Lab03-GreedyStrategy

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

∗ If there is any problem, please contact TA Shuodian Yu.
∗ Name:Yulong Hui Student ID:518030910059 Email: qinchuanhuiyulong@sjtu.edu.cn

1. There are $n+1$ people, each with two attributes $(a_i, b_i), i \in [0, n]$ and $a_i > 1$. The i-th person can get money worth $c_i = \frac{\prod_{j=0}^{i-1} a_j}{b_i}$. We do not want anyone to get too much. Thus, please design a strategy to sort people from 1 to n, such that the maximum earned money $c_{max} = \max_{1 \le i \le n} c_i$ is minimized. (Note: the 0-th person doesn't enroll in the sorting process, but a_0 always works for each c_i.)

   (a) Please design an algorithm based on greedy strategy to solve the above problem. (Write a pseudocode)

   (b) Prove your algorithm is optimal.

Solution.

(a)The pseudocode is as follow, and it is based on quicksort. The sequence is sorted nondecreasingly with the product of a and b.

□

Algorithm 1: Sort

Input: An array $Person[1, \cdots, n]$
Output: $Person[1, \cdots, n]$ sorted nondecreasingly with $Person.a * Person.b$

```
1  if n ≤ 1 then
2      return;
3  pivot ← A[n]; i ← 1;
4  for j ← 1 to n − 1 do
5      if Person[j].a ∗ Person[j].b < pivot.a ∗ pivot.b then
6          swap Person[i] and Person[j];
7          i ← i + 1;
8  swap Person[i] and Person[n];
9  if i > 1 then Sort(Person[1, ··· , i − 1]);
10 if i < n then Sort(Person[i + 1, ··· , n]);
```

(b) Firstly, in a sequence, we can say there is a inversion pair. if there is a number i satsifying:

$$Person[i].a * Person[i].b > Person[i+1].a * Person[i+1].b$$

If we need to prove the correctness of the greedy algorithm, we just need to prove that swappping a inversion pair will not make the final result worse.

Then we just use a_i and b_i instead of $Person[i].a$ and $Person[i].b$. We suppose the oringinal case is: $a_i * b_i > a_{i+1} * b_{i+1}$, which is a inversion pair. Obviously, the swapping will not affect other values, so we just pay attention to c_i and c_{i+1}.

Assume $c_i = \frac{A}{b_i}$, then $c_{i+1} = \frac{A * a_i}{b_{i+1}}$

Because: $a_i * b_i > a_{i+1} * b_{i+1} > b_{i+1}$, we can get: c_{i+1} is bigger.

Then we do the swapping, and let $d_i = \frac{A}{b_{i+1}}$, then $d_{i+1} = \frac{A * a_{i+1}}{b_i}$

Then, both d_i and d_{i+1} are smaller than c_{i+1}, because $a_i * b_i > a_{i+1} * b_{i+1}$.

So, after the swapping, the maximum of the two is smaller than the old one.

Then the algorithm is proved.

2. **Interval Scheduling** is a classic problem solved by greedy algorithm and we have introduced it in the lecture: given n jobs and the j-th job starts at s_j and finishes at f_j. Two jobs are compatible if they do not overlap. The goal is to find maximum subset of mutually compatible jobs. Tim wants to solve it by sort the jobs in descending order of s_j. Is this attempt correct? Prove the correctness of such idea, or else provide a counter-example.

   Proof. This idea is correct, we can prove that by contradiction.

   First we use the greedy algorithm to get a subset. Then we sort its elements (jobs) in chronological order, which will not change the number of elements but make our following analyze clearly.

   Assume that the greedy algorithm is not the optimal algorithm, and the last k jobs (k is the largest possible value) in the greedy subset are the same as these in the optimal subset.(Both subset is ordered chronological)

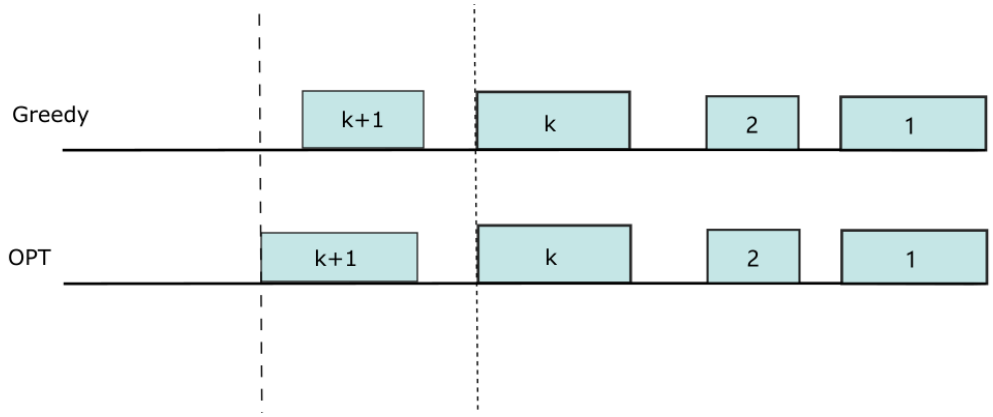

Figure 1: Job Sequence

   As you can see in the Figure1, the elements in job always have a bigger start time. Therefore, we can change the $k+1$ job in OPT to the $k+1$ one in Greedy sequence without affect other earlier jobs.

   So after the change, the solution will still be feasible and optimal.However it contradicts the largest attribute of k, so the assumption is not right. The greedy algorithm is optimal.

   □

3. There are n lectures numbered from 1 to n. Lecture i has duration (course length) t_i and will close on d_i-th day. That is, you could take lecture i **continuously** for t_i days and must finish before or on the d_i-th day. The goal is to find the maximal number of courses that can be taken. (Note: you will start learning at the 1-st day.)

   Please design an algorithm based on greedy strategy to solve it. You could use the data structrue learned on Data Structrue course. You need to write pseudo code and prove its correctness.

Solution.
I will explain the line-5 in the following algorithm:

This "compatiable" means that : $t_j + t_{all} < d_j$ (t_{all} is the total time to finish all lectures in the que).

Algorithm 2: Greedy Schedule

Input: An array $Lec[1, \cdots, n]$
Output: The maximal number of the lectures

```
1  Sort Lectures by deadline so that d_1 < d_2 < d_3 < ··· < d_n;
2  Create a PriorityQueue called que, which is sorted by duration(Lec[i].t) ;
3  que← ∅;
4  for j ← 1 to n do
5      if Lec[j] is compatiable with que then
6          Push Lec[j] into the que;
7      else
8          if Lec[j].t is smaller than the biggest one in que then
9              Pop the old biggest one out and push Lec[j] in
10 Return the number of elements in que ;
```

Proof. To prove the algorithm, we first need to clear two points: (1) if there are two lectures with (t_i, d_i) , (t_j, d_j) and $d_i < d_j$, then the lecture-j should be arranged after the lecture-i.

Then,prove it:

Let the already arranged lectures have a t_{all}. Then if we arrange lecture-i after lecture-j, it must satisfy:

$$t_{all} + t_j \leq d_j \text{ and } t_{all} + t_j + t_i \leq d_i \tag{1}$$

while another arrangement needs:

$$t_{all} + t_i \leq d_i \text{ and } t_{all} + t_i + t_j \leq d_j \tag{2}$$

Because $d_i < d_j$, if (1) is true, then (2) must be true. Otherwise it doesn't hold.

That's to say the second arrangement can permit not less lectures.

(2) If the next lecture is incompatible, we replace the one of biggest t in the old arrangement with the new lecure of smaller t (line7–line9 in algorithm). This opertion will not make the matter worse.

It is because that after the exchange, the number (before the larger d) will not lower, and the total time of this lectures are reduced, which will provide the later lecture with more space.

Then we can prove the algorithm by induction.

Basis. If there is only one lecture in arrangement, the greedy one is optimal.

Assumption. If there have been k lectures in arrangement, the greedy one is optimal.

Induction. Now we want to solve k+1 lectures.

Because of the point(1), the next lecture to consider will have smallest d, which is optimal.

If the next one is compatible with the old sequence, just add it into the consequence, then the optimal number of will be k+1 .

If the next one is incompatiable then we just keep doing the replacement (as line7–line9 in algorithm) until the compatiable one appears or the lecures are traversed. According to the point(3), it will still remain optimal.

Therefore, the algorithm is proved optimal.

□

□

4. Let $S_1, S_2, \ldots, S_n$ be a partition of S and $k_1, k_2, \ldots, k_n$ be positive integers. Let $\mathcal{I} = \{I : I \subseteq S, |I \cap S_i| \leq k_i \text{ for all } 1 \leq i \leq n\}$. Prove that $\mathcal{M} = (S, \mathcal{I})$ is a matroid.

Proof.

• **Hereditary** Assume $I_n \in \mathcal{I}$ and $I_m \subseteq I_n$.

Then $|I_n \cap S_i| \leq k_i$ for all $1 \leq i \leq n$, and I_m just has some of elements in I_n.

Therefore, $|I_m \cap S_i| \leq k_i$ for all $1 \leq i \leq n$, which means $I_m \in \mathcal{I}$.

Then $(S, \mathcal{I})$ is an independent system.

• **Exchange Property**

Consider $A, B \in \mathcal{I}$, and $|A| < |B|$. Then there must be a partiton of A, B:

$$A_1, A_2, \cdots, A_n \ (A_j = A \cap S_j)$$

$$B_1, B_2, \cdots, B_n \ (B_j = B \cap S_j)$$

Because $|A| < |B|$, which means $|A_1| + \cdots + |A_n| < |B_1| + \cdots + |B_n|$, there must exists a number i satisfying $|A_i| < |B_i| \leq k_i$.

Then we choose $x \in B_i \backslash A_i$, and $|A_i \cup x| = |A_i| + 1 \leq k_i$.

Since other part don't have any change, $A \cup x \in \mathcal{I}$. And the matroid is proved.

□

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.