Mini Report (Task 2.1)

Arthur Milner – 21035478

Task 1.1

I spent a lot of time planning out my approach, this included considering many data structures and researching relevant libraries. I decided upon the use of dictionaries for my final solution, one to contain **number: occurrence to keep** and the other to track the current occurrence when looping through the number list, **number: current occurrence**. It is also notable I used the collections module to create a **number: overall occurrences** dictionary and converted it to the **number: occurrence to keep** dictionary. I chose dictionaries as they appeared to be efficient in solving the problem with minimal memory usage, I also used a tuple to store the list of numbers for memory efficiency. I simply loop through the numbers and if **occurrence to keep == current occurrence** I write that number to the solution file for an O(N) solution.

Task 1.2

Task 1.2 uses given functions and the bisect module. I used merge sort for sorting as it gives more consistent performance on larger datasets than what would've been my second option, the quick sort. I also selected merge sort due to its time complexity of O(nlogn) in the worst case, compared to quick sort's worst case of O(n^2). Using a tuple for storing operations required less memory than a list and it doesn't matter if it is immutable. For the search operation I used the given binarySearch, which is very quick at O(logN) time. For insertion I used bisect.insort_right, again this is O(logN) as its operation is very similar to binarySearch. Finally, for deletion I used binarySearch times the number of occurrences to remove, giving O(mlogN) time.

Task 1.3

1.3 uses the multiprocessing Pool class, I felt Pool provided much simpler code and also handled the management of processes effectively. It creates a pool of worker processes, the number of which will be dependent on the number of CPU cores and how many will be needed to perform the task. This makes it both efficient and dynamic. I used the apply_async method from the Pool class as blocking was not needed, the result of each process did not rely on one another. The default number of processes passed to the search function is mp.cpu_count(), user can modify this as desired. I also used the KMPSearch function as it is an efficient pattern matching algorithm.

Task 1.4

I created the weighted graph from the CSV file. To calculate the route and cost I used the return value of the dijkstraSP method with the find path method, I did so because the path in a nodes shortest path tree will be the shortest route. I also converted the program into a tkinter application. The application allows users to get a route from departure and destination and allows for the adding of connections. The app has all relevant error messages as I modified the code of the graph weight class to account for this. Styling is also present, following recommendations for a Design for All/accessible approach from **C. Sik-Lanyi (2012)**.

Pseudocode for all tasks can be seen in the appendix.

Appendix

Figure 1 (Task 1.1 Pseudocode)

1. Get path for the reading of files

2. Read the file number by number into a tuple
3. Create a dictionary of **number: overall occurrences** from the tuple of numbers, where each unique number only appears as a key once
4. Loop through all values of the **number: overall occurrences** dictionary applying this rule: (value//2 + 1), this creates the **number: occurrence to keep** dictionary
5. Copy the **number: occurrence to keep** dictionary and set all values to 1 to create the **number: current occurrence** dictionary
6. Loop through the number tuple, if **occurrence to keep == current occurrence** for the current key/number write it to the solution file, add one to **current occurrence** each time its key value/number occurs
7. End

Figure 2 (Task 1.2 Pseudocode)

1. Get path for reading of files
2. Read all numbers from the number file into a list, each index indicating a number
3. Apply merge sort algorithm to the list
4. Store operations into a tuple from the operations file, format: ((OPERATION, NUMBER), (OPERATION, NUMBER), …)
5. Loop over each operation
6. Get the number from the current operation, store it as currentNum
7. If statement, if operation == 1: search, if operation == 2: insert, if operation == 3: delete
8. If operation == 1
    a. Get index of number using binarySearch
    b. If returned index == None then number not in list
    c. Else number in list
9. If operation == 2
    a. Use bisect.insort_right to insert number, uses binary search then inserts on the right
10. If operation == 3
    a. Get index of currentNum using binarySearch
    b. While currentNum is still in the list of numbers:
        i. Remove number at currentNum index
        ii. Use binarySearch again to check if number still present
        iii. If binarySearch returns None, all instances removed and break
11. Print list after all operations

Figure 3 (Task 1.3 Pseudocode)

1. Get path for reading of files
2. Read text line by line into a list from text file
3. Read names name by name into a list from name file
4. For each name:
    a. Call parallelSearch function
    b. Get appropriate number of chunks of text, dependant on number of processes
    c. Calculate last chunk separately to ensure whole text is passed to processes
    d. Create process for each chunk of text, using KMPSearch which returns occurrences from that chunk of text
    e. Sum up all results from the processes as the occurrences of that pattern
    f. Write name followed by occurrences to the solution text file

Figure 4 (Task 1.4 Pseudocode)

1. Get path for reading CSV file
2. Read CSV file, store edges as a list and places as a set to prevent duplicates
3. Create undirected weighted graph from the places and edges
4. Get user input for departure and destination
5. Apply dijkstra's algorithm to get the shortest path tree for the departure and destination
6. Use findPath to get the route and cost from departure to destination
7. Allow for repeat input from user

Bibliography

- C. Sik-Lanyi. (2012) Choosing Effective Colours for Websites. *Woodhead Publishing Series in Textiles* [online] 22, pp.600-621. [Accessed 22nd April 2023]