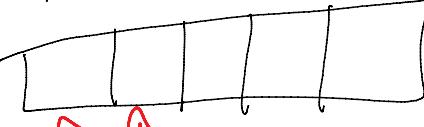
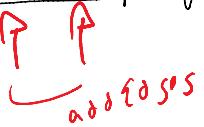


3 part b-d

Wednesday, March 25, 2020 6:34 PM

(2 points) (You must submit code for this question!) In your Main class, create a non-recursive method called Graph createRandomUnweightedGraphIter(int n) that creates n random nodes with randomly assigned unweighted, bidirectional edges. You should use some of the methods you implemented in part (a). Make sure you're either implementing an adjacency list or an adjacency matrix to keep track of your edges!

```
import graph (IGS)
for ( i in range(n) ) {
    graph.addNode(i)
    graph.getRandom()
    // next add adjancency matrix
    myAdjacencyMatrix = Graph.GetAdjacencyList()
    // Put Nodes in list
    
    
```

Question/
Feed Back
They should
be < below
wg

We are currently
Populating the graph
then putting all nodes
in a list, and use
a 2 pointer/sliding window
method to add edges

c) (2 points) (You must submit code for this question!) In your Main class, create a non-recursive method called Graph createLinkedList(int n) that creates a Graph with n nodes where each node only has an edge to the next node created. For example, if you create nodes 1, 2, and 3, Node 1 only has an edge to Node 2, and Node 2 only has an edge to Node 3.

```
1st create graph
graph g = new Graph();
for i in range(n):
    i = random.int(n)
    x = NewNode(i)
    myList.append(x)
    // update the edges for linkedList
    - , . . . , i, len(myList)):
```

↑ Order
matters
here

↑ Create
node before
put in list

```

// update edges for x1, ...
for (i in range(1, len(myList))):
    myList[i-1].neighbors.append(myList[i])
    myList[i-1].neighbors.append(myList[i])
// Put Back in graph
for nodes in myList:
    graph.add(nodes)

```

(3 points) (You must submit code for this question!) In a class called GraphSearch, implement `ArrayList<Node> DFSRec(final Node start, final Node end)`, which recursively returns an `ArrayList` of the Nodes in the Graph in a valid Depth-First Search order. The first node in the array should be start and the last should be end. If no valid DFS path goes from start to end, return null.

My helper function

def DFSRec(int target value, MySet)

 myNeighbors = node.start.neighbors

 if (node.val == target) list.append(node):

 if (node in mySet):

 if (node not in set):

 set.add(node)

 DFS(arraylist.append(node), node, target, set)

 if (target not in arraylist):

 return null

 else return arraylist