动态规划问题

PPt report template of Hunan university

第七次第十四周小班课个人资料

湖南大学
HUNAN UNIVERSITY

概述

概念　动态规划

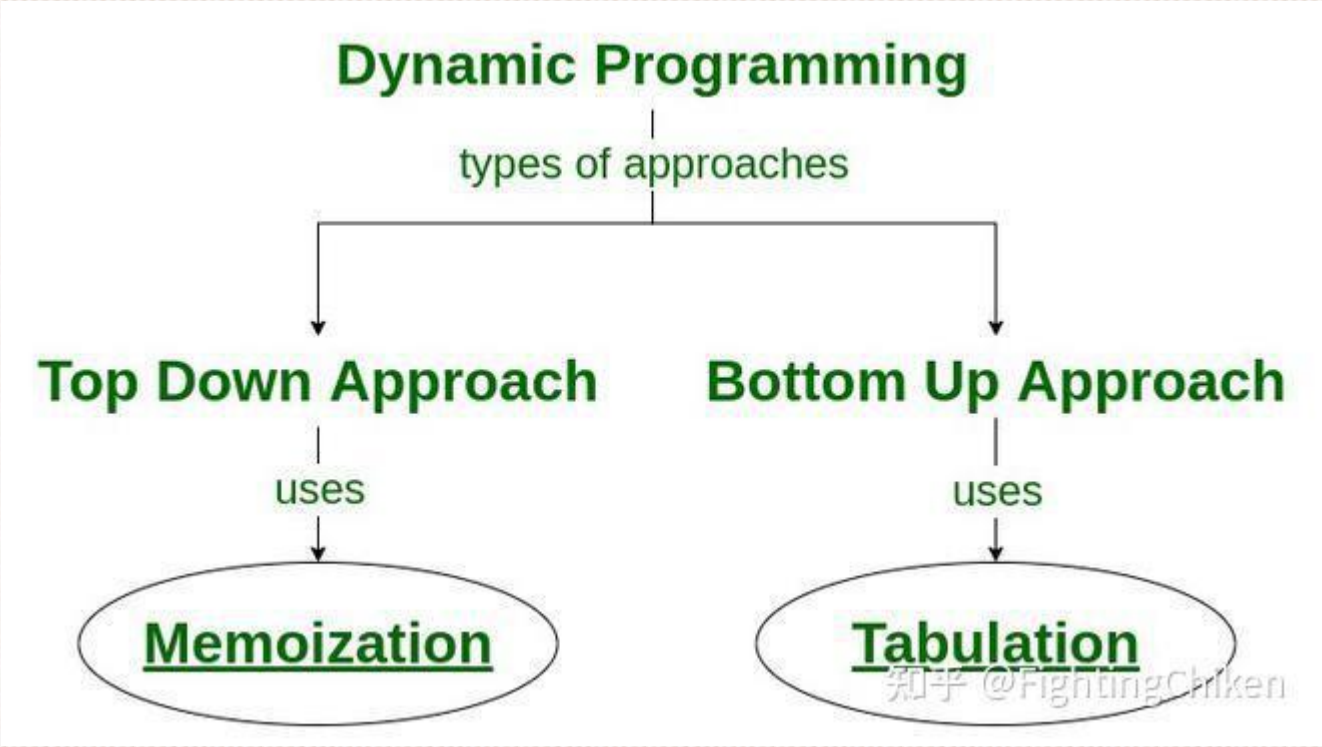
动态规划（Dynamic Programming,DP）算法通常用于求解某种具有最优性质的问题。在这类问题中，可能会有许多可行解，每一个解都对应一个值，我们希望找到具有最优值的解。动态规划算法与分治法类似，其基本思想也是将待求解的问题分解成若干个子问题，先求解子问题，然后从这些子问题的解中 得到原有问题的解。与分治法不同的是，动态规划经分解后得到的子问题往往 不是相互独立 的。

（分治算法也可以解决分解后得到的子问题不是相互独立的情况，只是要对公共子问题进行单独求解。这样会使分治法求解问题的复杂度大大提高。）

概念

动态规划

动态规划是一种算法设计范式，它为广泛的问题提供了有效和优雅的解决方案。其基本思想是递归地将一个复杂的问题划分为许多更简单的子问题，存储这些子问题的每个子问题的解，并最终将存储的答案用于解决原始问题。通过缓存子问题的解，动态规划有时可以避免指数级的浪费。



概念 动态规划

动态规划可以应用于表现出两个性质的优化问题:

最优子结构：一个问题的最优解可以从更小的子问题的最优解中计算出来。

重叠子问题:递归地将问题分解成子问题、子问题的子问题、子问题的子问题的子问题等等的过程，会导致一些子问题被重复。

动态规划有两种实现策略:自顶向下(记忆 Memorization)和自底向上(制表tabulation)。

记忆(Memorization):将动态规划实现为递归过程。为了解决子问题，我们简单地在子问题上调用递归过程

自下而上的动态规划:(Bottom-Up dynamic programming):通过识别给定问题的所有子问题以及子问题之间的依赖关系来实现动态规划。

回溯(Backtracing)。分两步解决优化问题往往很方便。第一步使用动态规划计算解的最优值。第二步，回溯，通过使用自顶向下的过程计算最优解。

目录

01

应用实例

由在此输入详细介绍，以表达项目工作的详细资料和文字信息。

02

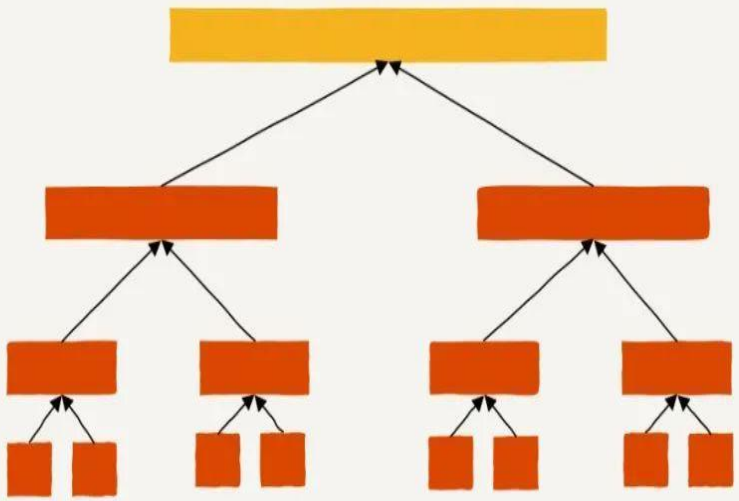
应用举例

由在此输入详细介绍，以表达项目工作的详细资料和文字信息。

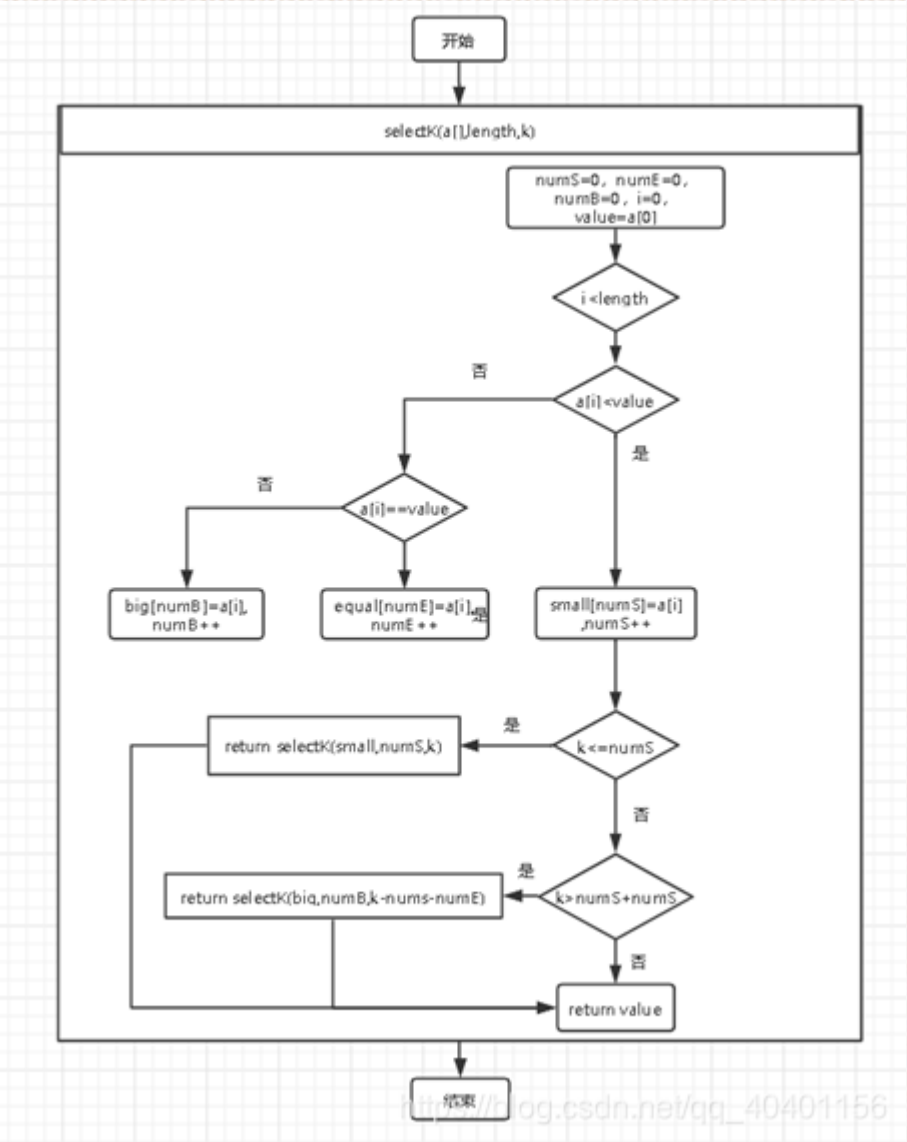
湖南大学
HUNAN UNIVERSITY

第一部分

应用实例



动态规划的应用场景非常广泛，包括但不限于计算机科学、数学、物理学、经济学等领域。在计算机科学中，动态规划被广泛应用于字符串匹配、背包问题、最短路径问题等经典问题。在数学中，动态规划可以用于求解微分方程、积分方程等复杂问题。在物理学中，动态规划可以用于模拟物理过程和优化实验设计。在经济学中，动态规划可以用于求解最优控制问题和博弈论中的纳什均衡。

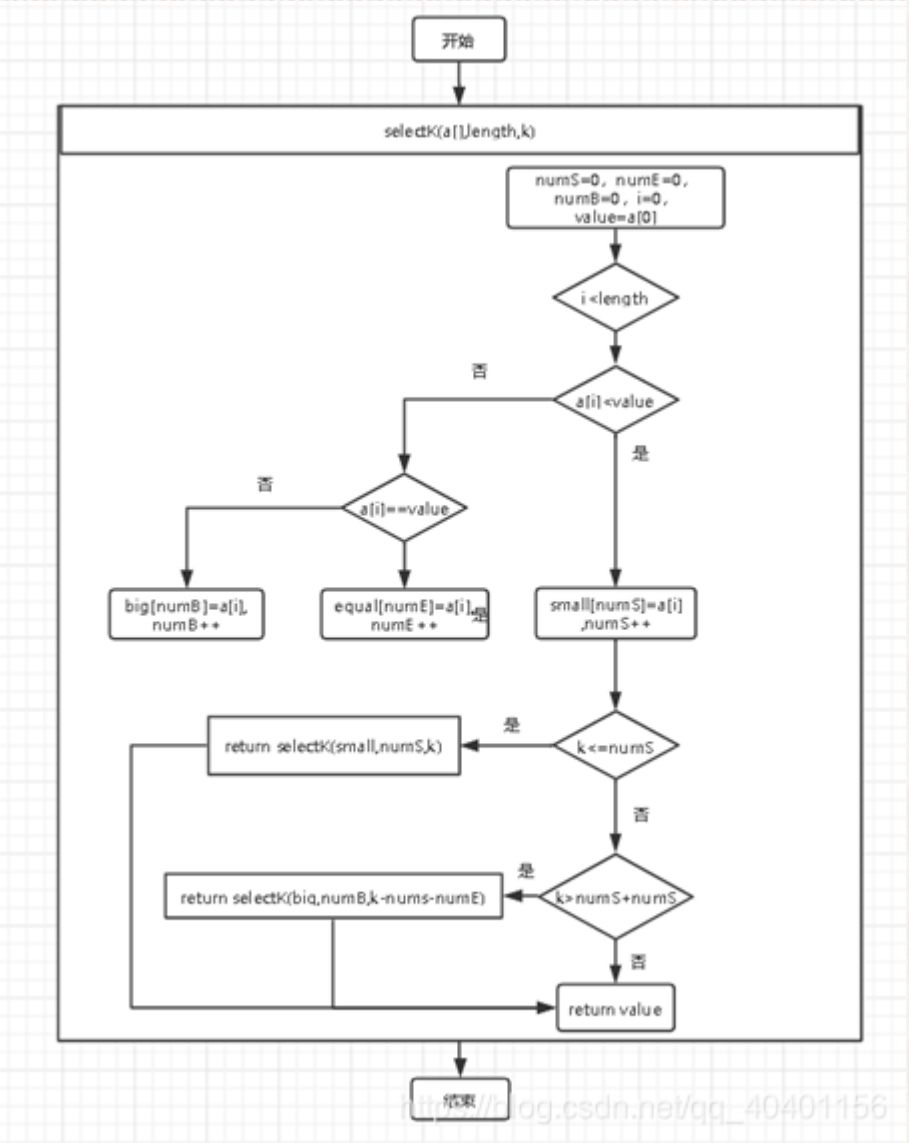


假设我们有一个长度为n的数组arr和一个目标值target。我们的任务是找出arr中是否存在一个连续子数组（长度至少为k），使得该子数组的和等于target。我们可以使用动态规划来解决这个问题。

首先，我们定义一个二维数组dp，其中dp[i][j]表示是否存在一个长度为j的子数组，使得该子数组的和等于i。然后，我们初始化dp数组为false。

接下来，我们遍历arr数组，对于每个元素arr[i]，我们将其加入到已经存在的所有长度为j的子数组中，并更新dp数组。如果加入arr[i]后，子数组的和等于i，则dp[i][j]为true。

最后，我们遍历dp数组最后一行，如果存在任何一个元素为true，则说明存在一个长度至少为k的子数组的和等于target，返回true；否则返回false。



在实际应用中，我们需要根据具体的问题来选择合适的动态规划算法。有时候我们需要对问题进行适当的预处理和数据结构选择，以便更好地利用动态规划的特性。此外，对于一些大规模问题，我们还需要考虑算法的时空复杂度，以便在实际应用中获得更好的性能。

爬楼梯问题

假设你现在正在爬楼梯，一共需要经过 n 阶楼梯你才可以到达楼顶。每次你可以爬 楼梯的 1 或 2 个台阶。请问一共有多少种不同的方法可以爬到楼顶？

现在，让我们按照动态规划算法的求解步骤我们来分析一下这个问题：

步骤 1： 刻画爬楼梯问题一个最优解的结构特征

情况 1：输入 n=1；输出为 1
解释 1：有一种情况可以爬上楼顶， 爬 1 步，记为 1
情况 2：输入 n=2；输出为 2
解释 2：有两种情况可以爬上楼顶，分别为连续两次爬一阶楼梯和一次爬两阶楼梯，记为 1+1=2
情况 3：输入 n=3；输出为 3
解释 3：有三种情况可以爬上楼顶，如情况 1 和 2 描述一样，记为 1+1+1,2+1,1+2

通过分析可以知道，爬楼梯问题主要在于我们可以一次爬两步或者一步，所以到达最后一阶楼梯 n 时，我们可以从第 n-2 阶楼梯爬两步或者第 n-1 楼梯爬一步完成。

爬楼梯问题

假设你现在正在爬楼梯，一共需要经过 n 阶楼梯你才可以到达楼顶。每次你可以爬 楼梯的 1 或 2 个台阶。请问一共有多少种不同的方法可以爬到楼顶？

现在，让我们按照动态规划算法的求解步骤我们来分析一下这个问题：

步骤 2： 递归的定义爬 n 阶楼梯最多的方法数

上 1 阶台阶：有 1 钟方法；
上 2 阶台阶：有 1+1 和 2 两种方法；
上 3 阶台阶：到达第 3 阶的方法总数是到达第 1 阶和第 2 阶方法的总和；
上 n 阶台阶：到达第 n 阶的方法总数就是到第 (n-1) 阶和第 (n-2) 阶的方法数之和。

综上所述，我们可以知道爬 n 阶楼梯的状态转移方程可以定义为：goStep (n) = goStep (n-1)+goStep (n-2)。动态规划算法最重要的就是去定义这个状态转移方程，通过这个状态转移方程我们就可以很清楚的去计算。

爬楼梯问题

假设你现在正在爬楼梯，一共需要经过 n 阶楼梯你才可以到达楼顶。每次你可以爬 楼梯的 1 或 2 个台阶。请问一共有多少种不同的方法可以爬到楼顶？

现在，让我们按照动态规划算法的求解步骤我们来分析一下这个问题：

步骤 3： 计算爬 n 阶楼梯最多方法数的值

楼梯阶数 n	爬 n 阶楼梯最多的方法数
1	1
2	2
3	goStep(1)+goStep(2)=1+2=3
4	goStep(2)+goStep(3)=2+3=5
5	goStep(3)+goStep(4)=3+5=8
6	goStep(4)+goStep(5)=5+8=13
7	goStep(5)+goStep(6)=8+13=21
8	goStep(6)+goStep(7)=13+21=36
9	goStep(7)+goStep(8)=21+36=57

爬楼梯问题

假设你现在正在爬楼梯，一共需要经过 n 阶楼梯你才可以到达楼顶。每次你可以爬 楼梯的 1 或 2 个台阶。请问一共有多少种不同的方法可以爬到楼顶？

现在，让我们按照动态规划算法的求解步骤我们来分析一下这个问题：

步骤 4： 利用计算出的信息构爬 n 阶楼梯每次走几步的方法

其实在爬楼梯这个问题中，我们并不需要统计每次的具体爬楼梯方法，如果需要统计每次具体走法时，需要在计算的时候记录之前的每一步走法，把信息全部记录保留下来即可。

我们可以很明显的发现，动态规划算法很多时候都是应用于求解一些最优化问题（最大，最小，最多，最少）

湖南大学
HUNAN UNIVERSITY

第三部分

应用举例

数字三角形问题

问题描述：

小渊和小轩是好朋友也是同班同学，他们在一起总有谈不完的话题。
一次素质拓展活动中，班上同学安排坐成一个 m行 n
列的矩阵，而小渊和小轩被安排在矩阵对角线的两端，因此，他们就无法直接交谈了。
幸运的是，他们可以通过传纸条来进行交流。
纸条要经由许多同学传到对方手里，小渊坐在矩阵的左上角，坐标 (1,1)
，小轩坐在矩阵的右下角，坐标 (m,n)
从小渊传到小轩的纸条只可以向下或者向右传递，从小轩传给小渊的纸条只可以向上或者向左传递。
在活动进行中，小渊希望给小轩传递一张纸条，同时希望小轩给他回复。
班里每个同学都可以帮他们传递，但只会帮他们一次，也就是说如果此人在小渊递给小轩纸条的时候帮忙，
那么在小轩递给小渊的时候就不会再帮忙，反之亦然。
还有一件事情需要注意，全班每个同学愿意帮忙的好感度有高有低
（注意：小渊和小轩的好心程度没有定义，输入时用 0表示），可以用一个 0~100
的自然数来表示，数越大表示越好心。
小渊和小轩希望尽可能找好心程度高的同学来帮忙传纸条
，即找到来回两条传递路径，使得这两条路径上同学的好心程度之和最大。
现在，请帮助小渊和小轩找到这样的两条路径。

数字三角形问题

问题描述：

输入格式
第一行有 2
个用空格隔开的整数 m
和 n，表示学生矩阵有 m行 n 列。
接下来的 m行是一个 m×n的矩阵，矩阵中第 i行 j列的整数表示坐在第 i行 j列的学生的好心程度，
每行的 n 个整数之间用空格隔开。
输出格式
输出一个整数，表示来回两条路上参与传递纸条的
学生的好心程度之和的最大值。

数据范围
1≤n,m≤50
输入样例：
3 3
0 3 9
2 8 5
5 7 0
输出样例：
34

数字三角形类问题

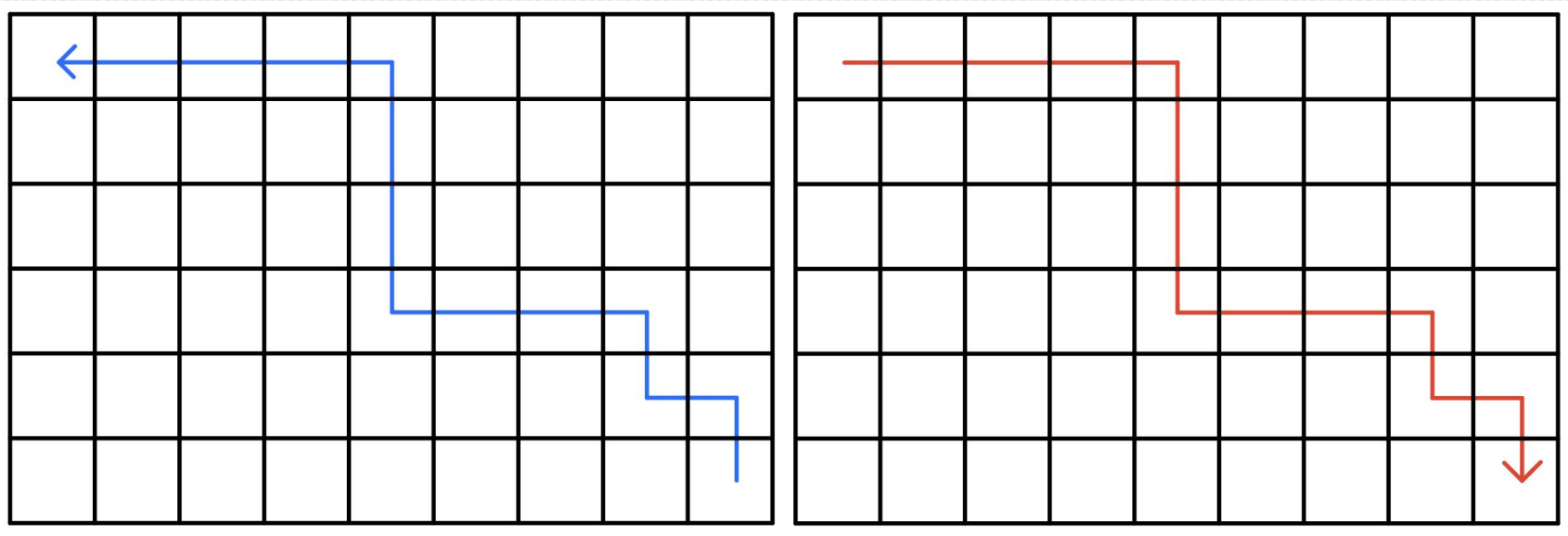
问题分析：

算法(线性DP) O(n3)
首先考虑路径有交集该如何处理。可以发现交集中的格子一定在每条路径的相同步数处。因此可以让两个人同时从起点出发，每次同时走一步，这样路径中相交的格子一定在同一步内。
状态表示：f[k, i, j] 表示两个人同时走了k步，第一个人在 (i, k - i) 处，第二个人在 (j, k - j)处的所有走法的最大分值。状态计算：按照最后一步两个人的走法分成四种情况：两个人同时向右走，最大分值是 f[k - 1, i, j] + score(k, i, j)；第一个人向右走，第二个人向下走，最大分值是 f[k - 1, i, j - 1] + score(k, i, j)；第一个人向下走，第二个人向右走，最大分值是 f[k - 1, i - 1, j] + score(k, i, j)；两个人同时向下走，最大分值是 f[k - 1, i - 1, j - 1] + score(k, i, j)
;
注意两个人不能走到相同格子，即i和j不能相等。

数字三角形问题

问题分析：

这题明显不是一个裸题，我们需要一层一层的拆解分析首先想到的是能不能往 方格取数 模型上靠
对于一个从 (n,m)(n,m) 出发到 (1,1)(1,1) 的路线，且只能向上或向右走，考虑将其方向调转，则必定对应一条从 (1,1)(1,1) 出发到 (n,m)(n,m) 的路线，且只能向下或向右走

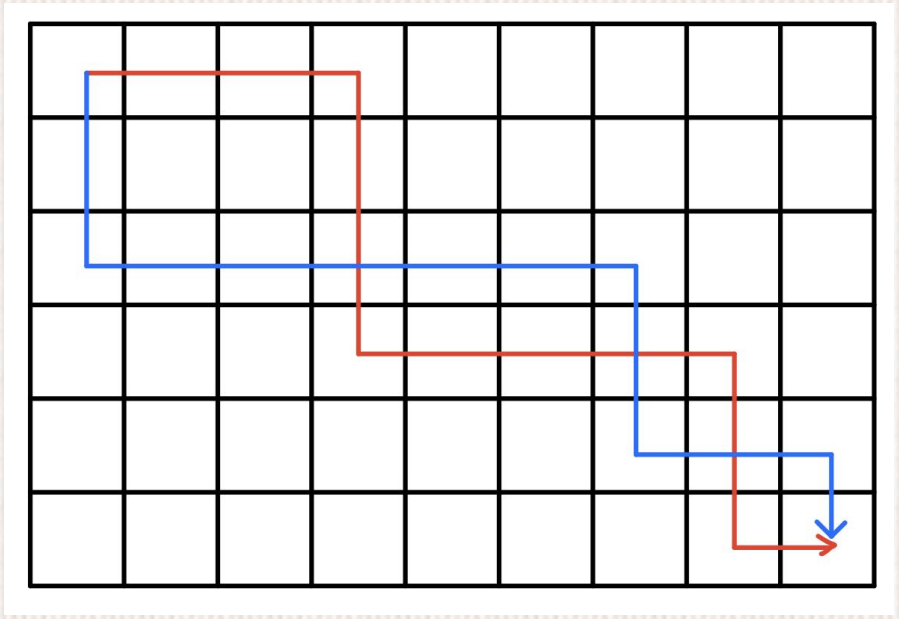


这两种走法的方案都是一一对应的（即任意一条路线都可以找到其对应的反向路线），因此该方案映射合法于是该问题就变成了寻找一条从 (1,1)(1,1) 出发到达 (n,m)(n,m)，每次只能向下或向右走，先后出发两次，且两次路线不能经过重复格子的最大价值方案

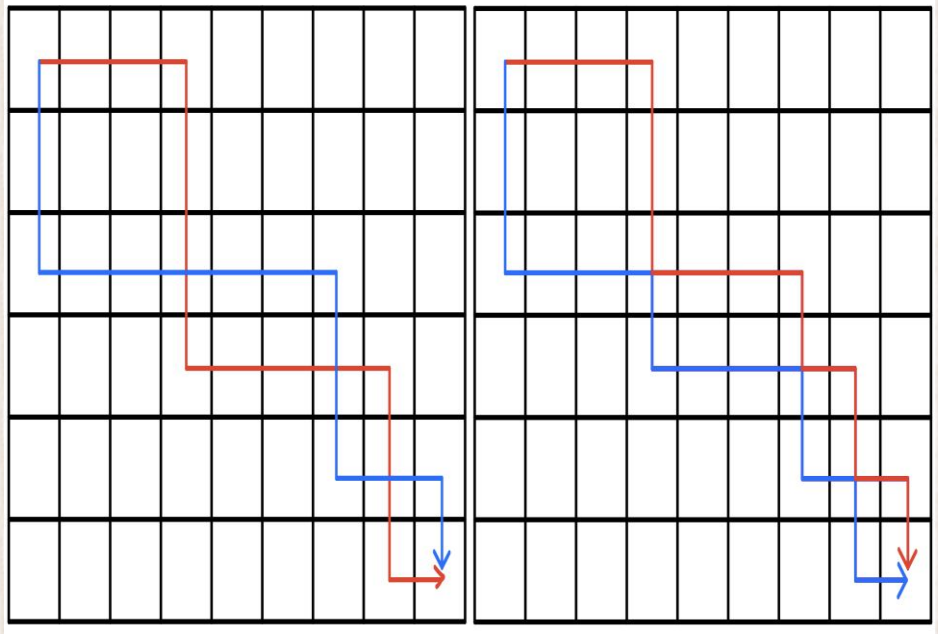
数字三角形问题

问题分析：

情况一：最优解的两条路线是相互交叉经过的



则我们可以对交叉出来的部分进行路线交换，如下图的操作

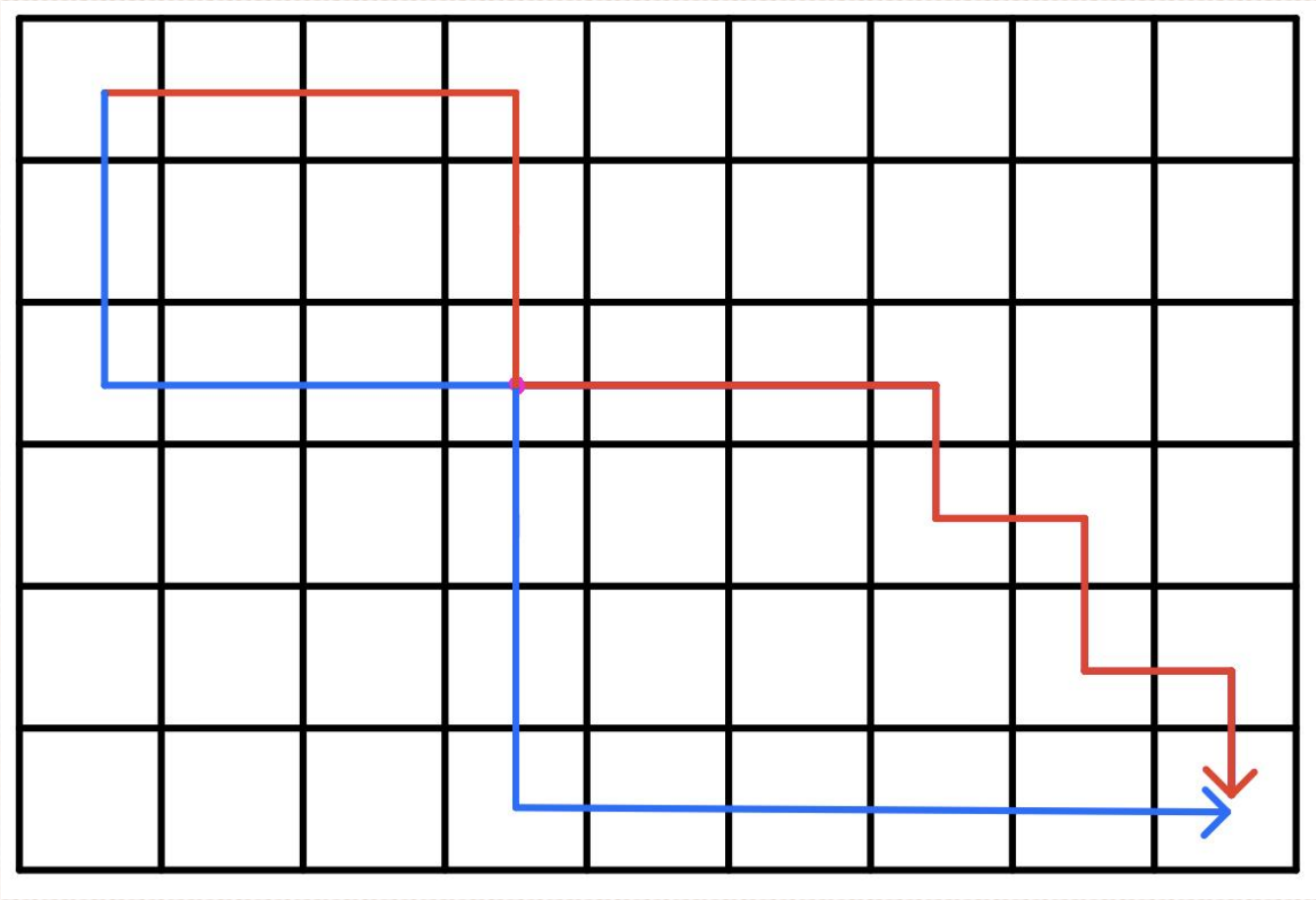


于是，我们可以发现，所有的交叉路线都会映射成一种一条路线只在下方走，一条路线只在上方走的不交叉路线因此我们只需集中解决情况二即可

数字三角形问题

问题分析：

情况二：最优解的两条路线不交叉，但在某些点有重合

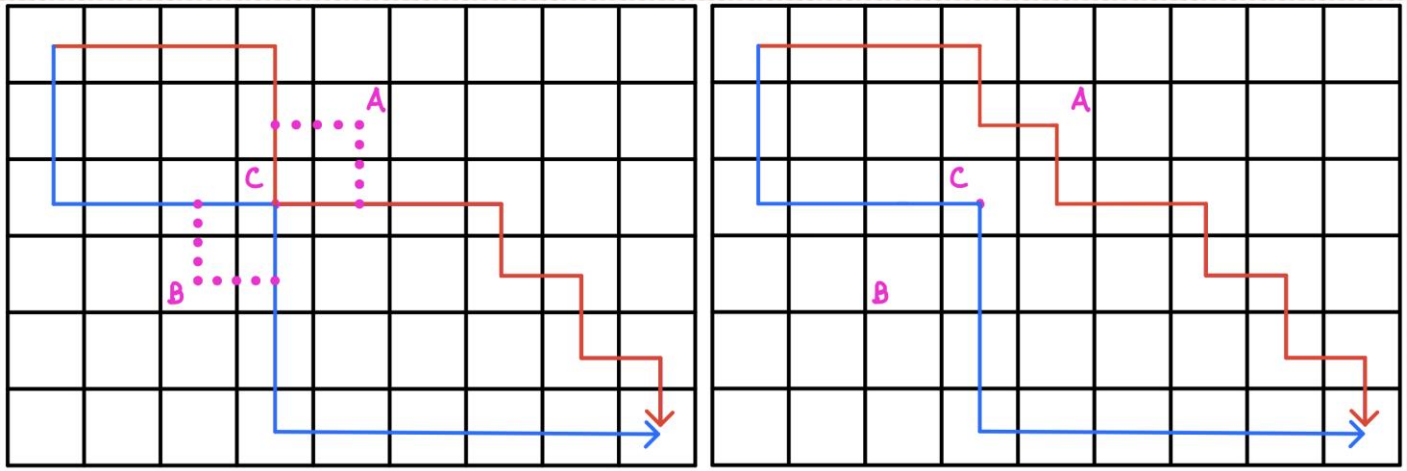


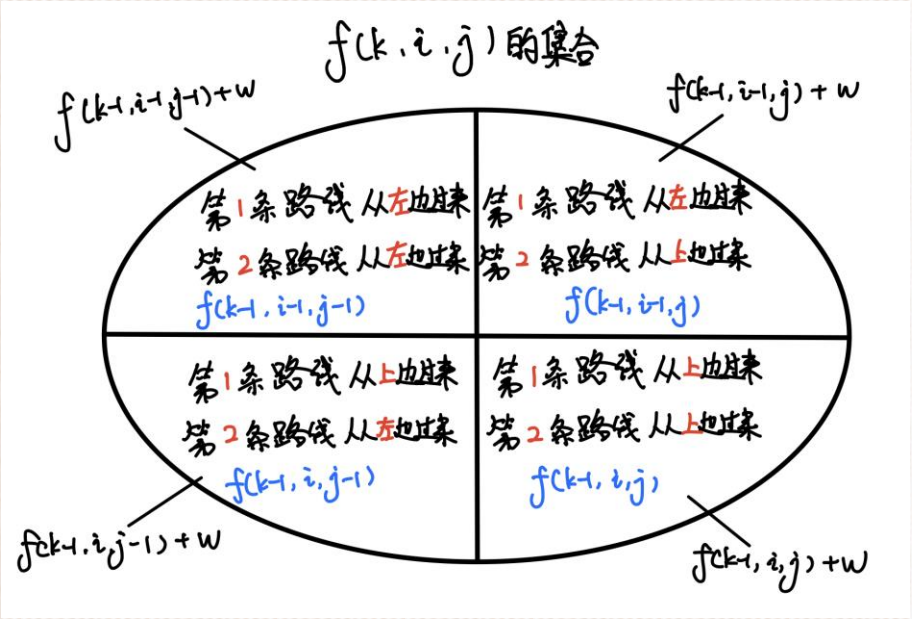
由于方格取数，对于走到相同格子时，只会累加一次格子的价值于是我们可以使用贪心中的微调法来进行这部分的证明对于重合的格子，我们必然可以在两条路线中找到额外的一条或两条路线，使得新的路线不发生重合

数字三角形问题

问题分析：

具体参照下图：





由于原路线是最优解，则必然 wA=wB=0，否则最优解路线必然是经过 A 或 B 的因此，我们可以通过微调其中的一条路线，使之不经过重合点 C，同时路线的总价值没有减少

得证：最优解路线可以是不经过重复路线的

数字三角形问题

问题代码：

```
int n,m;
scanf("%d %d",&n,&m);
for (int i=1;i<=n;i++)
    for (int j=1;j<=m;j++)
    {
        scanf("%d",&w[i][j]);
    }
int res=0;
```

数据读入与初始化

数字三角形问题

```
int dp(int k, int i, int j)
{
    if (f[k][i][j] >= 0) return f[k][i][j];

    if (k == 2 && i == 1 && j == 1) return f[k][i][j] = w[1][1];

    if (i <= 0 || i >= k || j <= 0 || j >= k) return -INF;

    int v = w[i][k - i];
    if (i != j) v += w[j][k - j];

    int t = 0;
    t = max(t, dp(k - 1, i, j));
    t = max(t, dp(k - 1, i - 1, j));
    t = max(t, dp(k - 1, i, j - 1));
    t = max(t, dp(k - 1, i - 1, j - 1));
    return f[k][i][j] = t + v;
}
```

dp部分代码（记忆化搜索版本）

数字三角形问题

```
for (int k = 2; k <= n + m; ++ k)
{
    for (int i = 1; i < k && i <= n; ++ i)
    {
        for (int j = 1; j < k && j <= n; ++ j)
        {
            int v = w[i][k - i];
            if (i != j) v += w[j][k - j];

            int &t = f[k][i][j];
            t = max(t, f[k - 1][i][j]);
            t = max(t, f[k - 1][i - 1][j]);
            t = max(t, f[k - 1][i][j - 1]);
            t = max(t, f[k - 1][i - 1][j - 1]);
            t += v;
        }
    }
}
```

dp部分代码（三重迭代版本）

谢 谢 观 看

thank you for watching

参考文献

[1] Xperia2著 .273传纸条及其题解

[2]打破砂锅问到底007著.Csdn .【算法】动态规划及其应用场景