Practice Final

Problem 1

A palindrome is a string that reads the same forwards and backwards, like x, pop, noon, redivider. Any string can be broken into sequence of palindromes. For example, the string bubbaseesabanana (‘bubba sees a banana’) can be broken into palindromes in several different ways:

- bub baseesab anana
- b u bb a sees aba nana
- b u bb a sees a b anana
- b u b b a s e e s a b a n a n a

Solution:

```
minPalindromes(string)
        minPals = list of 0s size+1

        isPal(i, j)
                if i >= j
                        true
                else
                        return string[i] == string[j] and isPal(i+1, j-1)

        for i in string
                if isPal(0, i)
                        minPals[i] = 1
                else
                        minPals[i] = minPals[i-1] + 1
                        for k in string
                                if isPal(k+1, i)
                                        minPals[i] = min(minPals[i], minPals[k]+1)

        return minPals[end]
```

Problem 2

Suppose you are given an n × n checkerboard with some of the squares deleted. You have a large set of dominos, just the right size to cover two squares of the checkerboard. Describe and analyze an algorithm to determine whether it is possible to tile the remaining squares with dominos-each domino must cover exactly two undeleted squares, and each undeleted square must be covered by exactly one domino.

Your input is a two-dimensional array Deleted[1..n, 1..n] of bits, where Deleted [i, j] = True if and only if the square in row i and column j has been deleted. Your output is a single bit; you do not have to compute the actual placement of dominos. For example, for the board shown above, your algorithm should return True.

Solution

```
function solveDominos(array, i, j)
        if array[i][j] open
                if array[i][j+1] open
                        place domino horizontal
                        horizontal = solveDominos(array, i, j+2)
                        remove domino horizontal
                if array[i+1][j] open
                        place domino vertical
                        vertical = solveDominos(array, i+2, j)
                        remove domino horizontal

                return horizontal or vertical
        else
                return false

solveDominos(array, 1, 1)
```

Problem 3

Problem 3. Clearly indicate the following structures in the directed graph below, or write NONE if the structure does not exsit.

a) A depth first spanning tree rooted at r

R - S - T - W - Z - Y - X - U - V

b) A breadth first spanning tree rooted at r

R - S, V, U

S - T, Z

V - X, Y,

U

T -W

c) A topological order.

No possible it is not a DAG, there are cycles

d) The strongly connected components.

S - T - W - S

U - V - X - U

Problem 4

Problem 4. A polygonal path is a sequence of line segments joined end-to-end; the endpoints of these line segments are called the vertices of the path. The length of a polygonal path is the sum of length of its segments. A polygonal path with vertices (x1, y1), (x2, y2), .., (xk, yk) is monotonically increasing if xi < xi+1 and yi < yi+1 for every index i; informally, each vertex is above and to the right of its predecessor.

Solution:

```
function pologonal()
        graph =
```

Problem 5

Describe and analyze an algorithm to solve arbitrary acute-angle mazes.

You are given a connected undirected graph G , whose vertices are points in the plane and whose edges are line segments. Edges do not intersect, except at their endpoints. For example, a drawing of the letter X would have five vertices and four edges; the first maze above has 13 vertices and 15 edges. You are also given two vertices Start and Finish.

Your algorithm should return True if G contains a walk from Start to Finish that has only acute angles, and False otherwise. Formally, a walk through G is valid if, for any two consecu-tiveedgesu→v→winthewalk,either∠uvw=180o or 0o<∠uvw<90o . Assumeyouhavea subroutine that can compute the angle between any two segments in O(1) time. Do not assume that angles are multiples of 1o.

Solution:

```
acute_path(G, start, finish)
        G1 = Create edge graph G1(V1, E1) from graph G such (ve1, ve2) is an element of
E1 if an only if e1 and e2 are acute or straight line

        G1 add start vertex
        G1 add end vertex

        BFS(G1, start as source)

        if end vertex exist in BFS then true
```

Time complexity $O(V + E)$

Problem 6

Problem 6. [For the following two subproblems, give clear pseudo code and English explana- tions, argue their correctness, and analyze its time complexity].

a) Suppose an arithmetic expression is given as a tree. Each leaf is an integer and each internal node is one of the standard arithmetical operations (+,−, ∗, /). For example, the expression 2 + 3 ∗ 4 + (3 ∗ 4)/5 is represented by the tree in figure below (left). Give an O(n) algorithm for evaluating such an expression, where there are n nodes in the tree.

b) Supposeanarithmeticexpressionisgivenasa DAG(directedacyclicgraph)withcommon subexpressions removed. Each leaf is an integer and each internal node is one of the standard arithmetical operations (+,−, ∗, /). For example, the expression 2 + 3 ∗ 4 + (3 ∗ 4)/5 is represented by the DAG in the figure below. Give an O(n+m) algorithm for evaluating such a DAG, where there are n nodes and m edges in the DAG.

[Hint: modify an algorithm for the tree case to achieve the desired efficiency]

a)
solution

```
function compute(operator, x, y)
        return x operator y

function evalulate(node)
        if node null
                return 0

        if node is leaf
                return node value

        x = evalulate(node.left)
        y = evalulate(node.right)

        return compute(node value, x, y)

evalulate(root)
```

b)
solution

```
function compute(operator, x, y)
        return x operator y

function evalulate(node)
        if node null
                return 0

        if node is leaf
```

```
                return node value

        if node value is number
                return node value

        x = evalulate(node.left)
        y = evalulate(node.right)

        return compute(node value, x, y)

evalulate(root)
```