《算法设计与分析》上机报告

姓名：	廖洲洲	学号：	PB17081504	日期：	2019.11.23
上机题目：	最佳调度问题的回溯算法				

实验环境：
CPU：Intel Core i7-8550U; 内存:8G ; 操作系统：Win 10 ; 软件平台: JetBrains CLion ;

一、算法设计与分析：

题目一:
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 t_i。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）

二、核心代码：

题目一:
(一)算法思想：

1. 采用子集树回溯算法，本问题解空间是一个 n 层满 k 叉树。
   - 搜索从开始结点(根结点)出发,以 DFS 搜素整个解空间
   - 开始结点就成为一个活结点,同时也成为当前的扩展结点。在当前的护展结点处向纵深方向移至一个新结点,并成为一个新的活结点,也成为当前扩展结点。
   - 如果当前的扩展结点处不能再向纵深方向扩展，则当前扩展结点就成为死结点。
   - 此时,应往回移动(回溯)至最近的一个活结点处,并使这个活结点成为当前的扩展结点;直至找到一个解或全部解。
2. 使用变量 min_time 存储任务结束的最早时间；best_scheduler[]数组存储最优调度，表示第 i 个任务由机器 best_scheduler[i]执行；time[i]表示机器完成任务 i 所需时间；x[]数组用于存储子集树的搜索路径。
3. 首先主函数调用 Backtrack(1)，从子集树第一层开始搜索，然后由第 t 层向第 t+1 层扩展，确定 x[t]的值。对于树的第 t 层，第 t 层的结点值为 x[t]，表示第 t 个任务由机器 x[t]执行。对每个任务 t，都有 k 种分配方法，即分配给 1~k。
4. 当 t 的值大于 n 时，说明搜索完叶子节点，是目前发现的最优解，将该解保存下来。当 t 的值小于等于 n 时，说明还在树中搜索，采用限界，若当前路径的分配方法所需时间已经超过当前最优时间，则不再往下搜索。若小于最优解，往下一层搜索。
5. 任务执行时间的计算：计算出每台机器的执行任务总时间，其中的最大值即为任务执行时间。

(二)核心代码

```
void Backtrack(int t){//子集树的回溯法，搜索到树的第 t 层
    //由第 t 层向第 t+1 层扩展，确定 x[t]的值
    int i;
    if(t>n){//说明是当前的最优解，将该解保存下来
        min_time=time_scheduler(n);//当前任务调度所需时间
        for(i=1;i<=n;i++)
            best_scheduler[i]=x[i];
    }
    else{
        for(i=1;i<=k;i++){//对每个任务，都有 k 种分配方法
            x[t]=i;
            if(time_scheduler(t)<min_time)//如果当前路径的时间已经超过了最小时间，则不再往前走
                Backtrack(t+1);//向下一层扩展
        }
    }
}

int time_scheduler(int m){//执行任务所需时间
    int i,max=0;
    int *machine_time;//每台机器所花费的时间，花费最多的时间值即为任务的时间
    machine_time=(int *)malloc((k+1)* sizeof(int));
    for(i=1;i<=k;i++)
        machine_time[i]=0;
    for(i=1;i<=m;i++){
        machine_time[x[i]]=machine_time[x[i]]+time[i];
    }//将完成任务 i 所需时间加到完成任务 i 的机器的花费时间上
    for(i=1;i<=k;i++){//找出机器所花时间的最大值，即为总时间
        if(machine_time[i]>max)
            max=machine_time[i];
    }
    return max;
}
```

三、结果与分析：

题目一:

(一)

```
请输入任务数量:
10
请输入机器数量:
5
请输入各个任务执行时间:
1 7 4 10 9 4 8 8 2 4
任务数: 10,机器数: 5
各任务时间: 1   7       4       10      9       4       8       8       2       4
最佳任务调度:
任务    1       2       3       4       5       6       7       8       9       10
机器    1       1       1       2       3       4       4       5       2       5
完成时间: 12
进程已结束, 退出代码 0
```

(二)

```
请输入任务数量:
5
请输入机器数量:
3
请输入各个任务执行时间:
1 1 1 4 2
任务数: 5,机器数: 3
各任务时间: 1   1       1       4       2
最佳任务调度:
任务    1       2       3       4       5
机器    1       1       1       2       3
完成时间: 4
```

总结：

通过子集树的回溯算法实现了最佳调度算法问题，加深了对回溯法的了解和运用。学习了回溯法求解问题的基本步骤：

1) 针对问题，首先定义问题的解空间。
2) 确定易于搜索的解空间结构（按树或图组织解，子集树还是排列树）。
3) 以深度优先方式搜索解空间，搜索过程中裁掉死结点以提高效率。

附录（源代码）

算法源代码（C/C++/JAVA 描述）

```
#include <stdio.h>
#include <stdlib.h>
int min_time=999999;//任务结束的最早时间
int *best_scheduler;//best_scheduler 是一个数组，表示第 i 个任务由机器 best_scheduler[i]执行
int n;//任务数量
int k;//机器数量
int *time;//time 是一个数组，time[i]表示机器完成任务 i 所需时间
int *x;//x 数组用于存储当前路径

int time_scheduler(int m){//执行任务所需时间
    int i,max=0;
    int  *machine_time;//每台机器所花费的时间，花费最多的时间值即为任务的时间
    machine_time=(int *)malloc((k+1)* sizeof(int));
    for(i=1;i<=k;i++)
        machine_time[i]=0;
    for(i=1;i<=m;i++){
        machine_time[x[i]]=machine_time[x[i]]+time[i];
    }//将完成任务 i 所需时间加到完成任务 i 的机器的花费时间上
    for(i=1;i<=k;i++){//找出机器所花时间的最大值，即为总时间
        if(machine_time[i]>max)
            max=machine_time[i];
    }
    return max;
}

void  Backtrack(int  t){//子集树的回溯法，搜索到树的第 t 层，第 i 层的结点值 x[i]表示第 i 个任务由机器 x[i]执行
    //由第 t 层向第 t+1 层扩展，确定 x[t]的值
    int i;
    if(t>n){//说明是当前的最优解，将该解保存下来
        min_time=time_scheduler(n);//当前任务调度所需时间
        for(i=1;i<=n;i++)
            best_scheduler[i]=x[i];
    }
    else{
        for(i=1;i<=k;i++){//对每个任务，都有 k 种分配方法
            x[t]=i;
            if(time_scheduler(t)<min_time)//如果当前路径的时间已经超过了最小时间，则不再往前走
                Backtrack(t+1);//向下一层扩展
        }
```

```
        }
}

int main(){
        int i;

        printf("请输入任务数量：\n");
        scanf("%d",&n);
        printf("请输入机器数量：\n");
        scanf("%d",&k);
        best_scheduler=(int *)malloc((n+1)* sizeof(int));
        time=(int *)malloc((n+1)* sizeof(int));
        x=(int *)malloc((n+1)* sizeof(int));
        printf("请输入各个任务执行时间：\n");
        for(i=1;i<=n;i++)
                scanf("%d",&time[i]);
        printf("任务数：%d,机器数：%d\n 各任务时间：",n,k);
        for(i=1;i<=n;i++)
                printf("%d\t",time[i]);


        Backtrack(1);//从第一层开始，子集树回溯算法

        printf("\n 最佳任务调度：\n");
        printf("任务\t");
        for(i=1;i<=n;i++)
                printf("%d\t",i);
        printf("\n 机器\t");
        for(i=1;i<=n;i++)
                printf("%d\t",best_scheduler[i]);
        printf("\n 完成时间：%d",min_time);
}
```