CSL356 Oct 6, 7, 9

TUTORIAL SHEET 8

1. **[KT-Chapter7]** Consider the following problem. You are given a flow network with unit-capacity edges: it consists of a directed graph $G = (V, E)$, a source s and a sink t, and $u_e = 1$ for every edge e. You are also given a parameter k. The goal is delete k edges so as to reduce the maximum $s - t$ flow in G by as much as possible. In other words, you should find a set of edges $F \subseteq E$ so that $|F| = k$ and the maximum $s - t$ flow in the graph $G_0 = (V, E \setminus F)$ is as small as possible. Give a polynomial-time algorithm to solve this problem.

   Solution: First observe that by removing any k edges in a graph, we reduce the capacity of any cut by at most k, and so, the min-cut will reduce by at most k. Therefore, the max-flow will reduce by at most k. Now we show that one can in fact reduce the max-flow by k. To achieve this, we take a min-cut X and remove k edges going out of it. The capacity of this cut will now become $f - k$, where f is the value of the max-flow. Therefore, the min-cut becomes $f - k$, and so, the max-flow becomes $f - k$.

2. **[KT-Chapter7]** In a standard $s - t$ maximum flow problem, we assume edges have capacities, and there is no limit on how much flow is allowed to flow through a node. In this problem, we consider the variant of the maximum flow and minimum cut problems with node capacities. Let $G = (V, E)$ be a directed graph, with source s, sink t , and non-negative node capacities u_v for each $v \in V$. Given a flow f in this graph, the flow though a node v is defined as $\sum_{e \in \delta^-(v)} f_e$. where $\delta^-(v)$ denotes the edges coming into v. We say that a flow is feasible, if it satisfies the usual flow-conservation constraints and the node-capacity constraint: the flow through a node v cannot exceed c_v. Give a polynomial-time algorithm to find a $s - t$ maximum flow in such node-capacitated network. Define an $s - t$ cut for node-capacitated networks, and show that the analog of the Maximum Flow Min Cut theorem holds true.

   Solution: We construct a new graph H where for every vertex v in G, we have two vertices v_i and v_o. There is an edge from v_i to v_o of capacity c_v. If the graph G has an edge (u, v), then we add an edge (u_o, v_i) in H of infinite capacity. Now notice that any flow in H entering v_i must go through the edge (v_i, v_o) and so the total amount of flow entering v_i (or leaving v_o) cannot exceed c_v. Now it is easy to check that a flow of value v in G results in a flow of the same value in H and vice-versa.

3. **[KT-Chapter7]** Let M be an $n \times n$ matrix with each entry equal to either 0 or 1. Let m_{ij} denote the entry in row i and column j. A diagonal entry is one of the form m_{ii} for some i. Swapping rows i and j of the matrix M denotes the following action: we swap the values m_{ik} and m_{jk} for $k = 1, 2, \ldots, n$. Swapping two columns is defined analogously. We say that M is re-arrangeable if it is possible to swap some of the

pairs of rows and some of the pairs of columns (in any sequence) so that after all the swapping, all the diagonal entries of M are equal to 1.

(a) Give an example of a matrix M which is not re-arrangeable, but for which at least one entry in each row and each column is equal to 1.

(b) Give a polynomial-time algorithm that determines whether a matrix M with 0-1 entries, is re-arrangeable.

Solution: Try (a) yourself. Draw a bipartite graph G where we have n vertices on both sides. We have an edge between vertex i (on the left side) and vertex j (on the right side) iff $M(i, j)$ is 1. Now, show that such a swapping exists iff G has a perfect matching.

4. Show that any $s - t$ flow can be written as a sum of flows along at most m $s - t$ paths and cycles.

Solution: Consider the edges which have positive flow – let this graph be H. If there is a cycle in H, let δ be the minimum flow along any edge in this cycle. We first remove δ units of flow along this cycle. This will reduce the flow along any edge in this cycle by δ units (and note that flow-conservation will still hold). Note that flow along some edge will become 0, and so, we will remove such edge(s) from H. Similarly, if we find a path from s to t in H we can remove δ units of flow along such a path, where δ is the minimum flow along any edge in this path. Again, flow on some edge will become 0. Thus, this process of removing a path or a cycle will happen at most m times.

5. Consider the following modification to the Ford Fulkerson algorithm: among all possible s-t paths in the residual graph, pick the one such that the maximum amount of flow can be sent along this path. How will you find such a path ? Show that you will need at most $\log(nU)$ iterations. What is the overall running time ?

Solution: Let $v^\star$ be the max-flow value in G. Suppose we have a flow f of value v in the graph G, and let G_f be the residual graph. Then, the max-flow in G_f is $v^\star - v$. Using the above problem, we know that there is a path from s to t in G_f where we can send at least $(v^\star - v)/m$ units of flow. Thus the max flow in new residual graph will be at most $(1 - 1/m)(v^\star - v)$. From this we can see that after i iterations, the max flow in the residual graph will be at most $v^\star(1 - 1/m)^i$. Since everything is integral, we will stop when $v^\star(1 - 1/m)^i$ becomes less than 1. Since $v^\star \leq nU$, we see that $i \leq O(m \log(nU))$.

6 **[KT-Chapter7]** Consider a set of mobile computing clients in a certain town who each need to be connected to one of several possible base stations. Suppose there are n clients, with the position of each client specified by its (x, y) coordinate in the plane. There are also k base stations; the position of each of these is also specified by its (x, y) coordinate in the plane. For each client, we wish to connect it to exactly one of the base stations. Our choice of connections is constrained in the following ways: (i) there is a range parameter r, and a client can only be connected to a base station that is within distance r, (ii) there is a load parameter L, and no more than L

clients can be connected to any single base station. Your goal is to design a polynomial time algorithm for the following problem – given the above data, decide whether it is possible to assign every client to a base station subject to the two constraints.

Solution: This is an application of max flow problem. Have one vertex for every client and one vertex for every base station. Add two extra vertices s and t. Now, add an edge from s to every client vertex and capacity of this edge is 1. Similarly add edges from every base station vertex to t of capacity L. Finally, add an edge from a client vertex to a base station vertex if the client can be assigned to this base station. The capacity is infinity (or 1, does not matter).

7 **[KT-Chapter7]** Give a polynomial time algorithm for the following minimization analogue of the max-flow problem. You are given a directed graph $G = (V, E)$, with a source s and sink t, and numbers (capacities) l_e for each edge $e \in E$. We define a flow f, and the value of a flow, as usual, requiring that all nodes except s and t satisfy flow conservation. However, the given numbers are lower bounds on edge flow, i.e., they require that $f_e \geq l_e$ for each edge e, and there is no upper bound on flow values on edges.

(a) Give a polynomial time algorithm that finds a feasible flow of minimum possible value (Hint: Start with any flow from s to t which obeys the lower bounds, and try to send a maximum flow from t to s in a suitable modification of the graph G).

(b) Prove an analogue of the max- flow min-cut theorem for this problem.

Solution: For part (a), first send a flow from s to t such that all lower bounds are satisfied. We can easily do this as follows: for every edge $e = (u, v)$ find a path from s to t which goes through (u, v), and send l_e amount of flow on this path. Let f denote this flow. Now that we have a feasible flow, we will try to reduce the flow as much as possible, but we do not want to reduce the flow on an edge below l_e. Therefore, we reverse every edge, and make it's capacity equal to $f_e - l_e$. Now we send a max flow from t to s in this "reversed" graph.