Contest overview and lecture.
Graphs and Flows.

Vasily Astakhov

1 Introduction

A short overview in English over the theme graphs and flows. This theme is very popular not only in ACM programming, but also in work process. We will make our path from the very basic problems and algorithms to there a lit bit more complicated appliance. Our today's lection purpose is not in learning the most efficient and hard algorithms in flow problems, but more in finding the ways where to apply simple flow methods. I hope it will be helpfull.

This is a draft material. I sincerely apologize for all the mistakes you can find here.

2 Preheat problems

These two problems were given to give your minds a starting burst, and also they are simple but interesting and can help you understand more the graph structures.

2.1 Problem A

Statement 1. *You were given a string of no more than 300 letters. Task was to find the number of trees for which it can be their description in left-right tour.*

Let the answer for substring from i to j will be $ans_{i,j}$. And $ans_{i,i} = 1$ for all i. Consider the first letter. It's a label of the root vertice, let us find in what moment we will first return back it can be every k from $i+1$ to j such that $s_i = s_k$. Than number of such a trees will be $a_{i+1,k-1} + a_{k,j}$. So we have to summarize over all k with above condition. This will give us the answer in $O(len^3)$ time.

It's a regular way to dealing with trees - using answer of subtrees and dealing with edges from root one by one.

2.2 Problem D

Statement 2. *You were given undirected weighted graph and two vertices S and F. You task was to find out all edges that increase the minimal distance between S and F.*

Deikstra algorithm is usefull in this problem. For a vertice A it finds shortest distances to all other vertices. It deal with graphs with non-negative weights. At every step we have a number of vertices with determined shortest distance and set (or a heap) with non-determined. We take the vertice with the smallest current undetermined distance and say that distance to it can be smaller than now and put it to the determined.(There we are using non-negativeness of edges) Than we take all the edges leading from this vertice to other vertices and refresh distances to them. And continue...

Back to the problem. Let us firstly find out what edges are on at least one minimal path between S and F. We can find the distances from the starting vertice $s[i]$ and from finish vertice $f[j]$. Than edge e from vertice i to vertice j and length c lies on the minimal path when $s[i] + c + f[j] = s[F]$. Let us sort all remaining ends of edges by $s[i]$ and than working with them as brackets sequences. If in some moment we have only one edge available moving between two numbers than this edge is important.

3 Matching problems

Finding maximal matching in bipartite graph is well-known problem. (In not bipartite graph it is more complicated). It can be solved by classical algorithm with alternating path. Given a matching M, an alternating path is a path in which the edges belong alternatively to the matching and not to the matching. Given not maximal matching we can find using dfs or bfs algorithms alternating path and therefore increase the size of matching. Algorithm works in $O(NM)$ operations. In the given problems graphs were of much bigger size and these simple algorithm did not work. But it gives us the opportunity to understand the structure of graph and find in the given cases more optimal ways to solve the problem.

3.1 Problem B

Statement 3. *You are given perfect matching in bipartite graph. Your task is to find all edges that can participate in perfect matching.*

Let's try to implement our previous ideas. If we are using the edge not from the matching we can consider that we already had the almost perfect matching and started to find alternating path from this edge. If we will succeed this edge is good, otherwise not. We will only succeed if we return back to the opposite vertice. So we have just to test it but it's too long! We have M edges to check and one test we make in M operations. But what our check looks like? We go from the first component by not matching edges and from the second by matching. So we can add orientation to the edges and also as a little trick we can add a matching adges from the first component too. In such a graph now we only need to know if we have gone by an edge, can we return back? And this problem of find the components of strong connectivity in directed graph. It has classical solution with the first tour by direct edges and second tour in order of time of exit by reverse edges.

3.2 Problem G

Statement 4. *You are given number of segments on the circle. Your task is to make the perfect matching of them with points.*

Of course the standart algorithm works here to long and we have to think about more clever algorithm. At first we must mention that if number of segments is greater than number of points the answer is no. As common method in circle problems we have to try to solve this problem on a segment and then try to somehow to cut the circle. This method works here but is a little bit more complicated.

How to solve segment problem? We have to sort our segments by their left end and consider points from left to right. The first point we must give to the segment with the leftmost right end available (if we give it to another segment in optimal solution then we can without any problem swap this point to this current segment). Then we go to the next point and add a bunch of segments here and continue our process. The success of this process is equivalent to the question.

How we get to the circle? We need to overview our problem. When we can to failure? When there are K segments that cover less than K points. Using the Hall's lemma we also get that this the necessary condition of our failure. Let us cut the circle somewhere and copy the segment to the right deleting right parts of segments on the left and left parts on the right. If we consider any N continues points we will get the cycle shifted picture of our circle on the segment. Therefore if there are K segments covering less than $K-1$ points on the circle we will find them on this doubled segment and vice versa. Now we can use our segment algorithm and solve the problem.

Unfortunately our solution doesn't give us required partion but answers on question Yes or No correctly.

4 Max-flow problems

Next important theme is Max-flow. Consider the directed weighted graph with two special vertices S and F. Flow is function from edges to numbers such that. $f[e] \leq capacity[e]$,

$$\Sigma_{start[e]=S} capacity[e] - \Sigma_{finish[e]=S} capacity[e] \geq 0,$$

$$\Sigma_{start[e]=F} capacity[e] - \Sigma_{finish[e]=F} capacity[e] \leq 0,$$

$$\Sigma_{start[e]!=S,V} capacity[e] - \Sigma_{finish[e]!=S,V} capacity[e] = 0$$

. Value of flow is equal to

$$\Sigma_{start[e]=S} capacity[e] - \Sigma_{finish[e]=S} capacity[e] = -(\Sigma_{start[e]=F} capacity[e] - \Sigma_{finish[e]=F} capacity[e])$$

. Maxflow is flow with maximal value.

4.1 Augmenting paths algorithm

Algorithm is following. We make for every edge the reverse edge with zero capacity and require that $f[e] = -f[reversed[e]]$. We can find every path from S to V using the edges satisfiyng $f[e] < capacity[e]$. Usually we take the shortest path. (In some special problems other ways can be considered). Than we find the minimal value on this path for $-f[e] + capacity[e]$ and add to every $f[e]$ on this path and update reversed edges using the condition. It can be easily seen that all above conditions are satisfied. To prove that this algorithm follows to maximal flow we need to consider the moment when it stops and all the vertices in the same component with S. We can see that all outgoing flow is equal to c, all outgoing edges are full and all ingoing edges are zero. So we have a cut of weight c(and every cut is not smaller than flow and if equal than we get mincut=maxflow)

4.2 Problem C

Statement 5. *You are given undirect graph and want to deliver K items from S to V moving each item on one edge in a second and not using an edge more than one every second in minimal time.*

Here we don't have a restriction on the overall capacity of edge but on the capacity per second. So we have to multiply our vertices (and edges) by moments of time. We can increase time one by one waiting for the moment where the overall flow of value K will be found. The constructing using flow values is also quiet simple. You only have to save the number of container in every vertice.

4.3 Problem I

Statement 6. *You are given direct non-cyclic unweighted graph and want to cover it with the least number of paths.*

In this problem all is reversed. You have to satisfy minimal capacities at edges and want to minimize the flow. We can find the difference between ingoing and outgoing flow. Our purpose now to send maximum extra flow to the vertices with negative balance - therefore we find the maximal flow between them. Other extra flow will be the answer for our problem. The proof is simple: if the vertice is negative - we start the route, than go by all initial edges and edges from the flow and finish in positive vertices. From the other side we can substract initial edges flow from the answer and get our special flow.

4.4 Word over more advanced algorithms

There are also a big number of more advanced flow algorithms which you can find for example in Wiki and then using links to materials. They are faster but a little more complicated.

5 Min-cut problems

As we already mentioned mincut is equal to maxflow but it leads to deeper understand of flow structure.

5.1 Problem M

Statement 7. *You are given a possible matrix of mincuts for undirected weighted graph. You have to find example of such a graph or say that they do not exists.*

Interesting fact is that every undirected weighted graph is flow-equivalent (or mincut) to just a chain. Now we will prove it.

Such a graph exists if and only if for every three ordered values of cuts between them $a \leq b \leq c$ is true that $a = b$. Two proof only if statement let us find a cut equal to a in graph. Than the third vertice is in one of the two components, so cut to one of the other two vertices is no more than a. Proof is ended. To prove if part we will construct our graph by induction (and it will be a chain!). Let us find the minimal value of cut d and two vertices i and j according to it. Let us put the vertice with cut to one of this vertice more than d in the component of this vertice and all other vertices in the component of i. Let us prove that cut between any two vertices from different components is equal to d. For vertice j it follows from the condition, for other vertice v in j-component we have a triplet k, j, v such that flow between k and v is equal to d and between j and v is greater than d. Now we can construct graphs in components by induction and connect the ends of chains by the edge of weight d.

5.2 Problem F

Statement 8. *You are given the set of vertices of different cost with the condition that if you get a vertice you get all of its descendants. You have to find the set with maximal sum.*

Why it's the cut problem when we cannot cut? Yes we can, but not the edges! We have to cut out some big negative vertices and take some this good positive vertices with the condition that second have now descendants between the first. So Let us add vertices S with edges of weight W_i to positive vertices, and F with edges of weight $-W_i$ from negative vertices. And all other vertices have infinite weight. let us consider the minimul cut in this graph. It destroys some S edges and some F edges so that there is no connection between not destroyed positve vertices and not destroyed negative vertices. So we can take not destroyed positive vertices and destroyed negative vertices as our set with sum equal to $P_i - N_i = SumAllPositive - MinCut$

5.3 Problem H

Statement 9. *In undirected unweighted graph you have to find set of vertices with maximal density of edges between them.*

This problem seems similar to max-clique problem, but has polynomial solution using flows! Let us do some magic! At first we will use binary search for the needed density level. Now we have to

check existance of subgraph (and find it) with $NumEdgesInside - M * NumVerticesInside \geq 0$. Than

$NumEdgesInside - M \cdot NumVerticesInside = NumEdges - NumEdgesToOutside - NumEdgesOutsideCompo$

$= NumEdges - SumDegreeOutsideVertice \cdot 0.5 - NumEdgesToOutside - M \cdot NumVerticesInside$

So we have to minimize the sum

$$SumDegreeOutsideVertice \cdot 0.5 + NumEdgesToOutside + M \cdot NumVerticesInside$$

Now we add edges from S to all vertices with weight M and from every vertice to F with weight $Degree_i * 0.5$ and find the minimal cut. The cutted edges from S go to the best inside component.

6 Min Cost Max Flow

Now beside the capacity we have the multiplier for cost on every edge's flow. What can we do if we want to find the mincost flow of value V? Just the same, but finding the shortest path in the graph. And if there are no negative cycles then we can find it and even more: no negative cycle will appear. If there are initially negative values then we can only use Bellman-Ford algorithm.

6.1 Using Deikstra

If all the edges are initially non-negative we can use Deikstra algorithm at first step. But what we can do next? We can change the weights of edges not changing the order of routes but leaving no negative edges. We have to $d_s - d_f$ to every edge.

6.2 Problem L

Statement 10. *You are given the graph where cost increases with the flow, you have to find mincost flow in it.*

The only negative edges here are edges from customers cities to F so we can add a constant to them to make it positive. We can divide each edge on edges of increasing weight and then use the standart algorithm (but finish not in max flow but in the mincost!). But it's to much edges. But it's simple to see that from bunch of edges it's enough to use only the first not taken values (others are just bigger).

6.3 Problem J

Statement 11. *You are given the cost of securing segment i, j. You have to get nessasary amount of guards each day for min cost.*

At first we optimize our matrix using the fact that we can use the segment i, j for all inside segments. With that matrix we have to see at the diagram of heights and add edges from the end of the height H to it's next appearance of weight zero (there is no need for H guards here). And of course all the edges from matrix. Now it's time to find min cost max flow. It's easy to see that constructed flow will give us good answer (there are at least H guards due to the fact that there are no more than $Max_H - H$ zero edges at this moment). From the other side it's a good exercise to see that every optimal settle of guards (in sence of cutting there time if it's not needed) leads to some flow in our system.

7 Counting number of cuts

Some times there are similar to the cut problems but with other structur. For example counting number of specifical cuts in graph. The problem with cuts of size 1 is well known and is the problem of finding bridges. It has a lot of solutions. But what if we are finding number of cuts of size two.

7.1 Problem E

Statement 12. *You have two find number of cuts of size two in the graph.*

Let us think about one cut. We can make dfs and for every subtree find the minimal entring number of all the vertices connected to this subtree (without parent edge) using dynamic programming. Now using this information it's easy to see when the edge is cut (when the previous number is greater then parents number). But what we can do with 2 edges? Simple solution is to look through all $N-1$ edges of dfs-tree cut it out and than use previous algorithm. But this is $N \cdot M$ and possibly too long. How to do it faster? There are three cases:

- edge from dfs-tree is a cut than we can take any other edge
- edge from dfs-tree is not a cut but there is only one edge from bottom to up not from dfs-tree (we can save to minimal entring values for each subtree to solve this case)
- there are two not-cuts from dfs tree but there are only edges from bottom part to upper part without middle part. This is most complicated case. For every not-cut edge while we have not reached the maximal outgoing edge (we can save them all in $N \cdot N$ matrix). For upper part we can save minimal outgoing edge and refresh it using newly added vertices and other branches for no more than N operations (finding two maximums).

8 Euler cycles and spanning trees

No more flows and cuts. Just a nice problem.

8.1 Problem K

Statement 13. *You are given a graph with vertice degrees 2 and 4. You need to find minimal euler cycle where for every pairing of edges in vertice is it's own cost.*

At first we can see that 2 out of 3 pairings in the 4-degree vertice keep the cycle connected. So we are interested only in 2 minimal values. Let us take all the minimal values and divide graph in cycles. Every switch in vertice connected two components we can use cycles as vertices and switches as edges and find minimal spanning tree. It will be the answer. As a proof we can use induction.

For finding minimal spanning tree we can use modified deikstra algorithm where distances are equal to weights of edges.

9 The end

This contest was prepared using materials from NCPC,GCPC,NEERC and Petrozavodsk Summer and Winter Series.