CS345/CS345A

Practice-sheet 2 : **Greedy Strategy, DAG, and DFS**

Date: *28th August, 2015*

1. (*Synchronizing the circuit*)

   This is the problem we discussed in the class. However, we shall now design an algorithm along the generic framework of the Greedy algorithms. Let us restate the problem for the sake of completeness.

   There is a complete binary tree representing a circuit. There are n leaves. Each edge has a positive no. which represents the delay along that edge. An electric pulse originates at the root and travels all the way to the leaves. The delay incurred along a path is sum of the delay on its edges. You need to ensure that the pulse reaches all leaves at the same time. As a result the entire circuit becomes synchronized. To achieve this objective, you are allowed to increase the delay along some edges. The delay enhancement is the sum of increase in the delay of all the edges. The aim is to synchronize the given circuit such that total delay enhancement in the circuit is minimum.

   (a) You have to design a greedy step that transforms the given instance to a smaller instance of the same problem. You might like to *generalize* the problem for this purpose.

   (b) Then you have to establish a relation (and prove it formally) between the optimal solution of the given instance and optimal solution of the smaller instance. The proof must be along similar lines as that of Huffman coding.

   (c) Design an $O(n^2)$ time recursive algorithm using the relation you derived in the previous step.

   (d) Using bottom up approach, describe an $O(n \log n)$ implementation of this algorithm.

2. (*Time varying MST*)

   Given an undirected weighted graph $G = (V, E)$, where each edge $e \in E$ has two parameters a_e and b_e. The weight of edge e at time t is equal to $a_e t + b_e$. Your aim is to monitor the minimum spanning tree of the graph during a time interval $[t_1, t_2]$. So this purpose, you wish to compute the minimum spanning tree which has the least weight in the interval $[t_1, t_2]$. Design an $O(m \log n)$ time algorithm for this problem.

3. (*Red-Blue cities*)

   There is a directed graph $G = (V, E)$ on n vertices and m edges where each edge has a positive weight. Each vertex has a unique color from the set { red, blue, green }. Let R, B, T denote the subset of vertices with color red, blue, and green respectively. Note that $V = R \cup B \cup T$. Furthermore, $|R| = \Theta(n)$ and $|B| = \Theta(n)$. Realize that the shortest path from a red vertex to a blue vertex may potentially pass through some green vertices as well. Our aim is to compute the minimum possible distance from

any red vertex to any blue vertex. In particular, we wish to compute the following quantity.

$$\text{min-RB-distance}(G) = \min\{\delta(x, y) \mid x \in R, y \in B\}$$

where $\delta(x, y)$ denotes the distance from x to y in G. Design an $O(m + n \log n)$ time algorithm to compute min-RB-distance(G).

4. (*Maxi-min spanning tree*)

   Comcast is building a cable network, which they want to use for broadcasting streaming video to n endpoints. We have a graph $G = (V, E)$, where each vertex $v \in V$ corresponds to an endpoint, and where each edge $e \in E$ has a (positive) capacity that represents the maximum data throughput that can be sent over that link. The capacity of a spanning tree is defined as the capacity of the edge in the tree of lowest capacity, since that will be the bottleneck link. In other words,

   $$\text{capacity}(T) = \min \{\text{capacity}(e) : e \in T\}.$$

   Devise an efficient algorithm to find a maximum capacity spanning tree in G. A running time of $O(|E| \log |V|)$ should be achievable.

5. (*Least label vertex using single DFS*)

   There is a directed graph $G = (V, E)$ where each vertex has a weight which is a real number. For each vertex v, you wish to compute the least label vertex reachable from it. Design an $O(m + n \log n)$ time algorithm which outputs n pairs $\{(v, L(v))|v \in V\}$, where $L(v)$ denotes the least label vertex reachable from v. You must use only a single DFS traversal of the graph.

6. (*Eulerian tour*)

   A strongly connected directed graph is said to be Eulerian if there exists a tour which starts from a node, traverses each edge exactly once, and returns to the same node. It is well known that a directed graph is Eulerian if and only if for each vertex the number of incoming edges is the same as the number of outgoing edges. Design an $O(m + n)$ time algorithm to determine if a graph is Eulerian. If the graph is found to be Eulerian, you must print an Euler tour of the graph.

7. (*Least toll tax path*)

   You are given a weighted directed graph $G = (V, E)$ that models the road network of a country. Each node corresponds to a city and an edge (u, v) represents a direct road from city u to city v. The label on each road is a positive real number that represents the toll tax that a vehicle has to pay if the vehicle travels along that road. After the election in the country, the new government decided to remove the toll tax for all those roads that connect cities of the same state. You are provided with a list of pairs $\{(u, s(u))|u \in V\}$ such that $s(u)$ is the state to which the city u belongs. Using all this information, you have to build an $O(n^2)$ size data structure such that

   (a) For any pair of cities x, y, it takes $O(1)$ time to report the the toll tax of the "least toll tax path" from x to y.

(b) You should have an efficient algorithm that can perform the following task: For any pair of cities x, y, it should be able to print the sequence of roads corresponding to the "least toll tax path" from x to y. The time taken by the algorithm should be of the order of number of roads on the path.

(c) The entire data structure should be built in $O(mn + n^2 \log n)$ time.

8. (*Counting the number of shortest paths*)

   There is a directed acyclic graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges. Each edge has a length which is a positive real number. There is a unique vertex $s \in V$ such that every vertex in G is reachable from s. Let $N(v)$ be the number of the **shortest paths** from s to v. Design an $O(m + n)$ time algorithm to compute $N(v)$ for all $v \in V$. Assume that the number of such paths are only polynomial in n. (Can you see why this assumption is important ?)

9. (*Semi-Connected*)

   A directed graph $G = (V, E)$ on $n = |V|$ vertices and $m = |E|$ edges is said to be **semi-connected** if for each pair of vertices $u, v \in V$, either there is a path from u to v or there is a path from v to u. Design an $O(m + n)$ time algorithm to determine if G is **semi-connected**.

10. (*The classification of edges*)

    We discussed the classification of edges by a DFS forest. Let G be a graph and let (u, v) be an edge. Try to answer the following questions.

    - Does there always exist some DFS forest of G such that (u, v) is present as a tree edge in it ? If not, what must be the necessary condition that G must satisfy for this to happen ?
    - Does there always exist some DFS forest of G such that (u, v) is present as a forward edge in it ? If not, what must be the necessary condition that G must satisfy for this to happen ?
    - Does there always exist some DFS forest of G such that (u, v) is present as a backward edge in it ? If not, what must be the necessary condition that G must satisfy for this to happen ?
    - Does there always exist some DFS forest of G such that (u, v) is present as a cross edge in it ? If not, what must be the necessary condition that G must satisfy for this to happen ?