Design and Analysis of Algorithms — Naman Jain Vimal Kumar
Assignment 3 — 1001670153

Problem 1

Write a step by step approach for finding a solution to a maze. (For this question You can also explain an approach to make a maze but you need to explain your approach (write a pseudo code) and do not use the exact step by step pictures that i used in class).

Solution

Answer

To Build your maze:

1. Divide the map into half and assign each different set value (Say 0 and 1).
2. Iterate the above set unless all the blocks are singleton and has different set values.
3. Add all edges separating 2 blocks into an array.
4. Apply Random Lotto |B|/|E| selection, which selects |B| random edges
5. These edges are removed from the maze and the edge joining the 2 blocks are assigned the same set value.

Let us consider the maze as a 2D matrix, where each index value represents a block and corresponding value represents the set value

Pseudo Code:

```
# Inputs: M – Maze represented as 2D matrix of size m, n
#              setValue – Disjoint set value given to each block
#              m_start, m_end, n_start, n_end – Concentration area of the maze
# Returns: M – Maze array with different disjoint values
function CreateBoard(Maze M, Int setValue, int m_start, int m_end, int n_start, int
 n_end)

  If m_start = m_end and n_start = n_end:
    M[m_start][n_start] <- setValue
    Return M

  Else: # Divide vertically
    If m_start = m_end:
      Int newSetValue <- setValue*2
      Int mid <- (int)(n_start+n_end)/2
      AssignSet(Maze M, newSetValue, m_start, m_end, n_start, mid)
      AssignSet(Maze M, newSetValue+1, m_start, m_end, mid + 1, n_end)
    EndIf

    # Divide horizontally
    If n_start = n_end:
      Int newSetValue <- setValue*2
      Int mid <- (int)(m_start+m_end)/2
      AssignSet(Maze M, newSetValue, m_start, mid, n_start, n_end)
      AssignSet(Maze M, newSetValue+1, mid+1, m_end, n_start, n_end)
    EndIf
  EndIf

  Return M
```

Naman Jain Vimal Kumar
1001670153

```
# Inputs: m, n - Size of the 2D maze
# Returns: A - Array of all possible edges. Each edge is represented by block1 and
block2
GetEdgeArray(m, n):
  A <- []
  For i = 0 to m-2:
  Do
    For j = 0 to n-2:
    Do
      # Adding blocks to the right
      Block1 <- i, j
      Block2 <- (i+1), j
      A.add([Block1, Block2])

      # Adding blocks at the bottom
      Block3 <- (i), (j+1)
      A.add([Block1, Block3])

    EndFor
  EndFor
  Return A

# Inputs blocks: block1 and block2 of the maze whose edge needs to be remove
#         m, n: Size of the maze
JoinSets(block1, block2, m, n)
  SetValue1 = M[block1[0], block1[1]]
  SetValue2 = M[block2[0], block2[1]]
  For i from 0 to m
  Do
    For j from 0 to n
    Do
      if M[i][j] = SetValue1:
      then
        M[i][j] <- SetValue2
      EndIf
    EndFor
  EndFor

# Inputs: m, n - Required size of the 2D maze
CreateMaze(m, n):
  Initialize Maze = CreateBoard(M, 0, 1, 1, m, n)
  Edges = GetEdgeArray(m, n)
  Block_count = m*n
  # Applying Lotto Block_count/Size(Edges)
  For i = 0 to Block_count
  Do
    Int randInt <- random integer between 0 and (Block_count - i)
    SelectedEdge <- Edges[randInt]
    Swap(Edges[Block_count - i - 1], Edges[randInt])
    JoinSets(SelectedEdge.Block1, SelectedEdge.Block2, m, n)
  EndFor
```

Naman Jain Vimal Kumar
1001670153

```
  return M
```

Method in class:

```
# Method 1L Method in class

# Returns array of size M*N each with a different disjoint set
construc2dtMapSet(m, n)
  N <- m*n
  Map <- Array of size = N
  for i from 0 to N-1
  Do
    Map[i] <- i
  Endfor

findEdges(m, n)
  # E is an edge of 2D array first array points at each edge, while 2nd is the bloc
k index
  E = []
  k = 0
  for i from 0 to n-2
  Do
    for j from 0 to m-2
    Do
      current_block = (n*i)+j
      next_block = current_block + 1
      E[k] = [current_block, next_block]
      Increament k

      below_block = current_block + m
      E[k] = [current_block, below_block]
    Endfor
  Endfor
  return E

# Function to find the set Value of a given node index
findMySubsetValue(Maze, node_index)
  if Maze[node_index] = node_index
  then
    return node_index
  else
    return findMySubsetValue(Maze, Maze[node_index])

# Inputs blocks: block1 and block2 of the maze whose edge needs to be remove
#        Maze which consists of set Values
JoinSets(block1, block2, Maze)
  SetValue1 = findMySubsetValue(Maze, block1)
  SetValue2 = findMySubsetValue(Maze, block2)
  Maze[SetValue1] = SetValue2


# Inputs: m, n - Required size of the 2D maze
```

Naman Jain Vimal Kumar
1001670153

```
CreateMaze(m, n):
  Initialize Maze = construc2dtMapSet(m, n)
  Edges = GetEdgeArray(m, n)
  Block_count = m*n
  # Applying Lotto Block_count/Size(Edges)
  For i = 0 to Block_count
  Do
    Int randInt = random integer between 0 and (Block_count - i)
    SelectedEdge = Edges[randInt]
    Swap(Edges[Block_count - i - 1], Edges[randInt])
    JoinSets(SelectedEdge[0], SelectedEdge[1], Maze)
  EndFor
  return M
```

To find the maze result:
We shall start with DFS from the start node

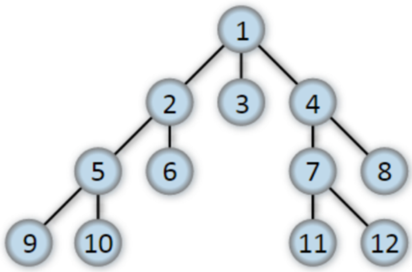
```
# To solve a maze
# Function to find the route to end node
findPath(endNode)
  path <- []
  temp <- endNode
  while(temp != null)
  Do
    path.add(temp)
    temp <- temp.parent
  Endwhile

  return reverse(path)

# Function to find the minimum distance between start node to end node
# Inputs - startNode and endNode
# Outputs - Path if exists, else 0
solveMaze(Node start, Node end)

  Let Object x <- {Node = start, parent = null}
  Stack <- []
  start.parent <- null
  Stack.push(start)
  While Stack is not empty
  Do
    Object x <- Stack.pop()
    if x is visited
    Then
      continue
    Else
      Mark x as visited
      if x = end
        return findPath(end)
      Else
```

Naman Jain Vimal Kumar
1001670153

```
        for each neighbour of x:
        Do
          neighbour.parent = x
          Stack.push(neighbour)

        Endfor
  Endwhile
  return -1
```

Problem 2

Consider the graph below:



a. *Write the pseudocode for finding the distance of vertex 1 from any other vertices.*
b. *Draw each step and show the final result.*

Solution

a. We can apply DFS for the same.
   - Since it is unweighted graph, we can take the weight of each edge to be unit, that is 1.
   - Also, based on the problem statement our end node is always 1

```
# Function to find the minimum distance between start node to end node
# Inputs - startNode and endNode
# Outputs - Distance and -1 if there is no path
find_distance(Node start, Node end)

  Let Object x <- {Node = start, distance = 0}
  Fringe <- []
  Fringe.push(x)
  While Fringe is not empty
  Do
    Object x <- Fringe.pop()
    if x.Node is visited
    Then
      continue
    Else
      Mark x.Node as visited
      if x = end
        return x.distance
      Else
        for each neighbour of x.Node:
        Do
```

Naman Jain Vimal Kumar
1001670153

```
        Fringe.push({Node = neighbour, distance = x.distance + 1))

        # Fringe is sorted by the ascending order of the distance of each object
        Fringe <- sort(Fringe)
      Endfor
  Endwhile
  return -1
```

b.
Let start = 11 and end = 1
Fringe:

Iteration	Index	0	1
0 Add start to the Fringe	Node Distance	11 0	
1 Selected Node: 11 | Distance: 0	Node Distance	7 1	
2 Selected Node: 7 | Distance: 1	Node Distance	4 2	12 2
3 Selected Node: 4 | Distance: 2	Node Distance	12 2	

Problem 3

How we can figure out if a graph contains cycles or not? Name two approaches and explain the steps for each.

Solution

a. DFS for directed graph

```
# Method 1: Using DFS - This method is used from undirected graphs

# Return true if cyclic, false if acyclic
# Function to check if a graph is cyclic
# Inputs - Graph G
# Outputs - True if the graph is cyclic
checkCyclic(Graph)

  Mark all vertex of the graph as not visited
  while all vertex are not visited
  Do
    start <- Random vertex which is not visited
    x <- {Node: start, parent: null}
    Fringe = []
    Fringe.push(x)
    while Fringe is not empty
    Do
      y <- Fringe.pop()
      Mark y.Node as visited
      for each neighbour of y.Node
      Do
```

```
        if neighbour not equal to y.parent
        Then
          if neighbour is visited
            return true
          Else
            Fringe.push({Node: neighbour, parent: y.Node})

    Endwhile
  Endwhile
  return false
```

b. Disjoint Sets for undirected graphs

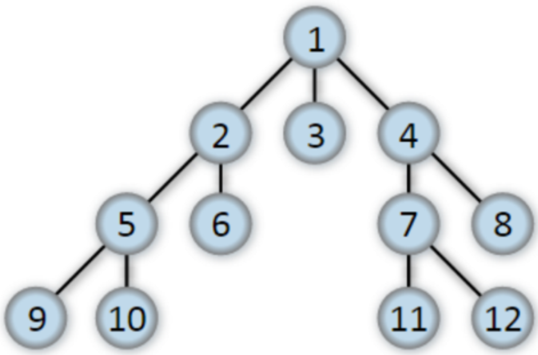
```
# Method 2: Using Disjoint Sets

# Function to union 2 subsets
union(subset_array, i, j)
  x <- findMySubsetValue(subset_array, i)
  y <- findMySubsetValue(subset_array, j)
  subset_array[x] <- y

# Function to find the set Value of a given node index
findMySubsetValue(subset_array, node_index)
  if subset_array[node_index] = -1
  then
    return node_index
  else
    return findMySubsetValue(subset_array, subset_array[node_index])

# Function to check if a graph is cyclic
# Inputs - Graph which as array of Vertex 's and Edge 's
# Outputs - True if the graph is cyclic, else false for acyclic
checkCyclic(Graph)

  # Mark all vertex of the graph as a set to itself (Say -1)
  subset_array <- Array of size(Graph.Vertex) with value -1

  for each E in Graph.Edge
  Do
    node1_index, node2_index <- E    # E connectes node1 and node2 indexses
    i <- findMySubsetValue(subset_array, node1_index)
    j <- findMySubsetValue(subset_array, node2_index)
    if i = j
      return true
    else
      union(subset_array, i, j)
  Endfor
  return false
```

Problem 4

Is any tree with at least 2 vertices is a bipartite graph? Explain your answer with examples

Solution

Yes. We know that a tree can be said as bipartile as all alternative depth level nodes will be a part of altering disjoint sets. For example below in the below graph:



Depth	Nodes
0	1
1	2, 3, 4
2	5, 6, 7, 8
3	9, 10, 11, 12

Let the 2 disjoint sets be represented by A and B.
'A' contains all nodes of depth level 2n where n is {0, +ve Integers}
'B' contains all nodes of depth level 2n+1 where n is {0, +ve Integers}

Therefore,
A = {Nodes at level 0} U {Nodes at level 2} = {1, 5, 6, 7, 8}
B = {Nodes at level 1} U {Nodes at level 3} = {2, 3, 4, 9, 10, 11, 12}

Since there are 2 nodes say N1 and N2 which are connected (trees), there one set would contain N1, while the other would have N2

References:

- GeekforGeeks.com