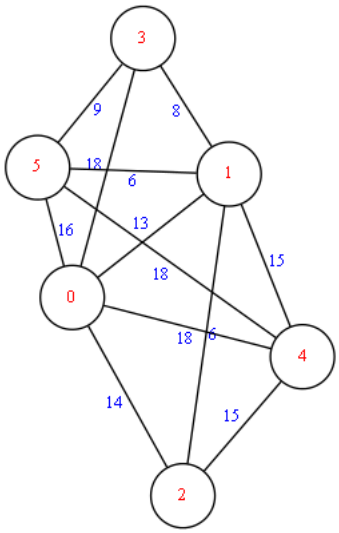
IE 172 Final Examination Practice Problems

Dr. T.K. Ralphs

1. This question concerns the graph below.



   (a) Execute Dijkstra's Algorithm on this graph by hand. Assuming the priority queue is stored as a heap in an array-based binary tree data structure, write down what the array containing the heap would look like at the beginning of each iteration. Please show as much work as possible if you would like partial credit in the case you make a mistake. Clearly indicate each arrays in your answer and what step it is associated with.

   (b) Show that Prim's Algorithm produces the same tree and give a general condition under which this would be true for a given graph.

   (c) Suppose you have an undirected graph with weighted edges, and perform a depth-first search, such that the edges going out of each vertex are always explored in order by weight, smallest first. Is the search tree resulting from this process guaranteed to be a minimum spanning tree? Explain why, if it is, or, if it isn't, provide a counterexample (specifying the start vertex for the DFS and showing both trees). (Hint: for the graph above, this algorithm does produce a minimum spanning tree. Think about the effect certain modifications to the graph would have on the outcome.)

2. The following is a search method that could be used with the `Graph` class we used in several of the laboratories.

```
1    def search(self, source, destination = None):
2
3        q = Stack()
4
5        nl = self.get_node_list()
6        for i in nl:
7            self.get_node(i).set_attr('color', 'black')
8
9        pred = {source:source}
10       if str(source) != str(destination):
11           found = False
12       while not q.isEmpty() and not found:
13           current = q.pop()
14           self.get_node(current).set_attr('color', 'blue')
15           if current == str(destination):
16               found = True
17               break
18           for n in self.get_neighbors(current):
19               if (self.get_node(n).get_attr('color') != 'green' and
20                   self.get_node(n).get_attr('color') != 'yellow':
21                   self.get_node(current).set_attr('color', 'yellow')
22                   pred[n] = current
23                   q.push(str(neighbor))
24           self.get_node(current).set_attr('color', 'green')
25
26        return pred
```

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

(b) What method would it implement if line 3 were changed to

```
3        q = Queue()
```

(c) What information about a vertex does the color of a node indicate? List all possible colors and what each one indicates.

(d) Write a Python function `find_path(pred, destination)` to print out the path from the source to any other node, taking the output of the `search()` method as input.

3. A d-dimensional box with dimensions $(x_1, x_2, \ldots, x_d)$ nests inside another d-dimensional box with dimensions $(y_1, y_2, \ldots, y_d)$ if there exists a permutation π of $1, 2, \ldots, n$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots, x_{\pi(n)} < y_n$.

(a) Give an efficient algorithm for determining whether a given box nests inside another one. Specify the running time of your algorithm.

(b) Suppose that you are given a sequence of n boxes $B_1, B_2, \ldots, B_n$. Give an efficient algorithm for determining the longest sequence of boxes $B_{i_1}, B_{i_2}, \ldots, B_{i_k}$ such that B_{i_j} nests within $B_{i_{j+1}}$ for all $j = 1, \ldots, k-1$. (Hint: use the fact that the nestedness property is transitive and construct an associated graph).

4. The *longest common subsequence* (LCS) problem is to find the longest subsequence that is common to each of two input sequences. Recall from your mathematics courses that a *subsequence* of a sequence is a sequence comprised of a subset of a given sequence in which the elements of the subsequence come in the same order as they did in the original sequence. For example, the sequence `[1, 3, 8, 2]` is a subsequence of the sequence `[1, 6, 3, 8, 9, 2]`. The following is a recursive function that calculates the length of the longest common subsequence of sequences `X` and `Y`, given as two Python lists.

```
def LCSLength(X, Y):
    if len(X) == 0 or len(Y) == 0:
        return 0
    if X[-1] == Y[-1]:
        return LCSLength(X[:-1], Y[:-1]) + 1
    else:
        return max(LCSLength(X, Y[:-1]),
                   LCSLength(X[:-1], Y))
```

   (a) Explain the basic logic of this function.

   (b) What is the worst case running time of this function? You can use big-O notation to give a rough answer, but be sure to explain how you got your answer.

   (c) The following is a non-recursive version of the function.

```
def NRLCSLength(X, Y):

    m, n = len(X), len(Y)

    C = [[0 for i in range(n)] for j in range(m)]

    for i in range(m):
        for j in range(n):
            if X[i] == Y[j]:
                C[i][j] = C[i-1][j-1] + 1
            else:
                C[i][j] = max(C[i][j-1], C[i-1][j])

    return C[m-1][n-1]
```

   How can we interpret `C[i][j]` when this function terminates?

   (d) What is the running time of the non-recursive version?

5. 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.

6. 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.

7. 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 $L[i] < L[i-1]$.

```
def firstDecrease(L):
    for i in range(len(L)):
        if L[i] >= L[i-1]:
            break
    return i
```

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

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

8. 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) Using the above formula as a basis, write a Python function for a recursive algorithm to multiply two n-bit integers.

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

   (c) 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.

9. 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) 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) Given an interval graph, describe an efficient algorithm for determining the union of all the intervals.

10. The following questions refer to a generic list data structure. Recall that a list can be stored either as an array (Python list) or a linked list.

    (a) Assuming the use of a Python list, write a Python function that moves the smallest item to the beginning of the list. What is the running time of this operation in the worst case?

    (b) Assuming a linked-list implementation, write a Python function that moves the smallest item to the beginning of the list. What is the running time of this operation in the worst case?

    (c) Describe how to use the subroutine from parts (a) and (b) to sort an array. What is the worst-case running time of this sorting algorithm. What standard sorting algorithm is this equivalent to?

11. A *Josephus election* is conducted as follows. The n candidates sit in a circle. Starting with a randomly chosen spot in the circle, the candidates count off and every m^{th} candidate is eliminated until only one remains.

    (a) Describe how to implement an algorithm for determining the winner of a Josephus election using modular arithmetic.

    (b) What are the requirements on m and n in order for the algorithm to produce a winner?

    (c) What is the running time of your algorithm, in terms of m and n.

    (d) How much memory does the algorithm use?

12. One difficulty in working with functions involving $n!$ is that it cannot actually be computed for large n without causing an overflow. Describe a recursive method for computing the quantity $(n! \mod M)$ such that overflow is not an issue.

13. Write a recursive function to find the maximum element of an Python list.

14. Draw the result of inserting the keys E A S Y Q U E S T I O N into a binary search tree in that order.

15. Consider the following recursive algorithm for topological sort of a directed graph. The algorithm either returns `None`, if there exists a directed cycle or `list`, containing a topologically sorted list of the vertices of the graph.

Algorithm 1 Topological sort method

```
if The graph has no nodes then
    Return list
end if
Find node i with no incoming arcs
if No such node exists then
    Return None
else
    Append i to list
    Delete i from graph
    Call topological sort method
end if
```

(a) Describe details of an implementation of this algorithm for the case when the graph is stored in a standard adjacency list data structure (such as the one described in the textbook) and analyze its running time. Be sure to describe details of the algorithm, such as how to find a node with no incoming arcs.

(b) Describe any modifications to the algorithm, the data structure, or the input graph itself that would improve the running time.

16. In this problem, we consider a data structure for storing a document in a simple text editor. The goal of our data structure is to allow basic editing operations, such moving the cursor and entering text, to be performed efficiently. We consider here a keyboard-based system, so the only way to move around the document is to use the arrow keys.

    The data structure that has been proposed is to store each line of the document as a linked list and to store the entire document as a (regular Python) list of these linked lists.

    (a) What is the running time of the following basic operations?

    - Typing a character
    - Deleting a character
    - Hitting one of the arrow keys (up, down, right, left).

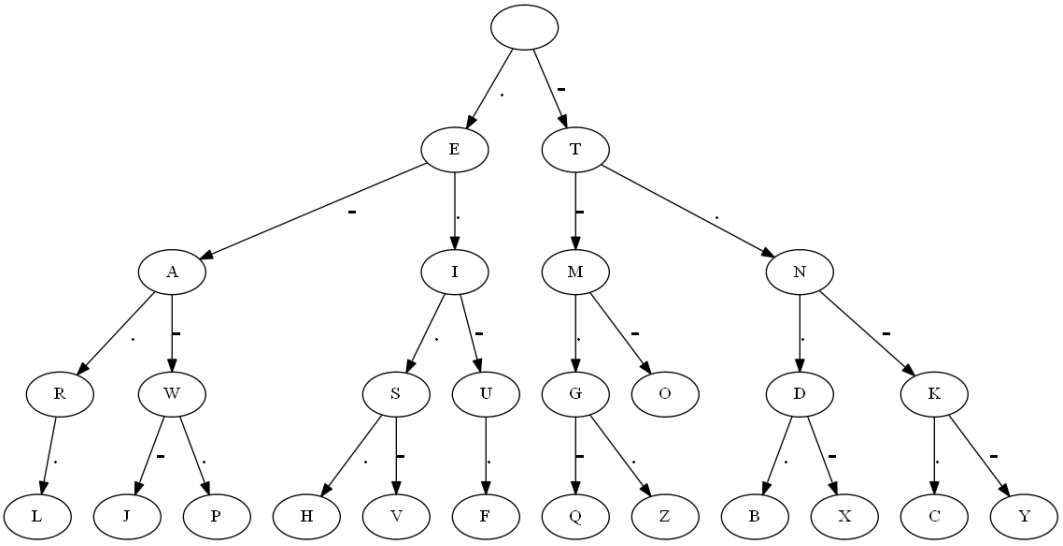
    You can consider the input to be a function of the maximum number of characters per line C and the number of lines N. Note that without some kind of line wrapping capability, the size of a single line can grow quite large. In your analysis, consider a carriage return (new line—what you get when hit the `Return` key) to be a character (you might want to treat this as a special case).

    (b) Suppose we want to include a "word wrap" capability. In other words, each time the typing of character causes a line to exceed a certain maximum length, the last word on the offending line must be moved to the next line. Write a function to insert a character into a line, taking into account word wrap, keeping in mind that the wrapping of one line may necessitate the wrapping of additional lines.

    (c) Suggest an algorithm for efficiently spell-checking the document and analyze its runing time. There are multiple ways of doing this. Try to choose the most efficient one.

17. The well-known Morse code converts each character of a given string into a series of up to four "dots" and "dashes." For example, the letter "X" becomes "- · · -". The scheme can be

pictured in the form of a tree in which each node contans a character and the corresponding code is the sequence of "dots" and "dashes" associated with the edged encountered on the path from the root. For Morse code, the tree looks as follows.



This yields a simple method of decoding in real time as each "dot" and "dash" is received. We simply trace the path in the trtee until we find the appropriate letter.

(a) The goal of Morse code is to make the transmission time of encoded messages as short as possible. Assuming that a "dot" takes 1 time unit to send and a "dash" takes three time units, how should we allocate the letters to nodes of the tree in order to minimize the transmission time?

(b) Suppose we try to use this scheme on a computer as a simple compression algorithm. We think of the "dots" as "0" and the "dashes" as "1", then we can re-interpret Morse code as a mapping of each letter to a 4-bigit binary number. By interpreting the entire sequence of binary digits as a single binary numbner, we hope to get a more compact representation of the original string. Will this scheme work? (Hint: try encoding the word "HIS").

(c) Huffman codes were invented as an alternative to the Morse code scheme just suggested. They work as just described except for two enhancement. First, the letters are only allowed to be associated with the leaves of the tree. Second, we also allow the tree to be unbalanced. What are the advantages of this scheme over the Morse code scheme. Be specific.

18. Consider the problems of finding the first non-repeated character in a string.

(a) The most straightforward algorithm is to move through the string character by character, searching the remaining part of the string linearly for a matching character. Any character found to be matching is marked and skipped when it is encountered later. What is the worst case running time of this algorithm?

(b) Describe a more efficient method of solving this problem and analyze its running time (Hint: One approach is similar to an algorithm we used in one of the labs).

19. A phone number can be converted into a string using the mapping given by the set of letters corresponding to each given number on the phone dial (2: ABC, 3:DEF, 4:GHI, 5:JKL, 6: MNO, 7: PQRS, 8:TUV, 9:WXYZ).

    (a) Write a recursive Python function for printing out all the strings corresponding to a given phone number and analyze its running time.

    (b) Describe how you might check as you go along whether the curent string consists of words in English?

20. Suppose we are given an undirected graph and an ordering of the nodes. Describe an algorithm for orienting the edges of the graph such that the resulting directed graph is acyclic and the given ordering is one of the associated topological orderings of the nodes of the new directed graph.

21. Consider the problem of finding the slice of a Python list with the greatest sum. When the list can contain negative items, this is a non-trivial problem.

    (a) The following one line of Python will solve this problem

    ```
    print max([sum(A[i:j]) for i in range(n) for j in range(i+1, n+1)])
    ```

    What is the running time of this algorithm?

    (b) To improve efficiency, we can avoid summing from scratch each time as follows.

    ```
    best = A[0]
    for size in range(1, n+1):
        cur = sum(A[:size])
        for i in range(n - size):
            cur += A[i+size] -= A[i]
            best = max(best, cur)
    ```

    Explain briefly how this algorithm works and and analyze the worst-case running time. Compare this running time to that of the previous algorithm.

    (c) Describe how to further improve the algorithm to linear time.