CS 5200 F 17 Take-Home Final

George Markowsky

2017-12-13

Due December 17, 2017 By Noon

This test is available as either a LaTeX file or Word file. You must use one of these two versions as the basis of your answer document. Your answer needs to be submitted as a PDF file, which you can easily do using either Word or LaTeX.

You may consult your textbook, class notes, and class handouts for this exam. You may also consult reference documents for Python. You may look up general facts and standard definitions, but you should not search for solutions to these problems on the Internet. When you write proofs clearly state what you are proving and how you are proving it. If you have any questions or are confused, please ask for clarification. You may ask questions about material presented during the semester either by email or on Canvas as long as it is not directly about any of the problems. When you finish your exam, your answer should consist of a ZIP file containing your Exam in the form of a PDF file generated from either the Latex or Word input file, and all other files that the individual questions ask for. Your ZIP file should contain a file Certify.txt that should contain the following sentence:

I, <your name>, certify that all the material in this ZIP file is my original work, that I did not discuss these questions with anyone other than my instructor, and that I did not use any sources not allowed for this examination.

Of course, <your name> needs to be replaced by your name.

1. (10 points) **Big O Notation**

   (a) (6 points)**Without using limits, but only the definition of O** prove that $81n^3+1300n^2+300n \in O(n^5-15000n^4-10n^3)$ but that $n^5-15000n^4-10n^3 \notin O(81n^3+1300n^2+300n)$. Show all work.

   (b) (4 points) Let $f(x) = 2\sin^3(x) - 4\sin^2(x)$ and $g(x) = 2\cos^4(x) + 5\cos(x)$. Determine whether $f(x) \in O(g(x))$ or $g(x) \in O(f(x))$. You must show all work and base it directly on the definition of O( ). **You may not use limits.**

2. (20 points) **Simple Graph Algorithms**

   Write a Python program to answer a series of questions about a graph that is given in the file GraphData.txt. **The file GraphData.txt will be available by 10 pm on 12/13/17. In the meantime you can work on the program and use your own sample data for debugging.** The structure of GraphData.txt is that each line is given in the form

   Vertex1,Vertex2

   To simplify things, the vertices are given as integers, but not necessarily consecutive integers. Your Python program should start by reading in all the data in GraphData.txt and finish by producing a text file called GraphDataOut.txt which is structured as described below. You should include your Python program as a .py file and GraphDataOut.txt in your submitted ZIP file. You do not need to include GraphData.txt in the ZIP file. If you can't do some parts of this problem place the text "I could not do this part" in the appropriate place in the output file.

(a) (2 points) The first line should read "The number of vertices in the graph is NumVert." where NumVert is the number of distinct vertices in the graph.

(b) (2 points) The second line should read "The number of edges in the graph is NumEdge." where NumEdge is the number of distinct edges in the graph. Note that GraphData.txt might contain duplicate edges.

(c) (1 point) The third line should read "Below is the adjacency list for this graph with the vertices sorted."

(d) (3 points) The next NumVert lines of the text file should contain the adjacency list for the graph with one list element per line in the form

Vertex,Neighbor1,Neighbor2,...

where all neighbors of the vertex are given. The lines of the adjacency list must be sorted by initial vertex and the list of neighbors must also be sorted. Make the lines as long as necessary to include all neighbors.

(e) (4 points) Following the adjacency list, the next line of the file should read "The number of connected components of this graph is NumComp." where NumComp is the number of connected components. Number the connected components using the numbers from range(#ConnectedComponents).

(f) (1 point) The next line should read "The number of connected components of the graph that have an Eulerian path is NumEulerPath." where NumEulerPath is the number of components that have an Eulerian Path.

(g) (3 points) The next section of the file should list one Eulerian path from each connected component that has an Eulerian path. Before each Eulerian path you should have a line that reads "The following NumEdges lines list the edges for an Eulerian path in Component NumEPComp." where NumEPComp is the number of the component that you are listing and NumEdges is the number of edges in that component. The lines must be listed in the format that describes a path, in other words if one line looks like

vertex1,vertex2

then the next line should look like

vertex2,vertex3

The components may be listed in any order. The first vertex on the first line should be the starting point of your Eulerian path and the last vertex on the last line should be the finishing point of your Eulerian path.

(h) (1 point) The next line should read "The number of connected components of the graph that have an Eulerian circuit is NumEulerCirc." where NumEulerCirc is the number of components that have an Eulerian circuit.

(i) (3 points) The next section of the file should list one Eulerian circuit from each connected component that has an Eulerian circuit. Before each Eulerian circuit you should have a line that reads "The following NumEdges lines list the edges of an Eulerian circuit in Component NumECComp." where NumECComp is the number of the components that you are listing and NumEdges is the number of edges in that component. The lines must be listed in the format that describes a circuit, in other words if one line looks like

vertex1,vertex2

then the next line should look like

vertex2,vertex3

The components may be listed in any order. The first vertex on the first line should be the starting point of your Eulerian circuit and the last vertex on the last line should be the finishing point of your Eulerian circuit, i.e., these two should be the same vertex.

3. **(15 points) Proof by Induction**

   Let function q be as below.

```
def q(n):
   if n <= 0:
      return 1
   elif n < 2:
      return 7
   else:
      return q(n-1) + q(n-2)
```

   Let function sq be as below.

```
def sq(n):
   if n < 0:
      return 0
   else:
      return sq(n-1) + q(n)
```

   (a) (5 points) Conjecture a very simple linear relationship between q and sq. If you can't figure this out, send me an email and I will give it to you. If I give you the answer you will not receive any points for this section.

   (b) (10 points) Prove, using induction, that the relationship that you conjectured in part (a) is correct. Be sure to set your proof up correctly and to list explicitly the steps of a proof by induction.

4. **(10 points) Probability**

   Let Vec$_n$ be the set of all vectors of length n each component of which comes from range(n). For example, (3, 0, 2, 2) $\in$ Vec$_4$. If v $\in$ Vec$_n$, a *quirk* is defined as a pair (i, j) such that $0 \leq i < j \leq n-1$, but v[i] > v[j].

   (a) (4 points) Create a sample space consisting of the elements of Vec$_3$. List all the elements of this space. Turn it into a probability space by using the uniform distribution. Finally, compute the average number of quirks in members of Vec$_3$.

   (b) (6 points) Generalize the results of Part (a) to determine the average number of quirks in members of Vec$_n$. You do not need to list the elements of Vec$_n$.

5. **(15 points) Dynamic Programming**

   (a) (8 points) Let G be defined by the following equations: G(0) = 5, G(1) = 15, G(2) = 40, and for n > 2, G(n) = G(n-1) + G(n-2) + G(n-3). Write a Python program that implements G directly from the definition. Submit this program as a .py in your ZIP file. Try to compute G(500) with this program. If you can't compute G(500) directly show how to use dynamic programming to write a more efficient program. Submit this program in your ZIP file.

(b) (7 points) Let H be defined by the following equations: H(0) = 6, H(1) = 7, H(2) = 8, and for n > 2, H(n) = H(n-1) - H(n-2) + H(n-3). Write a Python program that implements H directly from the definition. Submit this program as a .py in your ZIP file. Try to compute H(500) with this program. If you can't compute H(500) directly show how to use dynamic programming to write a more efficient program. Once you compute H(500) see if you can discover a more efficient program. Submit all of these programs in your ZIP file.

6. **(10 points) Amortized Analysis**
Suppose you are dealing with a dynamic table that follows the following rules:

   (a) The table size doubles when the table is full and another element is added.

   (b) The table contracts to 2/3-rds of its size when its load factor falls below 1/3.

   Using the potential function

$$\Phi(T) = |2 \times T.num - T.size|$$

   show that the amortized coase of a TABLE-DELETE for this strategy is bounded above by a constant. For the definition of all terms, consult your textbook.

7. **(10 points) Greedy Algorithms**

   (a) (5 points) Consider the following two sets of coin denominations: {1 cent, 8 cents, 20 cents}, {1 cent, 6 cents, 18 cents}. Describe a greedy algorithm that will express any sum given in pennies in terms of the denominations given in a set. Determine whether the greedy algorithm always produces the optimal solution for these two sets or not. Give a convincing reason for your conclusion.

   (b) (5 points) Suppose we have an unlimited number of rooms and a finite number of activities, each of which can be staged in any of the rooms. Give an efficient algorithm that can schedule all the activities using the smallest number of rooms. Each activity is represented by a start time and an end time and can only be performed during that interval.

8. **(10 points) NP-Complete Problems**

   (a) (5 points) Consider the following problem: given a graph, determine whether it can be colored using exactly 4 colors. Prove that this coloring problem is NP-Complete.

   (b) (5 points) Prove that for all k > 4, determining whether a graph can be colored using exactly k colors is NP-Complete.