CS 141, Fall 2017 Homework 5

Posted: October 30th, 2017 Due: November 6th, 2017

Name:Zhenxiao Qi

Student ID #:500654348

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted

Problem 1. (25 points)

Consider the Activity Selection problem we discussed in class. Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start (as long as it is compatible with all the previously selected activities). In other words, instead of considering the activities by "earliest finish", we consider them by "latest start". Explain why this approach is a greedy algorithm, analyze its time complexity, and prove that it yields an optimal solution (greedy choice and optimal substructure).

Answer:

Firstly, we sort the activities by its start time. the first one is the latest to start and so forth. Then we choose the activity in order.

According to the definition of greedy method, the greedy choice is whatever choice that seems to be the best at this moment and then solve the sub-problem after the first choice is made.

Our greedy choice is last to start. We sort the activities and every time we choose the last one to start in the current sub-problem. So this approach is a greedy algorithm.

The time complexity is $O(n \log n)$ due to the sort process.

Proof of greedy choice property:

Suppose that $A \subseteq S$ is an optimal solution for S.

Then we order the activities in A by start time(last start is the first)

Let K be the first one in A

If K is also the first one in S, the optimal solution begins with our greedy choice

Else, 1_{th} always start later than K, which means we can always replace K with 1_{th} to construct a better solution.

In conclusion, the optimal solution begins we our greedy choice

Proof of optimal substructure(by contradiction)

Suppose that $A = A' \cup \{1\}$ is not an optimal solution for S, then there is a solution $B = B' \cup \{1\}$ is optimal. Then B' should be a solution for S', but B' cannot be the optimal solution because the start time of activities in A' are always later than that of activities in B'. So $A = A' \cup \{1\}$ is an optimal solution for S

Overall, this algorithm yields an optimal solution.

Problem 2. (25 points)

A server has n customer waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. So if, for example, the customers are served in order of increasing i, then the i-th customer has to wait $\sum_{j=1}^{i} t_j$ minutes. We want to minimize the total waiting time:

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)$$

Give a greedy (efficient) algorithm for computing the optimal order in which to process the customers. Prove why your algorithm is correct (i.e., always returns an optimal solution).

Answer:
Our greedy choice is the least waiting time, which means every time we make the customer wait for as least as possible.
We can do this by sort the service time, and serve the customers in order.
So the time complexity is $O(n \log n)$

Proof of optimal substructure(by contradiction)
Suppose that $A = A' \cup \{1\}$ is not the optimal solution for S, then there is a solution $B = B' \cup \{1\}$ is optimal. however, for the sub-problem except $\{1\}$, the total waiting time of A' is less than B', So $A = A' \cup \{1\}$ is an optimal solution for S.
In conclusion, the algorithm always returns an optimal solution.

Problem 3. (25 points)

Assume that you are given two arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of n real numbers.

1. Describe a greedy algorithm to determine an ordering of the elements of A and B such that $W = \sum_{i=1}^{n} |a_i - b_i|$ is minimized
2. Analyze the time complexity of your algorithm
3. State and prove the greedy-choice property of your algorithm
4. State and prove the optimal substructure property of your algorithm

Hint: consider using the following fact. Given real numbers $x_1 \leq x_2$ and $y_1 \leq y_2$, then

$$|x_1 - y_1| + |x_2 - y_2| \leq |x_1 - y_2| + |x_2 - y_1| .$$

Answer:

1. Every time we can choose the a_i and b_i which makes $a_i - b_i$ is the smallest among the other choices. So we can sort these two arrays and choose the elements in order.

2. The time complexity is $O(n \log n)$ due to the sort process.

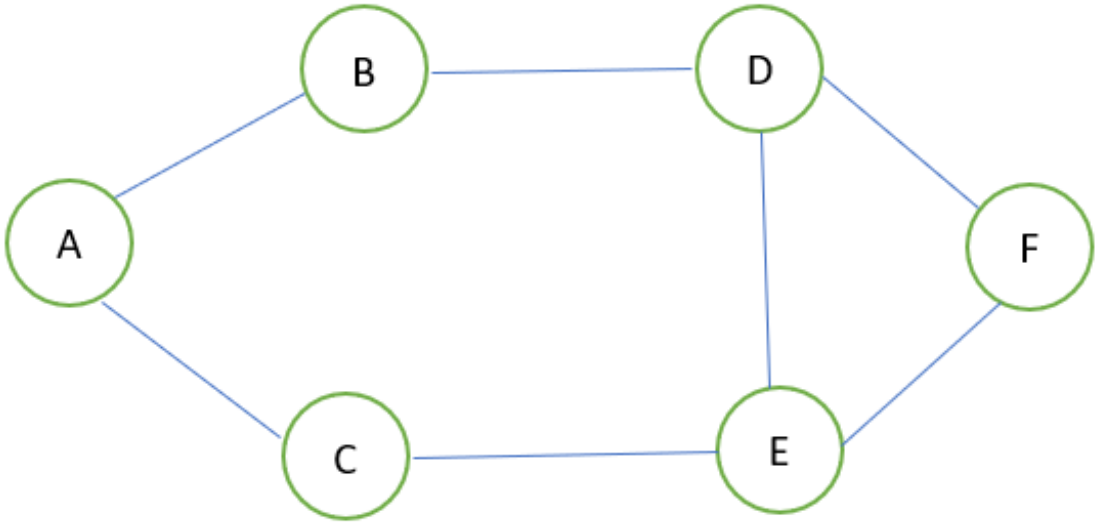
3. Suppose that $A \subseteq S$ is an optimal choice for S.
Then we order the pairs by the difference between them.
Let K be the pair which has the smallest difference.
If K is the our greedy choice, then the optimal begins with our greedy choice.
Else, our greedy choice can always replace K because our greedy choice has the smaller difference than does K.

4. Suppose that $A = A' \cup \{1\}$ is not an optimal solution for S, then there is a solution $B = B' \cup \{1\}$ which is optimal. However, according to our premise, the W of A' is smaller than that of B'. So A should be the optimal solution for S.

Problem 4. (25 points)

Given an undirected graph $G = (V, E)$, an *independent set* in G is any set $I \subseteq V$ of vertices such that no two vertices in I are connected by an edge. In the maximum independent set problem (MIS), for a given graph G, we want to find an independent set of maximum size. Here is our proposed greedy algorithm: (1) Set $I \leftarrow \emptyset$; (2) Repeat (3-4) until no nodes are left; (3) Choose a vertex v in G of minimum degree (breaking ties arbitrarily). (4) Add v to I and remove from G vertex v and all its neighbors. Does this greedy algorithm always return the optimal solution? If you think it does, give a proof for the greedy choice property. If you think it does not, give a counterexample in which it fails.

Answer:
Apparently, this greedy algorithm does not always return a optimal solution.
A counterexample can be found as follow:



According to the greedy algorithm, we can choose A first, for the degree of A is 2, then we eliminate B and C. Next we can choose F and then eliminate D and E, in which case we get A and F finally as an independent set. However, (A,F) is apparently not a maximum independent set.