# 实验报告

## 一、802 题正确性证明

| 编号 | 题目 | 状态 | 分数 | 总时间 | 内存 | 代码 / 答案文件 | 提交者 | 提交时间 |
|---|---|---|---|---|---|---|---|---|
| #546047 | #802 Weighted Interval Scheduling | ✔ Accepted | 100 | 577 ms | 2432 KB | cpp / 2930 B | 李馨雨(2020202279) | 2022/5/27 09:18 |

## 二、思路说明

首先将区间按照它们的结束时间从早到晚进行排序。

对于某个活动 j, 定义 p[j]表示排在活动 j 之前的所有活动里面离 j 最近并且相容的活动，如果没有相容的活动，则 p[j] = 0。

定义 M[i]，表示从 i 个活动{1, 2, ⋯, i}中能够得到的不重叠的子集的最大权值和。

当求解 M[n]时，有两种选择：如果不保留区间 n，则选择的区间组合没有发生变化，与{1, 2, ⋯, n-1}中能够获取的最大权重和相等，即 M[n] = M[n-1]；如果保留区间 n，则必须去掉{1, 2, ⋯, n-1}中所有跟 n 冲突的区间，并且由 p 数组的定义，即只能从区间{1, 2, ⋯, p(n)}中选取要参加的区间组合，因此 M[n] = M[p[n]] + vn

因此，得到 M 数组的递推公式：M[n] = max( M[p[n]] + vn, M[n-1])。根据该递推公式即可求得最大权值和。

此外，值得注意的是，为了减小时间复杂度，在初始化 p 数组时我使用了二分查找的方法，使复杂度降低为 O(nlogn)。并且在最终求解权值和时将函数的递归转化成了 for 循环迭代的方式以减少用时。

## 三、代码说明

首先是基本数据结构的定义，包括区间、p 数组和 M 数组等

```
struct Interval {  //区间，包括开始时间、结束时间和权重
    int beginTime;
    int endTime;
    int weight;
};
```

```
class DP {
public:
    Interval reqs[MAX_NUM + 1];
    int IntervalNum;
    int p[MAX_NUM + 1]; //p数组，表示最近并且相容的活动
    int M[MAX_NUM + 1]; //M数组，表示最大权重和
```

在 DP 类内添加成员函数，用于求解本问题。其中 prepare 函数做预处理，先对所有区间根据结束时间排序，然后初始化 p、M 数组。binarySearch 函数则用二分查找法来确定 p 数组的值。solveWIS 函数则迭代求解 M 数组的值，最终得到最大权值和。

```
void prepare() {
    sort(reqs + 1, reqs + IntervalNum + 1); //根据结束时间对所有区间排序
    memset(p, 0, sizeof(p));
    memset(M, 0, sizeof(M)); //初始化数组M
    for (int i = 1; i <= IntervalNum; i++) { //初始化数组p
        p[i] = binarySearch(i, reqs[i].beginTime);
    }
}
```

```
int binarySearch(int n, int key){ //二分查找最近并且相容的活动
    int low = 1;
    int high = n - 1;
    int mid;
    while(low < high){
        mid = (low + high + 1)/2;
        Interval midVal = reqs[mid];
        if(midVal.endTime <= key)
            low = mid;
        else if(midVal.endTime > key)
            high = mid - 1;
        else
            return mid;
    }
    if(reqs[low].endTime > key) return 0;
    if(reqs[low].endTime <= key) return low;
    else return high;
}
```

```
// 动态规划算法
int solveWIS() {

    M[0] = 0;
    for(int i = 1; i <= IntervalNum; i++){
        if(reqs[i].weight + M[p[i]] >= M[i - 1]){
            M[i] = reqs[i].weight + M[p[i]];
        }else{
            M[i] = M[i - 1];
        }
    }
    return M[IntervalNum];
}
```