剑指offer30 连续子数组的最大和【1】【初级动态规划】

- **题目**
  HZ偶尔会拿些专业问题来忽悠那些非计算机专业的同学。今天测试组开完会后,他又发话了:在古老的一维模式识别中,常常需要计算连续子向量的最大和,当向量全为正数的时候,问题很好解决。但是,如果向量中包含负数,是否应该包含某个负数,并期望旁边的正数会弥补它呢？例如:{6,-3,-2,7,-15,1,2,2},连续子向量的最大和为8(从第0个开始,到第3个为止)。给一个数组，返回它的最大连续子序列的和，你会不会被他忽悠住？(子向量的长度至少是1)
- **思路**
  动态规划（根据上一个状态推到）
  F(i)：是以**当前array[i]为末尾元素的子数组**和。（当前的子数组，不一定是最大值）
  F(i) = max(F(i-1)+array[i]，array[i])
  result：为全部子数组中和最大的。
  result = max(result , F[i])；
- 代码实现

```
public:
    int FindGreatestSumOfSubArray(vector<int> array) {
        int len = array.size();
        if(!len)
            return -1;
        int cur_sum = array[0], res_max = cur_sum;
        for(int i = 1; i < len; ++i){
```

```
            cur_sum = cur_sum + array[i] > array[i] ? cur_sum + array[i]:array[i];
            res_max = res_max > cur_sum ? res_max:cur_sum;
        }
        return res_max;
    }
};
```

换钱数

简述题目：用1元，2元，5元，换成20元。一共有多少种换法。每一种纸币数量不限。

```

```

换钱的最少货币数

给定数组arr, arr中所有的值都为正数且不重复。每个值代表一中面值的货币，每种面值的货币可以使用任意张，再给定一个整数aim代表要找的钱数，求组成aim的最少货币数。

- 思路:
  如果arr的长度为N, 则生成一个行数为N, 列数为aim+1的动态规划表dp[N][aim+1], dp[i][j]的含义为:在可以任意使用arr[0...i]货币的情况下，组成j所需的最小张数。
  设: arr=[5,2,3,1] aim = 5
  1、dp[0..N-1][0]的值表示找钱数为0时需要的最少张数，所以全设为0。（矩阵的第一列）
  2、dp[0][0...aim]的值表示只能使用arr[0]货币也就是5的情况下，找0 ,1,2,3,4,5的钱的情况下。其中无法找开的一律设为32位的最大值，记为inf.
  3、剩下的位置依次从左到右，再从上到下计算。假设计算到(i,j)位置，dp[i][j]的值可能来自下面的情况:
  3.1 完全不使用当前货币arr[i]情况系的最少张数，即dp[i-1][j]的值；
  3.2 当j >= arr[i] && dp[i][j-arr[i]] != INF时，可以考虑使用arr[i]，即 **dp[i][j] = min(dp[i][j], dp[i][j-arr[i]] + 1);**

```

```

```cpp
#include <bits/stdc++.h>
using namespace std;
#define INF 0x3f3f3f3f

int minConins(vector<int> arr, int aim){
 if(arr.size()==0 || aim < 0){
  return -1;
 }
 vector< vector<int>> dp(arr.size(),vector<int>(aim+1));
 for (int j = 1; j <= aim; ++j){  //初始化第一行
  dp[0][j] = INF;
  if(j >= arr[0] && dp[0][j-arr[0]] != INF){
   dp[0][j] = dp[0][j-arr[0]] + 1;
  }
 }
 for(int i = 1;i < arr.size(); ++i){
  for (int j = 1; j <= aim; ++j){
   dp[i][j] = dp[i-1][j]; //默认不使用当前货币arr[i]
   if(j >= arr[i] && dp[i][j-arr[i]] != INF){ //可使用货币arr[i]时
    dp[i][j] = min(dp[i][j], dp[i][j-arr[i]] + 1);
   }
  }
 }
 return dp[arr.size()-1][aim];
}
```

055 跳跃游戏（middle）

- **题目**

  给定一个非负整数数组，你最初位于数组的第一个位置。
  数组中的每个元素代表你在该位置可以跳跃的最大长度。
  判断你是否能够到达最后一个位置。

  > 示例 1:
  > 输入: [2,3,1,1,4]
  > 输出: true
  > 解释: 从位置 0 到 1 跳 1 步, 然后跳 3 步到达最后一个位置。

- **思路**

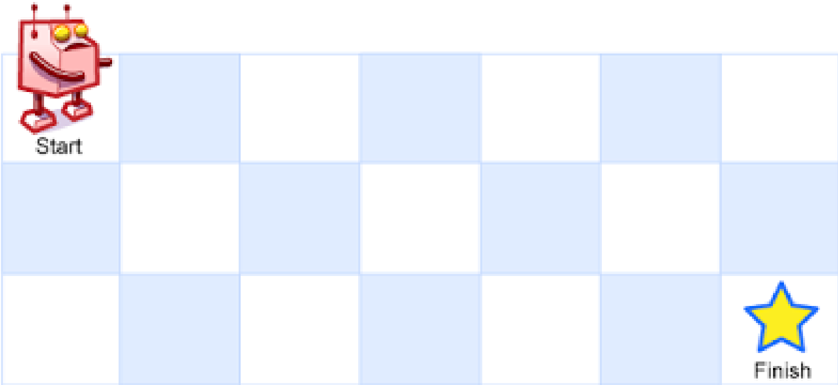
动态规划的思路：用dp[i]保存在i状态剩余的最大步数，由上一步推出，即dp[i - 1] 和nums[i - 1]中较大的值。所以dp[i - 1] > nums[i - 1] ? dp[i] = dp[i - 1] - 1: dp[i] = nums[i - 1] - 1; 目前觉得动态规划的要点是状态 i 的数据是由状态 i - 1推出的。贪心策略：只关注能不能到达最后那个位置，用一个变量reach保存当前能到达的位置，如果reach < i 或者 reach >= n - 1则跳出。否则reach更新为reach和nums[i] + i中的较大值。

- **代码实现**

```
//动态规划
class Solution {
public:
    bool canJump(vector<int>& nums) {
        vector<int> dp(nums.size(), 0);
        for (int i = 1, len = nums.size(); i < len; ++i) {
            dp[i - 1] > nums[i - 1] ? dp[i] = dp[i - 1] - 1: dp[i] = nums[i - 1] - 1;
            if (dp[i] < 0)
                return false;
        }
        return true;
    }
};

//贪心策略
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int len = nums.size(),reach = 0;
        for (int i = 0; i < len; ++i) {
            if(reach < i || reach >= len -1) break;
            if(reach < nums[i] + i)
                reach = nums[i] + i;
        }
        return reach >= len -1;
    }
};
```

062 不同路径（middle）

- **题目**
  一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。
  机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。
  问总共有多少条不同的路径？
  

例如，上图是一个7 x 3 的网格。有多少可能的路径？
说明：m 和 n 的值均不超过 100。

> 示例 1:
> 输入: m = 3, n = 2
> 输出: 3
> 解释:
> 从左上角开始，总共有 3 条路径可以到达右下角。
>
> 1. 向右 -> 向右 -> 向下
> 2. 向右 -> 向下 -> 向右
> 3. 向下 -> 向右 -> 向右

- **思路**
  动态规划：dp[i][j] = dp[i - 1][j] + dp[i][j - 1];dp[0][j]和dp[i][0]都等于1。注意起始位置的值。
  此题采用二维数组过于浪费空间。定义dp的长度为 m，即一行的长度。

1	1	1	1	1
1			dp[i]	
1	dp[i - 2]	dp[i - 1]	dp[i]+dp[i - 1]	

为1的都是边界行，对于第 i 行 dp[i] = 上一行的dp[i] + 左边的dp[i - 1]

- **代码实现**

```cpp
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<int> dp(m, 1);
        for(int i = 1; i < n; ++i)
            for(int j =1; j < m; ++j)
                dp[j] += dp[j - 1];
        return dp[m - 1];
    }
};
```

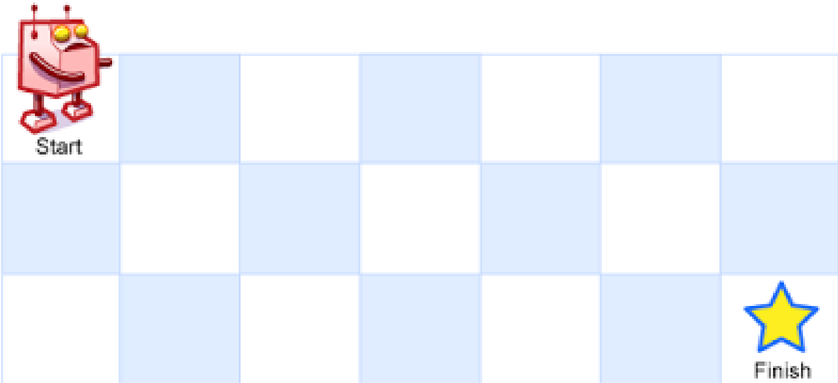
063 不同路径II（middle）

- **题目**

  一个机器人位于一个 m x n 网格的左上角 （起始点在下图中标记为“Start” ）。

  机器人每次只能向下或者向右移动一步。机器人试图达到网格的右下角（在下图中标记为“Finish”）。

  现在考虑网格中有障碍物。那么从左上角到右下角将会有多少条不同的路径？



网格中的障碍物和空位置分别用 1 和 0 来表示。

说明：m 和 n 的值均不超过 100。

> 示例 1:
> 输入:
> [
> [0,0,0],
> [0,1,0],
> [0,0,0]
>]

> 输出: 2
> 解释:
> 3x3 网格的正中间有一个障碍物。
> 从左上角到右下角一共有 2 条不同的路径：
>
> 1. 向右 -> 向右 -> 向下 -> 向下
> 2. 向下 -> 向下 -> 向右 -> 向右

- **思路**
  动态规划：我认为动态规划的**初始状态**很重要。
  此题加入了障碍物，所以得考虑得从第0行开始，每行从第0列开始。
  dp[0] = 1 这点很重要

1				
			dp[i]	
		dp[i - 1]	dp[i]+dp[i - 1]	

每一行从0列开始，若有障碍物则赋为0；无障碍物时，0列为dp[0]不表白呢，其他列dp[i] = dp[i]+dp[i - 1]

- **代码实现**

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        if(obstacleGrid.empty() || obstacleGrid[0].empty() ||
obstacleGrid[0][0] == 1) return 0;
        int m = obstacleGrid[0].size();
        int n = obstacleGrid.size();
        vector<long> dp(m, 0);
        dp[0] = 1;
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < m; ++j){
                if(obstacleGrid[i][j] == 1) dp[j] = 0;
                else if(j > 0) dp[j] += dp[j - 1];
            }
        return dp[m - 1];
    }
```

```
};
```

064 最小路径和（middle）

- **题目**
  给定一个包含非负整数的 m x n 网格，请找出一条从左上角到右下角的路径，使得路径上的数字总和为最小。
  说明：每次只能向下或者向右移动一步。

> 示例:
> 输入:
> [
> [1,3,1],
> [1,5,1],
> [4,2,1]
>]
> 输出: 7
> 解释: 因为路径 1→3→1→1→1 的总和最小。

- **思路**
  动态规划的关键的点，得处理好初始状态的值，若固定的值可灵活赋值，否则得提前完成处理。
  此题通过二维数组dp[i][j] 保存每个状态的最小路径和。dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];

- **代码实现**

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {
        if(grid.empty() || grid[0].empty()) return 0;
        int len_x = grid[0].size();
        int len_y = grid.size();
        int dp[len_y][len_x];
        dp[0][0] = grid[0][0];
        for(int i = 1; i < len_x; ++i)  //处理第 0 行
            dp[0][i] = grid[0][i] + dp[0][i - 1];
        for(int i = 1; i < len_y; ++i)  //处理第 0列
            dp[i][0] = grid[i][0] + dp[i - 1][0];
```

```
        for(int i = 1; i < len_y; ++i)
            for(int j = 1; j < len_x; ++j)
                dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];
        return dp[len_y - 1][len_x - 1];
    }
};
```

070 爬楼梯（easy）

- **题目**
  假设你正在爬楼梯。需要 n 阶你才能到达楼顶。每次你可以爬 1 或 2 个台阶。你有多少种不同的方法可以爬到楼顶呢？
  注意：给定 n 是一个正整数。
- **思路**
  dp[n] = dp[n-1] + dp[n-2]
- **代码实现**

```
    int climbStairs(int n) {
        if (n < 3)
            return n;
        int curentWays = 2, LastWays = 1;
        while (n > 2) {
            int temp = curentWays;
            curentWays += LastWays;
            LastWays = temp;
            --n;
        }
        return curentWays;
    }
```