp(i,j) = max\{dp[i-1][j-k*v[i]] + k*w[i]\}$$

第 i 种物品选择 k 件，$k \in [0, s_i]$

3、时间复杂度分析

N种物品，第 i 种 k_i 件$\sum k_i = K$，背包的最大承重 V，时间复杂度为 O(KV)

HZOJ-49：多重背包

1、优化拆分方法

假设某种物品有14件，拆分的本质目的是为了转换成0/1背包，可以组合得到这种商品的0件到14件，对比如下两种拆分方法：

(1,1,1,1,1,1,1,1,1,1,1,1,1,1)

(1,2,4,7)

2、时间复杂度分析

一种商品有k_i件，则最多会被拆分成$\log(k_i)+1$份

则时间复杂度为$O(VN + V\sum \log(k_i))$

HZOJ-49：多重背包

1、优化转移过程

假设，某种物品 i 的状态为 (5,6,3)，大家观察下表

dp[i][7]	dp[i-1][7]	dp[i-1][2]+6	-	-
dp[i][12]	dp[i-1][12]	dp[i-1][7]+6	dp[i-1][2]+12	-
dp[i][17]	dp[i-1][17]	dp[i-1][12]+6	dp[i-1][7]+12	dp[i-1][2]+18
dp[i][22]	dp[i-1][22]	dp[i-1][17]+6	dp[i-1][12]+12	dp[i-1][7]+18

HZOJ-49：多重背包

1、优化转移过程

假设，某种物品 i 的状态为 (5,6,3)，大家观察下表

-6	dp[i][7]	dp[i-1][7]	dp[i-1][2]+6	-	-
-12	dp[i][12]	dp[i-1][12]	dp[i-1][7]+6	dp[i-1][2]+12	-
-18	dp[i][17]	dp[i-1][17]	dp[i-1][12]+6	dp[i-1][7]+12	dp[i-1][2]+18
-24	dp[i][22]	dp[i-1][22]	dp[i-1][17]+6	dp[i-1][12]+12	dp[i-1][7]+18

HZOJ-49：多重背包

1、优化转移过程

假设，某种物品 i 的状态为 (5,6,3)，大家观察下表

dp[i][7]	dp[i-1][7]-6	dp[i-1][2]	-	-
dp[i][12]	dp[i-1][12]-12	dp[i-1][7]-6	dp[i-1][2]	-
dp[i][17]	dp[i-1][17]-18	dp[i-1][12]-12	dp[i-1][7]-6	dp[i-1][2]
dp[i][22]	dp[i-1][22]-24	dp[i-1][17]-18	dp[i-1][12]-12	dp[i-1][7]-6

2、单调队列优化

例如：求解 j%5=2 的答案

Step1：将 dp[i-1][2]-0*6 加入到单调队列中，求解 dp[i][2]的值

Step2：将 dp[i-1][7]-1*6 加入到单调队列中，求解 dp[i][7]的值

Step3：将 dp[i-1][12]-2*6 加入到单调队列中，求解 dp[i][12]的值

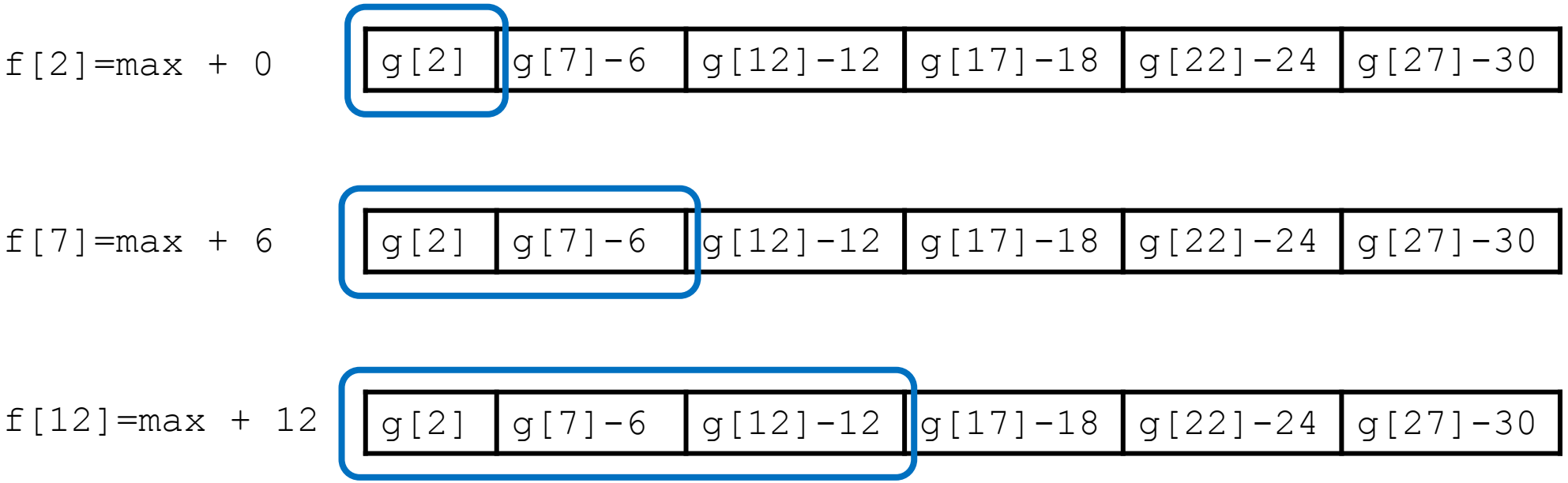
…

HZOJ-49：多重背包

1、优化转移过程

假设，某种物品 i 的状态为 (5,6,3)，单调队列示例如下：

dp[i] = f, dp[i-1]=g

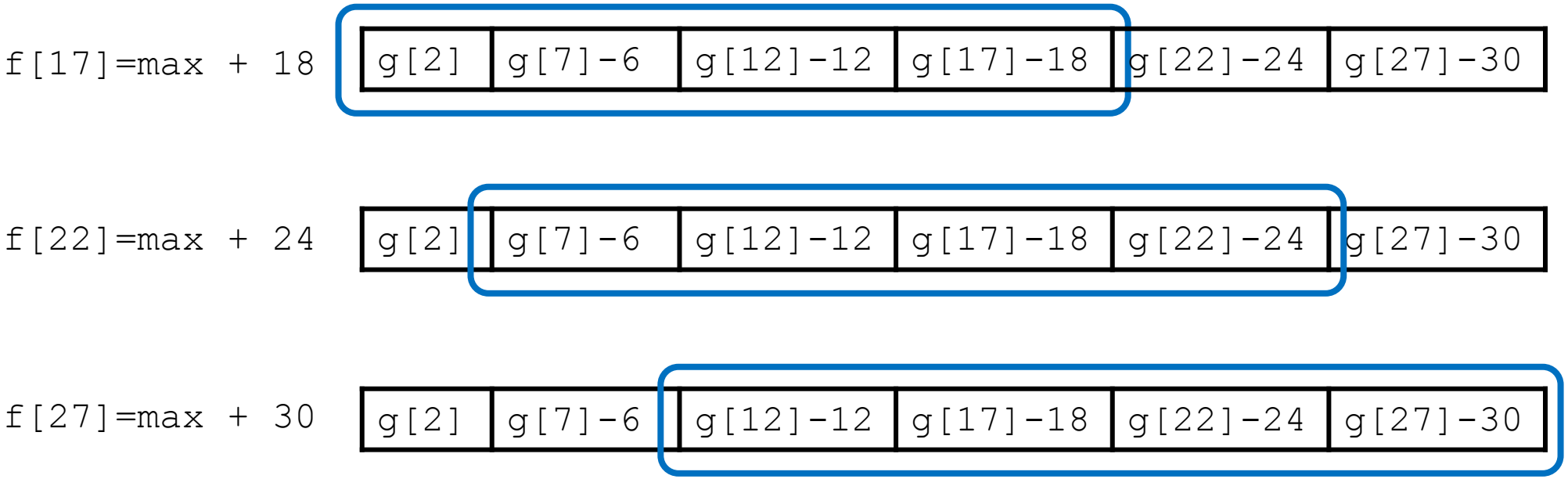


HZOJ-49：多重背包

1、优化转移过程

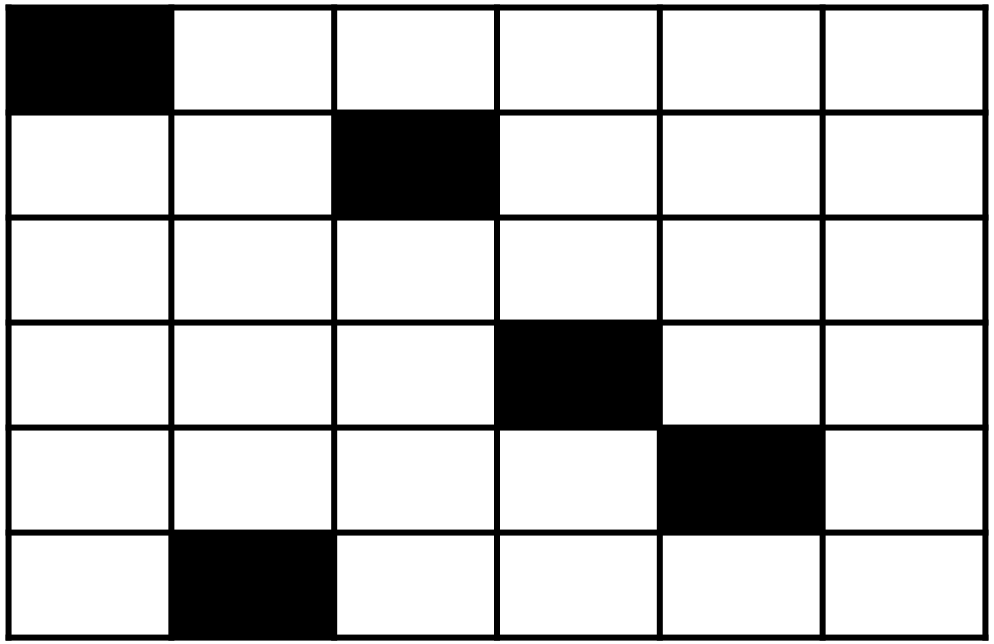
假设，某种物品 i 的状态为 (5,6,3)，单调队列示例如下：

dp[i] = f, dp[i-1]=g



HZOJ-51：矩形

【编写程序】在一个黑白相间的矩形中，有多少个全白色的矩形。



HZOJ-51：矩形

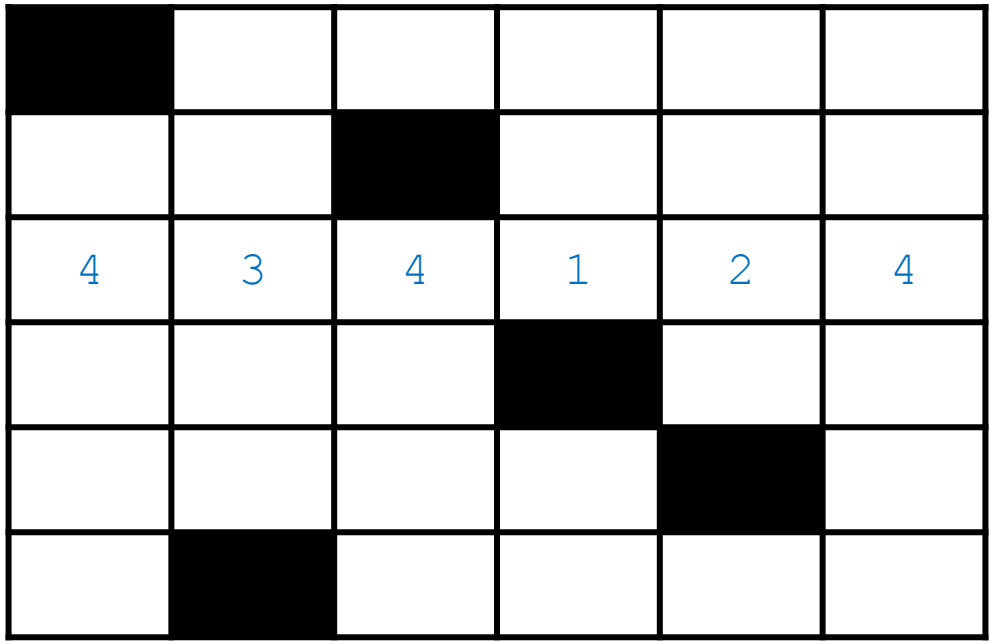
1、确定动归状态

dp(i,j)代表以(i，j)点作为左上角坐标的合法全白矩形的数量

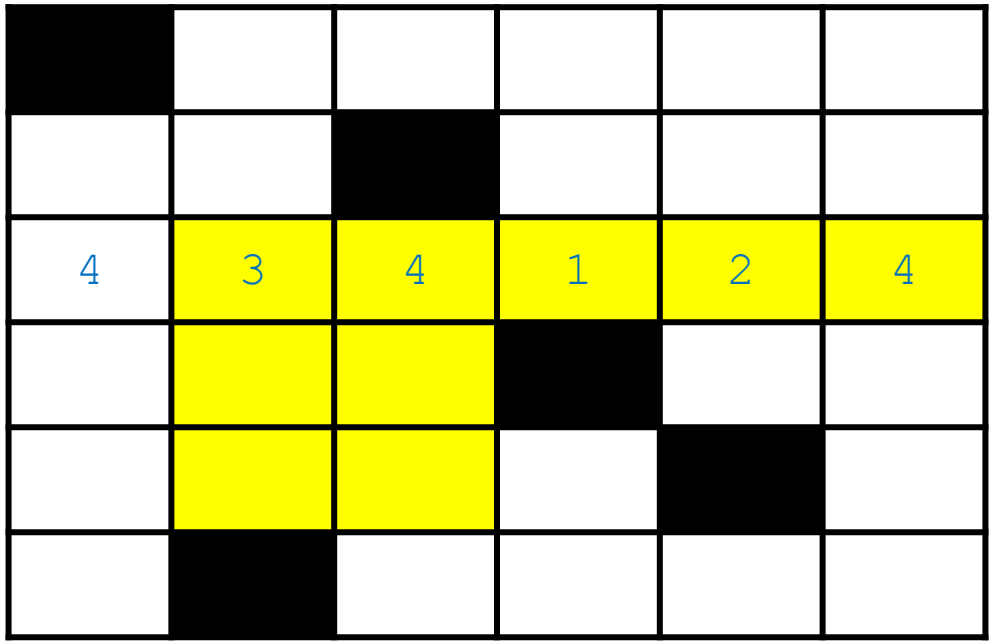
f(i,j)代表(i，j)点向下有多少个连续的白色格子

2、确定状态转移方程

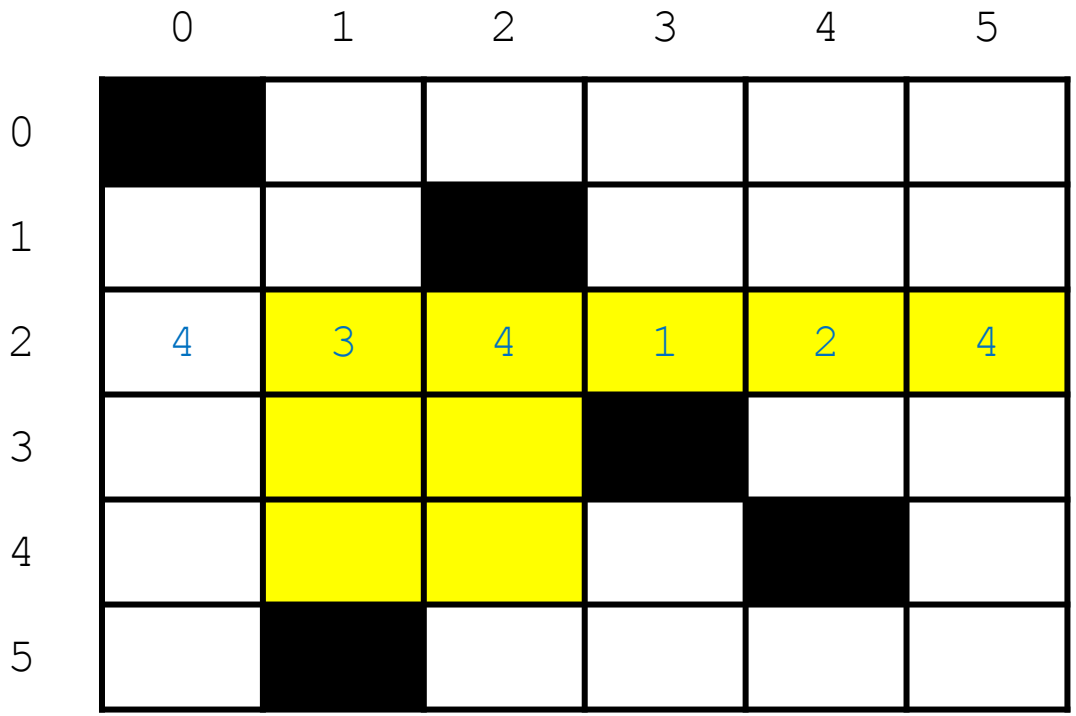
HZOJ-51：矩形



HZOJ-51：矩形



HZOJ-51：矩形




```
dp[2][1] = f[2][1] * (3 - 1) + dp[2][3]
```

HZOJ-51：矩形

对于待求值 dp[i][j]，可以找最小的一个 k，使得 k 满足如下性质：

1. k>j
2. f[i][k]<f[i][j]

则 dp[i][j] = f[i][j]*(k-j) + dp[i][k]

白话解释：

找到 (i,j) 右边最近的一个点(i,k)，使得f(i,k)<f(i,j)

对于这种最近最小关系的维护，当然**单调递增栈**

4-4. 斜率优化

HZOJ-52：古老的打字机

【编写程序】有一台古老的打字机和一篇待打印的文章，文章中的每个字符会有一个消耗值 C_i，打字机工作一次会打印 k 个字符，则打字机的消耗为

$$(\sum_{i=1}^{k} C_i)^2+M$$

其中 M 为打字机启动一次的固定损耗，现在给你 n 个字符的消耗值，问你打字机打印这 n 个字符的最小消耗为多少？

HZOJ-52：古老的打字机

1、确定动归状态

dp(i)代表打印到第 i 个字符的最小消耗

2、确定状态转移方程

$$dp(i) = min(dp[j] + (sum[i] - sum[j])^2 + M) | \, j \in [0, i-1]$$

这里的 sum 数组，代表前缀和

3、时间复杂度分析

从方程上看，时间复杂度为 O(N^2) 的

HZOJ-52：古老的打字机

思考：如果把转移方程中的平方去掉，面对这样的转移方程，怎么优化？

$$dp(i) = min(dp[j] + sum[i] - sum[j] + M)$$

HZOJ-52：古老的打字机

思考：如果把转移方程中的平方去掉，面对这样的转移方程，怎么优化？

$$dp(i) = min(dp[j] - sum[j] + sum[i] + M)$$

查找值　　确定值

优化方案：维护 i 之前最小的 dp[j]-sum[j] 的值即可

HZOJ-52：古老的打字机

$$dp(i) = min(\underbrace{dp[j] + sum[j]^2}_{\text{查找值}} - \underbrace{2sum[i]sum[j]}_{\text{混合值}} + \underbrace{sum[i]^2 + M}_{\text{确定值}})$$

碰到这种存在【混合值】的情况，通过推导最优解存在的性质，从而找到优化方法。

HZOJ-52：古老的打字机

假设 i 点通过 j 和 k 转移，那么 j 优于 k 的情况是(k<j)，当且仅当下式成立：

①： $dp[j] + sum[j]^2 - 2sum[i]sum[j] + sum[i]^2 + M$

②： $dp[k] + sum[k]^2 - 2sum[i]sum[\mathrm{k}] + sum[i]^2 + M$

式① < 式②

$$dp[j] + sum[j]^2 - 2sum[i]sum[j] < dp[k] + sum[k]^2 - 2sum[i]sum[\mathrm{k}]$$

请将确定值(i项)与查找值(j，k项)分别置于不等式两侧

HZOJ-52：古老的打字机

假设 i 点通过 j 和 k 转移，那么 j 优于 k 的情况是(k<j)，当且仅当下式成立：

①： $dp[j]+sum[j]^2-2sum[i]sum[j]+sum[i]^2+M$

②： $dp[k]+sum[k]^2-2sum[i]sum[\mathrm{k}]+sum[i]^2+M$

式① < 式②

$$dp[j]+sum[j]^2-2sum[i]sum[j]<dp[k]+sum[k]^2-2sum[i]sum[\mathrm{k}]$$

$$dp[j]+sum[j]^2-dp[k]-sum[k]^2<2sum[i](sum[j]-sum[k])$$

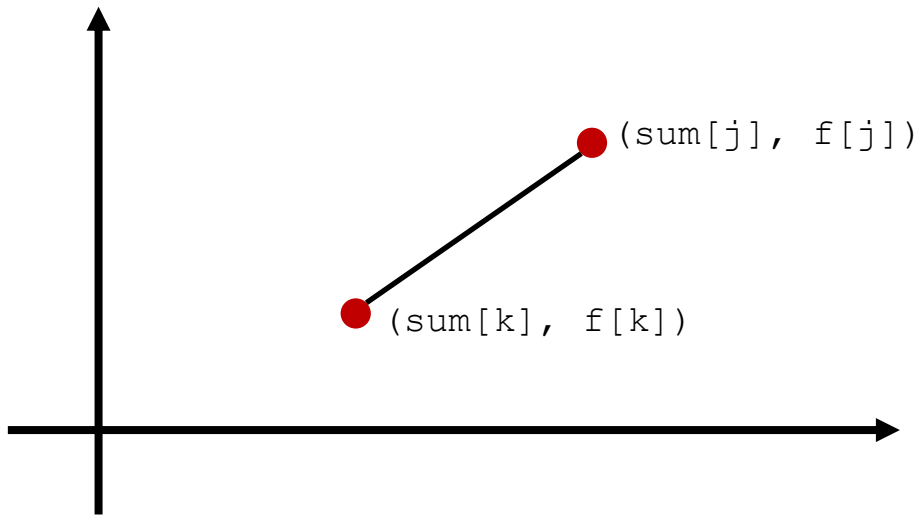
$$\frac{dp[j]+sum[j]^2-dp[k]-sum[k]^2}{sum[j]-sum[k]}<2sum[i]$$ *很重要

HZOJ-52：古老的打字机

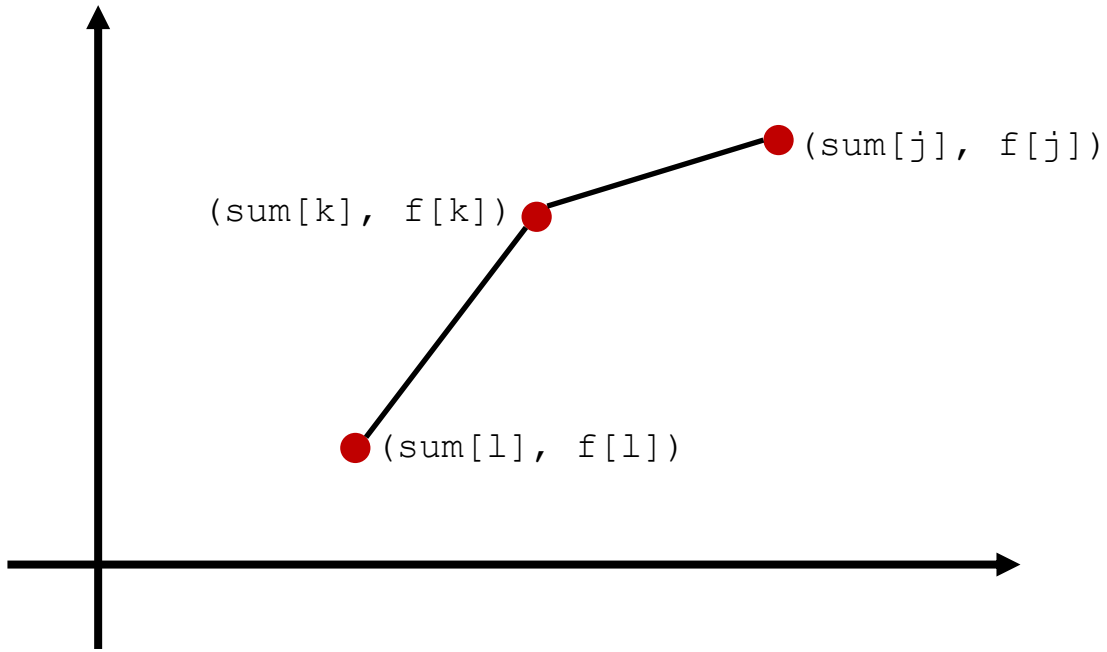
设置 f 函数，令 $f[x] = dp[x] + sum[x]^2$ ，则

$$\frac{f[j] - f[k]}{sum[j] - sum[k]} < 2sum[i]$$

根据题目性质，sum 数组单调增，上式可以看成是两点之间的斜率

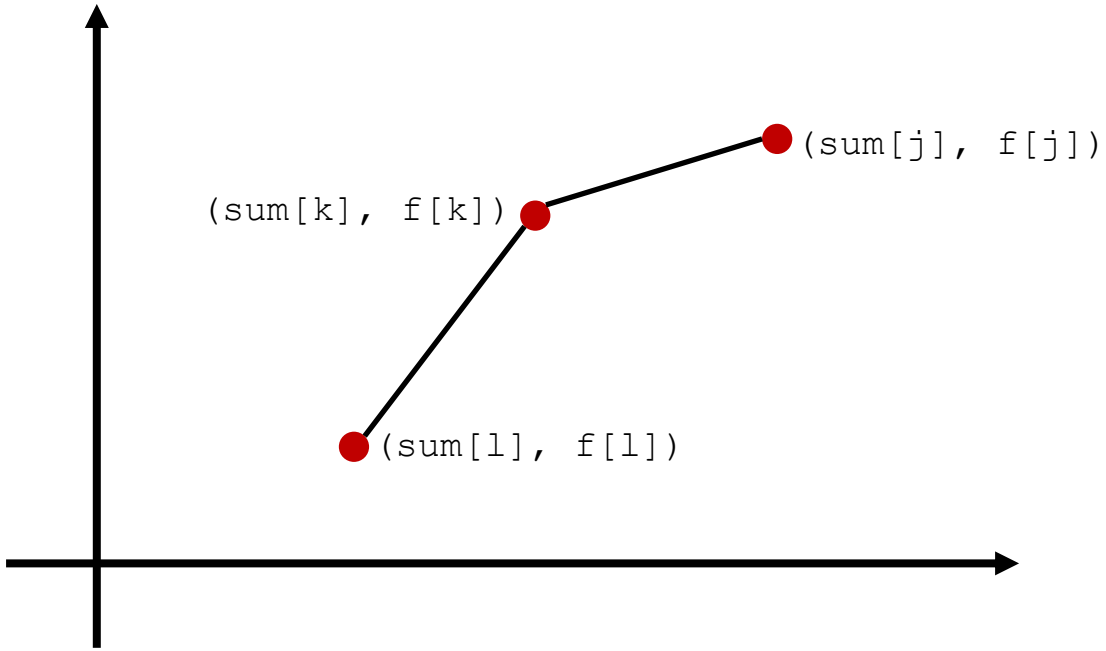


HZOJ-52：古老的打字机



思考：在如图的这种关系下，看看能推导出什么结论?

HZOJ-52：古老的打字机



$$\frac{f[j]-f[k]}{sum[j]-sum[k]} < \frac{f[k]-f[l]}{sum[k]-sum[l]}$$

HZOJ-52：古老的打字机

①：$\frac{f[j]-f[k]}{sum[j]-sum[k]} < \frac{f[k]-f[l]}{sum[k]-sum[l]} < 2sum[i]$

②：$\frac{f[j]-f[k]}{sum[j]-sum[k]} < 2sum[i] < \frac{f[k]-f[l]}{sum[k]-sum[l]}$

③：$2sum[i] < \frac{f[j]-f[k]}{sum[j]-sum[k]} < \frac{f[k]-f[l]}{sum[k]-sum[l]}$

思考：这三个式子分别说明了什么？

HZOJ-52：古老的打字机

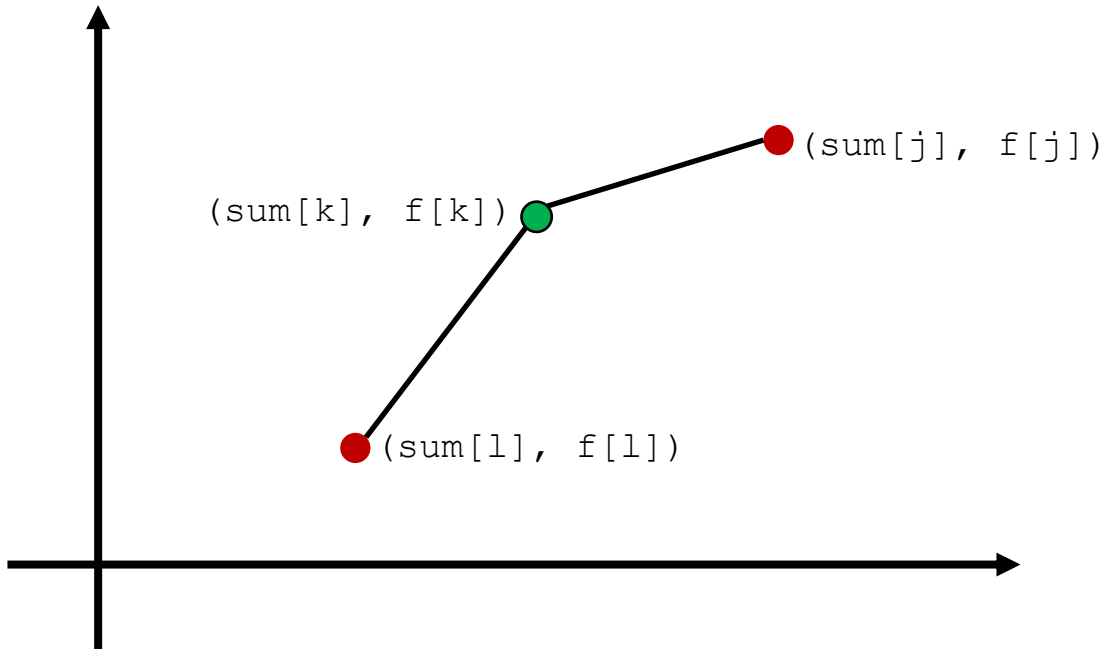
①：$\frac{f[j]-f[k]}{sum[j]-sum[k]} < \frac{f[k]-f[l]}{sum[k]-sum[l]} < 2sum[i]$

②：$\frac{f[j]-f[k]}{sum[j]-sum[k]} < 2sum[i] < \frac{f[k]-f[l]}{sum[k]-sum[l]}$

③：$2sum[i] < \frac{f[j]-f[k]}{sum[j]-sum[k]} < \frac{f[k]-f[l]}{sum[k]-sum[l]}$

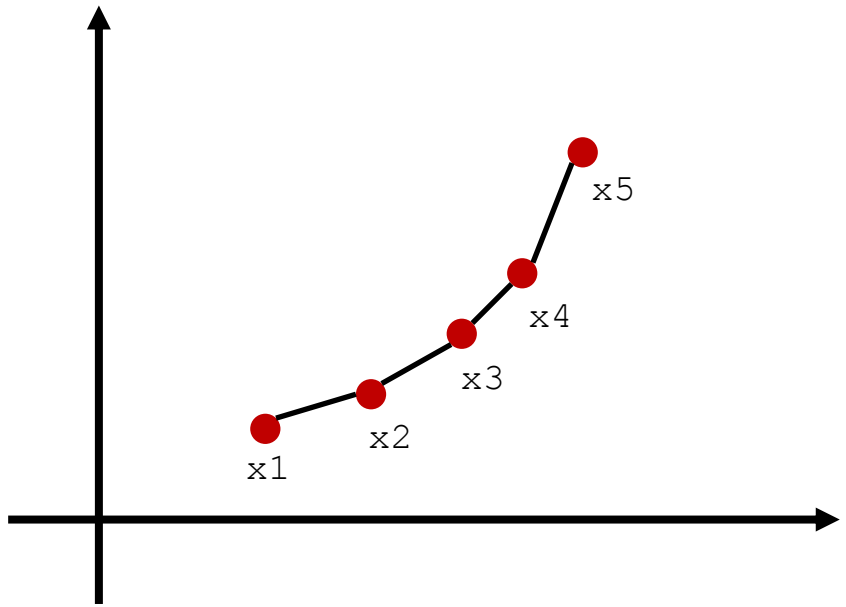
	j	k	l
①	✅	❌	❌
②	✅	❌	✅
③	❌	❌	✅

HZOJ-52：古老的打字机



结论： 呈现此种形状时，k 点肯定不是后续任何状态的备选答案

HZOJ-52：古老的打字机



结论：备选答案的集合应该长成如图所示的样子

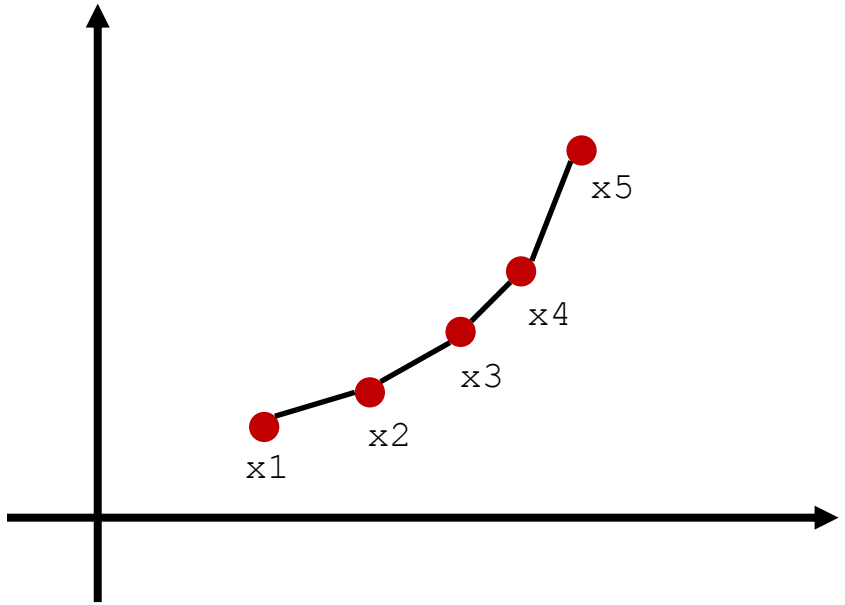
HZOJ-52：古老的打字机

设置 g 函数，令 $g[x,y]=\frac{f[y]-f[x]}{sum[y]-sum[x]}$，则

$g[x1,x2]<g[x2,x3]<g[x3,x4]<g[x4,x5]$

考虑 $2sum[i]$ 的位置：

$g[x1,x2]<g[x2,x3]<2sum[i]<g[x3,x4]<g[x4,x5]$



HZOJ-52：古老的打字机

1、确定动归状态

dp(i)代表打印到第 i 个字符的最小消耗

2、状态转移优化

一、维护一个备选答案集合，每次二分一个位置，以此求得 dp[i] 的结果

二、由于 sum[i] 单调，所以可以用单调队列进行维护

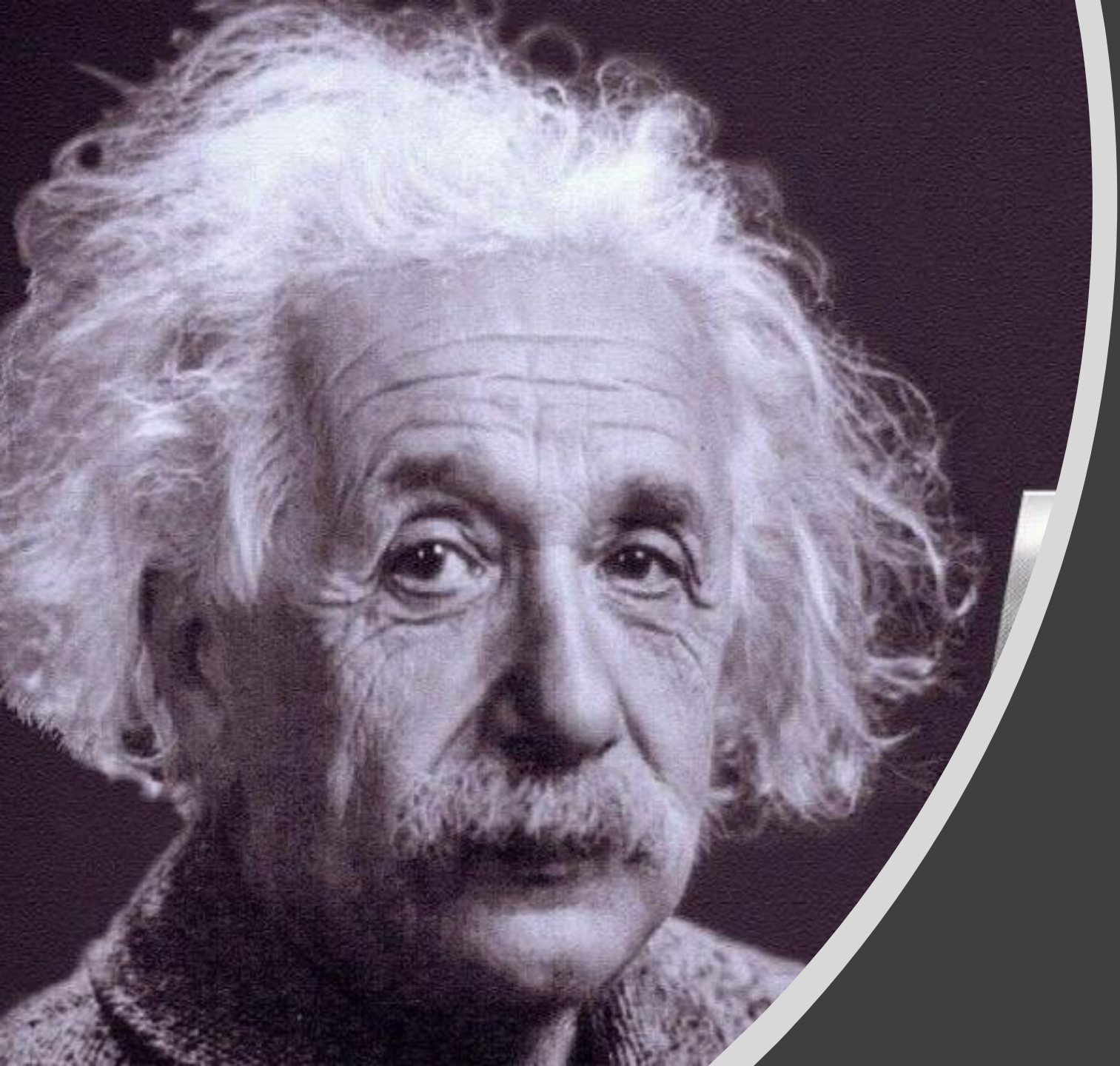
3、时间复杂度分析

一、用二分优化，O(nlogn)

二、用单调队列优化，O(n)

五、动态规划-课后实战题

1. 低价购买-P1108
2. 杂务-P1113
3. 最大子段和-P1115
4. NASA 的食物计划-P1507
5. 魔族密码-P1481
6. 找啊找啊找 GF-P1509
7. 书本整理-P1103
8. 删数-P2426
9. 垃圾陷阱-P1156
10. 最大正方形 II -P1681
11. 导弹拦截-P1158
12. 三元上升子序列-P1637
13. 多人背包-P1858
14. 仓库建设-P2120



为什么
会出一样的题目？

《船说：算法与数据结构》
第14章-动态规划