CSE 5311: Advanced Algorithms

PROGRAMMING PROJECT TOPICS

Several programming projects are briefly described below. You are expected to select one of these topics for your project. However, in exceptional circumstances I am also willing to consider other programming proposals that you may have on your own, provided they have a very strong relation to the contents of this course. Once you have selected a topic, please make appointments with me and/or Arjun/Muhammed asap to discuss the requirements of your topic in more detail.

Intermediate Project Checkpoint:

At the project checkpoint, you will be required to submit a project design for approval. The design should be a rough draft of a document that should contain:

1. An outline of the basic architecture of your system, such as main data structures, main components of the algorithm, design of the user-interface for input/output, etc.
2. An outline of how you plan to experiment with your system. This should include how you will prepare input data to test the time and space complexity of your system, as well as inputs that will be useful in demonstrating your system.

While I will give you guidance in specifying the basic requirements of your project, I urge you to be ambitious and creative in your design. You may wish to read up about your topic in more detail, and may even plan to implement and experiment with several competing algorithms. Initiative shown by you at this stage will be rewarded.

Final Project Demonstration:

Once the project is completed, the following is expected of you:

1. A demonstration of your project to me and/or Arjun/Muhammed, in which you show the various features of your system, such as its correctness, efficiency, etc. You should be prepared to answer detailed questions on the system design and implementation during this demo. We will also examine your code to check for code quality, code documentation, etc.
2. You should also hand in a completed project report, which is essentially a polished version of the project design document, but should also include some experimental results, e.g. charts of running time versus input size, etc.
3. You should also turn in your code and associated documentation (e.g. README files) so that everything can be backed up for future reference.

Project Topics:

1. **Shortest Paths and Minimum Spanning Trees:** In this project you will implement the Dijkstra's shortest path algorithm as well as Prim's minimum spanning tree algorithm (read up about Prim's algorithm). Assume adjacency list representations of the input graphs. Implement the fringe data structure using a binary heap and alternatively, as a linked list, and compare the two methods. Some animation which shows each major iteration of the algorithms will be very desirable. Have a user interface in which you can incrementally add edges to the graph and update the trees accordingly.
2. **Median finding, Order Statistics and Quick Sort:** In this project you will implement the median-finding algorithms. The user should be able to select the "k", i.e., the rank of the number desired as output ($k = n/2$ is the median). You should also be able to select groups of 3, 5, 7 etc in the linear-time median finding algorithm and be able to compare the performance of each. Also read up and implement the randomized median finding algorithm and compare against the linear-time one. Implement quick sort using both algorithms and compare the performances of these different versions.

3. **Convex Hull:** Implement several versions of the convex hull algorithm: (a) the divide and conquer version, (b) the Graham Scan version, (c) the version in which a preprocessing step removes all points in the extremal quadrilateral, and (d) a convex hull algorithm when the input is not a set of points, but is a non-convex polygon. Have a GUI where one can insert new points (for (d), the user can add a new vertex to the input polygon) and the application recalculates the hull. Compare the performances of all algorithms.
4. **Network Flow:** Implement the Ford-Fulkerson algorithm for computing network flow in bipartite graphs. While your algorithm should be able to handle large graphs, for demo purposes design a GUI in which any bipartite graph of up to 20 nodes can be specified. Some animation which shows each major iteration of the algorithm will get extra credit. Run experiments that measure the performance of the algorithm over large inputs.
5. **String Matching algorithms:** Implement both the KMP pattern matching algorithm as well as the Longest Common Subsequence (or Edit Distance) algorithm. Your algorithm should be tested out on real datasets, e.g., text files, biological sequence data, times series databases, etc. Compare the performances of each algorithm against naive algorithms for the same problems.
6. **Matrix Multiplication:** Read up about how large matrices are multiplied in the text book. Implement the Soloway-Strassen matrix multiplication algorithm, as well as the naïve algorithm. Compare the running time of the two algorithms experimentally using synthetically generated large matrices.