INDIAN STATISTICAL INSTITUTE

Mid Semestral Examination

M. Tech (CS) - II Year, 2011-2012 (Semester - III)

Advanced Algorithms for Graphs and Combinatorial Optimization Problems

Date : 24.09.2011 Maximum Marks : 60 Duration : 3.0 Hours

(Q1) A weighted directed graph $G = \{V, E\}$ satisfies the following conditions:

(i) the edges leaving the source vertex s can have negative weight edges,

(ii) all other edges have non-negative weights and

(iii) there are no negative weight cycles.

Prove or disprove the following statement: Dijkstra's shortest path algorithm finds the correct shortest path in G from s. [8]

(Ans:) Refer Figure 1. The notations in the Figure 1 have the usual meaning as discussed in the class. The vertex $w \in Y$ is the first vertex along path p ($= p_1 \rightsquigarrow x \rightarrow w \rightsquigarrow p_2$) in Y. Path p is the shortest path from s to y. For w, the vertex x is its immediate predecessor in X, the set for which the shortest paths have been determined. Let $y \in Y$ be the first vertex for which $\lambda[y] \neq \delta(s, y)$, where λ denotes the estimate of the shortest path and δ denotes the shortest path. We will basically obtain a contradiction to prove that $\lambda[y] = \delta(s, y)$. So, for a

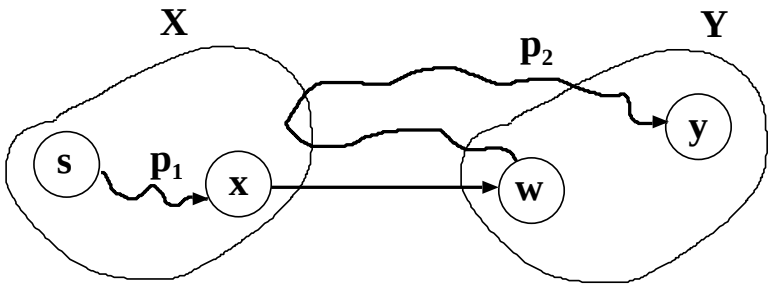

Figure 1: Correctness proof of Dijkstra's shortest path.

contradiction, we assume $\lambda[y] \neq \delta(s, y)$. First, we claim that $\lambda[w] = \delta(s, w)$ when y is added to X. Note that $x \in X$, and y is the first vertex for which $\lambda[y] \neq \delta(s, y)$ when it is added to X. So, $\lambda[x] = \delta(s, x)$ when x was added to X. Surely, edge (x, w) was relaxed at that time and as because a sub-part of a shortest path is also a shortest path, $\lambda[w] = \delta(s, w)$.

$$\begin{aligned} \lambda[w] &= \delta(s, w) && (1) \\ &\leq \delta(s, y) \text{ (edge weights are non-negative between w and y.)} && (2) \\ &\leq \lambda[y] && (3) \end{aligned}$$

On the other side, $\lambda[w] \geq \lambda[y]$ because with both w and y in Y, y was chosen first. So, both these inequalities taken together,implies $\lambda[w] = \lambda[y]$. Now, $\lambda[w] = \delta(s, w)$, and $\lambda[y] \geq \delta(s, y)$. With $\lambda[w] = \lambda[y]$, we surely have $\delta(s, y) = \lambda[y]$. So, it is the line number 2 in the above equation, that demands non-negative edge weights.

First observe that s is the first vertex to be added to X and $w, y \neq s$ as shown in Figure 1. The above proof only requires non-negativity of edge weights when it deduces $\delta(s, w) \leq \delta(s, y)$

to jump from Equation (1) to Equation (2). Both, w and y are in Y and cannot be in X. The vertices y and w can be the same, but in that case also, the proof holds; you write $\lambda[w] = \delta(s, w) = \delta(s, y) \leq \lambda[y]$.

(Q2) Show that for any bipartite graph, the maximum size of a matching is equal to the minimum size of a vertex cover. [10]

(Ans:) This question basically asks you to prove König's theorem.

Let $G = (V, E)$ be a bipartite graph with $V = X \cup Y$ where X and Y are the partite sets and M is a maximum matching in G. Recall that there should not be any augmenting path in G corresponding to a maximum matching M.

Let $V_c \subseteq V$ be the minimum vertex cover in G. Obviously, $|V_c| \geq |M|$.

We first need to pick a vertex cover $U \subseteq V$ from M. From every edge in M, we choose one of its endpoints in U as follows:

Choose the endpoint in Y if some alternating path ends in that endpoint.

Else, choose the endpoint in X.

We can see that $|U| = |M|$. So, if we can show that U forms a vertex cover, then surely U is a minimum vertex cover.

Let xy be an edge in G with $x \in X$ and $y \in Y$. If $xy \in M$, then there is nothing to prove. So, we consider the case when $xy \notin M$. Since M is a maximal matching, both x and y cannot be free vertices. So, there exists and edge $x'y' \in M$ so that either $x' = x$ or $y' = y$. We now consider the two cases:

$y = y'$**:** Here y is a matched vertex. If x is not matched, then xy is an alternating path and then y has to be chosen in U.

$x = x'$**:** Here x is a matched vertex and if $x \in U$ then we have nothing to prove. So, assume that $x \notin U$. Now for the edge xy' (same as $x'y'$), $y' \in U$ because at least one endpoint of a matched edge was picked in U. Therefore, some alternating path P ends at y'. But now there should also be another alternating path P' ending at y. Also note that M is a maximum matching and hence P' cannot be an augmenting path. So, y cannot be a free vertex; it has to be matched. Then, surely y has been picked in U as all matched vertices in Y were picked in U which had an alternating path ending at it.

(Q3) Let $G = \{V, E\}$, be a bipartite graph with $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \phi$. Let G' be the corresponding flow network where you have a source vertex s connected to all vertices in V_1 and a sink vertex t connected from all vertices in V_2 and the edges in G are directed from V_1 to V_2 in G'. Find out a non-trivial upper bound on the length of any augmenting path found in G' during the execution of the Ford-Fulkerson algorithm. [6]

(Ans:) An augmenting path is a simple path from the source s to the sink t in the residual graph G'_f of G'. G is a bipartite graph. So, there will be no edge between any two vertices in V_1. Similarly, there will be no edge between any two vertices in V_2. So, these edges will also not be present both in G' and G'_f. To go from s to t, we have to take the following route: $s \rightarrow v_1 \in V_1 \rightarrow v_2 \in V_2 \rightarrow t$. Though, edges in G' only go from $v_1 \in V_1 \rightarrow v_2 \in V_2$, but the edges in the residual graph G'_f can go in both directions. Thus, the augmenting path will be s followed by a sequence of vertex pairs, one from V_1 and the other from V_2, and finally the sink t. Now, the question is how many such vertex pairs can there be? Obviously, $\min\{|V_1|, |V_2|\}$ as no vertex can be used twice. So, the number of edges will be $2 \min\{|V_1|, |V_2|\} + 1$.

(Q4) $G = (V, E)$ is a directed graph with a source vertex $s \in V$ and a sink vertex $t \in V$. Each edge e has a positive integral capacity value. Someone has solved for you the integer maximum $s-t$ flow in G defined by a flow value $f(e)$ on each edge e. Now, that person picks up an arbitrary edge $e \in E$ and increases its capacity by 1 and asks you to find out an integral maximum $s-t$ flow in the new capacitated graph. Design and analyze an efficient algorithm to solve the problem. [12]

(Ans:) As the change done to the network is minor in terms of a change only in an edge capacity, we can do better than starting afresh. First note that, the maximum flow in the new capacitated network cannot be less than the previous max-flow. For this, we can have a small observation.

Observation 1 *Let the new capacitated network as stated above be G'. Also, let the value of the maximum flow in G be $\vee_f$. The value of the maximum flow, $\vee'_f$ in G' is either $\vee_f$ or $\vee_f + 1$.*

Proof: Since, the max-flow in G satisfies all the conditions for being a feasible flow in G', we have $\vee'_f \geq \vee_f$. By the Max-Flow Min-Cut theorem, consider an $s-t$ cut (A, B) in G. If the the edge whose capacity was altered in G', does not belong to the cut (A, B), then the capacity of the cut remains the same. Else, the capacity of the cut (A, B) increases by at most 1. So, the statement follows. □

So, it is all about finding an augmenting path, if it exists, in the residual graph G'_f of G_f. If an augmenting path does not exist in G'_f, then we have a max-flow. Else, an augmenting path exists which can increase the flow at most by 1 to get $\vee_f + 1$. Now, as flows are integral, there will exist no more augmenting path, implying a max-flow. To get this augmenting path, we spend only $O(\mid V \mid + \mid E \mid)$ time to get to t from s.

(Q5) Let $G = (V, E)$ be a graph and let $\overline{G}$ be its complement graph. Let $\chi(G)$ be the chromatic number of the graph.

A *clique cover* of size k is a *partition* of the vertices in V into sets of vertices $V_1, \ldots, V_k$ such that each set of vertices in V_i, $i = 1, \ldots, k$, forms a clique. Let $k(G)$ denote the size of the smallest possible clique cover of G.

Now prove or disprove the following statement: $k(G) = \chi(\overline{G})$. [8]

(Ans:) Take a clique cover of G of size $k(G)$. If we now look at $\overline{G}$, then each partition of G can be colored using a single color in $\overline{G}$. Thus, $k(G) \geq \chi(\overline{G})$.

Similarly, one can start with a coloring corresponding to $\chi(G)$ and show that vertices in the same color class would form a clique in $\overline{G}$. Thus $\chi(G) \geq k(\overline{G})$.

Taking the above two together, the result follows.

(Q6) A graph is called k-regular if the degree of each vertex of the graph is k, where $k > 0$ is a positive integer. Let $G = (X \cup Y, E)$ be a k-regular bipartite graph with $|X| = |Y|$. Now, prove or disprove the following statement.

G has a perfect matching. [6]

(Ans:) For the given graph, any subset of vertices would have its neighbor set to be at least equal or larger because of the k-regular condition. So, Hall's theorem applies and the graph has a perfect matching.

(Q7) Construct a simple graph $G = (V, E)$ with $|V| = 2n$ and $|E| = n^2$ such that G has exactly one unique perfect matching. [12]

(Ans:) Let $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$ be two sets of n vertices each. Let $V = X \cup Y$ where $X \cap Y = \emptyset$. Connect all vertices in X among themselves to have $\binom{n}{2} = \frac{n(n-1)}{2}$ edges. For any vertex $y_i \in Y$ $(1 \leq i \leq n)$, connect it to all vertices $x_j \in X$ where $1 \leq j \leq i$. This gives us $\frac{n(n+1)}{2}$ edges. Thus, in total we have $\frac{n(n-1)}{2} + \frac{n(n+1)}{2} = n^2$ edges in E. An example with $|X| = |Y| = 4$ is given in Figure 2.

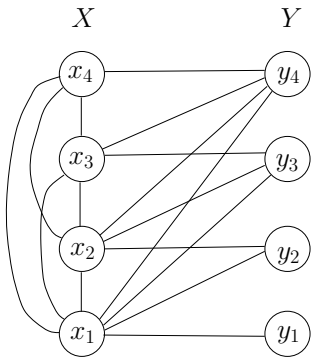

Figure 2: An example with 8 vertices and 16 edges.

(Q8) Let $G = (V, E)$ be an undirected weighted graph with weights $w(e) > 0$ for each edge $e \in E$. A *cut* in G is a partition of V into two non-empty subsets $X, \overline{X}$, such that $X \subset V$ and $\overline{X} = V \setminus X$. The *weight* of the cut is the sum of the weights of the edges going across the cut, i.e., the edges going from vertices in X to vertices in $\overline{X}$. Design and analyze an algorithm to compute the *minimum cut* in G. A *minimum cut* in G is the cut of minimum weight. [10]

(Ans:) We would do it using network flows and the minimum cut would come via the max-flow min-cut theorem. We design the flow network as follows.

Set aside any vertex as the source s. For each edge $e = (x, y) \in E$, make two directed edges – one from x to y and the other from y to x. For each such edge, make the capacity equal to the weight of that edge $w(e)$. Now, fix any other vertex in V apart from s as the sink vertex t. Now, run a max-flow algorithm.

The above gives the minimum cut when s and t are constrained to lie in two different cuts. So, we need to repeat this process $n - 1$ times with the sink t as all other vertices in V apart from s. We take that partition as giving us the minimum cut which gives us the lowest min-cut value.

The proof that we finally get the minimum cut of the graph is an easy consequence of the max-flow min-cut theorem.