实验七：最佳调度问题的回溯算法

姓名：陈鸿绪　　学号：PB21000224　　日期：11.29

实验内容：设有 n 个任务由 k 个可并行工作的机器来完成，完成任务 i 需要时间为 it 。试设计一个算法找出完成这 n 个任务的最佳调度，使完成全部任务的时间最早。(要求给出调度方案)。实验中给出了三个文件，需要读取三个文件得到最终结果。

实验目的：熟悉回溯算法，可以在回溯算法中进行适当剪枝优化。

算法设计思路：

1. 首先利用贪心给出一个较优的解，并算出其对应的完成时间作为开始的阈值。
2. 为了使得回溯的机会变得更多，考虑将原来的事务按照权重进行从大到小的排序。
3. 然后普通回溯，利用当前最短完成时间为阈值，一方面如果当前分配对应的完成时间超过当前阈值，则进行剪枝，另一方面如果当前的分配情况和之前某个分配的数值相同，则直接跳过这个分配，这样就又能进行一个剪枝。
4. 最后将最短完成时间对应的分配方案给出。

主代码以及解释：

```
struct pair_{
    int i;
    int weight;
};//为了在排序之后便于将每个权重和序号对应起来

typedef struct pair_ pair_;
```

```
int cmp_2(const void *a, const void *b){
    return ((pair_*) b)->weight-((pair_*) a)->weight;
}//快排的比较函数

int cmp(const void *a, const void *b){
    return *((int*) b)-*((int*) a);
}//快排的比较函数

void dfs(int *solution,int *good_solution,int *weight,int *all_time,int m,int k,int n,int &temp_min){
//solution 表示当前分配，good_solution 表示当前最优分配，weight 传递权重，all_time 实
//时更新当前分配的时间情况，m 是当前回溯所在层数，k 是当前回溯所在列，n 为事务数，
//temp_min 是当前最好分配对应的时间
    while(true){
        if(m==-1) break; //回溯完毕
        else if(m>=n){
            m=m-1; //回溯
            int temp_max_time=0;
            for(int i=0;i<k;i++)  if(temp_max_time<all_time[i]) temp_max_time=all_time[i];
            if(temp_max_time<temp_min){
            //第一次剪枝，利用当前阈值进行剪枝
                temp_min=temp_max_time;
                for(int i=0;i<n;i++){
                    good_solution[i]=solution[i];
                }
            }
        }
        int flag=0;
        do{
            flag=0;
            solution[m]++;
            if(solution[m]>k) break;
            for(int i=0;i<solution[m];i++){
                if(all_time[i]==all_time[solution[m]]){
                    flag=1;break;
                }//第二次剪枝，利用当前所在位置的分配与之前某个情况相同，将其跳过
            }
            if(solution[m]-2>=0) all_time[solution[m]-2]-=weight[m];
            all_time[solution[m]-1]+=weight[m];
        }while(all_time[solution[m]-1]>temp_min||flag);
        if(solution[m]>k){
            all_time[k-1]-=weight[m];
            solution[m]=0;
            m=m-1;
```

```
                continue;
            }//回溯
            else{
                m++;
                continue;
            }
        }
    }
```

算法测试结果：

```
test1.txt
total time: 112
machine1 solution: 10 7 9 5
machine2 solution: 6 8 3
machine3 solution: 1 4 2
time: 0 ms
```

```
test2.txt
total time: 182
machine1 solution: 13 14 3
machine2 solution: 8 9 5
machine3 solution: 11 10 12
machine4 solution: 1 2
machine5 solution: 6 15 4 7
time: 12.957 ms
```

```
test3.txt
total time: 126
machine1 solution: 19 1 2
machine2 solution: 10 17 13
machine3 solution: 15 18 3
machine4 solution: 6 14
machine5 solution: 8 4
machine6 solution: 12 9
machine7 solution: 16 7
machine8 solution: 5 11
time: 80.362 ms
```

实验过程遇到的困难和收获：

一开始没有进行剪枝的时候，程序几乎跑不出来，后来只考虑了一种剪枝方案，仍然没有跑出来，所以我尝试将其进行排序，排序后可以在三到五分钟之类跑出来，但是仍然时间很长，所以最后我尝试将两种剪枝方案结合剪枝，将 test3 时间优化到 80ms。收获：回溯的时间复杂度非常高，要尝试着用多次剪枝操作降低时间代价。