Homework4 Solution

洪方舟
2016013259
Email: hongfz16@163.com

2018 年 3 月 26 日

Problem 16-2

a.

算法设计：

将所有任务按照执行时间从小到大排序，就按照这个顺序执行，得到的平均运行结束时间是最小的

正确性证明：

首先，观察到该问题有最优子结构，如果我们选择第一个执行的任务的时候就选择最优解，在选择第二个以及之后的执行任务顺序时，按照同样的方式选择执行任务的顺序，那么将会得到最优解。下面说明如果每次选择任务的时候贪心选择耗时最短的那一个，将会达到最优。设 a 是当前待选择任务中耗时最短的那一个，b 为剩下任务中任意一个，设解法 A 将 a 排在第一个，而解法 B 将 a 和 b 的执行顺序对调，对于 A 中 b 之后执行的任务运行完成时间将没有变化，但是对于 a 和 b 两个任务之间的所有任务 S，由于在 B 中首先需要运行时间较长的 b，所以 S 中所有任务的平均完成时间在解法 B 中将会长于解法 A，因此在每次选择下一个要执行的任务时，最优的方案就是贪心的选择耗时最短的任务。因此该算法具有正确性。

时间复杂度分析：

只需要对所有任务按照执行时间从小到大排序即可，因此时间复杂度为 $O(nlgn)$

b.

算法设计：

使用最短剩余时间调度算法，伪代码如下

```
function SRTF(S)
    let tc=1, pc=MAX_NUM ,idc=-1
    let result[] be an empty array
    while not S.empty()
        for i in S.size()
            if tc >= S[i].r and pc > S[i].p
                if pc != MAX_NUM and pc != 0
                    for j in range(0,S.size()-1)
```

```
                    if S[j].p <= pc and pc <= S[j+1].p
                        S.insert(s(r=0,p=pc),i)
                        break
                pc=S.delete(i).p
                idc=i
                break
        tc+=1
        pc-=1
        result.append(idc)
    return result
```

正确性证明：

不难发现，该算法实际的完成顺序为 $S.r + S.p$ 的升序，如果重新构造一组任务 S_n，每个任务的执行时间对应为 $(S[i].r + S[i].p)$，并且使用上一问的无抢占的执行规则，则该算法产生的平均执行时间等于 S_n 采用上一问的最优解计算得到的平均执行时间，则利用上一问的结论，可知在本小题有抢占的执行规则下，最短剩余时间调度算法能够实现平均执行时间最优。

时间复杂度分析：

如果令 T 为完成所有任务所需的时间总长度，则外层 $while$ 需要循环 T 次，内层 for 循环耗时 $O(n)$，因此总的时间复杂度为 $O(nT)$

Problem 16-5

a.

使用 $FarthestinFuture$ 算法，伪代码如下

```
function FFSchedule(R,k)
    let cache[k] be hashtable initalized with random ri
    let schedule[n] be new array
    for i in range(0,n)
        if R[i] in cache
            schedule[i]=READ_FROM_CACHE
        else
            for j=n to i+1
                if R[j] in cache
                    schedule[i]=EVICT(R[j])
    return schedule
```

哈希表操作时间为 $O(1)$，一共两重循环各 $O(n)$，因此总的时间复杂度为 $O(n^2)$

b.

定义 R 为请求序列，S_{ij} 为对请求序列中从 i 到 j 的请求进行的操作序列，T_{ij} 表示 S_{ij} 中 $CacheMiss$ 的次数。考虑对 R_{in} 的规划，若 S_{in} 为最优的调度算法：若此时 $R[i]$ 在 $Cache$ 中，那么此时 $T_{in} = T_{(i+1)n}$，也即

此时 S_{in} 中必然包含 $S_{(i+1)n}$ 的最优子问题；若此时 $R[i]$ 不在 $Cache$ 中，那么就产生一次 $CacheMiss$，则此时 $T_{in}=1+T_{(i+1)n}$，也即此时 S_{in} 包含 $S_{(i+1)n}$ 的最优子问题。综上，该问题具有最优子结构。

c.

假设 S_{FF} 为按照 $FarthestinFuture$ 规则规划的序列，S^* 为具有最少 $CacheMiss$ 的序列；下面证明，可以通过不增加 $CacheMiss$ 数的一个过程将 S^* 转化为 S_{FF}；为了证明这个命题，下面证明一个该命题的递推版本：假设 S 为与 S_{FF} 前 j 个操作相同的序列，那么存在 S' 使得它与 S_{FF} 的前 $j+1$ 个操作相同，但是 $CacheMiss$ 数不多于 S。

设 $d=d_{j+1}$，如果 d 存在于 S 和 S_{FF} 的 $Cache$ 中，或者 d 不存在与两者的 $Cache$ 中，但是 S 和 S_{FF} 在第 $j+1$ 步弹出了相同的元素，那么此时 S,S',S_{FF} 在前 $j+1$ 个操作中均相同。

如果 d 不在 S 的 $Cache$ 中，并且 S 弹出 f，S_{FF} 弹出 $e\neq f$，下面就要找出一个 S'，使得在第 $k>j$ 次操作之后拥有和 S 第 k 次操作之后相同的 $Cache$，在此之后只需要采取和 S 一致的操作就可以实现 k 次操作之后的操作中 $CacheMiss$ 数和 S 相同，下面只需要说明在 j 到 k 之间做的一系列构造，使得 S' 在第 $j+1$ 次操作和 S_{FF} 一样，并且 $CacheMiss$ 数不多于 S。

考虑 $d'=d_{j+2}$；若 $d'\neq e,f$，且 S 此时弹出 e，那么此时让 S' 弹出 f，则此时 S' 和 S 有相同的 $Cache$；若 S 弹出 $h\neq e$，那么让 S' 也弹出 h，重复本步骤继续往下找；若 $d'=f$，且 S 弹出 e，那么此时 S' 无需做任何操作已经达到相同 $Cache$；若 S 弹出 $e'\neq e$，那么让 S' 同样也弹出 e'，那么也可以达到相同 $Cache$。

综上，已经证明了“假设 S 为与 S_{FF} 前 j 个操作相同的序列，那么存在 S' 使得它与 S_{FF} 的前 $j+1$ 个操作相同，但是 $CacheMiss$ 数不多于 S”的命题，那么只需要递归的构造下去，总可以将 S^* 转化为 S_{FF}，并且不增加 $CacheMiss$，那么就可以说明 $FarthestinFuture$ 方法是最优的。