IE 170 Final Examination Practice Problems

Dr. T.K. Ralphs

1. (from David Eppstein, UC Irvine) Suppose you are implementing a spreadsheet program in which you must maintain a grid of cells. Some cells contain values, while other cells contain formulas that depend on other values. After the user enters a new formula into a given cell or updates an already-existing formula, the spreadsheet program must first determine whether there are any *circular dependencies*, meaning a sequence of calculations that all depend on each other in a circular fashion. For example, if the expression in cell E1 depends on the value in cell C5 and the expression in cell C5 depends on the value in cell D3, then the expression in cell D3 is not allowed to depend on the value in cell E1. If a circular dependency exists, then the program must notify the user of the error and prompt for a correction.

   After it has been determined that there are no more circular dependencies, the program must then recalculate the values in the spreadsheet. In order to do this, the program must determine (1) which values may have changed and (2) in what order the changed values need to be recalculated.

   Implementing the spreadsheet requires a data structure for storing the spreadsheet in memory and one for storing the dependencies. In addition, we might have a separate file format for storing the spreadsheet on disk when not in use. The data structure for the spreadsheet itself is used to store the contents of each cell, including the formula for computing the value in the cell (if there is one) and the value itself. The data structure for the dependencies is used to look for circular dependencies and determine the order of calculation during an update. (Actually, there is a way to link these two data structures into one, but we will consider them separately).

   (a) Describe two alternative data structures for storing the spreadsheet in memory while it is being edited and discuss the advantages and disadvantages of each. Hint: Consider the time-space tradeoff. For each data structure, consider both the time required to look up the contents of a cell and the space required to store the spreadsheet.

   (b) Suppose the spreadsheet user is editing a particular cell. State an algorithm that either (1) determines all cells that must be recalculated after a change to the cell being edited or (2) detects a circular dependency. Determine the running time of the algorithm. Hint: You may want to construct a graph. If so, please state explicitly how the graph is constructed.

   (c) Assuming no circular dependencies exist, state an algorithm for determining the order in which the cell values should be recalculated and analyze its running time.

2. Because most computers represent numbers in binary format, certain operations can be accomplished very efficiently using *bit operations*. For example, to multiply a number by 2, we simply add a 0 to the end of its binary representation (this is the same as shifting all the bits one spot to the left).

(a) Describe how to divide a number by 2 and truncate the answer (i.e., throw away the fractional part) using a bit shift operation.

(b) Describe how to quickly determine whether a binary number is odd or even by examining one bit (this is called a parity check).

(c) Consider the following properties of the greatest common divisor of a and b.

- If a and b are both even, then $\gcd(a, b) = 2\gcd(a/2, b/2)$.
- If a is odd and b is even, then $\gcd(a, b) = \gcd(a, b/2)$.
- If a and b are both odd, then $\gcd(a, b) = \gcd((a - b)/2, b)$.

Using the bit operations from (a) and (b), state an efficient algorithm for computing $\gcd(a, b)$.

(d) Analyze the number of bit operations necessary to execute your algorithm (in the worst-case), in terms of a, assuming $a \geq b$. Justify your answer.

3. The so-called Fibonacci numbers arise in many applications and are defined by the following recursive formula:

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$

Calculating F_n for large n is a challenging exercise and there are a number of algorithms for doing so. This question explores a few of these algorithms.

(a) It is easy to show that $F_n = \phi^n/\sqrt{5}$ rounded to the nearest integer, where $\phi = (1+\sqrt{5})/2$ and is called the *golden ratio*. One algorithm is simply to use this formula to calculate F_n directly. What are the potential difficulties with implementing this algorithm using floating point arithmetic? Hint: Consider large n.

(b) Another obvious algorithm is to use recursion to calculate F_n based on the definition. What is the running time of this algorithm? Hint: Write a recurrence for the running time. What is the form of the recurrence?

(c) An improvement of the algorithm of part (b) is the following iterative function for calculating F_n.

```
int fibonacci(int n)
{
   int a(1), b(1);
   for (int i = 3; i <= n; i++) {
      int c = a + b;
      a = b;
      b = c;
   }
   return b;
}
```

What is the running time for this algorithm? Explain why this implementation is more efficient than the one in part (b) (don't simply state that the running time is lower). Hint: What calculations are done in part (b) that are avoided here?

4. Consider Prim's Algorithm to find the minimum spanning tree of a weighted graph. The worst case running time of the algorithm discussed in class is $O(m \lg n)$, where m is the number of edges and n is the number of vertices. Suppose we know that the weights of all the edges are integers between 1 and M.

   (a) Describe an implementation of the algorithm that takes advantage of our knowledge of the edge weights to decrease the running time. Hint: The running time will depend on M in some way. Think about the data structures used in the algorithm.

   (b) Analyze the running time of your algorithm from (a). Justify your answer.

5. This question refers to Prim's Algorithm for finding a minimum weight spanning tree in a graph $G = (V, E)$. Please note that you can do part (b) without knowing how to do part (a)!

   (a) (10 points) Describe an alternative implementation of Prim's algorithm that has a worst-case running time in $O(|V|^2)$. (Hint: You must find a different way of determining the node with the smallest estimate in each step.)

   (b) (10 points) Under what circumstances would this algorithm be better than the implementation we discussed in class? (Hint: You can do this problem even if you couldn't answer part (a).)

6. (Thanks to David Eppstein) This problem will lead you through the worst-case and average-case analysis for a simple algorithm. The following subroutine takes as input an array L that is a random permutation of the integers 1 through n and returns the first value of i for which $L[i] < L[i-1]$.

```
int firstDecrease(int *L, int n)
{
   for (int i = 1; i < n && L[i] >= L[i-1]; i++);
   return i;
}
```

   (a) (10 points) What is the worst-case running time of this function?

   (b) (10 points) If the numbers in the array L are ordered randomly, then the probability that `firstDecrease(L)` returns the value k is $k/(k+1)!$ except for the special case when $k = n$ for which the probability $1/n!$. Use this fact to write an expression for the average value returned by the algorithm. The answer can be expressed as a sum. You do not have to simplify this sum.

   (c) (10 points) Use your answer from (b) to determine the running time of this algorithm in the average case.

7. Suppose you are working on a computer system that is only capable of multiplying k-bit integers (integers that can be represented in binary using k bits) and suppose you want to multiply two numbers u and v, at least one of which is bigger than k-bit. This can be done using a formula that reduces the multiplication of a pair of n-bit integers to several pairs of

$\lceil n/2 \rceil$-bit numbers, as follows. First define $m = \lceil n/2 \rceil$ and let

$$\begin{aligned} w &= \lfloor u/2^m \rfloor, && (1)\\ x &= u \mod 2^m, && (2)\\ y &= \lfloor v/2^m \rfloor, \text{ and} && (3)\\ z &= v \mod 2^m, && (4) \end{aligned}$$

so that $u = 2^m w + x$ and $v = 2^m y + z$. Then we have

$$\begin{aligned} uv &= (2^m w + x)(2^m y + z) && (5)\\ &= 2^{2m} wy + 2^m wz + 2^m xy + xz. && (6) \end{aligned}$$

Note that w, x, y, and z all have at most m digits. for this problem, you may assume that all integers are positive and that the addition of arbitrary integers can be done in constant time.

(a) (10 points) Using the above formula as a basis, write pseudo-code for a recursive algorithm to multiply two n-bit integers.

(b) (10 points) Write a recurrence for the running time of your algorithm and solve it.

(c) (10 points) Note that if we define

$$r = (w + x)(y + z) = wy + (wz + xy) + xz, \quad (7)$$

then we have

$$\begin{aligned} uv &= 2^{2m} wy + 2^m wz + 2^m xy + xz && (8)\\ &= 2^{2m} wy + 2^m (r - wy - xz) + xz. && (9) \end{aligned}$$

Explain how you might use this observation to improve your algorithm and analyze the running time of the improved algorithm.

8. Consider the following system of equations

$$\begin{aligned} -x_1 + 2x_2 - x_3 &= -1 && (10)\\ 3x_1 - x_2 &= 0 && (11)\\ -x_2 + 2x_3 - x_4 &= 1 && (12)\\ -x_3 + 3x_4 &= 0 && (13) \end{aligned}$$

(a) (10 points) Solve this system using LU decomposition.

(b) (10 points) The most expensive operation in computing an LU decomposition is computing the outer product $vw^\top$ in each step. In many applications, v and w are sparse (they have a small number of nonzero entries). Suppose two vectors x and y have at most c nonzeros each and are stored in sparse format, as discussed in lecture, i.e., for each vector we store the positions and values of the nonzero entries in two auxiliary vectors. Describe an algorithm for finding the outer product of x and y (also in sparse format) and analyze its running time. Use pseudo-code as necessary.

9. The following is the search method from the `Graph` class of Laboratory 7.

```
1  void Graph::search()
2  {
3    for (int i = 0; i < vertNum_; ++i) {
4      if ( vertices_[i]->getColor() == WHITE ){
5        Vertex* source = vertices_[i];
6        source->setColor(GRAY);
7        source->setDistance(0);
8        stack< Vertex* > vertList;
9        vertList.push(source);
10       while (vertList.empty() != true) {
11         Edge * ed;
12         Vertex* v;
13         Vertex* u = vertList.top();
14         vertList.pop();
15         node<Edge> * head = u->getAdjList().getHead();
16         for (ed = &(head->getData()); ed != 0; ed = &(head->getData())){
17           v = vertices_[ed->getEndPoint2()];
18           if (v->getColor() == WHITE) {
19              v->setColor(GRAY);
20              v->setDistance(u->getDistance() + 1);
21              v->setPred(u);
22              vertList.push(v);
23           }
24           head = head->getNext();
25         }
26         u->setColor(BLACK);
27         std::cout << u->getIndex() <<"\t" << u->getDistance() << std::endl;
28       }
29     }
30   }
31 }
```

(a) (5 points) What specific method does this function currently implement?

(b) (5 points) What method would it implement if line 8 were changed to

```
8       queue< Vertex* > vertList;
```

and the corresponding function calls for adding and deleting from `vertList` on lines 9, 13, and 14 were changed appropriately?

(c) (5 points) What information about a vertex does the `getColor()` method provide?

(d) (15 points) Assuming the input graph is directed and acyclic, describe how you would modify this function to perform a topological sort of the vertices. Describe the method both in words and by writing specific lines of pseudo-code and indicating where they would appear with respect to the given lines of code above.

10. Suppose you are given a collection of intervals $[a_i, b_i]$ for $i = 1, \ldots, n$. An *interval graph* is a graph $G = (V, E)$, where each vertex corresponds to an interval and the edges consists of pairs of vertices whose corresponding intervals overlap.

    (a) (10 points) Describe a $\Theta(|V| \log |V| + |E|)$ algorithm for determining the edge set of an interval graph. Note that the "naive" algorithm of just comparing every interval to every other interval is $\Theta(|V|^2)$. Under what circumstances is your algorithm better than the naive algorithm?

    (b) (10 points) Given an interval graph, describe an efficient algorithm for determining the union of all the intervals.