EL-GY 9343 Homework 10

Yihao Wang
yw7486@nyu.edu

1. A string is a palindrome if it is the same read left to right or right to left, e.g. ababa. The longest palindrome subsequence of a string x is its longest subsequence (of not necessarily consecutive symbols) that is a palindrome. We want an algorithm to determine in time O($|x|^2$) the length of the longest palindrome sub-sequence of a string x.

   (a) A cheap solution is, let y be the reverse of x, and apply the standard Longest Common Subsequence (LCS) algorithm to x and y. This approach is numerically correct, but flawed, since the character locations for a LCS in x and y does not have to reference the same sequence of physical characters. Suppose x is ZABZA, what are the LCS for x and its reverse? How many are they? Are they all palindrome?
   The LCS for x(ZABZA) and its reverse(AZBAZ) are AZA, ABA, ZBA, ABZ, ZBZ, ZAZ. There are 6 LCS in total, among which ZBA, ABZ are not palindrome.

   (b) A better way is to design a new DP algorithm. Let $l(i, j)$ be the length of longest palindrome in $x[i, \ldots, j]$, please write the recurrence equations, including the initial conditions. Please analyze the time and space complexity.
   To specify, if the given string is of length n, then i, j are integers within $[1, n]$.

$$l[i,j] = \begin{cases} 0 & \text{if } i > j \\ 1 & \text{if } i = j \\ l[i+1, j-1] + 2 & \text{if } i < j \end{cases}$$

   The space complexity is $O(n^2)$ because we need a 2-D matrix of shape $n \times n$ to store $l[i, j]$. The time complexity is $O(n^2)$ because it takes $O(1)$ to fill in each entry of the matrix.

2. (**Maximum Independent Set in Trees**[1]) In an undirected graph $G = (V, E)$, an *independent set* S is a subset of the vertex set V that contains no edge inside it, i.e. S is an *independent set* on $G \Leftrightarrow S \subseteq V, \forall u, v \in S \rightarrow \{u, v\} \notin E$.
Given a rooted tree $T(V, E)$ with root node r, find an independent set of the maximum size. Briefly describe why your algorithm is correct.

1. Run BFS/DFS on the tree $T(V, E)$ from the root node r, during which mark the distance from the root node.
2. Gather all vertices (or nodes) into two groups, depending on whether the distance from root node is odd or even.
3. Check the number of vertices in each group, which yields $|V_{odd}|$ and $|V_{even}|$
4. Return $\max(|V_{odd}|, |V_{even}|)$

Since the given graph is a tree, all nodes are naturally divided into "layers" according to the distance from root node. And all edges only exist between adjacent layers. There are no direct connection between layers that are not adjacent. Therefore, layers with odd distance and layers with even distance are two possible largest independent set, among which we can tell which one is truly maximum by computing the sizes.

3. You are given a set of n intervals on a line: $(a_1, b_1], ..., (a_n, b_n]$. Design a greedy algorithm to select minimum number of intervals whose union is the same as the union of all intervals. Please also analyze the time complexity, and prove the correctness by showing the two properties.

Algorithm:

1. Sort all intervals in increasing order by a_i.
2. Set s to be the minimal value of a_i and initialize $T = 0$ as the total number of intervals in the final solution set.
3. Try to find an interval $(a_i, b_i]$ such that $a_i \leq s < b_i$:
   - If such an interval exists, set $T \leftarrow T + 1$ and $s \leftarrow b_i$;
   - If such an interval does not exist, set s to the minimal a_i that is greater than s.
4. Repeat step 3, until all intervals have been looked up.

Time complexity:

It takes $O(n \log n)$ to sort all intervals and $O(n)$ to traverse all intervals (since each interval will and only will be checked once), thus the total time complexity is $O(n \log n)$.

Correctness:

- Greedy-choice property: the algorithm tries to cover as much range as possible with a minimal number of intervals.
- Optimal substructure property: after each iteration, the algorithm maintain the invariant that the current solution set covers all possible points on the left side of s.

[1]Finding maximum-sized independent sets in general graphs is NP-complete. So we will focus on tree cases.

4. Suppose you have an unrestricted supply of pennies, nickels, dimes, and quarters. You wish to give your friend n cents using a minimum number of coins. You need to:

(a) Describe a greedy strategy to solve this problem, and analyze its time complexity;

1. Find out the coin with largest value v that is less than currently remained cents n_{rem}.
2. Add $m = n_{rem}/v$ coins with value v to the solution set, and update $n_{rem} \leftarrow \mod (n_{rem}, v)$
3. Repeat steps 1 - 2 until $n_{rem} = 0$.

Apparently, the time complexity is $O(n)$.

(b) Prove the correctness of your algorithm.

Greedy-choice property: the algorithm tries to cover as many cents as possible with a minimal number of coins.

Optimal substructure property: after each iteration, the algorithm maintain the invariant that the current solution set is of value $n - n_{rem}$.