Slide 1.

Hello, dear students. That's the eighth laboratory work for algorithms and data structures. Our topic is graph algorithms.

Slide 2.

Today we will analyze possible solutions of problem "Russian Pipelines" from your previous homework, problem "Network" and will have small overview of fourth test's tasks

Slide 3.

Problem 1450 - "Russian Pipelines". You can go through the link to check full description. I will remind you the most important part

- Russian pipeline system consists of N transfer stations. For each of M pipelines the numbers of stations A and B, which are connected by this pipeline, and its profitability C are known. Profitability of a pipeline is an amount of dollars, which will be yielded in taxes by transferring the gas through this pipeline.
- Pipelines are unidirectional. If it is possible to transfer the gas from station X to Y (perhaps, through some intermediate stations), then the reverse transfer from Y to X is impossible.
- You need to find a route to transfer gas from the starting to the final station. A profitability of this route should be maximal.

Slide 4.

Pipelines and stations are edges and vertices of the graph. Target for this problem is to find path between two vertices with maximum profitability or cost.

One of useful methods to solve problems is to interpret it as a well-known problem. So main idea is adaptation of shortest path algorithm.

First approach is to use Dijkstra's algorithm. Profitability is positive integer, so we can try to change sign of all costs to negative, so path with minimum cost will have maximum cost in real, or change usage of min function to max function, so, on each step algorithm will choose maximum subpath.

Slide 5.

Let's check changing of signs. On the first step we will get start vertex S with Dijkstra's Score equal to zero. We should find edge to unused vertex, which gives minimum DS. This edge is edge between vertices S and F. As you can see on the third image, after relaxation DS is equal to -4, and vertex F is added to list of used.

Next edge to relax is edge between vertices S and A (4^{th} image). The last edge to be chosen is edge between vertices S and B (5^{th} image). On the 6^{th} image we have situation, where all vertices are covered. So, we can take DS of vertex F and it will be length of shortest path between S and F? This value is -4 or 4 if change signs again. But it is obvious, that maximum path should have length of 6 (vertices of the path is S-A-F). Result is wrong. Why does it happen?

Answer is hidden in logic of algorithm. Dijkstra's algorithm expects that after vertex was added to set of vertices with calculated weights, its score is already a length of shortest path to this vertex. All other paths were skipped, because they already have greater score, and if we add any edge with positive weight to them, they become even greater! But in case of negative weights, it isn't true, and some path can become shorter unexpectedly to Dijkstra's algorithm. That's why Dijkstra's algorithm doesn't work with negative weights.

Slide 6.

If we try to use max function instead of min function for all purpose inside algorithm, we will have the same issues with the same reason as on previous slide. It means, that Dijkstra's algorithm isn't applicable for this problem.

Slide 7.

Another approach is to use Bellman-Ford algorithm, because this algorithm can work with negative weights, and our graph doesn't have any negative cycles. Both ways will work well. But the main issue for Bellman-Ford algorithm, is that complexity equals to O(n*m), where n is number of vertices and m is number of edges. It isn't slow algorithm, but what if we can solve this problem for linear time?

Slide 8.

The third approach is to use topological sort and dynamic programming.

Let's check the problem's description again. "If it is possible to transfer the gas from station X to Y (perhaps, through some intermediate stations), then the reverse transfer from Y to X is impossible." It means, that our graph has no directed cycles. Such graph can be called as directed acyclic graph or DAG. As it was known from lectures, algorithm of topological sorting can be applied to DAG. But why do we need sorting?

We can calculate cost of path to vertexes from S in order of topological sort. It is important, because in this case, it is guaranteed, that paths to all predecessors of the vertex are already calculated, and we can easily choose the maximum sum of such path and the edge between predecessor and current vertex.

Please note, that this algorithm also can be used to calculate shortest paths but can be easily adapted. Let's check how it works and find its complexity.

Slide 9.

First step is topological sorting of graph. We will not focus on sorting algorithm; it was fully explained on lectures. Complexity of topological sorting O(n). Result of sorting on the slide: first vertex is S, next A, B and finally F.

Slide 10.

Let's set maximum cost of path to S as 0, and as some very big negative value to all other vertices, minus infinity, for example. In order of topological sort, we can start to calculate cost for each vertex. First vertex is source, skip it. In this case source is equal to S.

Next vertex is A. It has only one parent S. Cost of S is zero, cost of edge is 3. It means that cost of greatest path between S and A is 3.

The same for vertex B. Cost of greatest path is 2.

Slide 11.

For vertex F situation is a bit more complicated because it has 3 predecessors. So, we can check all of them. S has cost 0, and edge between S and F has cost 4. Total 4. Vertex A has cost of path 3 and edge between A and F is 3. Total 6, greater than previous candidate to greatest path. For vertex B cost is 2 and edge is 1. Total 3. It is less than 6. So maximum possible cost on path from S to F is 6.

Because of sorted order, on each step it is guaranteed that cost can't be greater, all predecessors are already taken into account.

Complexity of this part is O(m), and total complexity is linear: O(n + m).

Don't forget, that if there is no path from S to F – result is "No solution".

Slide 12.

Problem 1160 - "Network". You can go through the link to check full description. I will remind you the most important part

- There will be N hubs in the company, they can be connected to each other using cables. Since each worker of the company must have access to the whole network, each hub must be accessible by cables from any other hub.
- Since cables of different types are available and shorter ones are cheaper, it is necessary to make such a plan of hub connection, that the maximum length of a single cable is minimal. There is another problem – not each hub can be connected to any other one because of compatibility problems and building geometry limitations.

Slide 13.

Let's represent this problem in terms of graphs. Network is a graph; hubs are its vertices and cables are edges. By description of the problem this graph is connected and weighted. And the task is to get such subgraph that maximum length of edges is minimized.

You can see graph from problems sample on the screen. As result four edges were chosen. It is obvious, that resulted graph also should be connected and cover all vertices. But are all edges of example required? If we remove edge from vertex 1 to vertex 3, graph is still connected, all vertices are covered, and maximum length of edges is the same.

Slide 14.

It means, that resulted graph always can be a spanning tree. Two main properties of each tree is that tree is connected graph and tree has no cycles. Let's suppose that it is possible, that sometimes resulted graph can't be a spanning tree. In this case it either not connected (but it breaks requirements of

problem), or it should have cycles. But it is obvious, that we can break cycle by removing edges with greatest weight and resulted graph will be still ok to be a solution of the problem.

So, we should build a spanning tree, where maximum length of edges is minimized. Such tree is called minimum bottleneck spanning tree or MBST.

Bottleneck is the value of the heaviest edge in the spanning tree. Bottleneck of MBST from the example on the slide is 1.

Slide 15.

But how to find this tree in the graph? We only know algorithm for finding minimum spanning tree or MST. Check again the graph from example. Red edges are MBST, but it is obvious, that it is also MST for this graph.

Slide 16.

MST is necessarily MBST. Let's prove this fact.

Imagine graph G and two trees T and T', where T is MST of graph G and T' is MBST of graph G.

Let edge e be a bottleneck of tree T. By cut property we know, that there is exist such cut, which crossing edge e, that weight of any other edge crossing this cut is not less than weight of e.

Tree T' also should have an edge, which crossing this cut. Let it be e'. It's weight by cut property, mentioned earlier, should be greater or equal than weight of edge e.

Slide 17.

There are 2 possibility for weight of edge e'. Let's check them both.

The first possibility, is that weight of e' is strictly greater than weight of edge e.

By definition of bottleneck we can say, that bottleneck of T' is greater or equal to weight of e', which is greater than weight of e. But e is bottleneck of tree T.

It means, that bottleneck of T' is strictly greater than bottleneck of T. But T' is MBST, and its bottleneck should be minimal of all other spanning trees of the graph. We have contradiction.

The second possibility, is that weight of e' is equal to weight of e.

In similar way we have, that bottleneck of T' is greater or equal to weight of e', which is equal to weight of e.

It means, that bottleneck of T' is greater or equal to bottleneck of T. But T' is MBST, and its bottleneck is minimal of all other spanning trees of the graph. So, bottleneck of T' is equal to bottleneck of T.

Finally, because T' is MBST, it means, that T is also MBST, this completes the proof.

Slide 18.

But the opposite statement is wrong, MBST is not necessarily MST.

Let’s check simple graph with 3 vertices and two of its spanning trees.

First tree is both, MST (sum of its edges’ weights is 3) and MBST (bottleneck’s weight is 2).

Second spanning tree is also MBST, its bottleneck’s weight is 2. But sum of its edges’ weights is 4. This spanning tree is not MST.

Slide 19.

Now we know that it is enough to find MST for input graph. It can be done by Prim’s algorithm, which was explained on lectures.

Slide 20.

Mandatory task. You should implement source code for problem 1160 “Network”, 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 21.

You will have 1 problem for homework. It is problem 1162 “Currency Exchange”. You should solve it by yourself. Please, note, that report for this problem should contain explanation, which algorithm was chosen

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

Slide 22.

Let’s briefly check the tasks of the fourth test.

The first task was related to topological sort algorithm

Main rules and steps for this algorithm.

- Topological sort can be applied for Directed acyclic graph only
- Call Depth-First Search algorithm for this graph
- When DFS returns from vertex – all its children are sorted, insert it onto front of linked list
- DFS always iterates in alphabetical order by description
- Linked list is sorted graph

Slide 23.

Let’s analyze one example with more details. We can start DFS only from vertex A. On first step DFS gets 2 children of vertex A – vertices B and D. Because of alphabetical order, let’s use vertex B. Now DFS gets children of vertex B – vertices C and F. Because of alphabetical order, let’s use vertex C. Vertex C has no

children, DFS will return. Let's add C to the front of empty linked list. You can see this state at the third image. Then DFS returns to B and goes to its next child F. F has no children, returns to B and add F at the front of linked list. You can check the last image on this slide.

Slide 24.

DFS checked all children of vertex B, so it can go to next children of vertex A, it is vertex D. Vertex B is added to the front of linked list. Vertex D has 3 children, but vertices B and C were already visited, skip them. Next vertex is E and its only child F (images 1 and 2 on the slide). Vertex F has no children, add it to the front of linked list. All children of vertex E are visited, E can be added to the front of linked list, and the same for vertices D and A. Now all vertices of graph are visited, check images 3 and 4 for full result.

Slide 25.

Let's check ourselves and visualize resulted output of sorting. For each edge from a to b we can say, that a is sorted earlier than b.

Answer is A-D-E-B-F-C.

Slide 26.

In similar way result can be found for this graph. Answer is F-E-A-B-D-C.

Slide 27.

The last one is a bit trickier. You should note that vertices B, E and F are in cycle. That means, that our graph is not a DAG and can't be topologically sorted.

Slide 28.

Task 2 was related to usage of Dijkstra's algorithm and priority queue. Let's analyze one example with more details.

On the first step we will add to queue started vertex A and get it. On the second image you can see result of relaxing edges, incident to vertex A. Current shortest path to vertex B is 3 and to vertex D is 5. Both are added to the priority queue. Next top element of queue is B. On the third image you can see result of relaxing edges, incident to vertex B. Current shortest path to vertex F is equal 3 + 5 = 8 and to vertex C is 3 + 4 = 7. Vertices C and F are added to the priority queue.

Next top element of queue is D. Result of relaxing its edges and state of priority queue you can see in fourth image.

Slide 29.

Next top element of queue is C. Result of relaxing its edges and state of priority queue you can see in first image. Next top element of queue is F. Result of relaxing its edges and state of priority queue you can see in second image. And the last image have all covered vertices. Result is A-B-D-C-F-E.

Slide 30.

In similar way result can be found for this graph.

If algorithm starts from vertex A, answer is A-D-C-B-F-E, if algorithm starts from vertex F, answer is F-B-E-C-D-A.

Slide 31.

The third task is related to minimum spanning tree. We can use Prim's algorithm to find MST. Let's analyze one example with more details. In this case we can start from vertex A, add it to MST and A's incidental edge with lower weight than others. It is edge AB, so we can also add B to the MST.

On the second image you can see intermediate result.

Let's choose next smallest edge from MST to other vertices. It is edge BD. Add it to the tree with vertex D. Result of it on image three. Next smallest edge is DC. Add it with vertex C.

Slide 32.

Next edge to be added is BF and final is EF, which you can see on images in the slide.

Answer is AB; BD; CD; BF; EF.

Other possibility to solve this problem is to analyze all variants of answers.

Don't forget, that MST is a tree. So, it has no cycles. And also sum of weights of all edges are minimum.

Slide 33.

In similar way result can be found for these graphs.

Answer for the first graph is AB; AC; CD; CF; EF and for the second graph is AB; BC; CD; DE; EF.

Slide 34.

Answers to questions of task 4 can be found in lecture presentations

Short versions of answers:

Dijkstra's algorithm has better complexity but can't work with negative weights.

If graph strongly connected, it means that for all pairs of vertices A and B in directed graph there is exist directed path from A to B and from B to A.

Just connected graph for undirected graph means, that all pairs of its vertices are connected.

Topological sort is linear ordering of all vertices of graph such that if graph has edge a->b, so a appears before b

Classic examples of its usage are shortest path problem for DAG or computing optimal order of execution in dependency graph.

Slide 35.

Thank you for attention.