CS 310 'Algorithms' **Assignment-2**
Due date: Monday, October 29, 2018 at 11:55 PM. [Total Marks = 105]

Greedy Algorithms

IMPORTANT: Read the instructions at the end of this document and follow the naming conventions.
Marks for writing well-structured and efficient code. **[5 marks]**

Q1. **[6 marks]**
Consider the following problem discussed in class. You are given two boxes, one of the boxes has 'n' red-colored sticks of lengths Lr_1, Lr_2, Lr_3 … Lr_n and the other box has 'n' blue-colored sticks of lengths Lb_1, Lb_2, Lb_3 … Lb_n. You have to make 'n' pairs of the red and blue sticks. Suppose a pair contains a red stick of length Lr_x and a blue stick of length Lb_y. The difference between the lengths of the two paired sticks is $| Lr_x - Lb_y |$. Your goal is to minimize the average of length-differences of all the pairs (i.e., minimize $\frac{1}{n}\sum(| Lr_x - Lb_y |)$, where x and y range from 1 to n.

Does the following greedy solution work? If yes, then **prove** it using Stays-Ahead or Exchange argument, otherwise provide a counter-example.
Sort the red and blue sticks in increasing order of their lengths. Suppose r_a, r_b, r_c … are the red sticks in sorted order and b_a, b_b, b_c … are the blue sticks in sorted order. Match r_a with b_a, r_b with b_b, r_c with b_c, and so-on until 'n' pairs are created.

Q2. **[20 marks]**
Your friend runs a company that delivers organs for transplantation to hospitals. Since lives are at stake, these organs have to be delivered as quickly as possible. Your friend transports these organs in special medical vehicles (MV) through fast highways. There are a number of petrol stations along the highway and your friend has to **minimize the time** spent in stopping for petrol. A full tank on the MV can hold ***T*** litres of petrol. The vehicle consumes petrol at the rate of ***R*** litres per km. If your friend decides to stop at a station then filling the tank takes ***F*** litres per minute (assume that filling the tank is the only delay incurred at the station). The starting point of the trip is your friend's office 'O' and ending point is the hospital 'H'. You are given the locations (L_1, L_2, L_3, … L_n) of petrol stations along the O-H route (assume that the O-H route is a straight line and L_1, L_2 are distances in kilometers from the starting point O along that line. The distance between any two stations will not be greater than T/R km). Assume that MV's fuel tank is full at starting point O.

You have to design a **greedy** algorithm that decides when and where to stop for petrol and how much to fill, while minimizing the overall time required in stopping for petrol.

1. Code an **efficient** algorithm in C/C++ that solves the above problem. Your code will take as input the values of ***T, R, F*** and the locations of the petrol stations. See the input format below. [7]

2. Your algorithm should output the locations where MV stopped for petrol and the time (in minutes) spent at each station. Also output the total time at all the stopping locations. [2]
3. **Prove** that your algorithm is correct using an Exchange argument. [5]
4. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]
5. You should create at least three different input files to test your algorithm for different scenarios (see format below). [3]

Your code will read input from a text file. The format of the file is as follows. The first three lines contain the values of *T, R, F*, respectively. The fourth line contains 'n', the number of petrol stations along the O-H route. This will be followed by n lines (one for each of the n locations) indicating the distance of that location from the starting point O. The last line in the input file indicates the distance of H (hospital) from O. An example is given below.

Input :
T 50
R 3
F 2
n 4
L1 6
L2 12
L3 25
L4 30
H 35

Output:
L1 6, L4 30
Total time: 36

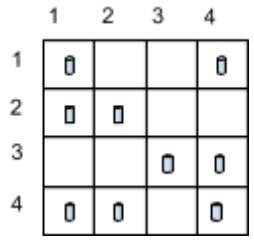
Note: The output values do not indicate the correct answer for this example. They are being used to illustrate the output format.

Q3. **[16 marks]**

You are given an `nxn` square game board and positive integers row_1, row_2, … row_n and col_1, col_2, .. col_n. You have to place game pieces in the squares in such a way that the number of game pieces in i^{th} row is equal to row_i and the number of game pieces in the i^{th} column is equal to col_i.

For example, let n=4. You are given as input a blank 4x4 board and the following integers:
$row_1=2$, $row_2=2$, $row_3=2$, $row_4=3$
$col_1=3$, $col_2=2$, $col_3=1$, $col_4=3$

One possible arrangement of game pieces on the board, satisfying the above row and column values, is as follows:



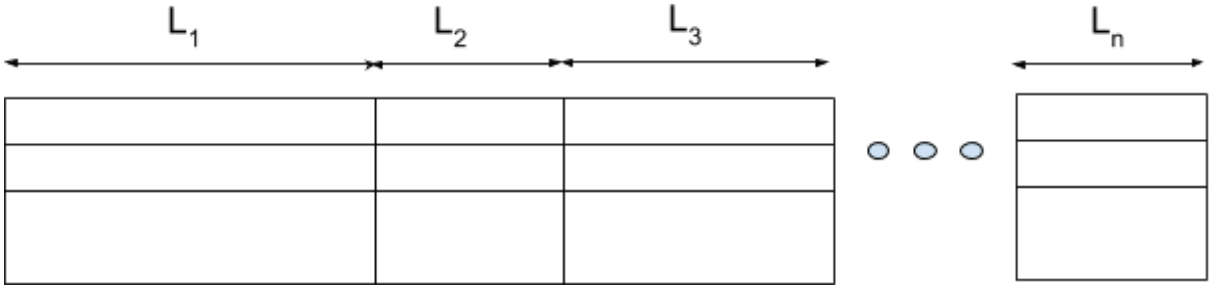
1. Code an **efficient greedy** algorithm in C/C++ that solves the above problem. Your code will take as input the value of 'n' and the row and column values. See the input format below. [8]
2. Your algorithm should output the (row, column) indices where game pieces are placed. See format below. If it is not possible to satisfy the row and column values given in the input, then your algorithm should output "Not possible". [2]
3. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]
4. You should create at least three different input files to test your algorithm for different scenarios (see format below). [3]

Input :
n 4
Row 2 2 2 3
Col 3 2 1 3

Output :
(1,1) (1,4)
(2,1) (2,2)
(3,3) (3,4)
(4,1) (4,2) (4,4)

Q4. **[16 marks]**

Consider a large warehouse where the inventory items for storage are stacked in 'n' racks. Assume all the racks are placed in a straight line as shown in the figure below. The racks are of different lengths L_1, L_2, .. L_n.



One type/category of items are placed together on one rack. Depending on the sales, different items require replenishment more often than others. Each of the racks have replenishment probabilities associated with them, which are denoted by p_1, p_2, … p_n.

A robot stands on the left of the first rack and when it receives a replenishment request, it goes to the appropriate rack and fills it up. We want to come up with an ordering of the racks, along a straight line, such that the **expected** replenishment time is **minimized**. That is, for all the 'n' racks, you have to decide which rack should be placed where, starting from the first position on the left to last on the right.

For example, suppose n=3 and $rack_3$ is placed first, followed by $rack_1$ and lastly $rack_2$. The expected replenishment time is given by: $p_3 \times L_3 + p_1 (L_3+L_1) + p_2 (L_3+L_1+L_2)$.

If we change the ordering in the above example to $rack_2$, $rack_3$, $rack_1$, then the expected replenishment time will be $p_2 \times L_2 + p_3 (L_2+L_3) + p_1 (L_2+L_3+L_1)$.

If the ordering is denoted by O_1, O_2, … O_n, then the general formula for expected time is given as follows:

$$\sum_{x=1}^{x=n} p_{Ox} \left(\sum_{y=1}^{x} L_{Oy} \right)$$

You have to come up with a **greedy** strategy for rack ordering that minimizes the expected replenishment time.

1. Code an **efficient greedy** algorithm in C/C++ that solves the above problem. Your code will take as input the value of 'n' and the lengths of the racks and their replenishment probabilities. See the input format below. [8]
2. Your algorithm should output the ordering of the racks as well as the expected time. See format below. [2]
3. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]
4. You should create at least three different input files to test your algorithm for different scenarios (see format below). [3]

Input :
n 4
L 12, 2, 8, 3

p 0.3, 0.2, 0.4, 0.1

Output :
rack1, rack4, rack2, rack3
Expected time: 18.5

Note: The output values do not indicate the correct answer for this example. They are being used to illustrate the output format.

Q5a. **[14 marks]**

Consider a railway network that connects a large number of cities across the country. A big oil company is laying pipelines along a rail network. The rail network has 'n' junctions and the oil company has set up booster stations at each junction. They want to **minimize** the total length of the pipes they have to lay such that there is a path for oil between every pair of junctions. The engineers at the oil company have constructed an undirected, weighted graph G(V,E) with 'n' vertices (i.e., junctions) and 'm' railway tracks. To minimize the total pipe lengths along the rail tracks they have constructed a minimum spanning tree 'T' of G and installed pipes along the edges of T.

One day, due to an earthquake, one of the railway tracks and also the pipe alongside it is permanently damaged. The engineers have to quickly construct a new minimum spanning tree T_2 after removing the damaged rail track from the graph G. You have to help the engineers. **Note:** if you say rerun a minimum spanning tree (MST) algorithm on the rail network, you will not get any marks.

A. Code an **efficient** algorithm in C/C++ to solve the above problem. The input to your program is a weighted undirected graph G (format is similar to one described in Q3, except that the graph is undirected. See example below.). You will find T of G using any of the MST algorithms studied in class. Now delete an arbitrary edge 'e' from T and find the new MST (T_2) without re-running the MST algorithm. Your program should output the MST 'T', the edge 'e' you have removed from T, and the new MST T_2. For outputting the MST, it is sufficient if you list all its edges. Note: it is possible that a new spanning tree T_2, including all the vertices of G, cannot be constructed after deletion of 'e' (think about why). In that case, simply print that T_2 cannot be constructed. [8]
B. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]
C. You should create at least three different input files to test your algorithm for different scenarios (see format below). [3]

An example of graph G is below.
n 5
C0: C1;3, C2;4
C1: C0;3, C3;5, C4;2
C2: C0;4, C3;4

C3: C1;5, C2;4, C4;2
C4: C1;2, C3;2

Output:
MST1: (C1,C2),(C3,C2)
Edge Removed: (C3,C1)
MST2: (C1,C2),(C3,C2)

Note: The output values do not indicate the correct answer for this example. They are being used to illustrate the output format.

Q5b. **[12 marks]**

This is a continuation of Q5a. You have the same initial rail network G(V,E) described in Q5a. This time there is no earthquake but the railway company has added a new railway track between two of the junctions (there was no direct track between these junctions previously). The railway engineers want to know what the new MST T_2 looks like after the addition of this new link. As in Q5a, you are not allowed to rerun the MST algorithm to find T_2.

A. Code an **efficient** algorithm in C/C++ to to find T_2. The input to your program is same as in Q5a. Add a new link to G between two vertices that did not have an edge previously. Your program should output the original MST 'T' (reuse code from Q5a), the edge 'e' you have added to G and the new MST T_2. For outputting the MST, it is sufficient if you list all its edges. [9]
B. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. **What is the running time of your algorithm and how much space does it use?** Your should include clear comments that explain your algorithm's time and space complexity. [2+1]
C. Reuse the input files you created in Q5a to test your algorithm for different scenarios.

Output format for the above question is the same as Q5a.

Q6. **[16 marks]**

Read Section-5.3 **'Horn Formulas'** of the book 'Algorithms' *by S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani*. You have to implement the greedy algorithm in Section 5.3 for Horn formula satisfiability in time that is **linear** in the length of the formula (i.e. the number of occurrences of literals in the formula).

1. Implement the **greedy** algorithm of Section 5.3 in C/C++ that runs in time that is linear in the length of the formula. The input file will contain each clause on a separate line. The variables will always be named x1, x2, … xn. Negation of a variable is indicated by a single apostrophe ('). The symbol ^ stands for logical AND and v stands for OR in the input. The first line of the input file will mention the number of variables 'n'. See the input format below. [8]
2. Your algorithm should output the satisfying assignment for each variable, if such an assignment exists. Otherwise, output "Not satisfiable". See format below. [2]

3. Give a clear description of your algorithm and the data structures used to implement it in the comments at the beginning of your code. Your should include clear comments that explain **how your algorithm achieves linear time complexity**. Also, mention how much space your algorithm uses. [2+1]
4. You should create at least three different input files to test your algorithm for different scenarios (see format below). [3]

Input:
n 4
(x1 ^ x2 ^ x3) => x4
(x4 ^ x3) => x1
x4 => x2
=> x4
(x4 ^ x2) => x1
(x1' v x4 v x2')
(x3')

Output:
x1=1
x2=1
x3=0
x4=1
Satisfiable

Instructions and policies

1. You must submit your ***own*** work. You may discuss the problems with other classmates but must not reveal the solution to others or copy someone's work. Remember to acknowledge other classmates if discussions with them has helped you.
2. You should name your code files using the following convention: Qx-rollno.c
3. Type your answer to the theoretical questions (Q1 & Q2 part-3) and submit a separate pdf file for each using the naming convention above.
4. Upload all your files in Assignment-1 folder on LMS. **There will be a 20% deduction for assignments submitted up to one day late. Assignments submitted 24 hours after the deadline will not be marked.**
5. There will be vivas during grading of the assignment. The TAs will announce a schedule and ask you to sign up for viva time slots. Failure to show up for vivas will result in a **70% marks reduction** in the assignment.
6. In most of the questions, you are asked to create test cases. Think carefully about good test cases that check different conditions and corner cases. The examples given in the assignment are for clarity and illustration purposes. You should not assume that those are the only test cases your code should work for. Your code should be able to scale up to larger input sizes and more complex scenarios.

7. Do not make arbitrary assumptions about the input or the structure of the problem without clarifying them first with the Instructor or the TAs.
8. For full credit, comment your code clearly and state any assumptions that you have made. As mentioned in the beginning of the assignment, there are marks for writing well-structured and efficient code.
9. Familiarize yourself with LUMS policy on plagiarism and the penalties associated with it. We will use a tool to check for plagiarism in the submissions.