Computer Science 373 – Analysis of Algorithms
Prof. Steven Skiena
Fall 2017

Homework 4 – Combinatorial Search
Due Thursday, November 9, 2017

The real reason we study algorithms is to find fast ways for solving problems. Analysis gives us a way of seeing how well we are doing, but it washes away questions about constants, difficulty of implementation, and performance in practice. For this assignment, you will be given an algorithmic problem requiring some form of combinatorial search, and your goal is to design and implement as fast a solution as possible. This is intended to be a competition - may the fastest program win! Modest prizes will be given for the fastest and slowest entries.

The *set cover problem* takes as input a collection S of m subsets of the universal set $U = \{1, \ldots, n\}$. The goal is to find the smallest subset $S' \in S$ such that the union of the subsets in S' is U. This is best understood through an example. Let $U = \{1, \ldots, 12\}$, and $S = \{S_1, \ldots, S_6\}$, where

$$S_1 = \{1, 2, 3, 4, 5, 6\}, S_2 = \{5, 6, 8, 9\}, S_3 = \{1, 4, 7, 10\}, S_4 = \{2, 5, 7, 8, 11\}, S_5 = \{3, 6, 9, 12\}, S_6 = \{10, 11\}$$

The minimum set cover consists of exactly three subsets, $S' = \{S_3, S_4, S_5\}$. Set cover arises whenever you try to efficiently acquire or represent items which have been packaged in a fixed set of lots. You want to obtain all the items while buying as few lots as possible. Finding *a* cover is easy, for you can buy one of each lot. However, by finding a small set cover you can avoid buying unnecessary lots. An interesting application of set cover is boolean logic minimization.

What is known about algorithms for set cover? The problem is NP-complete, meaning that it is *exceedingly* unlikely that you will be able to find an algorithm with polynomial worst-case running time. It remains NP-complete even for restricted classes of subsets.. However, since the goal of the problem is to find a subset of S, a backtracking program which iterates through all 2^m possible subsets and tests whether it is a set cover gives an easy $O(2^n \cdot n \cdot m)$ algorithm. But the goal of this assignment is to find as practically good an algorithm as possible.

Rules of the Game

1. Everyone must do this assignment separately. Just this once, you are not allowed to work with your partner. The idea is to think about the problem from scratch.
2. If you do not completely understand what the definition of minimum set cover is, you don't have the *slightest* chance of producing a working program. *Don't be afraid to ask for a clarification or explanation!!!!!*
3. There will be a variety of different data files of different sizes. In a combinatorially explosive problem such as this, adding one to the problem size can multiply the running time by n, so test on the smaller files first. Do not be afraid to create your own test files to help debug your program.
4. The data files will be available via http://www.cs.sunysb.edu/∼skiena/373/setcover/. Each file has a name like "s-X-10-20", meaning that the file contains a set of 20 subsets of special type X, with

$U = \{1, \ldots, 10\}$. (The meaning of the type is irrelevant for your program, just that I might create a variety of different types of graphs of the same size). The first line of each file will contain the size of the universal set, and the second line the number of subsets in S. Each subsequent line contains a list of the elements in the associated subset. For example, the example described above would be given:

```
12
6
1 2 3 4 5 6
5 6 8 9
1 4 7 10
2 5 7 8 11
3 6 9 12
10 11
```

Your program must output the subset of subsets defining the minimum set cover, and what the size of that cover is.

5. Writing efficient programs is somewhat of an iterative process. Build your first solution so you can throw it away, and start it early enough so you can go through several iterations.

6. You will be graded on how fast and clever your program is, not on style. Incorrect programs will receive *no credit*. A program will be deemed incorrect if it does not find a subset corresponding to a minimum set cover for some system of subsets.

7. If you feel your programming background is weak, use a *simple* backtracking approach first. Don't get fancy until you have something that works. As Clint Eastwood says, "a man's got to know his limitations." It is possible to get a working program about 100 lines long.

8. You may use any programming language if you have a preference, but if you don't please use C for uniformity/efficiency. The programs are to run on the whatever computer you have access to, although it must be vanilla enough that I can run the program on something I have access to.

9. Don't forget to use the code optimizer on your compiler when you make the final run, to get a free 20% or so speedup. The system profiler tool, *gprof*, will help you tune your program.

10. You are to turn in a listing of your program, along with a brief description of your algorithm and any interesting optimizations, sample runs, and the time it takes on sample data files. Report the largest test file your program could handle in one minute or less of wall clock time. The top five self-reported times / largest sizes will be collected and tested by me to determine the winner.

11. What kind of approaches might you consider? Instead of testing every subset of S, you should be able to develop a backtracking algorithm which prunes partial solutions, ie. there will be no way to complete a given collection of subsets which could reduce the cost of our best solution to date. Can you order the subsets so that the search is likely to proceed more quickly? Starting off with a good approximate solution may help achieve faster cutoffs. There is room for cleverness in selecting data structures to minimize the time need to test whether a given collection of subsets cover U. Good luck!

12. Note that daily problems continue during this period It is very important to do these to make sure you are keeping up with the material you will later get more problems on.