Assignment #2

430.329-002 Introduction to Algorithms
Bohyung Han

Due 23:59 10th November, 2020

Note

- T.A. contacts
  - Sanghyeok Chu: sanghyeok.chu@snu.ac.kr
  - DongHwan Jang: jh01120@snu.ac.kr

- If you have any questions about the assignments, please upload them to eTL.

- **You must complete the assignments by yourself.** Otherwise, there will be a severe penalty in your grade.

- Assignments must be completed and submitted by 23:59 as a zip file to the eTL assignment board.

- You should submit one zip file which contains one pdf (for document) and one python(for code) file. **Please write your name and student ID on the top right of the document.**

- Please title your files as mentioned below: **All cases that do not strictly follow the format will be deducted <u>10%</u> from the total score.**

  - [zip] [StudentID]_assignment2.zip
  - {e.g.} 2020-12345_assignment2.zip

  - [code] [StudentID]_assignment2-1.py
  - {e.g.} 2020-12345_assignment2-1.py

  - [document] [StudentID]_assignment2-2.pdf (Document)
  - {e.g.} 2020-12345_assignment2-2.pdf

1 [Code] Edit distance and Palindrome [100 pts]

Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other. Different types of edit distance allow different sets of string operations. In this problem, we are going to design two edit distances with following options and play with palindrome[1] by using those edit distances:

- LCS (Longest Common Subsequence) distance
- Levenshtein distance: the minimum number of single-character edits by insertion, deletion and substitution required to change one word into the other.

Considering the above definitions, please implement 4 functions below. Please write your code in a given file that contains skeleton codes. Any external packages are **NOT** allowed.

1) [30 pts] Design a function **LCS(X, Y)** that returns L, the length of the LCS between two strings.

2) [30 pts] Design a function **LD(X, Y)** that returns L, the Levenshtein distance between two strings X and Y.

3) [20 pts] Design a function **LPS(X)** that returns the longest palindromic subsequence of X.

4) [20 pts] Design a function **P_Min(X)** that returns the minimum number of single-character edits by insertion, deletion and substitution to make a palindrome that could be made from the input string X.

Example)
Input: X = ‘apple’, Y = 'place'

- Longest common sequence of X, Y= ‘ple’
  - LCS(X, Y) = 3
- The minimum number of operations required to transform X -> Y = 4
  - delete(a) -> delete(p) -> pass(p) -> pass(l) -> insert(a) -> insert(c) -> pass(e)
  - LD(X, Y) = 4
- The longest palindromic subsequence of X = ‘pp’, Y=’p’ or ‘l’ or ‘a’ or ‘c’ or ‘e’
  - LPS(X) = 2, LPS(Y) = 1
- The palindromes with the minimum number of single-character edits (insertion, deletaion and substitution) of X = ‘apppa’ or ‘alpla’… and Y = 'plalp' or 'pcacp'…
  - P_Min(X) = 2, P_Min(Y) = 2

[1] Palindrome is a word, number, phrase, or other sequence of characters which reads the same backward as forward, such as madam, racecar, 12321.

2 [Document] Solve the following problems [100 pts]

1. [Tree; 20 pts] Prove or disprove the following statements. Note that the root node has height of 1.
    a. RBT with 5 nodes always has a height of 3.

    b. RBT with 6 nodes always has a height of 3.

2. [Heap; 20 pts] Solve the following questions.
    a. Build a min heap from the following unsorted array. Please follow BUILD-MIN-HEAP algorithm in the lecture slides.

    [8, 6, 10, 16, 12, 15, 7]

    b. After removing the three smallest elements from 2.a., draw the output heap **in an array**.

3. [Hashing; 20 pts] Student X implemented chaining algorithms. However, by mistake, he inserted the collision node to the tail rather than the head of a linked list. That is, when a collision occurs during search, the node inserted the longest ago is searched first instead of the node that was inserted most recently.
   In this case, what is the expected time complexity of successful search? Please solve with a formula.

4. [Dynamic Programming; 20 pts] Construct the dynamic programming table for the following 0-1 knapsack problem where the weight capacity = 6. From the table, derive the optimal itemset.

item	**1**	**2**	**3**	**4**	**5**
w	4	5	3	2	1
b	5	5	4	2	2

5. [Huffman Code; 20 pts] Solve the following questions.

a. Construct Huffman Tree and find the code for each character. You do NOT have to illustrate the process of each step in detail. Please assign 0 to the less frequent sibling to solve 5.b.

(item: frequency) a: 82, c:28, e:130, n:67, o:75, t:91

b. Decode the following binary sequence based on the code that you get in 7.a.

"001001101011100"