Revised Problem Set 2—Due Wed., Febuary 2 3PM

Guidelines for writeups

When describing an algorithm first give a high level overview of your approach, then fill in details as need to justify your time bound and correctness. Your pseudo-code should always be commented (unless it is self commenting). You are also encouraged to describe your solutions in terms of known algorithms and data structures (e.g. sort the numbers, insert into a balanced binary search tree, delete the minimum of a heap, could all be described with little or no further elaboration).

For proofs, strive for clear, clean arguments. It will always be somewhat a matter of taste which steps can be skipped, but try to avoid proofs of obvious points, and be clear on anything tricky. Define your notation carefully. This will often allow you to give a much crisper argument.

This problem set is intended to let you explore some properties of graph representations as well as timing and testing simple algorithms. There are also paper and pencil questions on network flows.

(60) Problem 1. Implement BFS using three graph implementations: graph is represented by i) an adjacency matrix, ii) adjacency list where the graph is represented by an array of linked lists. List i has the neighbors of vertex i, iii) an adjacency list, but use two arrays instead of linked lists. One array is a **neighbor array** of length $2m$ for an undirected graph, or length m for a directed graph. It has the neighbors of vertex 1, followed by the neighbors of vertex 2, ... The second array is an **index array** that has in position i the array index of the first neighbor of i in the neighbor array.

For example, if vertex 1 has neighbors 2,3; vertex 2 has neighbor 1 and vertex 3 has neighbors 1 and 2, the neighbor array is: 2,3,1,1,2 and the index array is: 1,3,4 (assuming the first element of the array is in position 1).

Do the BFS starting at vertex number 1. Find the shortest path to each vertex in the graph from vertex 1 (path length= number of arcs). At the end output for each vertex its shortest path length from vertex 1 (or that it couldn't be reached).

Test your implementations on the random graphs generated using the program rand-graph.c (note this produces a graph in adjacency matrix form, you will need to convert this to an adjacency list form for two of your implementations.) First, code for correctness, then for efficiency. Try to make your programs as fast as possible (profiling may help).

For timing, run your programs on graphs with 5000 vertices and edge probabilities of 1%, 20% and 70 %. For timing purposes you may avoid doing the final output of distances.

What to turn in: please turn in your BFS code, a description of any improvements you made, and your timing results. Also, try and explain why the different graph representations differ in their performance (or why some are about the same if true).

(25) Problem 1a. Extra credit: use your BFS solution to implement a network flow algorithm (that is, use BFS in the ford-fulkerson algorithm to find augmenting paths).

(30) Problem 2. Suppose we have a maximum flow f in a flow network G with flow value V.

a) suppose that we want to know if there are any *critical edges* (x, y) such that if we reduced the capacity of (x, y) it would lower the value of the maximum flow. Describe a polynomial time algorithm to test if there is such an edge in G (if there are multiple edges, you need to find only one). Try and make your algorithm efficient and justify its correctness.

b) Now suppose that G is a **unit** network where all capacities are one. Given an integer parameter $k \leq V$, we want to find a set of k arcs such that removing those arcs will reduce the maximum flow value by k. Describe an efficient algorithm to find such a set of arks.

(20) Problem 3. In the successive shortest A-path algorithm we discussed in class we got a run time of $O(mn) + O(pn)$ where p was the number of augmenting paths used. We also showed that $p \leq mn$ to get our final run-time of $O(mn^2)$.

a) Suppose we have a unit-network where all capacities are one. Give an improved run time bound for the algorithm (note, you do not have to change the algorithm at all, just improve the analysis for this case). Justify your answer.

b) Now suppose that instead of a unit capacity network, we have a network where all capacities are integers and each is at most some constant k. Give an improved run time analysis for such networks in terms of m, n, k. Justify your answer.

(30) Problem 4. Use network flows to solve each of the following problems; Give the run time of your solution (try and make it as fast as possible)

a) You are given a directed strongly connected graph (that is, for each pair of vertices x, y has a directed path from x to y and from y to x). Each arc has a cost associated with it. We want to find a minimum cost set of arcs such that removing that set of arcs makes the graph no longer strongly connected.

b) This involves assigning classes to classrooms. As input you are given k classes, and the *ith* class has a_i students. There are also p classrooms, and the *ith* classroom can hold up to b_i students. Further, the *ith* classroom can be used for up to u_i different classes. A class i can be assigned to a classroom j if and only if $a_i \leq b_j$ (it has room for all the students).

Give an efficient algorithm to assign classes to rooms. Give its run time in terms of k and p.

c) Same as b) but now we assume classrooms are not all the same (e.g some have special equipment). Thus each class i has a list L_i of the classrooms it can be scheduled in. As before, find an efficient way to assign classes and give its run time.