Lab03-GreedyStrategy

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

∗ If there is any problem, please contact TA Shuodian Yu.

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 pseudo code is as follows (Alg. 1)

Algorithm 1: Minimize the maximum earned money

Input: Two arrays of attributes $a[0, \cdots, n], b[0, \cdots, n]$
Output: The minimized maximum earned money c_{max}

```
Sort people from 1 to n by a[i] · b[i] in non-decreasing order.
prod ← a[0]; c_max ← 0;
for i ← 1 to n do
    c[i] ← prod/b[i];
    if c[i] > c_max then
        c_max ← c[i];
    prod ← prod · a[i];
return c_max;
```

(b) First of all, if we exchange two adjacent people in a sequence, it is obvious that the earned money of other $n-2$ people will not change.

Definition. Given a sorting result S, an adjacent inversion is a pair of people i and j such that: $i < j$ but $a_i \cdot b_i > a_j \cdot b_j$.

We have to prove our greedy strategy S is optimal, and define S^* to be an optimal result. If S^* has no inversion, then $S = S^*$. If S has an inversion, let $i - i + 1$ be an adjacent inversion.

For these two adjacent people i and $i+1$ in S^*, and let $p = \prod_{j=0}^{i-1} a_j$, the maximum earned money between them is $\max(\frac{p}{b_i}, \frac{p \cdot a_i}{b_{i+1}})$. After exchange, the value will come to $\max(\frac{p}{b_{i+1}}, \frac{p \cdot a_{i+1}}{b_i})$.

As $a_i, a_{i+1} > 1$, we have $\frac{p \cdot a_i}{b_{i+1}} > \frac{p}{b_{i+1}}, \frac{p \cdot a_{i+1}}{b_i} > \frac{p}{b_i}$. Moreover, while this is an inversion and $a_i \cdot b_i > a_{i+1} \cdot b_{i+1}$, we have $\frac{p \cdot a_i}{b_{i+1}} > \frac{p \cdot a_{i+1}}{b_i}$.

So we can know that $\max(\frac{p}{b_i}, \frac{p \cdot a_i}{b_{i+1}}) > \max(\frac{p}{b_{i+1}}, \frac{p \cdot a_{i+1}}{b_i})$, which means if we swap such inversion $i - i + 1$, the result c_{max} will not increase. This contradicts the definition of S^*. Thus, the greedy strategy is already optimal.

□

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 greedy strategy is optimal. We prove it by contradiction.

   Assume greedy is not optimal. Let $\{i_1, i_2, ..., i_k\}$ denote the set of jobs by such strategy.

   Let $\{j_1, j_2, ..., j_m\}$ denote the set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \cdots, i_r = j_r$ for the largest possible value r. And $m \geq k$ as it is the optimal solution.

   From the greedy strategy we know that the job i_{r+1} starts after j_{r+1}, which means the job i_{r+1} remains more available time for the further jobs. So the greedy solution is also feasible and optimal. This contradicts with the maximality of r.

   □

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. The pseudo code is as follows (Alg. 2)

Algorithm 2: Maximize the number of taken courses

Input: Lecture duration $t[1, 2, \cdots, n]$, close day $d[1, 2, \cdots, n]$
Output: The maximum number of courses that can be taken

```
1  Sort courses (t[i], d[i]) by d[i] in non-descending order;
2  h ← ∅ //an be a max-heap
3  time ← 0;
4  for i ← 1 to n do
5      if time + t[i] ≤ d[i] then
6          time ← time + t_i;
7          h.push(t[i]);
8      else if h ≠ ∅ and h.top() > t[i] then
9          time ← time − h.top() + t[i];
10         h.pop();
11         h.push(t[i]);
12 return size(h);
```

First, we can find that for two course (t_1, d_1) and (t_2, d_2) satisfy $d_1 \leq d_2$, it is always be optimal to learn (t_1, d_1) first, because we have $x + t_1 + t_2 \leq d_2$ and $x + t_1 \leq d_1$, where x is the time that start learning. If we learn the second course first it may break the requirement that $x + t_1 + t_2 \leq d_1$.

Then we use induction to prove our greedy strategy optimal.

For $n = 1$, obviously it is optimal.

For $n = k + 1$, assume we have taken m in the previous k courses which is optimal, and for all $1 \leq i \leq m$, $\sum_{j=1}^{i} t_j \leq d_i$. Then we have to decide whether to take the $(k+1)$-th course.

Case 1: If $\sum_{j=1}^{m} t_j + t_{k+1} \leq d_{k+1}$, then take this course is also feasible. And these $m + 1$ courses is still optimal.

Case 2: We cannot take the $(k+1)$-th course with m courses simultaneously. In this case, if from the previous m taken courses, there is a course j that $t_j > t_{k+1}$. When we replace course j with $n + 1$, it is still feasible as the total duration will be reduced. Moreover, we have sort all the courses by d_i and $d_j \leq d_{k+1}$, the deadline of $(k+1)$-th course also meet the requirement. So taking course $n + 1$ instead of j is optimal.

Therefore, the correctness of such greedy strategy is proved. □

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. We prove the two requirements: hereditary and exchange property as follows:

Hereditary:

For $A \subset B$ and $B \in \mathcal{I}$, then $A \subseteq S$ and $\forall 1 \leq i \leq n, |A \cap S_i| \leq |B \cap S_i| \leq k_i$. Therefore $A \in \mathcal{I}$ as well.

Exchange:

For $A, B \in \mathcal{I}$ and $|A| < |B|$, because $\sum_{i=1}^{n} |A \cap S_i| = |A| < |B| = \sum_{i=1}^{n} |B \cap S_j|$, there must exist an index j that $|A \cap S_j| < |B \cap S_j| \leq k_j$. It is easy to be proved by contradiction.

Thus, $\exists x \in (B \cap S_j) \backslash (A \cap S_j)$, that $|(A \cup \{x\}) \cap S_j| \leq k_j - 1 + 1 = k_i$. Therefore, $A \cup \{x\} \in \mathcal{I}$. □

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