CS 602

Algorithm Design and Implementation

Assignment 3

[Question 1] Palindromic Zuma

Zuma was a popular game where players shoot and eliminate three or more colored balls of the same color before they reach the end of the path. In this variant of the Zuma game, balls that form a palindromic pattern are eliminated, instead of balls of the same color. Given a sequence of colored balls with colors represented by integers, each shot specifies a beginning index and an ending index. If the balls in this interval form a palindromic pattern, they are eliminated, and the two ends will then concatenate and merge together.

In this question, your task is to implement function `min_shot` to compute the minimum number of shots to eliminate all colored balls. For example, eliminating the color sequence `[1, 2, 3, 2, 1]` requires minimally one shot, while eliminating the sequence `[1, 2, 3, 4, 5, 5, 4, 2, 1]` requires minimally two shots.

Each set of test cases starts with the number of sequences, which can be up to 10, and each sequence has a maximum length of 500. 8 sets of test cases carry 1 mark each, and the remaining 2 marks are awarded with the correct justification of the time complexity.

Sample Input

```
4
1 2 1
1 2 3
1 2 3 4 5 5 4 2 1
1 3 5 6 6 2 5 7 1
```

Sample Output

```
1
3
2
4
```

[Question 2] Project Selection (II)

In order to get listed at an exchange, a company hired an accountant to ensure their account look smart. The profit of this company comes from taking projects, if it takes a project with cost c and revenue r, the company will then have a profit of $r - c$. You may assume the overall economy is so good that the company always has choices for their next project to take. The accountant suggests to the company that the next project they select should always be "larger" than the current project, where "larger" is in terms of project cost. In other words, the costs of projects they select should form an ascending order.

The accountant has a second suggestion that the cashflow is better to be consistent, which means selected projects should be spread out over time. To achieve this, project schedules are carefully examined, and projects are partitioned into J groups, with project j_i being the i^{th} project in group j ($1 \leq j \leq J$), with cost c_{ji} and revenue r_{ji} . The groups are arranged in chronological order, a project in group j should be completed before the start of another project in group j' where $j < j'$, if both projects are selected. In addition, for each group j, you can only select at most 1 project since your team can only work on one project at a time. If no project were selected from group j, there would be an accounting period with no operating income, which does not look smart on their account. Therefore, the objective is to understand if it is possible to have a feasible plan that covers all period. If it is possible, what the maximum final capital can be. Of course, the accountant still wants the project costs to be ascending.

Suppose there are n projects, grouped into J groups. There are n_j projects in group j ($1 \leq j \leq J$) where $\sum_{j=1}^{J} n_j = n$. Project j_i $(1 \leq j_i \leq n_j)$ is the i^{th} project in group j, and it is associated to a pair of values c_{ji} and r_{ji}, where c_{ji} is the cost to work on project j_i and r_{ji} is the revenue of project j_i after its completion. Assume your team start with a capital value c, and you can only select one project with its cost not exceeding your team's capital. Upon completion of the project j_i, your team's capital will increase by the profit of that project, which is $r_{ji} - c_{ji}$. For each group, your team can only select at most one project to work on. You may also assume all project are all valid with $r_{ji} \geq c_{ji} \geq 1$.

Implement the function `project_selection`. Please also state and justify the time complexity of your algorithm as comments in your code.

Test inputs begin with the number of groups J ($J \leq 1000$) and the number of scenarios. This is followed by J lines, each of which contains a list of integer pairs of cost and revenue separated by ':'. There are at most 50 pairs for each group, and these pairs are all distinct, it is possible for two projects to have the same cost or the same revenue, but not both. The

last line describes initial capital C for each scenario. All capitals, costs and revenues of projects are integers and ranging between 1 and 1,000,000.

Your task is to produce the maximum final capital which can be achieved by a selection plan that satisfies the criterion above. In the case that there is no valid selection plan, print impossible, otherwise, print the maximum final capital for each scenario.

There will be 8 sets of test cases with 1 mark each. The remaining 2 marks are awarded with the correct justification of the time complexity of your algorithm in comments.

Sample Input

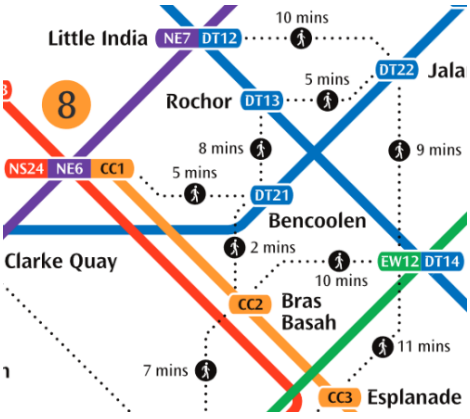
```
3 3
214:246 212:220 232:259 189:267 221:263
496:536 419:603 498:560 509:585
722:815 665:905 699:780 708:768 827:953 775:859
300 400 600
```

Sample Output

```
impossible
impossible
1102
```

[Question 3] Shall we walk more?

On 22nd March 2024, the Ground Transport Authority (GTA) has announced to remove the additional boarding charges for commuters who make rail transfers between any two different MRT/LRT stations within 30 minutes. This allows passengers to tap out from station A, walk to a nearby station B and continue their journeys. GTA has therefore published a rail system map with walking time indicated between nearby stations. For example, passengers can walk from Bencoolen station to Bras Basah station in two minutes via Singapura Magnificent University's concourse.



Suppose you are now hired by GTA as a data scientist to work on the comparison between time taken with and without walking between nearby stations. You are given the following information:

1. **`waiting[A]`**: the expected waiting time for train service running on line `A`. Assume the expected waiting time are the same for both directions at all stations on the same line.
2. **`traveling[B, C]`**: the travelling time between station `B` and station `C` if `B` and `C` are on the same line. Assume there are only local services, no express services.
3. **`transfer[D, E, F]`**: the transfer time from line `E` to line `F` at an interchange station `D` if both line `E` and `F` serve station `D`. Due to the platform structure of interchange stations, transfer time depends on the distances between platforms of different lines, and such transfer time can be as long as walking time outside stations, which should be taken into consideration.
4. **`walking[G, H]`**: the walking time from station `G` to station `H` if station `H` is walkable from station `G` within 15 minutes. These are the dotted line on the map, and we assume `H` is not walkable from `G` if key `(G, H)` does not exist in the data structure walking.

Design an algorithm to compute time savings by possible walking among MRT stations, which is the time saving of allowing walking versus not allowing walking. Given a pair of stations `U` and `V`, let x be the shortest travelling time between `U` and `V` via trains only and y be the shortest travelling time between `U` and `V` with both taking trains and walking allowed for each segment of a possible route, the algorithm computes the difference between x and y, i.e., x - y. This value should always be positive since y lower bounds x.

Before giving an example, let me explain the format of the input. Each input starts with two integers in the first line, n and m, where n is the number of MRT stations and m is the number of MRT lines. Assume all MRT routes are simple lines, e.g., there is no Y-shape route like the circle line. The next m lines describe m MRT lines each, indexed from L_0 to L_{m-1}. The first integer of each line represents the waiting time of the local services, in seconds. Starting from the second word, train stations interleaves with traveling time between two stations. For example, "`S0 90 S1`" represents S_0 and S_1 are two consecutive stations on this line, and the traveling time between S_0 and S_1 is 90 seconds. Each line ends with a terminal station ID. The next line indicates the number of interchange stations and is followed the transferring time at each interchange station. Note that there may be multiple lines intersecting at an interchange station, so the transferring time is specific to each pair of the train lines. For example, "`S0 L0:L3:48 L0:L5:75 L3:L5:122`" represents the symmetrical transferring time at station S_0, the transferring time between L_0 and L_3 is 48 seconds and between L_0 and L_5 is 75 seconds. Please do not worry about the validity of the input, station S_0 must appear in earlier descriptions about L_0, L_3 and L_5. The last section of the input is the description of the walking time among MRT stations, beginning with the number of walks.

Each walk is described by two stations, and number of seconds taken if walking from one to the other. The walks are symmetrical and transitive. If there are walks from `P` to `Q` and from `Q` to `R`, it is also possible to walk from `P` to `R`. It is possible to have multiple walks with total time longer than 30 minutes, since our primary concern is the time saving by walking.

With the description of the sample input, and takin the journey from Bencoolen (`S20` in sample input) to Bras Basah (`S1` in sample input) as an example, the transit time by train is

- ✓ 99 seconds (waiting time for `L2`)
- ✓ 186 + 192 seconds (traveling time from `S20` to `S18`)
- ✓ 40 seconds (transferring time at `S18`)
- ✓ 80 seconds (waiting time for `L5`)
- ✓ 89 + 134 seconds (traveling time from `S18` to `S0`)
- ✓ 75 seconds (transferring time at `S0`)
- ✓ 140 seconds (waiting time for `L0`)
- ✓ 90 seconds (traveling time from `S0` to `S1`)

This list sums up to be 1125, comparing with walking from `S20` to `S1` of 120 seconds, the time saving would be 1005 seconds, which is very significant compared to 120 seconds.

For each test input, there are at most 20 MRT lines with at most 50 stations on each line. The number of interchange stations is at most 100 with at most 4 lines interchange at one station, and the number of walks is at most 50 with multiple walks allowed for one trip.

Sample Input	Sample Output
Please refer to the file A3Q3.in .	Please refer to the file A3Q3.out .

Running python skeleton with sample input:

1. Open “Anaconda Prompt”
2. Go to the directory where you put the file **A3Q1.py** and **A3Q1.in**, using command **`cd`**
3. Run command **python A3Q1.py < A3Q1.in**
4. You may want to create a test input called **my_own_test.in** to design a test case for your own program, the command would then be **python A3Q1.py < my_own_test.in**
5. Same applies to Question 2, so you may run **python A3Q2.py < A3Q2.in**