Anna Karlin, 426C Sieg

Extra Credit Project

Proposal due November 13
Project due December 7

The project includes a proposal and a written final report. The purpose of the project is to evaluate alternative approaches to algorithmically solve an applied problem. I am fairly flexible as to exactly what it is though. It may include readings from literature. It may include an implementation study. It may be done in teams of up to 3 people.

There are two documents expected in the project: a one page proposal and a final report. The proposal is due by Monday, November 13. Naturally, the earlier you get your proposal to me the earlier you can start on the project. The final paper is due by Thursday, December 7. Both of these are firm deadlines.

In your proposal you should describe in one page precisely what you are planning to do, and what resources you will be using. You should describe the applied problem that you are interested in. You should list papers that you will read and other resources you plan to use in order to research your applied problem. If you are planning to do an implementation, please describe in some detail what you propose to do. Discuss briefly what techniques you will use to evaluate the algorithms you find or invent. If you are working in a team, then one proposal (and final report) from the team is adequate. In doing your proposal you are welcome to use the web or library resources like INSPEC to find the material you need. I will evaluate each proposal, so it is a good idea to get the proposal to me early so that I can approve it at the earliest possible date.

In your final report, you should describe the alternative algorithms hopefully with illustrative examples. You should describe your evaluation criteria. You should then evaluate your algorithms accordingly. The audience for your paper should be computer professionals with a bachelors degree or equivalent. Your paper should contain proper scholarly citations, and informative title and section headings. If you are doing an implementation, you should be prepared to demo it (and it is your responsibility to make sure that it runs on some machine in Sieg Hall that you have access to or on a laptop that you bring to our meeting). Your paper should describe the implementation, and its performance.

Listed below are a number of examples of applied algorithms problems. I will describe each problem briefly to give you some inspiration. However, it is your job to find the resources where algorithms are found to solve the problem. Naturally, I or Ashish can give some guidance to help get you started, but in the end you are responsible for finding the papers and other resources. You are **not** required to choose your project from this list – these are just examples. The problems listed below are quite general. Part of your job will be to focus on just a few papers that are interesting, accessible with your background, and include the topic.

- Program Analysis using Graphs: Compiler writers are always trying to ways to generate better code. Interestingly, they have found ways to represent code as graphs or graphs with weights on the edges to indicate the frequency that one piece of code is executed just after another one is. Using some graph algorithms the code can be mapped to memory so that it has good instruction cache performance.

- Graph Drawing: There are many cases when we would like to automatically generate a good drawing of a graph, for example a tree. For example, a big company does a reorganization, so the management hierarchy must be redrawn. A company that prepares family trees would like to generate them automatically. Drawing trees is just one example of graph drawing.
- Web Page Layout: It is more and more popular to have a personal web page where you receive things you are interested in. For example, I get my stock prices, local team sports scores, and weather on my personal web page. It would be nice to have a good layout for my web page. How can that be done automatically.
- VLSI Layout: Another layout problem that is even more complicated is laying out components on chip automatically. In this case we have to consider not only where the components will go but where the wires will go that connect them. This is a huge problem area but there are subareas that are in themselves quite interesting. For example, let us suppose that the components are already placed, then what is the best way to route the wires.
- Exact String Matching: A common problem that comes up in text databases is, given a relatively short query string, find all instances of it in a very long string. For example, the classic unix utility "grep" uses exact string matching. If the length of the query string is m and the length of the database string is n, then we would like to answer the query in O(m+n) time. There are some techniques around that even do better on real data. That is the time is sublinear!
- Approximate String Matching: Another common problem that comes up in some text databases and in DNA sequence databases is that of approximate string matching. Here for each query string we want to find all instances of very similar strings in a very long string. A key problem is to make concrete the concept of "very similar". Udi Manber invented a companion to grep called "agrep" that does approximate string matching. In this case the measure of similarity between strings is edit distance, how many insert and delete operations it takes to transform one string into the other.
- Intersection Detection: A common problem in computer game design is given a set of objects, detect all intersections quickly. Given, a missile and a target, each defined as a polygon how do you detect an intersection between the two very quickly. This is a classic problem in the field of computational geometry where there are many other interesting problems to find. As a starter problem, can you come up with a fast algorithm that will detect if a circle (given by its center and radius) and a rectangle (given by its lower left and upper right corners) intersect? Now do this for hundreds of such objects.
- Nearest Neighbor Search: This is a classic problem that is seen in many different contexts. There is a fixed database of points. A query point is given and the job is to find the database point that is nearest to the query point. Take as an example the FBI database of fingerprints. A query fingerprint arrives and we want to find the nearest fingerprint in the database. We probably want the top 20, but this is just an example. Naturally, we need to make concrete what we mean be the distance between two fingerprints.
- Prime Testing: For RSA public key encryption we need a way of finding large prime numbers. There are several algorithms that do a good job of this. All of them use randomness. It is still an open problem whether or not there is a polynomial time algorithm to text primality.
- Public Key Encryption: There are several competing algorithms to provide public key encryption. The first and most famous is RSA. There is a new method that uses elliptic curves that shows a lot of promise.
- Network Routing Algorithms: There are several competing network routing algorithms. Network people call them shortest path routing and distance vector routing, and there may be others. Both are in common use today. Why are both still in use? Isn't one better than the other.