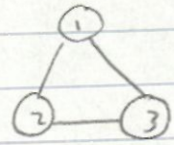
Algorithms HW 1 3.1

Elizabeth Port

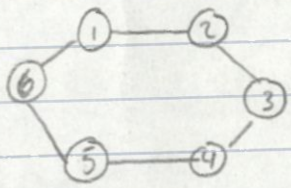
#5

n vertices ; n = 3 n x n



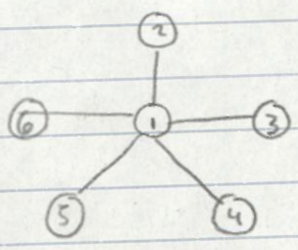
$$\begin{array}{c|ccc} & n=1 & n=2 & n=3 \\ \hline n=1 & & & \\ n=2 & & & \\ n=3 & & & \end{array} \Longrightarrow \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

For ring, boolean matrix might look like



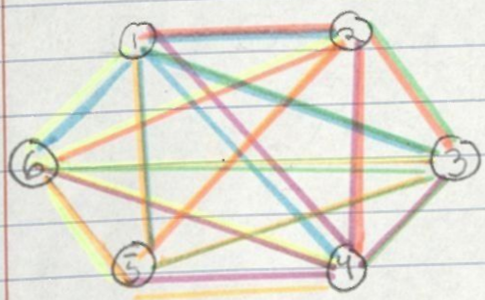
$$\begin{array}{c|cccccc} n= & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline n=1 & 0 & 1 & 0 & 0 & 0 & 1 \\ n=2 & 1 & 0 & 1 & 0 & 0 & 0 \\ n=3 & 0 & 1 & 0 & 1 & 0 & 0 \\ n=4 & 0 & 0 & 1 & 0 & 1 & 0 \\ n=5 & 0 & 0 & 0 & 1 & 0 & 1 \\ n=6 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

For star, matrix might look like



$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}$$

For fully connected, matrix might look like



$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 & 1 & 1 \\ 4 & 1 & 1 & 1 & 0 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 0 & 1 \\ 6 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

Identifying a Ring:

```
// Input: Boolean adjancency matrix
// Output: True / False if Matrix is
// a representation of a ring network topology

For y=0 -> n-1 {
    • Check that exactly two indecies
      are non zero
      For x = 0 -> n-1 {
        Array[x] != 0 {n++}
          if n++ != 2 return Not Ring }
    }
    • repeat for all y-indecies.
    return isRing
```

identifying a Astar:

```
//input: Boolean adjacency matrix
//output: True / False if matrix is Star network topology

For y=0 -> n-1 {
• Check if all indecies are non-zero except 1
  For x=0 -> n-1 {
  Array[x] = 0 {n++} }
  if n ≤ 1 : fullyConnected++ }
  • if less or more than 2 fully connected lines exist,
    adjacency matrix does not represent a star
  if (fullyConnected == 2): return isStar
   else (return NotStar)
```

Identifying a fully connected mesh

//input: Boolean adjacency matrix
// output: T/F depending if matrix represents fully connected mesh
· General approach: ensure each x row, & y column does not contain more than 1 non-zero

```
for y=0 → n-1 {
    for x=0 → n-1 {
    if ( Array[x] ==0) { n++ }    //Check all x indecies
            if n > 1 { return NotMesh} // only contain 1 Ø
    }
    if (Array[y] ==0) { z++ }    //Check all y indecies
            if z > 1 { return NotMesh} // only contain 1 Ø
}
```

Run the above three programs sequentially to identify which of the 3 topologies the matrix represents.
If none return true, matrix is not a known topology.

Because the time this algorithm takes is directly related to the # of nodes (and i.e. the size of the matrix) it is a linear time complexity.

3.1 #7

a) // Assume weight of each stack is stored in A & B
```
if (A > B) {
        A is stack of fake coins}
   else {A is stack of real coins}
```

This algorithm is O(1), because no matter how many coins are in the stacks, only 1 comparison is needed.

b) 2 weighings are required each time new coins are added. This is because we only need the current weights of each stack to determine which is heavier.

3.2 #5 How many comparisons will be made in searching for the following patterns in a text of 1,000 zero's

a) ... 0 0 0 0 0 0 0 0 0 0 ... looking for: 00001

0 0 0 0 1 • Five comparisons made

0 0 0 0 1 • Five comparisons made.

For each digit of the 1,000 0's 5 comparisons will be made

1,000 × 5 = **5,000 comparisons will be made**

b) ... 0 0 0 0 0 0 0 0 0 0 ... looking for: 10000

1 • 1 comparison

1 • 1 comparison

for each digit of the 1,000 0's, 1 comparison is made.

So **1,000 total comparisons will be made**

c) ... 0 0 0 0 0 0 0 0 0 0 ... looking for: 01010

0 1 • 2 comparisons

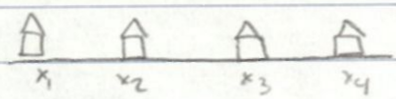
For each digit of the 1,000 0's, 2 comparisons will be made. So **2,000 total comparisons**

3.2 #7 Depending on the string & pattern be searched / searched for, there may be a very great advantage. For example for

text = 0000000000 & pattern = 0000000001

searching left to right would require 90 comparisons, but searching right to left would require 10. The opposite could also be true for a different string, or it may make no difference: it depends on the string & pattern used.

3.3 #3



a) Find average distance: $\frac{1}{n}\sum|x_i - x_j|$

```
//input: Array of n ordered values
//output: location of post office w/ min average distance
  if (n>1)
      index ← ⌈n/2⌉
  else
      index ← 0
  return index
```

b)
```
//input: Array of n ordered values
//output: location x so that x to post office is minimized
  define: m = (x_1 + x_n) / 2
  while x_i < m : i = i+1
  if x_i - x_1 < x_n - x_{i-1}
      location = i
  else
      location = i-1
  return location
```