CS-2009 Design and Analysis of Algorithms, Spring-2023
Project

Due Date and Time: 2nd May 2023 (1425 hrs) **Weight: 10%**

Instructions:

1. Late submission will not be accepted.
2. Project can be done in a group. Max. group size is 3 members. In case of a group size of 3 members, each student is required to solve exactly one problem. In case of group size of 2 members, one member is required to solve any one problem while the other is required to solve the remaining 2 problems. In case of group size of 1 member, the student is required to solve all the problems. Carefully form your group. Complaints related to group members not performing their tasks will not be entertained.
3. Groups are required to list their information on the following spreadsheet latest by 30th April, 2023 (you are required to lock the cells on the spreadsheet related to your group): https://docs.google.com/spreadsheets/d/1wPX4_R5qKgKPsKb4BAFraUgZFXX_s3PX0-7W69PQgfk/edit?usp=sharing
4. Changes in the groups are possible till 30th April, 2023, i.e., after 30th April, the groups cannot be changed. The schedule of demonstrations will be made on the basis of groups' information submitted till 30th April, 2023.
5. Only the listed student (on the spreadsheet) will give the demo. related to a problem, i.e., other member(s) of the group (if any) cannot give demo. on behalf of some group member. The other members of the group must not be present at the time of demo.
6. There may be 2 demos for every problem. One will be conducted by the concerned TA while the other may be conducted by one of the course instructors.
7. Secured project marks of each group member = (Secured marks in the assigned problem(s) × 5 / Total marks of the assigned problem(s)) + (group marks in all problems / 2).
   For example, if there are 3 members in a group; the 1st member secures 3 out of 3 points, the 2nd group member secures 2 out of 4 points, and the 3rd group member secures zero out of the 3 points, then the 1st group member will obtain (3×5/3 + 5/2=7.50) points out of the 10 points of the project, the 2nd group member will obtain (2×5/4 + 5/2=5.0) points out of the 10 points of the project, and the 3rd group member will obtain (0×5/3 + 5/2=2.50) points out of the 10 points of the project.
8. There will be no credit if the given requirements are changed.
9. Your solution will be evaluated in comparison with the best solution.
10. Plagiarism in any problem may result in zero marks in the whole project regardless of the percentage plagiarized. If you use a website link (URL) for help, provide it in your project report. You should have a complete understanding of the solution provided by you and should be able to fully demonstrate it during evaluation.
11. Use the additional material provided with this project.
12. Additional datasets can be used to test your work so make a generic solution of each problem.
13. All programs must be written in C++. You are required to submit the source code files only (and not the complete project folder).
14. Submit the PDF file of a "single" integrated project report having the algorithms (in pseudocode form) designed for the project and asymptotic time complexity analysis of your algorithms. Every group member will contribute to the report related to his/her allocation of problem(s). Handwritten reports will not be accepted.
15. Each group member will submit the source code file(s) of the problem(s) allocated to him/her. However, the single integrated project report will be submitted by all group members. So each group member will submit a zipped folder having; i) source code file(s), ii) PDF of the project report. *You are required to double-check the zipped folder that you are going to submit. Excuses like corrupt zipped folder, submitting a wrong or earlier version, etc., will not be accepted.*

Problem 1: Strings [Weight: 3%]

Suppose you are preparing a meal for a party and you need to prepare several dishes. Each dish has a list of ingredients. You also have a list of all the ingredients that are available. You need to check if you have enough ingredients altogether to prepare all the dishes in any order, without using the same ingredient of one set (or of one dish) for multiple dishes. The ingredients of one dish need to be present together, in any order, to be able to prepare that dish.

Formally, you are given "groups", with n sets of ingredients required for particular dishes and an array called "nums". So, n is the total number of dishes. You need to check if you can select n disjoint subarrays from the "nums" array. The elements of the sub-arrays can be in any order but they should be together.

If you can find n such disjoint subarrays, then return true. Note that a subarray is considered disjoint if and only if no element in the "nums" array is a part of (or considered in) more than one subarray.

For example, suppose you need to prepare the following dishes:
groups = {{'lettuce', 'tomato', 'cucumber'},
{'pasta', 'tomato', 'chicken'},
{'flour', 'sugar', 'eggs', 'milk'},
{'milk', 'sugar', 'custard'}}
Test case 1
nums = ['onions', 'potatoes', 'pasta', 'lettuce', 'tomato', 'cucumber', 'mangoes', 'pasta', 'tomato', 'chicken', 'bananas', 'apples', 'flour', 'sugar', 'eggs', 'milk', 'milk', 'custard', 'sugar']
Expected Outcome**: True**
Explanation:
In the example above, you can choose four disjoint subarrays from the nums list that match the groups array: {'lettuce', 'tomato', 'cucumber'}, {'pasta', 'tomato', 'chicken'},{'flour', 'sugar', 'eggs', 'milk'} and {'milk', 'custard', 'sugar'}. Note that each subarray appears in the groups array and no element is a part of more than one subarray. Therefore, you have enough ingredients to prepare, and the answer is true.
Test case 2
nums = ['pasta', 'tomato', 'chicken', 'onion', 'lettuce', 'cucumber', 'tomato', 'potatoes', 'milk', 'sugar', 'custard', 'bananas', 'flour', 'sugar', 'eggs', 'milk', 'mangoes']
Expected Outcome**: True**
Explanation:
In the example above, you can choose four disjoint subarrays from the nums list that match the groups array: {'lettuce', 'tomato', 'cucumber'}, {'pasta', 'tomato', 'chicken'}, {'flour', 'sugar', 'eggs', 'milk'} and {'milk', 'sugar', 'custard'}. Note that each subarray appears in the groups array and no element is a part of more than one subarray. Although, you have extra ingredients but that do not affect your dishes. Therefore, you have enough ingredients to prepare all the dishes, and the answer is true.
Test case 3
nums = ['pasta', 'tomato', 'chicken', 'lettuce', 'cucumber', 'milk', 'sugar', 'custard', 'flour', 'sugar', 'eggs', 'milk', 'tomato']
Expected Outcome**: False**
Explanation:
In the example above, you cannot choose four disjoint subarrays from the nums list that match the groups array: {'lettuce', 'tomato', 'cucumber'} is not present in its entirety while {'pasta', 'tomato', 'chicken'},{'flour', 'sugar', 'eggs', 'milk'}, and {'milk', 'sugar', 'custard'} are present. Note that **'tomato'** is present in the end which was missing earlier but cannot be used as it is not among other ingredients. Therefore, you do not have enough ingredients to prepare all the dishes and the answer is false.
Test case 4
nums = ['pasta', 'chicken', 'tomato', 'lettuce', 'cucumber', 'milk', 'sugar', 'custard', 'flour', 'sugar', 'eggs', 'milk']
Expected Outcome**: False**
Explanation:
In the example above, you cannot choose four disjoint subarrays from the nums list that match the groups array: {'lettuce', 'tomato', 'cucumber'} and {'pasta', 'tomato', 'chicken'} require two separate **'tomato'** ingredients as

one instance of an ingredient cannot be used in more than one dish. Hence, overlapping is not allowed and the answer is false.

Test case 5

nums = ['lettuce', 'tomato', 'cucumber', 'pasta', 'tomato', 'chicken', 'flour', 'sugar', 'eggs', 'milk', 'milk', 'sugar']

Expected Outcome**: False**

Explanation:

In the example above, you cannot choose four disjoint subarrays from the nums list that match the groups array: last dish requires {'milk', 'sugar', 'custard'} whereas in the 'nums' array 'custard' is missing. So, the answer is false.

Note that the above example is given as a sample scenario for multiple test cases but your solution should be generic with different number of dishes in a group and variable length of 'nums' array. Your solution should take input 'groups' and 'nums' from a text file and output the results of the test cases on console. In the demonstration, 'groups' and 'nums' will have different values. So, your solution/program should work for a generic case without changing anything in the code for a different text file. The sample text file "p1_input.txt" is enclosed. Write the program in C++ and name the source code file as string.cpp.

In your project report, write all your algorithms (in pseudocode form) and perform the asymptotic time and storage complexities analysis. Your solution should list all steps in the form of primitive operations, i.e., without the use of any built-in function call.

Problem 2: Graphs [Weight: 4%]

Suppose you are designing a delivery robot that must visit a number of locations in a warehouse. The robot starts at a designated "home" location and must visit every other location exactly once before returning to the home location. Additionally, the robot has a limited battery life and must return home before its battery runs out.

More formally, you are given an undirected graph G = (V, E) representing the warehouse layout, where each vertex $v \in V$ represents a location and each edge $(u, v) \in E$ represents a direct path between locations u and v that the robot can traverse. The home location is a designated vertex $h \in V$, and each vertex $v \in V$ (other than h) has a delivery time t(v) representing the amount of time it takes the robot to deliver a package at location v.

Your goal is to find a Hamiltonian circuit in G that starts and ends at h and visits every other vertex exactly once, subject to the constraint that the total time taken by the robot (including delivery times and travel times) does not exceed a given time limit T.

Design an algorithm to solve this problem. If a feasible Hamiltonian circuit exists, your algorithm should output the sequence of vertices visited in the circuit. If no feasible circuit exists, your algorithm should output "NO FEASIBLE CIRCUIT".

You may assume that the input graph is connected, and that the battery life of the robot is sufficient to travel between any two locations in the warehouse.

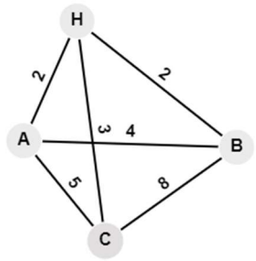
- Write the program in C++ and name the source code file as hcp.cpp.
- In your project report, write your algorithm (in pseudocode form) and perform the asymptotic time-complexity analysis. Your solution should list all steps in the form of primitive operations, i.e., without the use of any built-in function call.

Note: You have been given several text files, each containing different test cases. Your task is to develop a code that can process the test cases in each file. Here is a possible approach you can take:

1. Define a function that can handle a single test case. This function may need to read data from the text file, perform calculations or operations, and generate an output, depending on the specific task.
2. Write a loop that can iterate over each text file in the directory. For each file, use the open() function to read its contents and separate the test cases based on a delimiter, such as a blank line.

3. For each test case in the file, call the processing function you defined in step 1 to generate an output.
4. You may choose to display the output on the screen or write it to a file.

Test case 1:



V = {h, A, B, C}

E = {(h, A), (h, B), (h, C), (A, B), (A, C), (B, C)}

Weights = {2,2,3,4,5,8}

t(A) = 5, t(B) = 10, t(C) = 8

T = 38

Expected output: (h, B, A, C, h)

Test case 2:

V={h, A, B, C}

E = {(h, A), (h, B), (h, C), (A, B), (A, C), (B, C)}

Weights = {1,2,3,4,5,6}

t(A) = 5, t(B) = 10, t(C) = 8

T = 20

Expected output: NO FEASIBLE CIRCUIT

Problem 3: Dynamic Programming [Weight: 3%]

Part A [1.5%]

Suppose that Aamir either receives one e-mail or two e-mails daily. The e-mail delivery to Aamir stops as soon as he receives "n" e-mails, where n is a natural number. When Aamir has already received "n-1" e-mails, he can receive only 1 more e-mail and the e-mail delivery to him will be stopped. You are required to compute the total number of ways (unique sequences) of e-mail-delivery to Aamir in not more than linear time. For example, if n=3, the sequences can be 1,1,1 or 1,2 or 2,1. So, the total number of ways (sequences) for this example is three. Note that this is just an example; your solution should work for all values of n.

- Write the program in C++ and name the source code file as p3a.cpp.
- In your project report, write your algorithm (in pseudocode form) and perform the asymptotic time-complexity analysis. Your solution should list all steps in the form of primitive operations, i.e., without the use of any built-in function call.
- You are also required to provide the following table in your project report:

"n" e-mails	**Number of ways**
3	3
8	?
75	?
1225	?

Part B [1.5%]

You need to travel by a rented car in a forward direction only. There are n stopping points in the route (including the starting and the ending points). You do not need to stop at every point in the route. Before starting your journey, you are given for each $1 \leq i < j \leq n$, the cost $c_{i,j}$ for renting a car from point i to point j. These costs are arbitrary. For example, it is possible that $c_{1,3} = 6$ and $c_{1,4} = 4$. You begin at point 1 and must end at point n (using rented cars). Your goal is to minimize the total rental cost.

- In the project report:

- Give the most efficient Dynamic Programming algorithm (in pseudocode form) you can (not more than quadratic time) to find out the optimal cost.
- Write an efficient algorithm (in pseudocode form) to find out the optimal path (not more than linear time).
- Perform asymptotic time and extra storage complexity analyses of your algorithms.
- Show trace of your algorithms by choosing an appropriate example.
- Provide the results of the following sample test cases. At the time of demonstration, your solution will be checked against two unseen test cases as well.

Sample test cases	**Outputs**
Number of points = n = 8 Cost Matrix {0,100,20,300,400,500,600,700}, {∞,0,50,10,100,200,300,400}, {∞,∞,0,30,10,90,10,150}, {∞,∞,∞,0,80,160,240,320}, {∞,∞,∞,∞,0,120,240,600}, {∞,∞,∞,∞,∞,0,120,240}, {∞,∞,∞,∞,∞,∞,0,120}, {∞,∞,∞,∞,∞,∞,∞,0}	Optimal cost = 150 Optimal path: 1→3→7→8
Number of points = n = 10 Cost Matrix {0,100,200,40,250,300,400,500,600,700}, {∞,0,50,10,100,200,300,400,500,600}, {∞,∞,0,30,10,90,10,150,200,250}, {∞,∞,∞,0,80,160,240,320,400,430}, {∞,∞,∞,∞,0,120,240,600,650,700}, {∞,∞,∞,∞,∞,0,120,240,300,350}, {∞,∞,∞,∞,∞,∞,0,120,140,160}, {∞,∞,∞,∞,∞,∞,∞,0,200,400}, {∞,∞,∞,∞,∞,∞,∞,∞,0,400}, {∞,∞,∞,∞,∞,∞,∞,∞,∞,0}	Optimal cost = 320 Optimal path: 1→2→3→7→10

- Write the program in C++ and name the source code file as p3b.cpp.
- You should define const int n = [some value] and then all your loops should run on n (instead of hardcoding any value), so that, at the time of demonstration, you only need to make the change at one place apart from changing the cost matrix.