Second Level Algorithms

Prof. Palash Dey

Department of Computer Science and Engineering

Indian Institute of Technology, Kharagpur

Week – 08

Lecture 38

Welcome to the 38th lecture of second level algorithms course. In this lecture we will start a new and very important topic namely min cut problem. So, let us begin. So, what is the min cut problem? In this problem the input are undirected graph G and the goal or output is a cut x comma v minus x. So, x is not equal to empty set or full set. So, a cut is a partition of the vertices into 2 non-empty sets. it is a there is no special vertices s or t in the s t cut problem. So, it is just a cut.

So, any partition of the vertex set into 2 non empty subsets is a cut. So, the output is a cut of maximum size, size or capacity of the cut is the number of edges, this is an undirected So, number of edges of the graph with one end point in X and other end point is in $V \setminus X$ ok. So, in this problem edges do not have weight. If edges have weight, then the capacity or size of the cut is defined to be the sum of the weights of the cut edges. The edges with one endpoint in X and the other endpoint is in not in X, they are called cut edges and for weighted graphs, the capacity of the cut is the sum of the weights of all those edges. The algorithm that we will see can be easily generalized to weighted graphs also. So, for the sake of simplicity and understanding the core concept clearly, we will focus on unweighted graphs. So, let us see an example of cut to ensure that all of us have understood the concept of cut in crystal clear way. So, a cut is just a partition of the vertices into two sets. So, the cut shown in the picture, the size is three because there are three edges whose one end point is in one partition and the other end point is in other part, but this is not the min cut. The min cut there are actually multiple min cuts, one min cut is this that you take g in one part and rest of the vertices in the other part and there are only two edges.

Size of the cut defined as ABCD is 3 whereas, the size of the other cut which is a min cut. So, we want to design an algorithm for finding a minimum cut of the graph of the input graph. So, here are some natural algorithms based on flow.

Polynomial time algorithms based on maximum s-t flow. So, here is the input graph G, and I am looking for a bipartition of the vertices into X and $V \setminus X$. But I know, given two vertices s and t. how to find a minimum s-t cut. So, I try all possible choices of s and t. There are $n(n-1)/2$ such choices, and I know for one of them, s will belong to the minimum cut X, which of course exists, but I do not know, and t will belong to $V \setminus X$. So, for that particular choice of s and t, the minimum s-t cut will coincide with the minimum cut.

So, the algorithm is we iterate over all possible $n(n-1)/2$ choices of s, t and V. For every choice of s and t, we use any maximum s-t flow algorithm to compute a minimum s-t cut. So, for all $n(n-1)/2$ choices of s and t, we find the minimum s-t cut, and then we output the minimum over all minimum s-t cuts found, ok. So, the algorithm is clearly correct, or the correctness of the algorithm follows from the observation that there exists at least one choice of s and t such that s belongs to X and t belongs to $V \setminus X$ for a min cut of G, ok.

So, this is clearly correct. What is the time complexity? So, the time complexity is $O(n^3)$ for finding any minimum s-t cut, as we need to run the maximum s-t flow problem, which takes $O(n^3)$ time. This is the best we know using the push-relabel algorithm, and we have to run this algorithm $n(n-1)/2$ times. So, the overall runtime is $O(n^5)$.

This can be improved using a simple observation. For undirected graphs, which we are considering here. We can fix the choice of s to any vertex and iterate over the remaining $n-1$ vertices for t and solve for the maximum s-t flow for all such pairs of using which we find a minimum s-t cut using any maximum s-t flow algorithm. For every $n-1$ choices of s and t, the correctness of this approach follows from the observation that there exists a minimum cut (X, $V \setminus X$), such that s belongs to V—sorry, s belongs to X, and t belongs to $V \setminus X$. Why is this the case? So, suppose this is V. Consider a min cut.

And suppose we have fixed s to a vertex and suppose s does not belong to x here. Then, if s belongs to $V \setminus X$, pick any vertex of x as t, and then instead of calling the left side x, call the right side x, and this is $V \setminus X$. There exists a partition of the vertices x comma v minus x such that s belongs to x and t belongs to v minus x, OK. So, with this, we have improved the running time substantially. So, what is the time complexity now? Big O of —we are now running $n-1$ instances of max flow, each takes $O(n^3)$ time using push-relabel. That is the best we know, so the overall runtime is $O(n^4)$.

Now we will see another very simple algorithm, but it is a randomized algorithm due to Karger, which goes by the name Karger's min cut algorithm. It is very simple because

notice that all these flow-based algorithms are substantially complicated. All the algorithms are very non-trivial. And the Karger's min cut algorithm that we will see is extremely simple. So, let us see that algorithm next—Karger's min cut algorithm. So, again, before discussing this algorithm with informal full details, let us understand the working of this algorithm using a toy example.

So, suppose this is an input graph. So what the Karger's algorithm does—it is an iterative algorithm, and this loop continues if we have more than two vertices. So what we do is we pick an edge uniformly at random. So there are so many edges. Pick any edge uniformly at random and contract them.

What do we min by contraction? So, let us first pick any edge. So, suppose I pick this edge CF uniformly at random and then let us understand the contraction operation. The other vertices remain the same. So, let us first draw the other part of the graph which are not endpoints of this contracted edge CF.

Then contraction means you delete the endpoints of the edge you are contracting, namely CF in this case, and introduce a new vertex which let us call CF. For every edge which has any of the endpoints in either C or F, make it incident on the new vertex. For example, I had an edge between D and C, so I draw an edge between D and CF. Similarly, I had an edge between B and C, so I put an edge between B and CF. Similarly, I had an edge between E and F, so I put an edge between E and CF.

And I had an edge between F and G, so I put an edge between G and CF, this new vertex. The edge CF, which had both endpoints in it or which is the edge being contracted, is not present in our contracted graph, OK? So, next again, let us uniformly pick another edge. So suppose I pick the edge between B and CF. Then how does it look like? Let us again draw the remaining graph. So let us not touch the edge between B and CF. EG is there and I delete the vertices B and CF and put a new vertex, let's call it BCF.

There was an edge between D and CF that edge I put here. there was an edge between A and B so I put that edge here there was an edge between D and B so I put that edge here also you see because of the contraction operation the graph may end up having parallel edges and we retain the parallel edges that is very important we do not get rid of parallel edges we retain the parallel edges in Kargar's min-cut algorithm then we have an edge between E and CF that I put here then we had an edge between E and B that also I put here and we had an edge between G and CF that I put here that is how the graph look like

next let us pick another edge uniformly randomly so suppose I pick AD So it's a randomized algorithm. So A and D gets deleted.

A new vertex which I am labeling A D gets introduced. The other vertices are there. And then now you see we have three parallel edges. Again I pick an edge uniformly randomly.

So, suppose I pick this edge EG, then the graph looks like ok and next I pick another edge uniformly randomly. So, suppose I pick this edge. So, the graph looks like and now we have only two vertices and the by partition of the original vertices that this new vertices ah represent that is the output of the Karger's min-cut algorithm.

Its output in this case is the following by partition. One part has vertices A, B, C, D, E, and the other part has E, F. And the other part has the remaining vertices. So, in the next class, we will see the pseudocode of this algorithm and analyze Karger's min-cut algorithm. Okay.

So, let us stop here. Thank you.