CISC 380 **Instructor: Sarah Miracle**
Student: Practice

Exam 2 Review

Topic Summary

Note that Test 2 covers all material from class and your assignments up to and including the last day of class before the exam. However, there will be a strong focus on the material covered since Test 1. Here is a *brief* outline of the topics covered in class since Test 1.

1. Dynamic Programming
    - Subproblem Definitions & Recurrences
    - Iteration & Memoization
    - Recovering the solution (back-tracking through the table)
    - Examples from Class
        - Fibonacci
        - Longest Increasing Subsequence
        - Longest Common Subsequence
        - Knapsack (with and without repetition)
        - Chain Matrix Multiply
        - Maximum Independent Set in a Tree
2. Graph Algorithms
    - Adjacency List & Adjacency Matrix Representations
    - DFS on graphs (Depth First Search)
    - Finding connected components (in undirected graphs)
    - Kosaraju's Algorithm
    - Forward, Backward and Cross Edges
    - Finding Cycles & Topological Orderings
    - Strongly Connected Component Algorithm (directed graphs)

Practice Problems

These practice problems are representative of the type of problems you might get on the test but do NOT cover all of the material you are responsible for.

1. In class we discussed a solution for the longest increasing subsequence problem. Here we are given $a_1, \dots, a_n$ and our goal is to find an increasing subsequence of maximum length. We defined our subproblems $L(j)$ as the length of the longest increasing subsequence in $a_1, \dots, a_j$ which ends at a_j AND includes a_j. We defined $L(j)$ recursively as follows:

$$L(j) = 1 + \max_i \{L(i) : i < j, a_i < a_j\}.$$

Use *memoization* to write pseudocode for computing $L(j)$ for $1 \le j \le n$. You must use the above subproblem definition and recursive formula.

(a) Write pseudocode for your algorithm (you must **USE MEMOIZATION**) .

Solution:

```
Global Array $L[1 \dots n] \leftarrow Null$
function LIS($A = [a_1, \dots, a_n]$)
    if $L[n] \neq Null$ then
        return $L[n]$
    $L[n] = 1$
    for $i = 1 \rightarrow n - 1$ do
        if $a_i < a_n$ AND (LIS($[a_1, \dots, a_i]$) +1) $> L[n]$ then
            $L[n] = L[i] + 1$
    return $L[n]$
```

(b) Analyze the running time of your algorithm (use O notation).

Solution: The ith step does $O(i)$ work in finding the maximum. There are n steps total. Thus this is a $\sum_{i=1}^{n} O(i) = O(n^2)$ algorithm.

2. Recall the palindrome problem given on Homework 4. A subsequence is a *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is *not* palindromic). In the homework solutions we gave a dynamic programming solution that takes a sequence $x[1 \dots n]$ and returns the length of the longest palindromic subsequence. The solution used the following subproblem definition. Let $Palin(i, j)$ be the length of the longest palindromic subsequence between (and including) characters $x[i]$ and $x[j]$. We use the convention that $Palin(i, i-1) = 0$ (a zero-length string), and that $Palin(i, i) = 1$, since an individual character is a palindrome. To write the recursive formula we considered two cases based on whether $x[i] = x[j]$.

$$Palin(i, j) = \begin{cases} 1 + Palin[i+1, j-1] & \text{if } x[i] = x[j] \\ \max(Palin[i+1, j], Palin[i, j-1]) & \text{if } x[i] \neq x[j] \end{cases}$$

We gave the following pseudocode for filling in the table $Palin$.

```
function LongestPalin(Array $x[1 \dots n]$)
    Array $Palin[1 \dots n \times 0 \dots n]$
    for $i = 1$ to $n$ do
    $Palin[i, i] \leftarrow 1$
    $Palin[i, i-1] \leftarrow 0$
                                                        ▷ Initialization of values
for $l = 1$ to $n - 1$ do                               ▷ Increasing lengths
    for $i = 1$ to $n - l$ do
        $j \leftarrow i + l$
        if $x[i] = x[j]$ then
            $Palin[i, j] \leftarrow 2 + Palin[i+1, j-1]$
        else
            $Palin[i, j] \leftarrow \max(Palin[i+1, j], Palin[i, j-1])$
return $Palin[1, n]$
```

This algorithm calculates the maximum value $Palin[1, n]$ by filling in the table $Palin$ but does not find the actual palindromic subsequence. Using the dynamic programming algorithm above, find a way to recover the palindromic subsequence of length $Palin[1, n]$. You may modify the pseudocode above and/or add an additional function to output the subsequence. The overall running time of the algorithm should not change.
(Hint: Recover the first $\lceil Palin[1, n]/2 \rceil$ characters and then very briefly describe the processing you would do to recover the entire subsequence.)

(a) Briefly describe your algorithm in words.

Solution: First we will obtain the first $\lceil Palin[1, n]/2 \rceil$ characters of the palindromic subsequence. Next if the length is odd, we remove the last character and reverse it to get the last $\lfloor Palin[1, n]/2 \rfloor$ characters. If the length is even just reverse it. At each step we determine which of three cases we are in. We start with $i = 1, j = n$. If $x[i] = x[j]$ then print $x[i]$ and recurse on $Palin[i+1, j-1]$. Otherwise if $Palin[i, j] == Palin[i+1, j]$ recurse on $Palin[i+1, j]$. Finally if $Palin[i, j] == Palin[i, j-1]$, recurse on $Palin[i, j-1]$.

(b) Write pseudocode for your algorithm (this may include modifications to the pseudocode above).

Solution: At this point, we assume that $Palin[i, j]$ contains the length of the longest palindromic subsequence between (and including) characters $x[i]$ and $x[j]$.

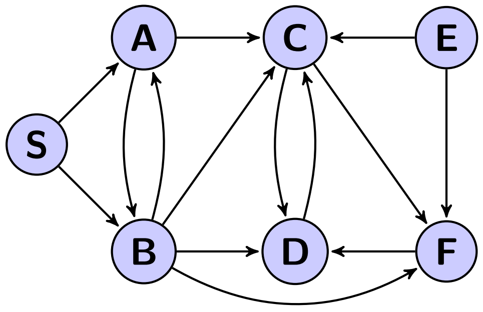
```
function RecoverPalin
    $i \leftarrow 1$
    $j \leftarrow n$
    Answer ← ''                     ▷ an empty linked list of characters to store our answer
    while $j - i \geq 1$ do
        if $x[i] = x[j]$ then
            Answer = Answer $+x[i]$                          ▷ append the matched character
            $i \leftarrow i + 1$
            $j \leftarrow j - 1$
        else if $Palin[i, j] == Palin[i+1, j]$ then
            $i \leftarrow i + 1$
        else
            $j \leftarrow j - 1$
    if $Palin[1, n] = 1 \mod 2$ then                         ▷ If the length is odd.
        AnswerF = RemoveOne(Answer)                          ▷ Remove the last character
    AnswerF = Flip(AnswerF)                                  ▷ Reverse the order of AnswerF
    return Answer + AnswerF                                  ▷ Append AnswerF to Answer
```

(c) Analyze the running time of your algorithm. (use O notation and explain your answer)

Exam 2 Review

Solution: At each step of the while loop we either increment i, decrement j, or both. Since the difference between i and j is initially $n-1$ and each step of the while loop takes $O(1)$ time the entire loop takes $O(n)$. Each of the string processing steps takes $O(n)$ time so the entire function has running time $O(n)$.

3. Suppose the DFS algorithm is run on the following graph, starting at node S.



   (a) Give the pre/post order numbers of each vertex if the DFS algorithm is run on the graph starting at vertex S.

   (b) Run the strongly connected component algorithm on the graph. Draw the component graph and indicate the order the components are found by the algorithm. (Hint: this involves running DFS twice!)

4. Consider the following alternative algorithm for finding the strongly connected components in a directed graph $G = (V, E)$.

```
Function SCC Algorithm 2 ($G = (V, E)$)
  1. Run DFS to compute the pre and post numbers for $G$.
  2. Compute $G^R$.
  3. Order $V$ by decreasing post number from step 1.
  4. Run DFS on $G^R$ using the vertex order determined by step 3 to obtain a forest (collection
     of trees). Output each tree in the forest as a SCC.
```

   (a) Explain why this algorithm is correct.

Solution: By the Lemma given in class (The vertex with the highest post number lies in a source SCC.) the first vertex explored by the 2nd run of DFS is in a source SCC in G. This implies that it lies in a sink SCC in G^R since all of the edges are reversed in G^R. The SCCs remain the same in G as in G^R. When running DFS on a sink SCC (as proved in class) we only return that SCC. By repeating this procedure on vertices with increasing post number we will continue exploring vertices whose only outgoing edges are to SCCs that are already found. Thus we will correctly find the SCCs of G^R which are the same as the SCCs in G.

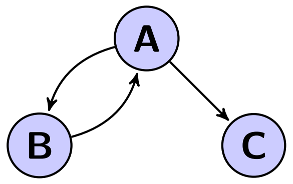
Alternative Explanation: Notice that this algorithm is the same as running the SCC algorithm described in class on the reverse graph G^R. Thus, by the proof in class it correctly returns the strongly connected components in G^R. By Lemma 3.2 the strongly connected components in G^R and G are the same so the algorithm correctly returns the strongly connected components in G.

(b) Analyze the runtime of this algorithm.

Solution: Let n be the number of vertices and m be the number of edges. Step 1 takes time $O(n+m)$ as shown in class. Step 2 takes time $O(m)$. Step 3 takes time $n \log n$ (note that you could also order the vertices by decreasing post number as you're running step 1 making this O(n) - either is fine). Step 4 takes O(n + m). This gives an overall runtime of $O(n \log n + m)$ or $O(n+m)$ with the simple improvement described in the analysis of step 3.

(c) Consider the following simplification to the algorithm: Skip step 2. In step 3 order the vertices by increasing post number. In step 4 use the original graph G (instead of G^R) and the new vertex order. Does this simpler algorithm always produce correct results? Explain why this doesn't work and give an example.

Solution: This simpler algorithm does **NOT** always produce a correct result. This is because the vertex with the lowest post number is not necessarily a sink. Consider the following example,



If we run DFS on the graph above starting with vertex A and exploring B first we get the following pre and post order numbers:

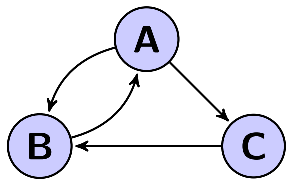
$$A:(1,6) \quad B:(2,3) \quad C(4,5)$$

In the second pass of DFS we start with the vertex with the lowest post order number which is B. From here we explore the whole graph and return $\{B,A,C\}$ as a SCC. However this is not correct - there is no path from C to A or to B. The SCC's should be $\{B,A\}$ $\{C\}$.

5. State whether each of the following statements is true or false and explain why.

(a) If the depth-first-search of a directed graph has a cross edge then the graph is not strongly connected.

Solution: False! Here's an example of a tree with a cross edge (c,a) that is strongly connected. The DFS explored the vertices in order A,B,C.



(b) If a graph is not strongly connected, then its depth-first-search must have a cross edge.

Solution: False! Consider starting a DFS at vertex A.