题目描述

给出n个牌数，在-100到100之间，求最大得分。

规则如下：连续翻牌，如果选当前牌，则总得分等于上一次翻牌总得分加上当前牌的数字，

如果当前总得分小于它前三次的总得分的话，那此次不翻牌，并且总得分就等于它前三次的得分。

1到3次翻牌数如果小于0的话就取0。

例子：1，-5，-6，4，7，2，-2

（1）1大于零 翻牌
（2）-5 加上1 小于0 不翻 结果为0
（3）-6 加上0 小于0 不翻 结果为0
（4）4 加上0 大于0（1）翻牌 结果为4
（5）7 加上4 大于0（2）翻牌 结果为11
（6）2 加上11 大于0（3）翻牌 结果为13
（7）-2 加上14 大于4（4）翻牌 结果为11

输入描述

无

输出描述

无

用例

输入	1,-5,-6,4,7,2,-2
输出	11
说明	请看题目描述

题目解析

本题比较迷惑人的是

> 如果当前总得分小于它前三次的总得分的话，那此次不翻牌，并且总得分就等于它前三次的得分

这里的前三次应该指的是三轮前，比如当前是第10轮，则前三轮是第7轮。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.on("line", (line) => {
  const arr = line.split(",").map(Number);

  const dp = [];

  for (let i = 0; i < arr.length; i++) {
    if (i === 0) {
      dp[0] = Math.max(0, arr[0]);
    } else if (i < 3) {
      dp[i] = Math.max(0, dp[i - 1] + arr[i]);
    } else {
      dp[i] = Math.max(dp[i - 3], dp[i - 1] + arr[i]);
    }
  }

  console.log(dp.at(-1));
});
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  // 输入获取
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    Integer[] arr =
        Arrays.stream(sc.nextLine().split(",")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(arr));
  }

  // 算法入口
  public static int getResult(Integer[] arr) {
    int n = arr.length;

    int[] dp = new int[n];

    for (int i = 0; i < n; i++) {
      if (i == 0) {
        dp[0] = Math.max(0, arr[0]);
      } else if (i < 3) {
        dp[i] = Math.max(0, dp[i - 1] + arr[i]);
      } else {
        dp[i] = Math.max(dp[i - 3], dp[i - 1] + arr[i]);
      }
    }

    return dp[n - 1];
  }
}
```

Python算法源码

```python
# 输入获取
arr = list(map(int, input().split(",")))


# 算法入口
def getResult():
    n = len(arr)

    dp = [0] * n

    for i in range(n):
        if i == 0:
            dp[0] = max(0, arr[0])
        elif i < 3:
            dp[i] = max(0, dp[i - 1] + arr[i])
        else:
            dp[i] = max(dp[i - 3], dp[i - 1] + arr[i])

    return dp[-1]


# 算法调用
print(getResult())
```