EL9343 Homework 10

Due: Nov. 30th 11:00 a.m.

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?

   (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.

   Solution:

   (a) x is ZABZA, and its reverse y is AZBAZ. There are in total **six** LCS: ZAZ, ZBZ, ZBA, ABZ, ABA, AZA. In the six LCS, four of them are palindrome, while the other two are not.
   Note: Following this observation, the length 3 returned by the LCS algorithm may come from any of these six answers. Even though length 3 is also the length of longest palindrome sub-sequence, actually it may not come from a palindrome one.
   The underlying problem is, we can easily know that the longest palindrome sub-sequence of x is going to a common sequence of x and its reverse, but why there doesn't exist a longer common sequence when comparing x to its reverse? If we want to use the easy solution, we have to prove this for correctness.
   We will do the proof by contradiction, as follows.
   Suppose the longest palindrome sub-sequence of x has length t, and there is a string z with a length $T > t$, being the common sub-sequence of x and its reverse.
   Since z is a sub-sequence of x, let p be the index array of z in x, i.e. for each $i \in \{1, \ldots, T\}$, p_i is the position of the character z_i in x. From the construction, we know p_i is monotonically increasing. Similarly, let q be the index array of z in the reverse of x, and q_i is monotonically decreasing.
   For simplicity let's add 0 and ∞ to both ends, and we have

   $$p_0 = 0 < p_1 < p_2 < \cdots < p_T < p_{T+1} = \infty$$
   $$q_0 = \infty > q_1 > q_2 > \cdots > q_T > q_{T+1} = 0$$

   There must be a index $k \in \{0, 1, \ldots, T\}$, where $p_k \leq q_k$ and $p_{k+1} \geq q_{k+1}$. Consider the two index arrays $\{p_0, \ldots, p_k, q_k, \ldots, q_0\}$ and $\{q_{T+1}, \ldots, q_{k+1}, q_{k+1}, \ldots, q_{T+1}\}$. Both of these two index arrays are monotonically increasing, so they can generate sub-sequences from the string x, and they are palindrome. The two palindrome sub-sequences have length $2k$ and $2T - 2k$ respectively, and one of the two values would be larger than t, making a contradiction to t being the longest palindrome length.

   (b) With $l(i, j)$ being the length of longest palindrome in $x[i, \ldots, j]$, the recursive is,

   $$l(i,j) = \begin{cases} 0, & \text{if } j < i, \\ 1, & \text{if } i = j, \\ 2 + l(i+1, j-1), & \text{if } x[i] = x[j], \\ \max\{l(i+1, j), l(i, j-1)\}, & \text{otherwise.} \end{cases}$$

   There are in total $O(n^2)$ sub-problems. The space complexity is $O(n^2)$.
   To solve each sub-problem, given that all the sub-problems of smaller size have been solved, to check conditions and do the calculation are constant time. With $O(n^2)$ sub-problems to solve, the total time complexity is $O(n^2)$.

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.
Solution:
Define:

$$IS(T) = \text{size of maximum independent set in } T$$
$$IS_{with\ root}(T) = \text{size of maximum independent set in } T \text{ that includes the root}$$
$$IS_{without\ root}(T) = \text{size of maximum independent set in } T \text{ that excludes the root}$$

So the maximum independent set has size $IS(T)$, where T is the given rooted tree. If the root is included in the set, then all the sub-trees must excluded their roots (otherwise not an independent set). If the root is excluded, then each sub-tree can include or exclude its root, whichever has the larger size. As the result, the recursive formula of the problem is,

$$IS_{without\ root}(T) = \sum_{T' \text{ is sub-tree of } T} IS(T')$$
$$IS_{with\ root}(T) = 1 + \sum_{T' \text{ is sub-tree of } T} IS_{without\ root}(T')$$
$$IS(T) = \max\{IS_{with\ root}(T), IS_{without\ root}(T)\}$$

3. You are given a set of n intervals on a line: $(a_1, b_1], \ldots, (a_n, b_n]$. 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.
Solution:
   (a) Algorithm:
   First, sort the intervals in increasing order by a_i. Our algorithm will essentially keep track of a right-most frontier s which we try to grow greedily while ensuring that we always cover everything possible to the left of s. Set s to initially be the minimal value of a_i. Then repeat the following:
   Find the interval with maximal b_i satisfying $a_i \leq s < b_i$.
      i. If such an interval $(a_i, b_i]$ exists, then add it to the solution set and update $s = b_i$.
      ii. If such an interval does not exist, then set s to be the smallest value of a_i which is greater than s.

   (b) Running time:
   $O(n \log n)$ for sorting. Each interval is only looked at once after the sorting, and hence the total running time is $O(n \log n)$.

   (c) Correctness:
   Optimal substructure property:
   Consider each update of the frontier s, each subset of the optimal solution with intervals whose right endpoints smaller or equal to s, is still the optimal selection of intervals that cover the union of all intervals before s.
   Greedy choice property:
   Assume contradiction, let S be the set of intervals in our solution of size l, and let some other set I of intervals of size $k < l$ be optimal. Sort both S and I in increasing order by the interval start points. Let S_j , I_j denote the union of the first j in intervals in S, I respectively (according to the above sorted order). Then we will show that $I_j \subseteq S_j$ by induction.
   This is true for $j = 1$ since we pick the interval that starts at the left most point, and if there are multiple such intervals, we pick the one with maximum b_i value.
   Assume it is true for j. Let s_j be the value of s in our algorithm after the first j intervals were added (If the j-th interval added was $(a_j, b_j]$, then $s_j \geq b_j$). Let t_j be the largest point contained in I_j so that $t_j \leq b_j \leq s_j$ by the inductive hypothesis.

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

Let the j-th interval in S and I be the intervals $(a_j, b_j]$ and $(a'_j, b'_j]$ respectively. $(a_{j+1}, b_{j+1}]$ is chosen based on the maximal b_{j+1} satisfying $a_{j+1} \leq s_j < b_{j+1}$.
Now, after the $j+1$-th addition, if I_{j+1} should have a point that is not in S_{j+1}, it CANNOT be the case that $a_{j+1} > s_j$, because that implies I_j must be equal to S_j , and therefore the interval $(a'_{j+1}, b'_{j+1}]$ must be a subset of $(a_{j+1}, b_{j+1}]$ because of construction.
In the other case, when $a_{j+1} \leq s_j$, we need $b'_{j+1} > b_{j+1}$ for there to be a point in Ij+1 that is not in S_{j+1}, since we already have $a'_{j+1} \leq s_j$, which is not possible based on the construction.
Therefore $I_{j+1} \subseteq S_{j+1}$. This completes the induction and shows that, if $|I| = k$ then S_k covers I. This contradicts our assumption that $|S| = l > k$ since we never include an interval unless it adds points. Therefore, $S = S_k$ and $|S| = |I| = k$ is optimal.

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.
   (a) Describe a greedy strategy to solve this problem;
   (b) Provide the pseudo-code and write down its time complexity;
   (c) Prove the correctness of your algorithm.

   Solution:

   (a) Use as many quarters as possible, then dimes, then nickels, and finally pennies.

   (b) The pseudo-code is as follows,

GREEDY-CHANGE(n)
$n_1 = n \mod 25$ (mod here means to get the remainder)
$a = (n - n_1)/25$
$n_2 = n \mod 10$
$b = (n_1 - n_2)/10$
$n_3 = n \mod 5$
$c = (n_2 - n_3)/5$
$d = n_3$
return a, b, c, d

The time complexity is $O(1)$.

   (c) Greedy property:
   First observe that no optimal solution can have ≥ 5 pennies (use nickel instead), ≥ 2 nickels (use dime instead), ≥ 3 dimes (use a quarter and a nickel instead).
   Next, we are going to prove the greedy property using a case by case analysis[2].
      i. $1 \leq n < 5$, this is clearly true that we have to use just pennies.
      ii. $5 \leq n < 10$, we pick a nickel in this case, as the alternative is to use ≥ 5 pennies.
      iii. $10 \leq n < 25$, we pick a dime in this case. Otherwise we have to use ≥ 5 pennies or ≥ 2 nickels (or both), none of which can be optimal.
      iv. $n \geq 25$. From the observation, the maximum cents we can generate with our constraints on the number of pennies, nickels and dimes is $(4 * 1 + 1 * 5 + 2 * 10) = 29$. So for any $n > 29$, we have to use quarters.
      Also, for $25 \leq n \leq 29$, notice that we have to use $n - 25$ pennies, and to get 25 cents the optimal way is to pick a quarter.

   Therefore, with any number of n, we can get the optimal solution following greedy choice.
   Optimal substructure property:
   If a coin of value v_i is removed from an optimal solution with number of coins as m, the remaining solution is optimal that uses $m - 1$ coins to give $n - v_i$ cents.

[2]This kind of proof doesn't work in all cases, but I find it the best way in this one. An alternative (and more general) way to prove this is to assume an optimal solution that is different from the one generated by the algorithm, and then to find some contradiction.