Greedy Algorithm

韦俊林 (201928016029023)

2020 年 1 月 8 日

1 Job Schedule

1.1 Problem Description

There are n distinct jobs, labeled $J_1, J_2, \cdots, J_n$, which can be performed completely independently of one another. Each job consists of two stages: first it needs to be *preprocessed* on the supercomputer, and then it needs to be $finished$ on one of the PCs. Let's say that job J_i needs p_i seconds of time on the supercomputer, followed by f_i seconds of time on a PC. Since there are at least n PCs available on the premises, the finishing of the jobs can be performed on PCs at the same time. However, the supercomputer can only work on a single job a time without any interruption. For every job, as soon as the preprocessing is done on the supercomputer, it can be handed off to a PC for finishing.

Let's say that a *schedule* is an ordering of the jobs for the supercomputer, and the *completiontime* of the schedule is the earlist time at which all jobs have finished processing on the PCs. Give a polynomial-time algorithm that finds a schedule with as small a completion time as possible.

1.2 Natural Language Description

先对输入的作业进行排序，按照需要在 PC 上花费的时间排序，这里进行升序排序。接着按花费在 PC 上的时间从大到小调度任务，完成所有任务所花的时间及所求。

1.3 Greedy Choice Property

任务工作在 *supercomputer* 上是串行执行，而在 PC 上是并行执行，串行上的工作时间是固定的。因此，越先执行在 PC 花费时间越长的作业，花费在 PC 上的时间与花费在 *supercomputer* 上的时间重叠度越高，则花费的总时间就越少。

1.4 pseudo-code

algorithm 1.

1.5 Prove of Correctness

设作业 i 在 *supercomputer* 与 PC 上运行耗费的时间分别为 $J_i.supercomputer_{time}$、$J_i.PC_{time}$。反证法：

algorithm 1 Job Schedule

INPUT: Each job of time cost,number of jobs

OUTPUT: The minimum of time

```
1: function JobSchedule(jobs[],N)
2:     Ascending sort the jobs by working time on PC.
3:     time ← 0
4:     temp ← 0
5:     for i = N − 1 to 0 do
6:         temp ← temp + jobs[i].supercomputer_time
7:         time ← MAX(time, temp + jobs[i].PC_time)
8:     end for
9:     return time
10: end function
```

假设有作业 i, j 并且有 $J_i.PC_{time} \leqslant J_j.PC_{time}$，先运行作业 i，花费的总时间最少。
此时花费的总时间：

$$TIME_1 = J_i.supercomputer_{time} + J_j.supercomputer_{time} + MAX(J_i.PC_{time} - J_j.supercomputer_{time}, J_j.PC_{time})$$

$$TIME_1 = J_i.supercomputer_{time} + J_j.supercomputer_{time} + J_j.PC_{time}$$

而假如先运行作业 j，此时花费的总时间：

$$TIME_2 = J_j.supercomputer_{time} + J_i.supercomputer_{time} + MAX(J_j.PC_{time} - J_i.supercomputer_{time}, J_i.PC_{time})$$

因为 $J_i.PC_{time} \leqslant J_j.PC_{time}$，所以

$$TIME_1 - TIME_2 = J_j.PC_{time} - MAX(J_j.PC_{time} - J_i.supercomputer_{time}, J_i.PC_{time}) \geqslant 0$$

假设与事实矛盾，所以算法正确。

1.6 Analyse of Complexity

该算法需要对输入数据进行预处理，即排序，这里使用快速排序时间复杂度为 $O(nlogn)$。排序之后只需遍历一次整个输入数据即可，时间复杂度为 $O(n)$。

因此，该算法时间复杂度为 $O(nlogn)$，其中 n 为工作数。

2 CrossTheRiver

2.1 Problem Description

Some people want to cross a river by boat. Each person has a weight, and each boat can carry a maximum weight of limit. Each boat carries at most 2 people at the same time, provided the sum of the weight of those people is at most boat’s weight limit. Return the minimum number of boats to carry every given person. Note that it is guaranteed each person can be carried by a boat.

2.2 Natural Language Description

在当前还未上船的人群中取体重最大以及最小的人进行配对。如果体重最大者上船后，该船剩余的空间还能装下体重最小者，则该体重最小者也上此上船。如果体重最大者上船后，该船剩余空间装不体重最小者，则该体重最大者单独乘这艘船，与他配对的体重最小者再重新与剩余还未上船的人群中体重最大者配对。按上述方法循环直至所有人上船。

因此，在算法执行之前需要对输入数据进行预处理，即对人群按体重升序或降序排序，这里我们按升序处理。

2.3 Greedy Choice Property

船数最小意味着尽可能的将船装满，而该问题中船有两个限制，分别时人数以及承重。因此，先后从两个维度做贪心选择。即先将体重最大的放入船上后，接着尽可能的将此船装够两个人。

2.4 Optimal Substructure

假设人体重（$p_1, p_2, ..., p_n$）按升序排序，$OPT(p_i, ..., p_j)$ 表示使用最少的船数将 $p_i, ..., p_j$ 人群搭载上船，L 表示船的承重量。

$$OPT(p_i, p_{i+1}, ..., p_{j-1}, p_j) = \begin{cases} OPT(p_{i+1}, p_{i+2}, ..., p_{j-1}) + 1 & p_i + p_j \leqslant L \\ OPT(p_i, p_{i+1}, ..., p_{j-1}) + 1 & p_i + p_j > L \end{cases}$$

2.5 pseudo-code

algorithm 2.

2.6 Prove of Correctness

使用最少的船数意味着尽量使每艘船能装上两个人。直观的贪心是尽可能地将一条船用完其承重能力，即一开始 $p_i + p_j \leqslant L$ 尽可能的接近 L。

假设分别有四个人（a, b, c, d）, 每艘船承重为 L，并且 $p_a \leqslant p_b \leqslant p_c \leqslant p_d$，其中 $p_c + p_d \leqslant L$。则按照直观的贪心选择有 $[p_a, p_b]$，$[p_c, p_d]$ 分别搭乘两艘船。

按照本算法，这四个人则以 $[p_a, p_d]$，$[p_b, p_c]$ 方式搭乘。

由 $p_b \leqslant p_d$，有 $p_b + p_c \leqslant p_c + p_d \leqslant L$；由 $p_a \leqslant p_c$，有 $p_a + p_d \leqslant p_c + p_d \leqslant L$。

因此，$[p_a, p_b]$，$[p_c, p_d]$ 与 $[p_a, p_d]$，$[p_b, p_c]$ 分配方式船数一样，即算法正确。

2.7 Analyse of Complexity

该算法需要对输入数据进行预处理，即排序，这里使用快速排序时间复杂度为 $O(nlogn)$。排序之后只需遍历一次整个数组即可，时间复杂度为 $O(n)$。

因此，该算法时间复杂度为 $O(nlogn)$，其中 n 为总人数。

algorithm 2 CrossTheRiver

INPUT: Number of people, weight limit and people's weights

OUTPUT: The minimum number of boats

```
1: function CrossTheRiver(N,L,WEIGHTS[])
2:     i ← 0
3:     j ← N − 1
4:     count ← 0
5:     QuickSort(WEIGHTS)
6:     while i <= j do
7:         if i = j then
8:             BREAK
9:         end if
10:        if WEIGHT[i] + WEIGHT[j] ⩽ L then
11:            i ← i + 1
12:            j ← j − 1
13:        else
14:            j ← j − 1
15:        end if
16:        count ← count + 1
17:    end while
18:    return count
19: end function
```