动态规划（DP）

> 动态规划，常用于：数学，管理科学，计算机科学，经济和生物信息学。
>
> 特征：一个问题，可以拆分成一个一个的子问题，解决子问题，顺带解决这个问题
>
> 核心思想：拆分子问题，记住过程，减少重复运算。
>
> 例子：
>
> - 1+1+1+1+1+1=？等于6
> - 在上面式子的在加上一个“1+”。答案是多少?
> - 直接通过以往计算过的答案，再加1

题型：青蛙跳阶问题：

> 一只青蛙，可以一次跳上1级台阶，也可以一次跳上2级台阶。求这只青蛙跳10级台阶有多少种跳法?

答案：**89**
暴力递归（还不是动态规划）

```java
public class Main {
    public static void main(String[] args) {
        System.out.println(f(10));
    }
    public static int f(int n){
        if(n == 1) return 1;
        else if(n == 2) return 2;
        else return f(n - 1) + f(n - 2);
    }
}
```

使用字典改进

```java
import java.util.HashMap;
import java.util.Map;

public class Main {
    static Map<Integer,Integer> map = new HashMap<>();//创建字典
    public static void main(String[] args) {
        System.out.println(f(10));
    }

    public static int f(int n){
        if(n == 1) return 1;
```

```
        if(n == 2) return 2;
        if(map.containsKey(n)){ //如果包含对应n的值
            return map.get(n);//直接返回n对应的值
        }
        else {
            map.put(n,f(n-1)+f(n-2));//如果没有包含，那就把对应n的键值对放进去
            return map.get(n);//然后直接返回，我们上一步计算好的值对
        }
    }
}
```

***自下向上的动态规划

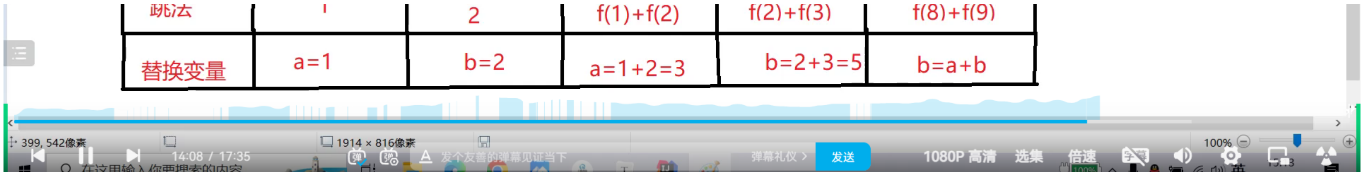
2.自底而上的动态规划

动态规划和带字典的递归有些不同，但解法类似。都有减少重复运算。

- 带字典的递归是从f(10)--->f(1),自顶而下。
- 动态规划，从f(1)--->f(10)，自底而上。

动态规划的特征：

- 最优子结构：f(n) = f(n-1) + f(n-2), f(n-1)和 f(n-2)就是f(n)的最优子结构。
- 状态转移方程：f(n) = f(n-1) + f(n-2)
- 边界：f(1) =1 ,f(2) = 2
- 重叠子：重复的运算：f(10) = f(9) + f(8),f(9) = f(8)+f(7). f(8)就是重叠子




```java
public class Main {
    static Map<Integer,Integer> map = new HashMap<>();//创建字典
    public static void main(String[] args) {
        int a = 1;
        int b = 2;
        for(int i = 3; i <= 10; i++){
            int tmp = a;
            a = b; //把a变为 b
            b = a + tmp;//b变为a+b;
        }
        System.out.println(b);
    }
}
```

写成函数

```java
public class Main {
    static Map<Integer,Integer> map = new HashMap<>();//创建字典
    public static void main(String[] args) {
        System.out.println(ways(10));
    }
    public static int ways(int n){
        if(n == 1) return 1;
        if(n == 2) return 2;
        int a = 1;
        int b = 2;
        for(int i = 3; i <= n; i++){
            int tmp = a;
            a = b;
            b = a + tmp;
        }
        return b;
    }
}
```

动态规划的解题思路

什么样的问题适用动态规划?

> 如果一个问题，可以把所有的答案穷举出来，而且存在重叠子问题，就可以考虑使用动态规划。

动态规划的经典应用场景:

> - 最长递增子序列 ---20年蓝肽子问题

> - 最小（大）距离 --- 数字三角形
> - 背包问题
> - 凑零钱问题
> - 等等等。。。

动态规划的解题思路

- 核心思想：拆分子问题，记住过程，减少重叠子运算
- 穷举分析
- 确定边界
- 找出规律，确定最优子结构
- 写出状态转移方程
- 代码实现

LCR 100. 三角形最小路径和

已解答

中等 相关标签 相关企业 Aあ

给定一个三角形 `triangle` ，找出自顶向下的最小路径和。

每一步只能移动到下一行中相邻的结点上。**相邻的结点** 在这里指的是 **下标** 与 **上一层结点下标** 相同或者等于 **上一层结点下标 + 1** 的两个结点。也就是说，如果正位于当前行的下标 `i` ，那么下一步可以移动到下一行的下标 `i` 或 `i + 1` 。

示例 1:

```
输入: triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]
输出: 11
解释: 如下面简图所示:
   2
  3 4
 6 5 7
4 1 8 3
自顶向下的最小路径和为 11 (即，2 + 3 + 5 + 1 = 11) 。
```

示例 2:

```
输入: triangle = [[-10]]
输出: -10
```

具体来说，代码中的双重循环用于从三角形的底部向上逐层计算最小路径和。在每一层中，对于当前位置`(m-1, n)`，选择下一层相邻的两个位置`(m, n)`和`(m, n+1)`中的较小值，将其累加到当前位置上。这样就能够逐步计算出从底部到顶部的最小路径和。

最终返回`triangleArray[0][0]`，即顶部的值，即为整个三角形的最小路径和。

```java
class Solution {
    public int minimumTotal(List<List<Integer>> triangle) {
        int len = triangle.size();
        int[][] triangleArray = new int[triangle.size()][];
        for (int i = 0; i < triangle.size(); i++) { //先变为二维数组
             List<Integer> row = triangle.get(i);
             triangleArray[i] = new int[row.size()];
             for (int j = 0; j < row.size(); j++) {
                 triangleArray[i][j] = row.get(j);
            }
        }
        for(int m = len - 1; m > 0; m--){
            for(int n = 0; n < m; n++){
                triangleArray[m - 1][n] += triangleArray[m][n] < triangleArray[m][n + 1] ?
triangleArray[m][n] : triangleArray[m][n + 1];
            }
        }
        return triangleArray[0][0];
    }
}
```