INF1008 Data Structures and Algorithms
Lab 1: Greedy Algorithm

Topics:

1. Gas Stops
2. Color Graph
3. Knapsack

Lab Assignment:

To help you better practice, you need to perform a set of tasks in one auto-grading system, **Gradescope** in LMS/xSiTe. Gradescope will provide you immediate feedback of your program, such that you will know the issue of your program.

In this lab, you need to finish three tasks in Gradescope system. Gradescope system will provide you immediate feedback about the correctness of your program. You can view the feedback of your system by clicking the failed case. Gradescope adopts a test cases-based approach to check your program. Your score is depending on the number of test cases that you program can pass. Note that to train you to have a good programming practice, you must write your program strictly according to the requirement of the tasks, including your input and output format. If there is any difference (even one space), your program will fail on the test cases. SO, TRAIN YOURSELF TO BE AN EXACT THINKER!

To help you practice, you are allowed to do multiple attempts for each task. Enjoy your learning!

Note:

1. **It is noted that usage of '*quit()*', '*exit()*' and '*sys.exit()*' functions to quit/exit a conditional statement or function in Gradescope submission results in test case failure.**
2. **Use '*return*' or other means.**

Task 1: Gas Stops

Task Description

You are driving a car from city A to city B. The distance between the two cities is D miles. Your car can travel up to M miles on a full tank of gas. Along the way, there are N gas stations at distances S1, S2, …, SN miles from city A (with S1 < S2 < ⋯ < SN < D). You start with a full tank of gas.

Your task is to determine the minimum number of stops you need to make to reach city B. If it is not possible to reach city B, return −1.

Input

1. An integer D representing the total distance to the destination.
2. An integer M representing the maximum distance the car can travel on a full tank.
3. An array S of length N, where S[i] is the distance of the i-th gas station from city A.

Output

The minimum number of refuelling stops required to reach city B, or −1 if the destination is not reachable.

Example 1 Input

Enter the total distance (D): 25
Enter the maximum fuel range (M): 10
Enter the distances of the gas stations separated by spaces: 10 14 20 21

Example 1 Output

Minimum number of stops: 2

Example 2 Input

Enter the total distance (D): 100
Enter the maximum fuel range (M): 30
Enter the distances of the gas stations separated by spaces: 40 70 90

Example 2 Output

Destination is not reachable.

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "**gas_station.py**" from the "Lab\Lab1" xSiTe folder.
2. Add your code to the function "***def min_stops(D, M, S)***".
3. Do not change file name, function names and function arguments.
4. Drag and drop your locally tested code to Gradescope assignment titled "**Lab1-1: Gas Stops**". You can submit and auto graded any number of times. Last submission counts.
5. Autograder will be timed out after 10 minutes.

Task 2: Color Graph

Task Description

You are given an undirected graph represented by n vertices and m edges. Write a program to determine the minimum number of colors required to color the graph such that **no** two adjacent vertices have the same color.

Input

1. Number of edges

2. Each edge as two space-separated integers

Output

Print the color assigned to each vertex and the total number of colors used.

Example Input

Enter the number of edges: 6

Enter each edge as two space-separated integers (e.g., '0 1'):

0 1

0 2

1 2

1 3

2 4

3 4

Example Output

Vertex	Color
0	0
1	1
2	2
3	0
4	1

Minimum number of colors required: 3

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "**graph_coloring.py**" from the "Labs\Lab1" xSiTe folder.
2. Add your code to the function "***def graph_coloring(graph, n)***".
3. Do not change file name, function names and function arguments.
4. Drag and drop your locally tested code to Gradescope assignment titled "**Lab1-2: Color Graph**". You can submit and auto graded any number of times. Last submission counts.
5. Autograder will be timed out after 10 minutes.

Task 3: Knapsack

Task Description

You are a backpacker, and you have a knapsack with a weight capacity of W. You are given n items, where each item has a weight and a value. Unlike the 0/1 knapsack problem, you are allowed to take fractions of items (i.e., you can take a part of an item, not necessarily the entire item).

Your goal is to maximize the total value of the items in the knapsack without exceeding the weight limit.

Input

a) n ($1 \leq n \leq 1000$): The number of items.
b) W ($1 \leq W \leq 10^4$): The weight capacity of the knapsack.
c) An array of n elements where each element represents an item, containing:
- weight[i] ($1 \leq weight[i] \leq 10^4$): The weight of the i-th item.
- value[i] ($1 \leq value[i] \leq 10^4$): The value of the i-th item.

Output

The maximum value that can be carried in the knapsack, up to two decimal places.

Example Input

Enter number of items and knapsack capacity, separated by space: 3 50
Enter the weights for each item (separated by spaces): 10 20 30
Enter the values for each item (separated by spaces): 60 100 120

Example Output

Maximum value that can be carried: 240.00

Instructions to submit and auto grade your code to Gradescope:

1. Download the skeleton code file "**knapsack.py**" from the "Labs\Lab1" xSiTe folder.
2. Add your code to the function "***def fractional_knapsack(W, items, n)***".
3. Do not change file name, function names and function arguments.
4. Drag and drop your locally tested code to Gradescope assignment titled "**Lab1-3: Knapsack**". You can submit and auto graded any number of times. Last submission counts.
5. Autograder will be timed out after 10 minutes.