CSE 2012- Design and Analysis of Algorithms Practice Problem Sheet (Maximum-flow network Problem and Path problems)

Practice makes you Perfect

Maximum flow network problem

Given a flow network $G = (V, E, c, s, t)$ where V is the set of vertices, E is the set of edges, c is the capacity of the edges, s is a vertex designated as source vertex, t is a vertex designated as sink vertex. Task is to prescribe a flow f such that the flow value is maximum.

1. Propose a sample flow network which when given as an input to Ford-Fulkerson algorithm, the algorithm will never terminate. Give a proper justification for your claim. Also modify the Ford-Fulkerson algorithm to overcome that limitation.

2. Modify the Ford-Fulkerson algorithm so that the algorithm returns a flow f such that flow value is maximum and the $f(u, v)$ is prescribed for every edge (u, v) in the graph G.

3. Ford-Fulkerson algorithm uses Max-flow min-cut theorem for computing the maximum flow value in G.
   Max-flow min-cut theorem:
   If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:
   1. f is a maximum flow in G.
   2. The residual network G_f contains no augmenting paths.
   3.$|f| = c(S, T)$ for some cut (S, T) of G.
   Ford-Fulkerson relies on the augumenting path of the residual flow netwrok for the computation of the maximum flow value. Instead, modify the Ford-Fulkerson algorithm to compute the maximum flow value based on the cut of the graph G. Analyse your algorithm with time-complexity.

4. Given a graph $G = (V, E)$, design an algorithm to compute all the paths from any vertex v_i to any vrtex v_j, where v_i and v_j are any two different vertices. Analyse your algorithm with time-complexity.

5. Given a graph $G = (V, E)$, design an algorithm to compute the path from a vertex $u \in V$ to another vertex $v \in V$ where only the adjacency matrix

of G is given as input to your algorithm. Your algorithm should process only the entries of the adjacency matrix of G to return the required path from u to v.

6. A graph G is said to be cyclic if there exists one cycle in G. Given an undirected graph $G = (V, E)$, design an algorithm to compute whether G is cyclic or not. Analyse your algorithm with time-complexity.

7. A directed graph $G = (V, E)$ is said to be semiconnected if, for all pairs of vertices $u,\ v\ \in\ V$, we have $u \rightsquigarrow v$ or $v \rightsquigarrow u$. Give the graph G,design an algorithm to determine whether or not G is semiconnected. Analyze your algorithm with time-complexity. its running time.

8. A directed graph $G = (V, E)$ is said to be connected if, for all pairs of vertices u, v, v is reachable from u through a path. Give the graph G,design an algorithm to determine whether or not G is connected. Analyze your algorithm with time-complexity. its running time.

9. An Euler tour of a connected, directed graph $G = (V, E)$ is a cycle that traverses each edge of G exactly once, although it may visit a vertex more than once. Given G, design an algorithm to compute the Euler tour in G, if it exists in G. Analyze your algorithm with time-complexity. its running time.

10. Given a directed graph $G = (V, E)$ on which each edge $(u, v) \in E$has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v)\ eq1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Design an algorithm to compute the most reliable path between two given vertices.

11. Express the single-pair shortest-path problem as a linear programming problem.

12. Express the single-source shortest-paths problem as a product of matrices and a vector. Design an algorithm to solve the single-source shortest path problem where inputs are matrices and vectors alone. Analyse your algorithm with time-complexity.

13. Given a graph $G = (V, E)$, a vertex $s \in V$, a vertex $t \in V$, design an algorithm to compute the longest path from s to t. Analyse your algorithm with time-complexity. .