Slide 1.

Hello everyone. That's a seventh laboratory work for algorithms and data structures. Our topic is graph algorithms. Problems of this laboratory work will require not only usage of graph algorithms, but also implementation of these algorithms.

Slide 2.

Today we will analyze possible solutions of problem "Billionaires" from your previous homework, and problems "Map Coloring" and "Mobile Telegraphs".

Slide 3.

Problem 1650 - "Billionaires". You can go through the link to check full description. I will remind you the most important part

- Your employer asks you to determine for each city the number of days during period of time on which this city exceeded all other cities in the total amount of money that billionaires staying in this city have.
- The N lines contain information about billionaires: names, cities where they were staying at the beginning of the period, and their fortunes. The K lines contain the list of travels: the number of days, name of the person, and city of destination.
- In each line of the output give name of a city and the number of days during which this city was the first with respect to the sum of fortunes of the billionaires staying there.

Slide 4.

This problem is mostly focused on accurate implementation and using of data structures from standard libraries rather than on inventing complex algorithm.

Restrictions for this problem allow us to simulate entities of the problem and interactions between them.

Should be noted, that this problem can be solved with slightly different approaches, depends on structure choice and the way to split data between structures.

We will focus on one example.

Slide 5.

At the start, it is required to understand what entities are presented in the problem, and what properties do they have.

First entity is billionaire. Each billionaire has his name, city, where he is currently located, and number of money, which he has.

Second entity is city. Each city has its name, number of total money in this city (or sum of money of billionaires, who are currently in the city), and number of days, when this city exceeded all other cities by total money.

Slide 6.

Because input calls all billionaires and cities with names, we can present them as map, where name is the key.

For cities regular map is normal choice, because it will be sorted by name during iteration, it will be useful for final output.

But there is one more question. How to fast detect which city is current leader? For this purpose, we can add one more data structure - set of pairs of city names and city properties. This set should have custom comparator to sort all set entries by total sum of money in the city, so its order can be maintained correctly. Instead of pairs you can use, for example, simple class with 2 fields and getter and setter methods.

Don't forget, that objects inside set are constant, because otherwise it is impossible to automatically maintain order of elements. If you want to change element, you should remove it from the set, make required changes and insert again, so set will be correctly resorted.

Slide 7.

Data structures were chosen, next part is simulation. What should be done on each billionaire's travel?

First step is updating city with maximum amount of money. For this purpose, we should calculate length of the last period. It is just a subtraction between date of current travel and previous one.

Because we have set, we can easily get city with greatest number of money. This city was leader for entire previous period. We can increment counter of leader days in city map. Should be noted, if there are 2 or more cities with the same maximum amount of money – there were no leaders for this period at all.

Slide 8.

Next step is updating information. For example, we have input line, for Billionaire2, who has travelled to City3. It means, that we should find previous location of billionaire in related map, update his current city and remember previous one. Previous location of billionaire in this example is City2.

Slide 9.

Why do we need previous city of billionaire? Because we should decrease total money inside this city. For this purpose, it is required to change our set. First, we find city of previous billionaire's location. In our example, it is line with City2. We should remove this entry from set, update total money, because billionaire leaves this city and insert it again.

Same procedure should be done and for new location of billionaire.

Slide 10.

After inserting set will resort itself, and again top value of the set will be city with greatest number of total money, City2 in our case.

Slide 11.

And the last step is printing result. There are several important moments. All names of cities should be sorted by name, more than that, city should be printed only if it was leader for at least one day.

Also, important not to miss first period (from day 1 till first travel) and last period (from last travel till day M), because some city has billionaires from the start and was leader for this period.

Slide 12.

Problem 1080 - “Map Coloring”. You can go through the link to check full description. I will remind you the most important part

- We consider a geographical map with N countries numbered from 1 to N. For every country we know the numbers of other countries which are connected with its border. From every country we can reach to any other one, eventually crossing some borders. Write a program which determines whether it is possible to color the map only in two colors — red and blue in such a way that if two countries are connected their colors are different. The color of the first country is red. Your program must output one possible coloring for the other countries, or show, that such coloring is impossible.

Slide 13.

One of the common ways to interpret map of countries is a plane graph, where countries are faces of graph, borders are edges and crossing of borders are vertices.

Let’s create a dual graph for this plane graph. Dual graph for plane graph is a graph that has a vertex for each face of plane graph. The dual graph has an edge for each pair of faces of original graph that are separated from each other by an edge. You can see it on the second image in red color. From description of problem, it is also known that there is always exist a path between each two countries. It means, that dual graph is connected.

Slide 14.

So, we can say, that problem of map coloring is the same as proper vertex coloring of dual graph.

It is known that graph can be properly colored in 2 colors if and only if it is a bipartite graph. In this graph all vertices can be split on 2 parts, without any edges inside each of the part. If we fully color first part into red color, and second part into blue color, coloring will be proper. Neighbor vertices have different colors.

Slide 15.

How to check that graph is bipartite or can be colored properly into 2 colors? It is enough to traverse the graph and check and color all vertices. So, it means, that if we imagine tree of search in our graph, it should have different colors for neighboring levels of vertices. On each step we should check that we can color next vertex in proper color.

So, both search algorithms can be used: DFS and BFS. We will focus on BFS as example. First country color is red by task description.

Slide 16.

Next level should be colored in blue. There is no issues because all vertices have no colors.

Next level should be red, first two vertices can be colored into red.

Slide 17.

As you can see on the slide, this time we find, that one vertex is already colored by left sibling. But in this case, it has “expected” color. We want to color it into red, it is already colored into red. It means, that everything is good, coloring is proper, and we can continue.

Slide 18.

Let’s check another example. Three levels were successfully colored. On the next step we should color fourth level and it’s turn to color all non-parent neighbors of vertex A in blue. But one of the neighbors is already a red vertex in the same level. It means, that coloring can’t be proper for this graph.

Can be noted, that bipartite graph can’t have any odd cycles. In the last example, problem with coloring is happened in odd cycle of 5 vertices.

Slide 19.

Problem 1806 - “Mobile Telegraphs”. You can go through the link to check full description. I will remind you the most important part

- Each device has a unique number, which is a string consisting of ten decimal digits. A message can be sent from a telegraph a to a telegraph b only if the number b can be obtained from the number a by changing exactly one digit or by swapping two digits, and the time of sending a message from the telegraph a to the telegraph b depends on the length of the longest common prefix of their numbers.
- Output “-1” if it is impossible to deliver the message. Otherwise, in the first line output the minimal time required to deliver the message. In the second line – number of fighters in the delivery path, and in the third line – their numbers

Slide 20.

For this problem we have graph of mobile telegraphs. Our task is to find shortest path between first and last telegraphs. Also, it is known, that all weights of edges are positive integers.

Based on this knowledge we can choose fastest algorithm for shortest path problem – it is Dijkstra’s algorithm, and problem can be solved with complexity O(m*logn), where n is vertices and m is edges.

But the main problem is building a graph.

Slide 21.

It is possible to prepare whole graph at the start of the program. But in this case for big graphs, it becomes harder to satisfy memory restrictions. Graph can have too many vertices. Also, it can lead to a problems with time limit.

To solve this issue, several optimizations can be applied. Let's check some of them.

Slide 22.

First of all, we should understand, that number of possible neighbors of each vertex is limited. Let's find this value.

We should check, how many possible combinations are available for telegraph. First, we can calculate amount of telegraph numbers, which can be generated by swapping of two digits.

Let's note, that first digit can be swapped with second, third and so on. 9 possible swaps. Second digit can be swapped with third, fourth and so on. 8 possible swaps. We have no need to calculate swapping second and first digit again, because this swap has already been taken into account.

Finally, we have sum of numbers from 1 to 9, formula for calculating this sum is known. Total 45 combinations.

Slide 23.

Second type of possible combinations can be generated by changing one digit. It means, that first digit can be changed to any of other 9 possible values. The same for second digit and so on. Each of 10 digits can be changed to 9 other possible values. Total 90 combinations.

So maximum number of possible neighbor vertices is 135.

Slide 24.

Second point is when and where generate the graph.

It is very important, that for Dijkstra's algorithm at each point we have no need in presentation of the whole graph.

Check example on the slide. We have undirected graph; algorithm starts from vertex A and now at vertex B. This is the first moment of time, when algorithm should know, what vertices are neighbors of vertex B, because otherwise relaxation would be impossible. At the same time, neighbors of vertex C is irrelevant at this stage of algorithm.

Slide 25.

As it was discussed earlier, there is no need to compare current number of telegraph with all other telegraph numbers to find connections between them. For current vertex it is possible to compute numbers of all its possible neighbors (not more than 135 neighbors) and then check with fast data structure, like hash table, is generated telegraph number presented in our graph or not.

Also, only at this point we can calculate weights of edges.

Slide 26.

The last, but, probably, most important note. We have no need to create 135 new strings each time for each neighbors. After we process vertex and relax its edges, we have no usage for numbers of its neighbors (in priority queue they can be present as index number in original order of telegraphs in input data).

It means, that we can create array of 135 strings, and always use it to generate all neighbors of vertex. More than that, these strings can be edited, so there is no need to allocate memory for new array or strings. It should save enough memory to pass memory limits. Should be noted, that in Java objects of class String is immutable. But instead of String you can use 2-dimensional array of chars, and if required build a String with related constructor.

Other optimizations are related to usage of integers instead of strings.

Slide 27.

Let's check how these optimizations work, based on first example from the problem.

Relationship between telegraph numbers and their vertex numbers is presented with hash table for fast search. List of possible neighbors is array of 135 strings.

We will start with first vertex. Its number is in the head of the table. Other rows of the table present different combinations, which are telegraph numbers of all possible neighbors for this vertex. As it was mentioned earlier, it is only possible to have 45 neighbors by swapping two digits and 90 neighbors by changing one digit. Let's generate all possible neighbors into the array.

On the next step we should iterate over this array and check in hash map, is this telegraph number exist in given graph? In this case, only one number from array is presented in hash map – it is number of vertex 2. So, there is an edge between vertex 1 and vertex 2.

Slide 28.

What is weight of an edge from current vertex to found one? We can easily calculate length of common prefix and get weight from input data.

Please, note, now we know only neighbors of current vertex, rest part of the graph isn't presented in memory.

We can continue with regular steps of Dijkstra's algorithm:

- Calculate Dijkstra's score to all found neighbors
- For each neighbor check if new Dijkstra's score to vertex is less than known value
    - If this condition is true, then add such vertex with its score to priority queue or update its score if vertex is already presented in a queue
    - And store new shortest paths to neighbors
- Finally, we can mark current vertex as vertex with already found shortest path and get next vertex with minimal Dijkstra's score from priority queue and continue.

Slide 29.

We will get next vertex from priority queue, and it is vertex 2.

Let's again generate all possible neighbors of this vertex. We can use the same array of 135 strings as on previous step. No need to clean it – we will again generate the same number of possible neighbors.

In this case we find two numbers, which are presented in hash map. But first number is related to vertex 1, which is already marked as vertex with found shortest path. No actions are required for this vertex. But another vertex is vertex 4. So, there is an edge from vertex 2 to vertex 4.

Now we should calculate weight of this edge and repeat usual steps of Dijkstra's algorithm.

Algorithm will stop when shortest paths to all vertices are found.

Slide 30.

Mandatory task. You should implement source code for problem 1080 "Map Coloring", upload it to Timus system and pass all tests. After this you should prepare a report and send it via email. Please, use template document for your report and carefully set correct subject for the mail.

Slide 31.

You will have 2 problems for homework. First problem is 1806 "Mobile Telegraphs", which was explained earlier. Second problem is 1450 "Russian Pipelines". You should solve it by yourself. Please, note, that report for this problem should contain analysis of graph type and comparison of possible solutions.

Homework is optional, but successful completion of this problems can give you extra points for better grade.

Slide 32.

Thank you for attention