1. Timezone converter: Write a program with the following input C1 C2 HH:MM where C1 and C2 represent the countries C1 and C2 and HH:MM represents how many hours and minutes in 24-hour format, C1 is ahead or behind C2, depending on HH is + or -. You then take another input, C HH:MM:Day where C is either C1 or C2 and HH:MM:Day is the current day of the week and hours and minutes in C, and you need to convert the time i,e HH:MM:day in the other country.

2. Intra-City Connectivity: Consider the blocks in a city map as a 2D matrix. Your aim is to determine the pair of blocks that are connected. This will help authorities determine where the next transportation systems must be set up. In the setup, alphabets represent blocks and 0 represents in-habitated areas.
   For example, for the given 2-D layout
   A 0 B C
   D 0 G 0
   0 0 0 F

   The pairs of connectivity are:
   (A, D)
   (B, C)
   (B, G)
   Note that pairs (A, D) and (D, A) are considered the same, and we keep (A, D) as it is in alphabetical order. For connectivity, we should check all surrounding directions of a block.

   In the above setup, we can also determine the blocks that are isolated, i.e., not connected to any blocks. Here we output:
   F

3. Ranking metrics: In information retrieval, some metrics are employed to determine how good/relevant the results are. Consider for a given query, google search returns results that are ranked with the most relevant link at the top and so on. MAP (mean average precision) determines the average of average precision where we first determine the result's precision at kth rank and then average it for all values of K for a given query's output. Finally, we average this over multiple queries to determine the system's MAP. Slides 1-6 here can be used for a visual understanding.

   For example, in the below setup, X represents a document that is marked irrelevant, and Q marks good results we have the following ranks from left to right, with left most at rank one space separated:
   Q X Q Q Q Q X X X Q
   We have relevant results at ranks 1, 3, 4, 5, 6, and 10
   Precision at each rank from 1 to 10 is
   P_1 = 1/1 i,e 100%
   P_2 = 1/2, i.e., 50%, because we have relevant at rank 1 but irrelevant at rank 2
   P_3 = 2/3 because we have relevance at rank 1 and 3 but irrelevant at rank 2.
   Similarly, for P_8, we have relevant at positions 1, 3, 4, 5, and 6 and irrelevant at rest thus, P_8 = ⅝
   Finally, AP is the average P_i for i from 1 to 10.
   In a general case, we have p_i go from 1 to K, where K is given by the user.
   Once we have computed the Average P for a given query setup, we repeat this for multiple documents returned by the system for other queries
   Thus, for the input setup of the form
   K
   Q X Q Q Q Q X X X Q
   X X Q Q Q Q X Q X Q
   X X Q Q X Q X X X Q
   Q Q Q X Q Q Q Q X Q
   Q X Q Q Q X X Q X Q
   Write a program to return the MAP value of the system.
   First, find AP for each row of results separately from p_i to p_k, then take an average of AP.

4. Buffering: Over the internet, information is transferred into small information units called packets. We assume each packet to be of form ID:START:END, where ID is the identifier of the packet, and START and END capture the time at which the packet left the source and reached the destination, respectively, in terms of milliseconds. We aim to determine the total buffer time that occurred in the system. The time gap for each packet is simply END-START, and then we sort the packets based on ID values, as packets may arrive out of order. Finally, once sorted, we can learn about packets that are still missing.
   Based on the input below, write a program to perform the above three tasks.
   P 5
   3:20:130
   1:100:123
   2:12:15

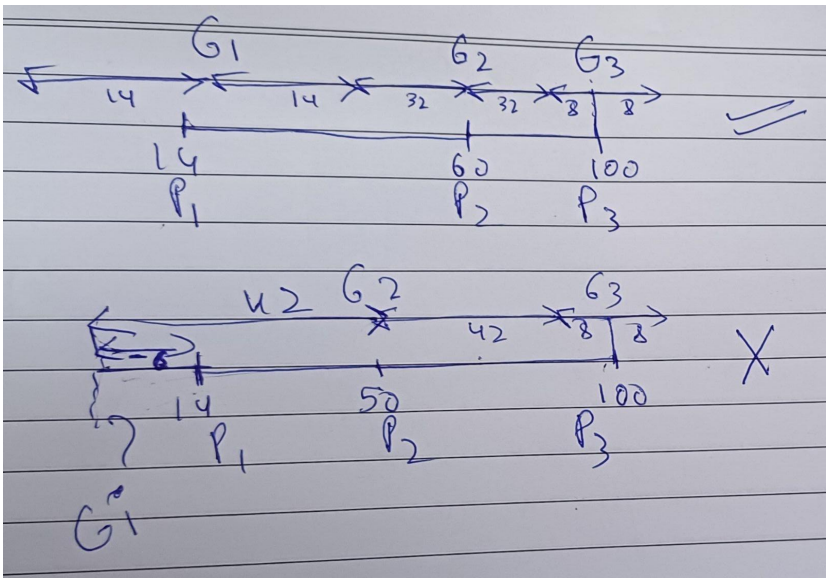
   Here, P states how many packets should be checked. The next n lines where n can be <= P in our case, P = 5.
   We see that buffer time is (130-20)+(123-100)+(15-12) milliseconds. We print the time in seconds.
   Additionally, we print sorted packets as 1,2,3
   Further, based on the sorted list, packets with IDs 4 and 5 are missing, printed as Missing: 4,5

5. Gears and Pegs: Consider a system where we have circular gears of radius R 1 to 100. We need to employ the fact that when the gear is just touching (not overlapping but just touching) the next gear, the motion of one can trigger the motion of the other. Assume gears are placed horizontally from left to right, with rightmost as the last gear, and you are given positions where the gears on the horizontal beam should be placed, along with the radius of the rightmost gear. You need to determine what will be the size of the rest of the gears and make a purchase of the gears of that size. The price of a gear is 0.75*(R^2).
   If for the given gear positions, no gears are available, we return -1.
   Example
   8
   14,50,100

   Here, we place the last gear at position 100 and of size 8, which will occupy space from 92 to 108. Now, from position 92 to 50, we need a gear of size 92-50, i.e., 42 radius. But when we place a gear of size 42 at location 50, we cover location 14 as well, which means we cannot place our first gear in the required position hence, this setup will be discarded. We discard it because 50-14<42 therefore, it is impossible to place another gear.
   We should be able to place only N gears for a system of N positions. Otherwise, the system will be discarded.

   8
   14, 60,100

   We again start with a gear of size 8, then we need a gear of size 92-60 = 32 radii. Now, from position 60, we have gear of size 32, which means we still have 60-14>32, and we see that radii of 14, i.e., 60-14-32, are apt to accommodate the 3rd gear.
   Total price will be 0.75[8^2 + 32^2 + 14^2]




6. Every interaction in the real world can be mapped into a sequence of states executing one after the after. However, it is possible that movement from some states to others is not possible. We also have a probability value associated with the state for each movement. We aim first to employ the best data structure to capture these transitions of states and then, for a given sequence of states, determine the probability of the event taking place as a product of the transition/movement probability.

   For example, in the above setup

   4
   S1:S2:0.7
   S1:S3:0.9
   S2:S1:0.2
   S2:S3:0.3
   S3:S4:1.0
   3
   S1,S4
   S1,S3,S4
   S4,S3

   4 means we have four valid movements, followed by movements and their probability S1:S2:07 means movement from S1 to S2 is possible with a probability of 0.7. Note that movement from A to B does not mean the same as movement from B to A.
   3 represents we are checking the movement for three separate sequences. For S1, S4, no movement is possible from S1 to S4 thus, output will be -1. S1 to S3 to S4, movement is possible with probability 0.9 * 1.0 i,e 0.9. For S4 to S3, movement is not possible, so we output -1
   Thus, results will be
   -1
   0.9
   -1

   Your program should be able to accommodate 100 unique states i.e., 100*100 movements are possible, including self-transition like S3:S3:0.4
   Then, for the next N inputs, you should output the final probability if and only if the whole sequence is possible.

   For 100 unique states, think about what data structure we should use if the transitions are possible for more than 90% of state pairs (9000 values to store). What if only 5% of the pairs have possible movements (500 values to store)?