Assignment 03

肖阳 201918008629001

1

(a) Firstly, sort the positions of bananas and the positions of monkeys. It's not hard to find out that assigning the monkey with the maximum index to the banana with the maximum index must be involved in the best solution of minimize the time. Assume T(i) is the time it takes to assign the monkey at index i to the banana at index i , and OPT(n) is the minimized time it takes to assign the n monkeys to n bananas which are not assigned. So we get $\mathrm{OPT(N)} = \max(\mathrm{OPT(N-1)}, \mathrm{T(N,N)})$. Further, every monkey can be assign to the banana which is at the same index, and the minimum amount of time is the maximum $\mathrm{T(i,i)}(1 \le \mathrm{i} \le \mathrm{N})$.

算法 1 分配香蕉给猴子所需最短时间

输入: *monkeys*数组,*bananas*数组

输出: *t*最小分配时间

```
1: function AssignBananasToMonkeys(monkeys, bananas)
2:     sort monkeys and bananas
3:     t ← 0
4:     n ← monkeys.size()
5:     for i = 1 → n do
6:         if t < abs(monkeys[i] − bananas[i]) then
7:             t ← abs(monkeys[i] − bananas[i])
8:         end if
9:     end for
10:    return t
11: end function
```

(b) Assume two array bananas and monkeys are sorted.

Greedy-choice property: We only need to assign every monkey to the banana which has the same index as the monkey.

Optimal structure: $\mathrm{OPT(N)} = \max(\mathrm{OPT(N-1)}, \mathrm{T(N,N)})$

T(N,N) represent the time assigning monkey N to banana N cost.

(c) Prove the correctness

Assume that OPT(N)=T(j,j) has a better value OPT'(N)=T(j,k) , where $1 \le \mathrm{k} < \mathrm{j} \le \mathrm{N}$, that means the monkey with index j is assigned to the banana with index k. There are two situations for the solution OPT'(N) :①T(j,k)<T(j,j), there must exist a monkey $\mathrm{t}(1 \le \mathrm{t} < \mathrm{j})$, so that T(t,j)>T(j,j)>T(j,k), contradiction occur! ②T(j,k)>T(j,j), that means banana k is not the best choice to monkey j, contradiction occur! In sum, the correctness is proved!

(d) Time complexity: O(nlogn)

Space complexity: O(1)

2

(a) Given that the supercomputer can only work on serial mode and the n PCs can be regarded as parallel operation, the best choice of each decision is to let PC to process the task i with the longest processing time f_i on PC.

算法 1 工作分配安排

输入: p每个任务在超级计算机上处理所需时间,f每个任务在PC上处理所需时间

输出: $schedule$工作安排顺序数组

```
1: function JobSchedule(p, f)
2:     n ← NumberOfJob
3:     for i = 1 → n do
4:         schedule[i].p ← p[i]
5:         schedule[i].f ← f[i]
6:     end for
7:     sort schedule according to f descendingly
8:     return schedule
9: end function
```

算法 1 最小完成时间计算

输入: $schedule$工作安排顺序队列

输出: $CompletionTime$ 最早完成时间

```
1: function MinimumCompletionTime(schedule)
2:     n ← NumberOfJob
3:     CompletionTime ← 0
4:     SupercomputerEndingTime ← 0
5:     for i = 1 → n do
6:         SupercomputerEndingTime+ = schedule[i].p
7:         if SupercomputerEndingTime + schedule[i].f > CompletionTime then
8:             CompletionTime = SupercomputerEndingTime + schedule[i].f
9:         end if
10:    end for
11:    return CompletionTime
12: end function
```

(b) Assume the job queue is sorted according to PC processing time descendingly.

Greedy-choice property: We only need to choose the job from schedule queue with the longest PC processing time on each decision.

Optimal structure:

$$\text{OPT(k~N)} = \max\begin{cases} p_{top} + f_{top} \\ p_{top} + OPT((k+1)\sim N) \end{cases}$$

p_{top} denotes the supercomputer processing time of the top job in schedule queue.

f_{top} denotes the PC processing time of the top job in schedule queue.

OPT(k~N) denotes the minimum completion time of the job set$\{k, \dots, N\}, 1 \le k \le N$. and I use OPT(1~N) to represent the question.

Every job will be poped from the schedule queue after being processed by the supercomputer.

(c) Prove the correctness

Assume we only have 2 job .First Job J_1 has$\{p_1, f_1\}$, Second job J_2 has $\{p_2, f_2\}$. Let $f_1 > f_2$. If we start from J_2, and the completion time will be $p_1 + p_2 + f_1$. If we start from J_1 and the completion time will be $\max\begin{cases} p_1 + f_1 \\ p_1 + p_2 + f_2 \end{cases}$. Obviously, starting from J_1 can complete the job earlier.

(d) Time complexity: O(nlogn)

Space complexity: O(1)