算法分析与设计基础作业7

软件71 骆炳君 2017013573

16-2

a.

算法：将任务a_i按执行时间p_i从小到大排序，然后按顺序执行任务.

```
Sort(S)
average=0
time=0
tasks=[]
for i=1 to n:
    tasks.append(S[i])
    time+=S[i].p
    average+=time
average=average/n
return tasks,average
```

时间复杂度：设排序算法的复杂度为$O(f(n))$，则该算法的时间复杂度为$O(f(n)+n)$，通常情况为$O(n\log n)$.

证明：假设这个算法不能最小化平均完成时间，即在最优任务序列的某一个连续子序列中，执行时间p_i最小的任务a_i不在该子序列的第一位，那么把a_i与当前的第一位互换位置，则原来在a_i前面的任务的完成时间减少，而原来在a_i后面的任务的完成时间不变，平均完成时间减少.这与假设矛盾，证明完毕.

b.

算法：和a.的算法类似，在每一步后都加入局部最优任务（当前已被释放且剩余执行时间最短的任务），直到所有任务被完成.

时间复杂度：寻找局部最优任务的过程要求遍历S，时间复杂度为$O(n)$，而需要寻找局部最优任务的时刻包括当前任务结束和有任务被释放的时刻，其数量为$O(n)$，所以该算法的时间复杂度为$O(n^2)$.

证明：假设这个算法不能最小化平均完成时间，即在最优任务序列的某一个连续子序列中，局部最优任务a不在该序列的第一位，那么把a与当前的第一位互换位置，则原来在a前面的任务的完成时间减少，而原来在a后面的任务的完成时间不变，平均完成时间减少.这与假设矛盾，证明完毕.

16-5

a.

设m为请求元素的种数($m\leq n$)，算法伪代码如下：

```
r[n]为访问请求,k为缓存规模
d[n]用于记录请求到r[i]时应该逐出缓存的元素,llist为辅助双向链表,t[m]为全为null的辅助数组
for i=1 to n:
    if t[r[i]] != null:
        llist.push_back(r[i])
        t[r[i]]=&llist.back
for i=n to 1:
    ret[i]=llist.back
    llist.move_to_front(t[r[i]])
return ret
```

算法的时间复杂度为$O(n)$.

b.

设$c[i]$为访问到r_i时的待逐出数据，则$c[i]$是r_i后的数据中距离r_i最远的元素，若r_i后存在未出现过的元素，则$c[i]$为未出现的元素，若r_i后所有种类的元素都已经出现过，则$c[i]$的求解依赖于$c[i+1, i+2, \cdots, n]$子问题的求解，即$c[i] = c[j]$, j为i+1,i+2,...,n中满足r_i到r_j间无c[j]出现且j与i距离最近的值，其子问题的求解方式与之相同，这是一个最优子结构.

c.

假设furthest-in-future策略不能保证最小缓存未命中次数，那么就存在一种策略G和i，在G中访问到r_i时选择逐出x($x \neq c[i]$)，且G的缓存未命中次数比furthest-in-future策略少.设r_i后下一个$c[i]$的位置为j，因为$c[i]$是r_i后距离最远的元素，在i到j之间必然有x出现，与原策略相比，G在i到j间必然会多一次缓存未命中x的情形，因此G的未命中次数多于原策略.这与假设矛盾，因此furthest-in-future策略能够保证最小缓存未命中次数.