Lab03-GreedyStrategy

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

∗ If there is any problem, please contact TA Shuodian Yu.
∗ Name: Futao Wei Student ID: 518021910750 Email: weifutao@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 $c_{max} = \max\limits_{1 \le i \le n} c_i$ 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)

Algorithm 1: Sort

Input: $P[0, \cdots, n]$ each with two attributes $(a_i, b_i), i \in [0, n]$ and $a_i > 1$

Output: $P[0, \cdots, n]$ sorted such that $c_{max} = \max\limits_{1 \le i \le n} c_i = \max\limits_{1 \le i \le n} \frac{\prod_{j=0}^{i-1} a_j}{b_i}$ is minimized.

```
pivot ← a_n * b_n; i ← 1;
for j ← 1 to n − 1 do
    if a_j * b_j < pivot then
        swap P[i] and P[j];
        i ← i + 1;
swap P[i] and P[n];
if i > 1 then Sort(P[1, ··· , i − 1]);
if i < n then Sort(P[i + 1, ··· , n]);
```

(b) First, let's prove that sorting two adjacent people by $a_i * b_i$ non-decreasingly will not result in a larger $c_{\max}$. Denote them as i, j $(j = i + 1)$.

Case 1: If neither of i, j earns the maximum money, the sorting won't have an effect on $c_{\max}$.

Case 2: If one of i, j earns the maximum money, but $a_i * b_i \le a_j * b_j$, the sorting won't make a difference.

Case 3: If one of i, j earns the maximum money, and $a_i * b_i > a_j * b_j$, let's denote $p = \prod\limits_{k=0}^{i-1} a_k$. Then the maximum earned money before sorting

$$c_{\max} = \max\{\frac{p}{b_i}, \frac{p \cdot a_i}{b_j}\}$$

after sorting

$$c'_{\max} = \max\{\frac{p}{b_j}, \frac{p \cdot a_j}{b_i}\}$$

Since $a_i * b_i > a_j * b_j$, we have $\frac{p \cdot a_i}{b_j} > \frac{p \cdot a_j}{b_i}$. And $\frac{p \cdot a_i}{b_j} > \frac{p}{b_j}$, since $a_i > 1$. Therefore

$$c_{\max} > c'_{\max}$$

i.e., the sorting results in a smaller amount of maximum earned money.

So it has been proved that sorting two adjacent people by $a_i * b_i$ non-decreasingly will not result in a larger $c_{\max}$.
Then we can prove my sorting is optimal by contradiction. If my sorting does not get an optimal solution, we can always convert an optimal solution to the result obtained by my sorting algorithm by sorting two adjacent people repeatedly, without loss of optimality. Thus my sorting algorithm is 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. Correct.
   Assume Tim's method is not optimal.
   Let $i_1, i_2, \cdots, i_k$ denote the set of jobs selected by Tim's method.
   Note that there may be several different optimal solutions. Among all the optimal solutions, there must be one which has the most continuous jobs from back to front the same as Tim's method. Let $j_1, j_2, \cdots, j_m$ denote the set of jobs selected by this specific optimal solution, with $i_k = j_m, i_{k-1} = j_{m-1}, \cdots, i_{k-(r-1)} = j_{m-(r-1)}$ for the largest possible value of r.
   Let's consider i_{k-r} and j_{k-r}. According to Tim's method, i_{k-r} starts after or at the same time when j_{k-r} starts. Hence if we replace j_{k-r} in the optimal solution with i_{k-r}, it'll become an optimal solution with one more job identical to Tim's method, which contradicts the aforementioned maximality.
   Therefore, Tim's method 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. Denote the n courses as an array $C[1, 2, \cdots, n]$, each element of which has two attributes t_i and d_i.

Algorithm 2: Schedule

Input: $C[1, 2, \cdots, n]$

Output: *count* = the maximum number of courses that can be taken

```
1  Sort courses by close times so that d_1 ≤ d_2 ≤ ··· ≤ d_n;
2  heap ← ∅; // a data structure
3  count ← 0; time ← 0;
4  for i ← 1 to n do
5      heap.push(C[i]); // function push(element) inserts element to the heap
6      time ← time + t_i;
7      if time ≤ d_i then
8          count ← count + 1;
9      else
10         time ← time − heap.top().t; // function top() returns the element with
             the maximum t_i in the heap
11         heap.pop(); // function pop() removes the element with the maximum t_i
             from the heap
12 return count;
```

Let's prove algorithm 2's optimality by induction.

For $n = 1$, apparently it's optimal.

Assume optimality holds for $n = k$, i.e., for the first k courses with $d_1 \leq d_2 \leq \cdots \leq d_k$. Denote the courses selected as $C[i_1], C[i_2], \cdots, C[i_m]$. The total time spent till now is $t_{i_1} + t_{i_2} + \cdots + t_{i_m}$

For the $(k+1)$-th course:

Case 1: If $t_{i_1} + t_{i_2} + \cdots + t_{i_m} + t_{k+1} \leq d_{k+1}$, the $(k+1)$-th course will be selected according to algorithm 2. Let's prove by contradiction that our selection $C[i_1], C[i_2], \cdots, C[i_m], C[k+1]$ is optimal among the first $k+1$ courses with $d_1 \leq d_2 \leq \cdots \leq d_k \leq d_{k+1}$.

Suppose that there exists a better solution π, which gets $\geq m+2$ courses from $C[1, 2, \cdots, k+1]$ or $m+1$ courses in $< t_{i_1} + t_{i_2} + \cdots + t_{i_m} + t_{k+1}$ time. If π selects $C[k+1]$, we can simply remove $C[k+1]$ and get a solution which gets $\geq m+1$ courses from $C[1, 2, \cdots, k]$ or m courses in $< t_{i_1} + t_{i_2} + \cdots + t_{i_m}$ time. That contradicts our induction hypothesis that optimality holds for $n = k$. If π does not select $C[k+1]$, our induction hypothesis guarantees that π gets at best m courses from $C[1, 2, \cdots, k]$ in $t_{i_1} + t_{i_2} + \cdots + t_{i_m}$ time.

So there's no such better solution, which contradicts our supposition.

Case 2: If $t_{i_1} + t_{i_2} + \cdots + t_{i_m} + t_{k+1} > d_{k+1}$, the course with the largest duration among $C[i_1], C[i_2], \cdots, C[i_m], C[k+1]$ will be removed after adding $C[k+1]$, according to algorithm 2.

This measure will achieve optimality for the first $k+1$ courses with $d_1 \leq d_2 \leq \cdots \leq d_k \leq d_{k+1}$, which can be proved by contradiction in a similar way to case 1.

Therefore, algorithm 2 is optimal for the first $k+1$ courses.

The correctness of algorithm 2 is thus 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. First, let's prove the hereditary property. If $A \subset B$ and $B \in \mathcal{I}$, meaning $B \subseteq S, |B \cap S_i| \leq k_i$ for all $1 \leq i \leq n$, then $A \in \mathcal{I}$.

Next we'll prove the exchange property by contradiction. Assume $A, B \in \mathcal{I}$ and $|A| < |B|$, but $\forall x \in B \setminus A$, $A \cup \{x\} \notin \mathcal{I}$, that is, violating $|(A \cup \{x\}) \cap S_i| \leq k_i$ for all $1 \leq i \leq n$. In

other words,

$$\forall x \in B \setminus A,\ \exists i_0 \text{ such that } |(A \cup \{x\}) \cap S_{i_0}| > k_{i_0}$$

and obviously $x \in S_{i_0}$. Hence

$$|A \cap S_{i_0}| = k_{i_0} \geq |B \cap S_{i_0}|$$

which leads to the contradiction that $|A| \geq |B|$. Exchange property is proved.
Therefore, we can conclude that $\mathcal{M} = (S, \mathcal{I})$ is a matroid. □

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