Design and Analysis of Algorithms

Graph Algorithms

May 5, 2000

Homework III: Turn in the solutions to problems 4 and 12 on May 23rd, 2000 in class. No late or early submissions.

1. Problem 23.2-7, Page 476; (CLR)
2. Problem 23-2, Page 495; (CLR)
3. Problem 23-3, Page 496; (CLR)
4. Problem 24-1, Page 511; (CLR)
5. Problem 24.2-7, Page 510; (CLR)
6. Problem 24.2-8, Page 510; (CLR)
7. Problem 25.3-6, Page 536; (CLR)
8. Problem 25.2-5, Page 532; (CLR)
9. Problem 25-1, Page 545; (CLR)
10. Problem 25-2, Page 546; (CLR)
11. Problem 25-3, Page 546; (CLR)
12. If P is a path in a weighted graph G, let $maxweight(P)$ be the maximum of all the weights of the edges in P.

    Give a polynomial-time algorithm to solve the following problem:

    Given an undirected graph whose edges have positive integral weights, and two distinct vertices s and t, among all $s - t$ paths P find one for which $maxweight(P)$ is minimized. (This is a *bottleneck shortest $s - t$ path.*)
13. Let G be a connected graph whose edges have positive integral weights. A *minimum product spanning tree* is a spanning tree in G, for which the *product* of the edge weights in minimized.

    Give a polynomial-time algorithm that finds a minimum product spanning trees in a given connected, weighted graph. Hint: Think logarithms, but you cannot actually use them in the final algorithm, since computers cannot exactly manipulate irrational numbers like logarithms.
14. Prove that a graph is bipartite if and only if it has no odd cycles.
15. Give an algorithm running in time $O(m + n)$ for finding the distance from s to every other vertex in an undirected graph, all of whose edges have weight 1 or 2.

16. Look at problem one to find the definition of a matching. A *maximum matching* is one of maximum cardinality. Prove that the following algorithm correctly finds a maximum matching in a forest F:

    If F has no edges, output the empty set and halt. Otherwise, let v be a leaf in F, let u be its neighbor and let $e = \{u, v\}$. Let F' be the forest obtained by deleting both v and u (and all incident edges) from F. Recursively find a maximum matching M' in F', and output $M' \cup \{e\}$.

17. Consider bipartite graphs of the following special form:

    - the vertex set is $\{x_1, ..x_n\}$ (boys), $\{y_1, ...y_n\}$ (girls).

    - if (x_i, y_j) is an edge, there are no edges of the form (x_u, y_v) with both $u > i$ and $v < j$.

    A *matching* is a set of edges, no two of wich are incident on the same vertex. Design an algorithm that finds a maximum cardinality matching in linear time.

18. **Edge-disjoint paths**

    a Given an undirected graph $G = V, E$ and two distinguished nodes, s and t, describe an algorithm that finds a maximum-sized set of edge-disjoint paths from s to t. (We say that two paths are $edge - disjoint$ if they don't share a common edge, even though they are allowed to go through the same vertex.) Give a time analysis for your algorithm in terms of $|V|$ and $|E|$.

    b Given a solution to (a), suppose we add one more edge to the graph. Give an efficient algorithm for updating the solution. Give a time analysis for your algorithm in terms of $|V|$ and $|E|$. (Your algorithm should be significantly faster than redoing the entire problem from scratch.)

19. **Special cases of shortest paths**

    Problem You need to solve all pairs longest paths (*i.e.*total delays) for a VLSI application. The input graphs are DAGs where each node has in-degree at most 2. The edge lengths are all either 1 or 2. What algorithm and which data structures would you use? What is the time complexity of the algorithm for these instances?

20. **One-Way streets**

    Problem A city currently has only two-way streets in its downtown. It wants to make all of the streets one-way, while keeping any location reachable from any other location. Formally, an instance of the problem is an undirected graph. A solution is a directed graph where each edge in the undirected graph appears in exactly one direction. The constraint is that the solution must be strongly connected. Give the most efficient algorithm you can for the problem of finding a solution that is strongly connected or showing that it is impossible.

21. **Always Non-negative Path**

    Problem Leg G be an n-node directed graph whose edges are labelled either -1 or 1. A path $e_1, \ldots e_t$ in G is $always non\text{-}negative$ if the sum of the weights of e_1 up to e_i is non-negative for all $1 \leq i \leq t$. A source s and a destination s' are given as inputs. Give an algorithm to decide whether there is an always non-negative path from s to s'.

22. **Broadcast Times for a Tree**

    Problem Let T be a rooted binary tree whose edges are given positive real weights, representing $messagedelaytimes$. For a node $x \in T$, define the $broadcasttime$ for x to be the maximum over $y \in T$ of the weighted distance in T between x and y. (In other words, the broadcast time is the time before a message originatin at x is received by every other processor y.) Give an efficient algorithm to compute all the broadcast times for nodes in T simultaneously.

23. **Fuzzy Connectedness**

Problem A fuzzy undirected graph is an undirected graph with edge weights $0 \leq w_e \leq 1$. The fuzzy connectedness of a path is the minumum weight of an edge on the path. The fuzzy connectedness of two nodes is the maximum fuzzy connectedness of a path between them. Give the best algorithm you can to compute the fuzzy connectedness between two nodes in a fuzzy undirected graph.

24. **More on Minimum Spanning Trees**

Problem The minimum spanning tree problem takes as input a connected undirected graph on n nodes whose edges have integer weights. Possible solutions are spanning trees of the graph, *i.e.*, subsets of the edges that contain no cycles but connect all n nodes. The cost of a spanning tree is the sum of the weights of its edges, and the problem is to find a spanning tree of minimal cost.

Kruskal's algorithms is a greedy algorithm for this problem where, in every phase of the algorithm, a sub-forest of the tree has been chosen. In each phase, the minimum cost edge that connects two distinct components of this forest is added to the forest.

Prim's algorithm is also greedy but is slight different. Here, at the beginning of a phase, a tree has been found that spans a subset C of the nodes. The minimum cost edge between a node in C and a node not in C is added to the tree, and the endpoint of that edge not already in is added to C.

(a) You are told that the inputs for your algorithm will all be planar graphs, and you know that planar graphs have at most $3n$ edges. Which algorithm do you pick, and which data structures do you choose to implement it? What is the overall time analysis?

(b) Say that you know, in addition, that all the edge weights of the graphs are integers between 1 and n. Does this change which algorithm you choose, how you would implement the algorithm, or the time analysis?

(c) Say that instead you want to find the $maximum$ const spanning tree. Can you use modified versions of thes algorithms? Why or why not?