

依澜的学习任务【1.29】

任务简介：

28 号的“这也许会是最后一次学习任务啦”

这个“也许”也是思考了一个早上的结果

算法：

今天开始学动态规划（DP）

动态规划是啥？一种把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解的思维

如何理解动态规划？请参考知乎第一篇回答：

<https://www.zhihu.com/question/23995189>

主要是人家讲的太好了

动态规划题目分为几种

比较常见的有线性动规、图形动规和背包问题

那我们的计划是今天学习线性动规和图形动规

明天学习背包问题

理解完动态规划之后呢

强调一下做题的关键性步骤：

1.寻找各阶段的关系（转移方程）

2.将大问题拆分成小问题

3.先解决小问题，然后解决大问题

线性动规和图形动规，可以当成一维和二维动规

分别从解例题来理解一下哈

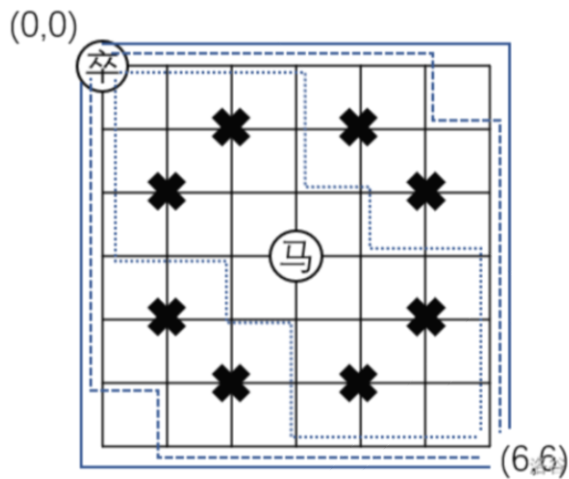
图形动规：洛谷题目 1002

题目描述

[展开](#)

棋盘上 A 点有一个过河卒，需要走到目标 B 点。卒行走的规则：可以向下、或者向右。同时在棋盘上 C 点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。

棋盘用坐标表示， A 点 $(0, 0)$ 、 B 点 (n, m) (n, m 为不超过20的整数)，同样马的位置坐标是需要给出的。



现在要求你计算出卒从 A 点能够到达 B 点的路径的条数，假设马的位置是固定不动的，并不是卒走一步马走一步。

输入格式

一行四个数据，分别表示 B 点坐标和马的坐标。

输出格式

一个数据，表示所有的路径条数。

是不是感觉和迷宫很像

马的控制点就是障碍，其它均为通路

之前我们用 DFS+回溯来走迷宫

而当迷宫过于空旷时，路径条数特别多

DFS 搜索遍历可能会累死（时间复杂度为 $O((m+n)!/m!n!)$ ），指数
那我们分析一下能不能用动规的思想

1.寻找各阶段的关系

首先我们知道，卒只能往下走或者往右走

So 走到终点只有两种方案：

从终点的上面一格往下走 or 从终点的左边一格往右走

So! 走到终点的路径数 = 左一格的路径数 + 上一格的路径数

即转移方程： $dp[i][j] = dp[i-1][j] + dp[i][j-1]$

（转移方程是不是有点像递推

2.将大问题拆分成小问题

有了上面的推导，我们只需解决左一格的路径数和上一格的路径数就可以啦，相应的，可以通过解决它们的左一格的路径数和上一格的路径数来解决它们…

3.先解决小问题，然后解决大问题

最小的问题：地图的上边界和左边界的点，只有直直的一条路径

然后剩下的问题都可以通过转移方程得到路径数啦

需要特别处理：马的控制点的路径数为 0

最后，上代码：

```

#include<iostream>
#include<algorithm>
using namespace std;

int main()
{
    int i,j,k,x,y,x2,y2;
    cin>>x>>y>>x2>>y2;
    long long a[x+2][y+2];
    //整个地图向右下平移一格, 便于处理
    x2++;y2++;
    //马的控制点
    int c[9] = {x2-1,x2-1,x2-2,x2-2,x2+1,x2+1,x2+2,x2+2,x2};
    int d[9] = {y2-2,y2+2,y2-1,y2+1,y2-2,y2+2,y2-1,y2+1,y2};
    //设定初始路径数
    for(i=0;i<x+2;i++){
        for(j=0;j<y+2;j++){
            a[i][j] = 0;
        }
    }
    //神奇的操作: 使得上边界和左边界路径数为 1
    a[1][0] = 1;
    for(i=1;i<x+2;i++){
        for(j=1;j<y+2;j++){
            for(k=0;k<9;k++){
                //如果是马的控制点, 路径数改为 0
                if(i == c[k] and j == d[k]) goto th;
            }
            a[i][j] = a[i-1][j] + a[i][j-1];
            th:;
        }
    }
    //输出终点的路径数
    cout<<a[x+1][y+1]<<endl;
}

```

时间复杂度 $O(mn)$, 平方级别

线性动规：洛谷题目 1115

题目描述

[展开](#)

给出一段序列，选出其中连续且非空的一段使得这段和最大。

输入格式

第一行是一个正整数 N ，表示了序列的长度。

第二行包含 N 个绝对值不大于 10000 的整数 A_i ，描述了这段序列。

输出格式

一个整数，为最大的子段和是多少。子段的最小长度为 1。

输入输出样例

输入 #1

[复制](#)

输出 #1

[复制](#)

```
7
2 -4 3 -1 2 -4 3
```

```
4
```

说明/提示

【样例说明】

2, -4, 3, -1, 2, -4, 3 中，最大的子段和为 4，该子段为 3, -1, 2。

这道题其实思路挺多的

一个简单的想法：遍历子段的起点和终点，然后遍历该子段，算出该子段的和，最后找出所有子段最大的和

该思路复杂度为 $O(n^3)$ ，代码优化一下可达到 $O(n^2)$

那我们用动规的思路看看呢？

1. 寻找各阶段的关系

分析一下知道，如果子段终点下标为 0，那只有一个子段 $[0, 0]$ ，这种情况的最大子段和为 2（例题数据），保存为 $dp[0]$

如果子段终点下标为 1，这些子段可以分为两种：子段终点下标为 0 的子段 + $a[1]$ 、只有 $a[1]$ ，第一种的最大子段和为子段终点下标

为 0 的最大子段和（即 $dp[0] + a[1]$ ，第二种子段和为 $a[1]$ 。所以总的子段终点下标为 1 的最大子段和为 $\max(dp[0] + a[1], a[1])$ ，保存为 $dp[1]$

如果子段终点下标为 2，这些子段可以分为两种：子段终点下标为 1 的子段 $+a[2]$ 、只有 $a[2]$ ，第一种的最大子段和为子段终点下标为 1 的最大子段和（即 $dp[1] + a[2]$ ，第二种子段和为 $a[2]$ 。所以总的子段终点下标为 2 的最大子段和为 $\max(dp[1] + a[2], a[2])$ ，保存为 $dp[2]$

.....

如果子段终点下标为 n ，这些子段可以分为两种：子段终点下标为 $n-1$ 的子段 $+a[n]$ 、只有 $a[n]$ ，第一种的最大子段和为子段终点下标为 $n-1$ 的最大子段和（即 $dp[n-1] + a[n]$ ，第二种子段和为 $a[n]$ 。所以总的子段终点下标为 n 的最大子段和为 $\max(dp[n-1] + a[n], a[n])$ ，保存为 $dp[n]$

So 经过复杂的推算，得出了转移方程：

$$dp[n] = \max(dp[n-1] + a[n], a[n])$$

（当做题熟练之后，可以自然跳过推算，直接猜出转移方程

2. 将大问题拆分成小问题

有了上面的推导，我们只需解决子段终点下标为 i ($0 \leq i \leq n$) 的最大子段和就好啦，相应的下标为 i 的可以由 $i-1$ 的得到

3. 先解决小问题，然后解决大问题

最小的问题： $dp[0] = a[0]$

最终的问题：最大子段和= $\max(dp[0], dp[1], \dots, dp[n])$

最后，上代码：

```
#include <iostream>
#include <algorithm>
using namespace std;

int main()
{
    int i, j, m, n;
    cin >> m;
    int a[m], dp[m];
    for(i=0; i<m; i++) cin >> a[i];
    n = dp[0] = a[0];
    for(i=1; i<m; i++){
        dp[i] = max(a[i], a[i]+dp[i-1]);
        n = max(n, dp[i]);
    }
    cout << n << endl;
}
```

时间复杂度 $O(n)$

(个人认为一维的比二维的要难理解许多)

任务：

算法：完成洛谷题目 1216、1130、1832、1057、1020、1091

1216、1130、1832、1057 为二维动规

1020、1091 为一维动规

动规是十分重要的算法/思路，因此题数较多，难度也较高

(重要到姐又搞这个搞到了两点才睡)

提交：把 6 道题的代码和 AC 截图以及**转移方程**发给姐就好啦

ddl：无