Algorithm Design and Analysis Assignment 3
Greedy

Xinmiao Zhang
202028013229129

December 3, 2020

1 Monkey and Bananas

There are N Monkeys and N bananas are placed in a straight line.

Each monkey want to have a banana, if two monkeys want to own the same banana, there will be a fight! A monkey can stay at his position, move one step right from x to $x+1$, or move one step left from x to $x-1$. Any of these moves consumes 1 second. Assign monkeys to banana so that no monkey fight each other and the time when the last monkey gets a banana is minimized.

1.1 Algorithm Description

1.1.1 Natural Language

Suppose that the position of monkeys on a straight line is $M = \{m_1, m_2, .., m_N\}$. Similarly, the position of bananas is $B = \{b_1, b_2, .., b_N\}$.

Notice that it is hard to find an optimal substructure when M and B are unsorted. Hence, first of all we sort M and B by *ascending* order. And we can obtain $M' = m'_1, m'_2, .., m'_N$ and $B' = b'_1, b'_2, .., b'_N$, where we have:

$$\begin{cases} m'_1 \leq m'_2 \leq .. \leq m'_N \\ b'_1 \leq b'_2 \leq .. \leq b'_N \end{cases}$$

The assigning principle is following: assign the banana at b'_i to the monkey at m'_i,for $1 \leq i \leq N$.

1.1.2 Pseudo Code

Let the array of monkeys' and banana's position be M and B respectively, and as the input to following algorithm. And it will return the minimized time when the last monkey gets a banana.

ASSIGN-BANANAS-TO-MONKEY(M, B)

```
1  M ← sort(M)
2  B ← sort(B) //Try to get the optimal structure
3  T_min = ∞
4  for i = 1 to B.length
5        if |M[i] − B[i]| < T_min
6              T_min = |M[i] − B[i]|
7  return T_min
```

1.2 Greedy Selection Rule and Optimal Substructure

It is clear that in original problem, where the array is not sorted, building optimal substructure is difficult. However, it is not hard to find this kind of structure when we transform the problem by simply sorting.

The greedy selection rule in this problem is to assign the kth banana in a straight line to the kth monkey.

We establish $OPT(k)$ as the optimal solution to the subproblem when utilizing $M[1\,..\,k]$ and $B[1\,..\,k]$ as input. Then we can find the following optimal substructure:

$$OPT(k) = \begin{cases} |m_1 - b_1| & k = 1 \\ \max(OPT(k-1), |m_k - b_k|) & k > 1 \end{cases}$$

1.3 Correctness

To prove the correctness, we need to prove that in this greedy algorithm, every single step can lead to the final optimal solution. That is, to prove the optimal substructure's correctness.

1. When $k = 1$, then $OPT(1) = |m_1 - b_1|$ apparently.
2. When $k > 1$, we prove that $OPT(k) = \max(OPT(k-1), |m_k - b_k|)$.

Now assume that $OPT(k) \neq OPT(k-1)$ or $|m_k - b_k|$. Then there must be monkey m_i and banana $b_j (1 \leq i, j \leq k-1)$, s.t. assigning m_k to banana b_j and m_i to banana b_k will lead to optimal solution, notice that $m_i < m_k$ and $b_j < b_k$. Then we know according to assumption above, the optimal solution can be $|m_k - b_j|$ or $|m_i - b_k|$.

We will prove that this will lead to a conflict.

1. If there is only one monkey-fetching reach the optimal solution, without losing generality, suppose it is $|m_k - b_j|$. That is to say:

$$\begin{cases} |m_k - b_j| < |m_k - b_k| \\ |m_i - b_k| > |m_k - b_k| \\ \max\limits_{s,t} |m_s - b_t| = |m_k - b_j| \end{cases}$$

Do some basic transform in the first inequal relationship, we have $m_k < \frac{b_j + b_k}{2}$.Therefore we have $m_i < m_k < b_k$. Then evaluate $|m_i - b_k|$:

$$\begin{aligned} |m_i - b_k| &= b_k - m_i \\ &> b_k - m_k \\ &= |b_k - m_k| \end{aligned}$$

It is conflict with the condition(3) where $\max\limits_{s,t} |m_s - b_t| = |m_k - b_j|$.

2. If there are two monkey-fetching reach the optimal solution, that is to say:

$$\begin{cases} |m_k - b_j| < |m_k - b_k| \\ |m_i - b_k| < |m_k - b_k| \end{cases}$$

Do some basic transform in both inequal relationship, it is easy to find the following facts:

$$\begin{cases} m_k < \frac{b_i + b_k}{2} \\ b_k < \frac{m_j + m_k}{2} \end{cases}$$

Add the inequality together, we will find

$$m_k + b_k < m_i + b_j$$

Apparently, it conflicts with $m_k > m_i$ and $b_k > b_j$. So the assumption is false.

In other words, we prove the optimal substructure in this problem along with the algorithm.

1.4 Complexity

As for sorting, using classic sorting algorithm can consume $O(N \log N)$ time.

As for process of finding solution, it is a single loop with iteration N, so the time complexity is $O(N)$.

So the total time complexity is $O(N \log N)$.

The space complexity is $O(N)$, we only need arrays to store M and B.

2 Job Schedule

There are n distinct jobs, labeled $J_1, J_2, .., J_n$, 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 completion time 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.

2.1 Algorithm Description

2.1.1 Natural Language

Suppose the array of last time on supercomputer for all jobs $J = \{J_1, .., J_n\}$ is $P = \{p_1, .., p_n\}$, the array of last time on personal computer for all jobs J is $F = \{f_1, .., f_n\}$.

We sort $J = \{J_1, .., J_n\}$ according to F in descending order, and obtain $J' = \{J'_1, J'_2 .., J'_n\}$.

The solution to the problem is utilizing this order above to schedule jobs.

2.1.2 Pseudo Code

We have array of jobs J. Let the array of last time on supercomputer and personal computer for all jobs be P and F respectively, and as the input to following algorithm. And it will return the minimized time under best scheduling scheme.

JOB-SCHEDULE(J, P, F)

```
1  J ← sort(J) according to F
2  P ← sort(P) according to F
3  F ← sort(F)
4  prev = 0, max = 0
5  for i = 1 to J.length
6        prev = prev + P[i]
7        if F[i] + prev > max
8              max = F[i] + prev
9  return max
```

2.2 Greedy Selection Rule and Optimal Substructure

In the original problem, the optimal substructure and greedy rule is unclear. However when we apply sorting algorithm, the optimal substructure appears.

The greedy selection rule is that in every single step, we choose the job whose running time on PC is longest preferentially.

Considering the choice in step k. Let $OPT(k)$ be the minimized time by choosing different schedule scheme, then we have the following optimal substructure:

$$OPT(k) = \begin{cases} p_1 + f_1 & k = 1 \\ \max(OPT(k-1), f_k + \sum_{i=1}^{k} p_i) & k > 1 \end{cases}$$

2.3 Correctness

For a job schedule order $S = (s_0, s_1, .., s_n)$. The kth job's finish time is $f_{s_k} + \sum_{i=0}^{k} p_{s_i}$. Then the optimal problem can be transferred to the following objective:

$$\min_{S}\{\max_{k}\{f_{s_k} + \sum_{i=0}^{k} p_{s_i}\}\}$$

Then we will prove the optimal structure:

1. When $n = 1$, then $OPT(1) = p_1 + f_1$ apparently.
2. When $n > 1$, we prove that $OPT(n) = \max(OPT(n-1), f_n + \sum_{i=1}^{n} p_i)$.

Suppose the order sorted by F is $S = \{s_0, s_1, .., s_n\}$. If our optimal substructure condition is not satisfied, that is to say, when adding nth item to job queue, we have:

$$\min_{S'}\{\max_{k}\{f_{s'_k} + \sum_{i=0}^{k} p_{s'_i}\}\} < \max(OPT(k-1), f_k + \sum_{i=1}^{k} p_i)$$

First of all, we will demonstrate that the left end can not be achieved when $k < n$. If $k < n$, there must have:$s_n = s'_r (r < k)$, otherwise it must less than or equal to $OPT(k-1)$. When J_{s_n} is before the value, the $f_{s'_k} + \sum_{i=0}^{k} p_{s'_i} > OPT(k-1) + p_{s_n} \geq OPT(k)$ for$k < n$. So we consider about the situation when $k = n$:

If $f_{s'_k} + \sum_{i=0}^{k} p_{s'_i} > f_{s'_n} + \sum_{i=0}^{n} p_{s'_n}$, we can obtain there exists one job sequence $S' = \{s'_0, s'_1, .., s'_n\}$, *s.t.*

$$f_{s'_n} + \sum_{i=1}^{n} p_{s'_i} < f_{s_n} + \sum_{i=1}^{n} p_{s_i}$$

Suppose that $s'_k = s_n$, we know that $f_{s'_n} > f_{s'_k} = fs_n$.
And we know that $\sum_{i=0}^{n} p_{s_i} = \sum_{i=0}^{n} p_{s'_i}$
So we have:

$$f_{s'_n} + \sum_{i=1}^{n} p_{s'_i} > f_{s_n} + \sum_{i=1}^{n} p_{s_i}$$

It is conflict!
Then we prove the correctness of our algorithm.

2.4 Complexity

Their is only three sort, whose time complexity is $O(nlogn)$.
The space complexity is $O(n)$ for 3 arrays.

3 Cross the River

Some people want to cross a river by boat. Each person has a weight, and each boat can carry an equal maximum weight 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.

3.1 Algorithm Description

3.1.1 Natural Language

Suppose that the people's weight can be expressed in one array $W = \{w_1, w_2, .., w_n\}$, and one single boat's load limit is M. First of all, we sort W in ascending order. Consider about the heaviest person, if he can stay in the same boat with the thinnest person, then they will be arranged a common boat. Else the heaviest person will be arranged a boat alone.

3.1.2 Pseudo Code

We have array of people's weight $W = \{w_1, w_2, .., w_n\}$ and one boat's load limit M as input respectively. And it will return the minimized boat needed.

```
CROSS-THE-RIVER(W, M)
1  W ← sort(W)
2  num = 0, i, j = 0
3  while i ≠ j
4       if W[i] + W[j] < M
5            j − −
6       num + +
7       i + +
8  return num
```

3.2 Greedy Selection Rule and Optimal Substructure

For the sorted weight array, the greedy selection rule is as follows: for the heaviest person, if he can stay in the same boat with the thinnest person, then load them together; else he has to stay in the boat alone. The optimal substructure is as follows:

$$OPT(i,j) = \begin{cases} 0 & i = j \\ OPT(i+1,j)+1 & w_0 + w_n \geq M \\ OPT(i+1,j-1)+1 & w_0 + w_n < M \end{cases}$$

3.3 Correctness

If the upper optimal substructure is false, then there must be one selection, although the heaviest person on bank can go with the thinnest person, he doesn't come boat with the thinnest one.

1. If he come to the boat alone, then it become $OPT(i+1,j)+1$. Notice that it is trivial that $OPT(i+1,j) \geq OPT(i+1,j-1)$. So this can not lead to best solution.

2. Else, the heaviest person P_1 come with another person P_a, the thinnest person is P_t, and P_a's original partner is P_b. Then it is easy to find that: $w_t + w_b < M$.Then we must arrange P_t and P_b two into one boat, else we will increase one boat.

So the algorithm is correct!

3.4 Complexity

The time complexity is sorting as $O(n \log n)$.

The space complexity is $O(n)$.