一.问题描述

给定一些活动的时间序列$intervals[i] = [start_i, end_i]$,返回最少需要删除活动的总数，使得其余活动都不重叠。

二.算法设计

2.1穷举思想

若要求最少的总数,对于活动序列可以从依次选取$0, 1, 2, 3.....n$个进行去除,然后判断剩余的活动是否重叠。

选取活动序列的时间
$T(n) = C(n,0) + C(n,1) + C(n,2) + ... + C(n,n) = \Sigma_{k=0}^{n} C(n,k)$,
判断剩余活动是否重叠的时间$G(n) = 0 + 1 + 2 + ... + k - 1 = \frac{k(k-1)}{2}$
,总的时间复杂度
$M(n) = \Sigma_{k=0}^{n} K^2 C(n,k) = n2^{n-1} + (n^2 - n)2^{n-2} = O(n^2 2^n)$。所以穷举的办法在这里行不通。

2.2贪心算法

想法简单来说就是：活动结束时间越早，就有更多的时间参加更多的活动，就会有更少的重叠。基于这样子的思想，本题适用于采用贪心算法的策略，具体思路为按区间结尾值的大小升序排序后，每次优先保留结尾小且与前一个区间不重叠的区间，选取足够的区间之后，剩余的就是需要删除的。

```
1.排序操作
template <class type>
void sort(vector<vector<type>>& arr) //按照结束时间进行排序
{
    int size = arr.size();
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size - 1 - i; j++)
        {
```

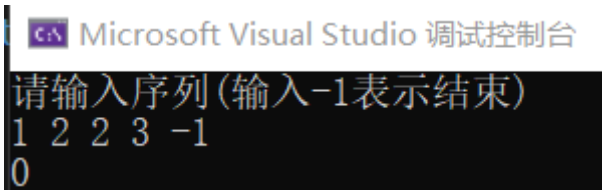
```
            if (arr[j][1] > arr[j + 1][1]|| arr[j][1] ==
arr[j + 1][1] && arr[j][0] > arr[j + 1][0])
            {
                vector<type> temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
2.贪心算法
template <class type>
int Solution(const vector<vector<type>>& arr) //贪心算法
{
    int size = arr.size();
    if (size == 0)
        return 0;
    type minend = arr[0][1];
    int count = 1;
    for (int i = 1; i < size; i++)
    {
        if (arr[i][0] >= minend)//如果前一个任务的结束时间小于
等于下一个任务的开始时间
        {
            minend = arr[i][1];
            count++;
        }
    }
    return size - count;
}
```

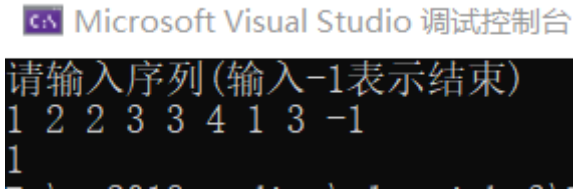
三.时间复杂度

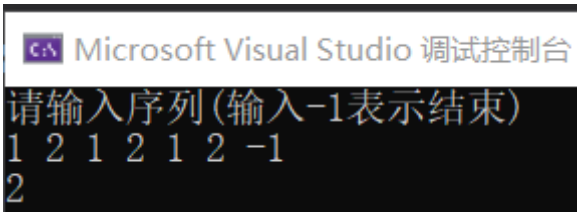
贪心算法时间复杂度主要由两个组成，一个是排序算法，一个是贪心算法。

- 排序算法最低时间复杂度为$O(nlogn)$,此处我用的冒泡排序所以时间复杂度为$O(n^2)$。
- 贪心算法需要遍历整个数组，所以时间复杂度是$O(n)$,总的时间复杂度就是$O(n^2)+O(n)=O(n^2)$或者$O(nlogn)+O(n)=O(nlogn)$

四.运行结果截图


Microsoft Visual Studio 调试控制台
请输入序列(输入-1表示结束)
1 2 2 3 -1
0


Microsoft Visual Studio 调试控制台
请输入序列(输入-1表示结束)
1 2 2 3 3 4 1 3 -1
1


Microsoft Visual Studio 调试控制台
请输入序列(输入-1表示结束)
1 2 1 2 1 2 -1
2