David Hatcher

U74842093

Homework 5

1.

a. The data structure I would use for this is a 2D array of size (n+1)x(m+1) where n is the length of the string a, and m is the length of string b.
b. I would iterate through this by starting at the bottom right of the matrix and going from right to left through the rows, then from bottom to top through the columns.
    i. For bottom row to top row as currentRow:
        1. For last column in currentRow to first column in currentRow:
            a. do stuff
        2. end
    ii. end
c. a is a string of length n
b is a string of length m
iterativeSC(a,b,n,m):
scArr = array[n+1,m+1]
for i = m+1, to 1, step -1:
    for j = n + 1, to 1, step -1:
        if a[j] is empty string or b[i] is empty string:
            scArr[i,j] = 1
        else if a[j] == b[i]:
            scArr[i,j] = 2 * a[i+1,j+1]
        else
            scArr[i,j] = scArr[i+1,j] + scArr[i,j+1] – scArr[i+1,j+1]
    end
end
return scArr[1,1]
d. The space complexity could be reduced by only keeping track of the previous row and the current row, as the values that are needed to build out the current value are either one row down, one column to the right, or one row down and one column to the right.

2.

a. Input: workshops, a list of n workshops
Input: start, a list of start times for workshops
Input: end, a list of end times for workshops
Input: n, number of workshops
GreedyWorkshopRooms
roomCount = 0
Sort workshops, start, and end by start time
unaddedWorkshops = set of sets {workshops,start,end} initialized with all workshops
addedWorkshops = set of sets {workshops,start,end} initialized as empty
while unaddedWorkshops is not empty do:

if addedWorkshops is empty:
    Remove unaddedWorkshops[1] and place it in addedWorkshops
    increment roomCount
if the unaddedWorkshops[1] does not overlap with all workshop in addedWorkshops:
    Remove unaddedWorkshops[1] to addedWorkshops
else:
    Remove unaddedWorkshops[1] to addedWorkshops
    increment roomCount
end
end
return roomCount

b. I think 1D array and tuple would be a useful data structure for this algorithm. I would use an array of sets that stores (workshop, start time, end time). Then use this to store the workshops that have already been checked and to compare the ones that have not been checked. And a basic integer value to store the count of the number of required rooms

3. A graph that models the WRM problem could be designed as a weighted graph, so that each of the vertices store the name of the workshop and the start time. For the previous example it could be designed as such: