Assignment 4

November 20, 2024

The assignment is due on **Saturday, 30 November, 9:00 PM**. Late submissions will be accepted until **Tuesday, 3 December, 9:00 PM** with a penalty of 10%. After that no submissions will be accepted.

1 Written Questions

Question 1: Priority Queues (15 points)

Simulating Priority Queue Using a Stack

Write a function `push_with_priority(stack, priority, element)` that pushes elements onto a stack based on a given priority. Lower priority elements should always appear closer to the top.
Your Answer:

```
def push_with_priority(stack, priority, element):
    # Temporary stack to hold elements until the right position is found
    temp_stack = []

    # 1) Move elements to temp_stack while maintaining priority order

    # 2) Insert the new element with its priority

    # 3) Move the elements back from temp_stack to stack to maintain
        their order

# Example usage
stack = []
push_with_priority(stack, 3, "A")
push_with_priority(stack, 1, "B")
push_with_priority(stack, 2, "C")
print(stack)  # Output: [(3, 'A'), (2, 'C'), (1, 'B')]
```

Question 2: Binary Search Tree Operations (20 points)
In this question, you will perform several operations on a Binary Search Tree (BST).
a) (10 points) Draw the binary search tree after inserting the following 12 elements into an empty BST in the given order

$$[50, 30, 70, 20, 40, 60, 80, 10, 25, 35, 65, 90]$$

b) (5 points) Insert Element: After constructing the BST with the above elements, draw the updated tree after inserting the element 17 into the tree.
c) (5 points) Delete Element: Draw the binary search tree after deleting the element 70 from the tree and describe how the tree restructures itself.

Question 3: Sorting (20 points)

Task 1: [10 points]
Demonstrate the quick-sort algorithm step by step, using **the last element of the current subarray as the pivot**. Given the array:

$$[8, -3, 7, -1, 10, 2, -5]$$

Complete the following table by showing the left partition (L), right partition (G), and the pivot at each step. Perform quicksort recursively on the subarrays and show each intermediate result.

Solution Template:

Step	**Pivot**	**Left Partition**	**Right Partition**
Initial Array	–	–	–
1			
2			
3			
...			
Final Array	–	–	–

Table 1: Solution Template

Notes for Students:

- Use the solution template above to format your answers.
- Use the last element of the current subarray as the pivot for each step.

Task 2: [2 points]
Complete the missing line in the following code fragment to correctly implement merge-sort's merge function.

```
    def merge(S1, S2, S):
        i = j = 0
        while i + j < len(S):
            if j == len(S2) or (i < len(S1) and S1[i] < S2[j]):
                S[i+j] = S1[i]  # copy ith element of S1 as next item of
                    S
                i += 1
            else:
                # Missing line
                j += 1
```

Task 3: [8 points]
You are given a function that performs quick sort using extra storage to partition the array. Your task is to fill in the missing lines in the code to complete the quick sort implementation.

Code:

```
def quick_sort_extra_storage(S, a, b):
    if a >= b:
        return
    pivot = S[b]  # Choose the pivot
    # Create new arrays for the partitions
    left_partition = []
    right_partition = []
    for i in range(a, b):  # Iterate only until b (exclude pivot)
        if S[i] < pivot:
            left_partition.append(S[i])
        else:
            right_partition.append(S[i])

    # Sort left partition
    quick_sort_extra_storage(left_partition, 0, len(left_partition) - 1)
    # Sort right partition
    quick_sort_extra_storage(right_partition, 0, len(right_partition) -
        1)

    # Missing lines here to correctly merge the partitions and pivot back
        into the original array
```

Hint:

After the recursive calls to the left and right partitions, think about how you can reconstruct the original array using the sorted left partition, the pivot, and the sorted right partition. The original array must be updated with these sorted parts.

2 Coding Questions

Question 1: Analyzing Character Frequencies (35 points)
Implement a Python function `analyze_character_frequencies(text)` that takes a string `text` as input and does the following:

Count the frequency of each character in the text and store the character-frequency pairs in a dictionary. The function should process the text as follows:

- Convert the text to lowercase.
- Consider only alphanumeric characters (letters and digits).
- Ignore whitespace and punctuation characters.
- Store the character-frequency pairs in a dictionary where the keys are the characters and the values are their respective frequencies.

Return a list of the characters sorted in descending order based on their frequencies. If multiple characters have the same frequency, sort them in alphabetical order.

Instructions:

- Implement the `analyze_character_frequencies(text)` function according to the given requirements.
- You can assume that the input text contains only alphanumeric characters, whitespace, and punctuation characters.
- The function should handle case-insensitivity when counting character frequencies.
- If the input text is an empty string or contains only whitespace and punctuation characters, the function should return an empty list.

Input:

> Knack for knick-knacks, the keen kangaroo quickly kicked the kooky koala into the knapsack by the kayak on the kooky creek.

Output:

> ['k', 'a', 'e', 'o', 'n', 'c', 't', 'h', 'y', 'i', 'r', 'l', 's', 'b', 'd', 'f', 'g', 'p', 'q', 'u']

Marking Rubric: This question will be marked out of 35 for correctness (pass test cases):

- 35/35: Pass all 5 of the test cases
- 21/35: Pass any 2 (or more) of the test cases
- 7/35: Pass any of the test cases

3 Submission Instructions (10 points)

Please follow these submission instructions carefully. Correctly submitting all files is worth 10 points. In these files, you must replace `ccid` with your own ccid. Your ccid is the first part of your UAlberta email (ccid@ualberta.ca). Do not zip any of these files. Please submit the following files to eclass:

- `ccid_writtenQuestions.pdf` : Your answers to all written questions should be in this one pdf file.
- `ccid_solutionQuestion1.py` : Edit the provided file for question 1. After your solution passes all test cases (which you must test using the driver), rename the file to include your ccid, that is, `ccid_solutionQuestion1.py` and submit it to eclass.