CECS 428 Syllabus

Darin Goldstein*

This course is meant to follow CECS 328. In CECS 328, the goal was to identify certain algorithmic tricks which, when applied cleverly to tractable problems, yield "good" solutions. We also learned certain analysis methods which helped us to quantify exactly how good the algorithm was. At the end of 328, we identified a set of problems which, though easy to state, seemed very hard to find good algorithms for.

In this course, we will continue where we left off in 328. We will start by learning one final analysis method and then move on to find out that certain problem do indeed look hard (though how hard, we are embarrassed to say we do not know). We will then learn multiple ways of dealing with these problems. Just because a problem is hard doesn't mean we don't need a solution to it...

1 Contact info

- Office: ECS 546
- Phone: (562)985-1507
- Email: daring90808@gmail.com
- Course time: 8:00-10:15 AM MW
- Office hours: 10:15-11:45 MW
- Web page: http://www.csulb.edu/∼dgoldst2/

There are at least two books for this course: Approximation Algorithms for NP-Hard Problems by Dorit Hochbaum and Online Computation and Competitive Analysis by Allan Borodin and Ran El-Yaniv. Lecture notes will be made available online.

2 Stuff you should already know

1. How to evaluate arithmetic and geometric sums
2. Log rules
3. $\sum_{i=1}^{n} i = \frac{(n+1)n}{2}$

* *daring90808@gmail.com*

4. limits, $\frac{d}{dx}, \int$
5. Absolutely everything from CECS 228/328 (including flawless asymptotics and midrange algorithmic analysis)
6. Elementary probability
7. Something about modular arithmetic

3 How this course will be graded

The programming assignments will be graded as follows: Each assignment is posted online and you may demonstrate the assignment on any day *before the deadline*. Once the deadline arrives and you have not demonstrated, your project will receive no credit. *Do not bother emailing code to me!* Methods for demonstration are noted on each assignment. If you have not demonstrated your assignment *in full* before the deadline, you will not necessarily receive credit for the assignment: be aware that you will need to demonstrate your code multiple times (roughly 3 for each assignment).

The programming assignments will be graded on a scale of 0 to 4. (You can think of a 4 as an A, 3 as a B, etc.)

These policies will be STRICTLY enforced: IF YOU LEAVE YOUR DEMONSTRATION FOR THE LAST MINUTE, IT IS VIRTUALLY CERTAIN THAT YOU WILL NOT BE ABLE TO GET ANY CREDIT FOR THE ASSIGNMENT!!! I WILL NOT STAY AFTER LAB TO ACCOMMODATE PEOPLE WHO DO NOT PLAN AHEAD!!! I will also *not* stay after lab to wait for a demonstration to finish running. If you have any questions or comments about the rules, it is your responsibility to let me know about them IMMEDIATELY so that they can be straightened out. Sorry for writing so much in capital letters, but I am serious about the rules. Please plan ahead or plan to not get credit.

If, at the end of the semester, you receive 18 points total for the assignments, you may keep an A in the class without taking the final exam. 15 points is equivalent to a B, and 12 points will be equivalent to a C. Anything below 12 points and you are required to take the final exam.

Should you take the final exam, the grading breakdown will be 60% programming assignments and 40% final exam. If you take the final exam, you will be entered into a curve where the final scores are based on this formula.

3.1 Assignments

Allow for multiple demonstration iterations for each assignment.

3.1.1 Gray codes

Your job is to write an extremely large Gray code to file. The input to your program will be a number $N \leq 23$. Your job will be to write a program that

creates a Gray code consisting of all binary strings N bits long starting with $0 \ldots 0$. These codewords should be written out to a text file with exactly one codeword per line.

For example, if $N = 3$, your output file could look like the following:

000
001
011
010
110
111
101
100

When you are ready to demonstrate, you will email me for a number N and respond with your file.

3.1.2 Almost-tree vertex cover

Assume that you are looking for a vertex cover for an undirected graph $G = (V, E)$ that "from a distance" looks like a tree. However, upon closer examination you find that it is not a tree, but would be a tree if at most a constant fraction of its edges are removed. (In other words, you can think of your input as being created via the generation of a large random tree to which random edges are added.) Your goal is to find the size of the smallest possible vertex cover for this graph.

The input to your program will be a graph that has the "almost-tree" property described above in edge list form. The vertices will be labelled by integers. So the edges connected to vertex i will come after i followed by a colon. An x will indicate the end of an edge list. A triangle might then be described by

$$0 : 1, 2x1 : 0, 2x2 : 0, 1$$

The output of your program will be a file with a list of vertex names followed by x. This will indicate your minimal vertex cover. So, for example, your program may return

$$0x1x$$

for the triangle above.

Once you are finished coding, you will email me and I will respond with a graph. You will then email the results of your calculation. Assuming that your file does in fact describe a vertex cover, your grade will be based on the size of your vertex cover as compared with my simple algorithm's solution.

3.1.3 Mazes

Your goal is to write a program for time-traveling aliens that creates a maze in 4 dimensions, given the maximum size of any given dimension $N \leq 45$ *using the algorithm that arises from the disjoint data structure presented in lecture.*

A point in 4 dimensions can be represented by (t, z, y, x) where x, y, z are the normal 3-dimensional coordinates and the fourth is the time coordinate. Assume that if an alien is situated at any point in the 4D maze, then it can, in one move, change any one of its coordinates up or down by 1 as long as it does not "go off the board" (i.e. while progressing through the maze, all coordinates must remain between 0 and $N-1$; you are required to keep the walls on the outside of the 4D cube intact) or hit a wall.

You are required to use every cell in $\{0, 1, \ldots, N-1\}^4$; the maze cannot contain any cells in the grid that are unreachable from any others. Because of this, the starting and ending positions of the alien in the maze are irrelevant. It is also required that you do not create any loops within the maze; the underlying graph of the maze *must* be a tree.

Finally. and most importantly, your maze must truly be *random* in the sense that any two runs of your program should yield different results for any given slice of the maze. In other words, if you hold any 3 coordinates constant, then two runs of your program should yield two different results for the array formed by the fourth coordinate.

The input to the program will be the value of N read from the keyboard.

The idea is to think of the 1 bits as walls.

Your output should consist of a series of bytes, one for each cell in the grid starting with $(0,0,0,0), (0,0,0,1), (0,0,0,2), \ldots, (0,0,0,N-1), (0,0,1,0), \ldots$, etc., and ending with $(N-1, N-1, N-1, N-1)$. Each byte will contain 4 2-bit codes: the 2 lowest order bits will represent the x-direction, the next 2 bits will represent the y-direction, and so on, until the 2 highest order bits that represent the t-direction. The 4 possible codes are as follows:

- 00: In this cell, along this coordinate axis, the alien may move ± 1 units. In other words, there are no walls barring movement along this axis.
- 01: In this cell, along this coordinate axis, the alien may move in the -1 direction only. There is a wall blocking the positive direction along this axis.
- 10: In this cell, along this coordinate axis, the alien may move in the $+1$ direction only. There is a wall blocking the negative direction along this axis.
- 11: In this cell, along this coordinate axis, the alien may not move at all. There are walls blocking both the positive and negative direction along this axis.

For example, a byte with ASCII code (0 from 255 possibilities) equal to 75 with corresponding bits 01001011 would indicate that there are walls preventing any movement in the x-direction. movement is possible in the positive y-direction and the negative t-direction, and any movement is possible in the z-direction.

These bytes should be written to a file. Please note that this file will *not* be a text file. When you are ready to demonstrate, you will email me for N and respond with your file.

3.1.4 Placing gas stations

Assume that you are in charge of setting up a series of gas stations across the country in the 1950s. The goal is to maximize efficiency while still serving the public: Gas stations are always required to be at intersections, and, from any given road intersection, you are *required* to have a gas station somewhere within k miles. Given these requirements, your goal is to place the minimum number of gas stations that you can throughout the network.

The input to your program will be a file containing an adjacency matrix that describes the network and the value of the number k (k will be entered via the keyboard). The entries in the adjacency matrix will correspond to the weights of the edges, and a 0 entry will indicate that no edge exists. There will be a comma between every entry, and an x at the end of a row. Thus, a triangle with 2 edges of weight 10 and one edge with weight 2 might look like the following.

$$0, 10, 10x10, 0, 2x10, 2, 0$$

You will output the vertex numbers on which the gas stations are to be placed, separated by x's. For example if $k = 3$, then the results for the triangle above might be

$$0x1x$$

Once you are finished coding, you will email me and I will respond with a graph. You will then email the results of your calculation. Assuming that your file does in fact describe a vertex cover, your grade will be based on the size of your vertex cover as compared with my simple algorithm's solution.

3.1.5 The day trader

You will assume that you are a day trader who is attempting to make money by buying one particular stock. During the day, you will make a decision what fraction of your remaining money to put into the stock. Each night, the stock's price will rise and fall. The next day, the money you put into the stock will have made money or lost money depending on the movement of the stock. You realize all of your gains and/or losses in cash the following day. Any money you did *not* use to purchase stock will not change in value.

You will be given the value of ϕ, the maximum amount an offline player could theoretically make if he made every choice perfectly, the number of nights n, and, one at a time, a file with a single number that represents x_i for the factor/change that occur during the night. (The product of the values in these files is guaranteed to be ϕ.)

Your goal is to consume these files one at a time and, within 2 minutes of receipt of the files, return a file with a single value for the ratio that you want to buy stock that day. For example, if your file contains the number 0.3, then you will spend 30% of your current money on stock. The first time that your file is run, it should be run without an input file (because the first day, there is no change). Your program will be judged based on how much money you make after the final day.

This assignment will need to be demonstrated personally due to the timing issue and there will be time set aside specifically for this purpose at the end of the semester.

4 Lectures

Some good news for this course is that you will never be explicitly penalized for missing a lecture. I will never give any pop-quizzes. All graded material is mentioned explicitly somewhere in this syllabus. The final exam date for this course is set by the University (totally independently of me and over which I have absolutely no control) and should be available via the University website.

On the other hand, this is definitely a lecture-based course. If you choose to miss a lecture, I will not penalize you for it or hold it against you in any way, but you are fully responsible for any material that I go over. If I mention something in lecture that is not in the book, YOU ARE STILL RESPONSIBLE FOR KNOWING IT. I will not redo a lecture for people that missed it the first time. It is your responsibility to get the notes/information. If you miss any kind of instructions about assignments that are given during lecture (including but not limited to due dates, methods for submission for assignments, etc.), it is STILL your responsibility to be aware of what occurred in lecture.

4.1 Exams

The final exam is at a date and time assigned by the University over which I have absolutely no control. You may not leave the room during an exam. During exams, you may not communicate with anyone other than me. You may not use any communication device (cell phones, etc.) during an exam. In particular, all cell phones are required to be turned off during an exam. Failure to observe these rules will result in an F in the course.

The policies I will follow in terms of exam grading are as follows.

- Partial credit will only be given out for steps LEADING TO THE CORRECT ANSWER. The answer itself is meaningless without clear reasoning. This means that if your reasoning is incorrect or not explicitly on the page, the ANSWER BY ITSELF WILL NOT BE COUNTED AS CORRECT.
- Write out every step of your reasoning. If it is not on the page, you will not get credit for it.
- Clearly mark out anything you don't want graded. If it is on your page and not crossed out, it will be graded as if it's part of your answer. If there is more than one answer on your page, I WILL GRADE THE INCORRECT ANSWER. Points may be deducted for anything incorrect written on the page that is not crossed out even if the final answer is correct.

- Exam questions are not required to look anything like the homework questions. Exams will be given to test whether you are able to APPLY the knowledge that you have learned.

5 Cheating

Cheating on any graded material in the course will lead to an automatic grade of F in the course. I don't give warnings.

6 The final word

If there is anything on this page that you have a question or comment about, it is very important to let me know about it on the FIRST DAY OF CLASSES. After the first day of classes, I will assume that you are aware of the grading policies. Any grading misunderstandings you have after the first day of classes are your responsibility. Good luck.

7 Course topics

The programming assignment is due the day of the lecture above.

1. The good subspace: simple problems
2. The good subspace: Satisfiability I
3. The good subspace: Satisfiability II
4. The good subsapce: Vertex cover in bipartite graphs
5. Fixed parameter tractability: 0-1 knapsack problem, Vertex cover problem

   Gray codes assignment due
6. Fixed parameter tractability: Planar independent set
7. Amortized analysis: The pancake stack
8. Amortized analysis: Dynamic tables, Binary counters
9. Binomial Heaps
10. Fibonacci Heaps I
11. Fibonacci Heaps II

    Almost-tree vertex cover assignment due
12. Disjoint Sets I

13. Disjoint Sets II
14. Disjoint Sets III
15. Simple approximations: Vertex cover problem, metric traveling salesman, inapproximability
16. Simple approximations: Scheduling jobs, bin-packing, inapproximability
17. Intermediate approximations: metric k-center problem

    Mazes assignment due

18. Randomized approximations: Maximum satisfiability
19. Simple Las Vegas Algorithms: Randomized quicksort (analysis)
20. Simple Monte Carlo Algorithm: Karger's
21. Intermediate Monte Carlo: Karger's II
22. Online problems: The file cabinet I
23. Online problems: The file cabinet II
24. Online problems: Buying at the right price

    Placing gas stations assignment due

25. Online problems: Keeping a portfolio
26. Online problems: Keeping a portfolio II
27. Splay Trees I: Introduction
28. Splay Trees II: Analysis
29. Splay Trees III: Information
30. Splay Trees IV: Asymptotic static optimality

    Day trader assignment due