/*
Jason Wilson
Thursday, April 16th 2020
CMPT435L 111 20S
*/

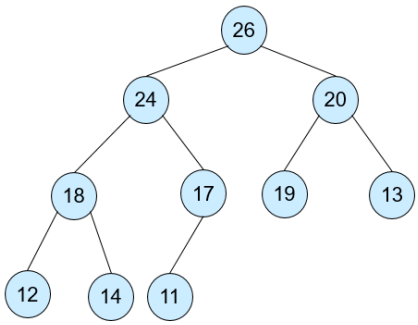
Assignment 8

Please read turn-in checklist at the end of this document before you start doing exercises.

Section 1: Pen-and-paper Exercises

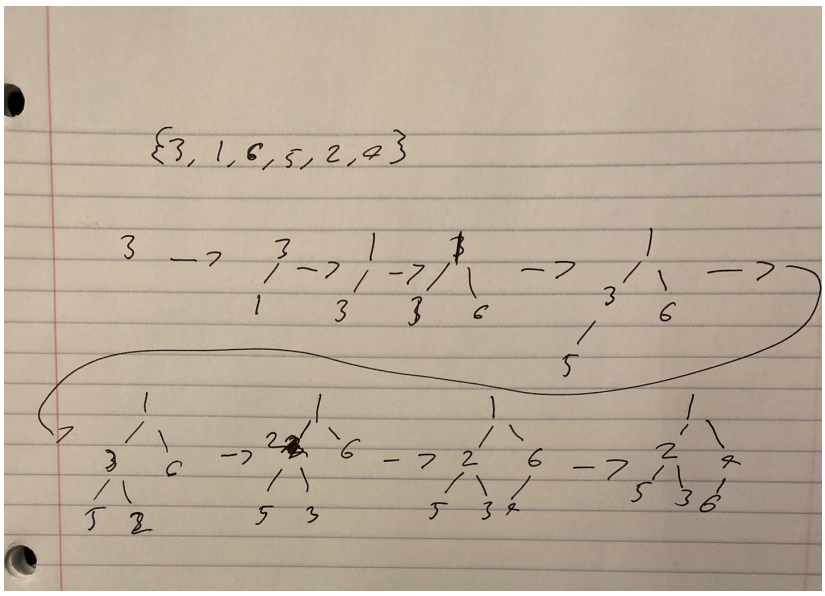
1. A heap can be stored as an array A.
   - Root of tree is A[0]
   - Left child of A[i] = A[2i+1]
   - Right child of A[i] = A[2i + 2]
   - Parent of A[i] = A[ë(i-1)/2û]
   - The elements in the subarray A[(ën/2û) … (n-1)] are leaves

   Following the algorithms above, what would be the array representation of the max-heap below? (5 points)

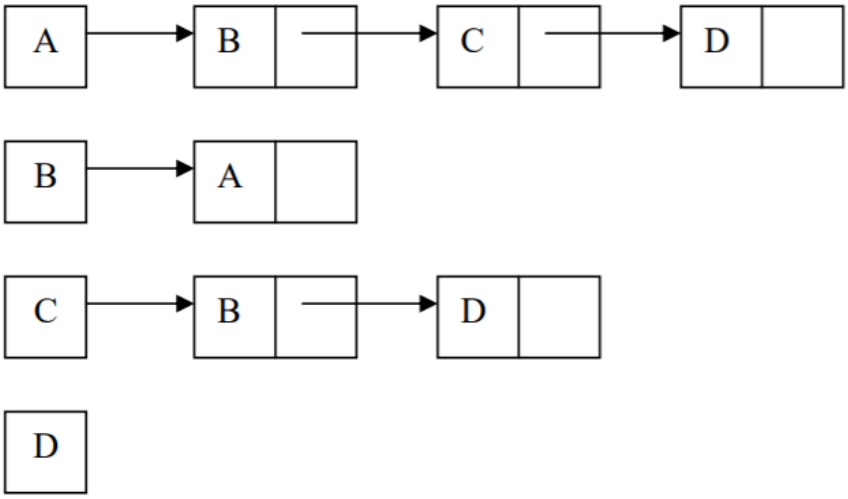


   26, 24, 20, 18, 17, 19, 13, 12, 14, 11

2. Explain how HeapSort sorts {3, 1, 6, 5, 2, 4}. Show all the steps (5 points).

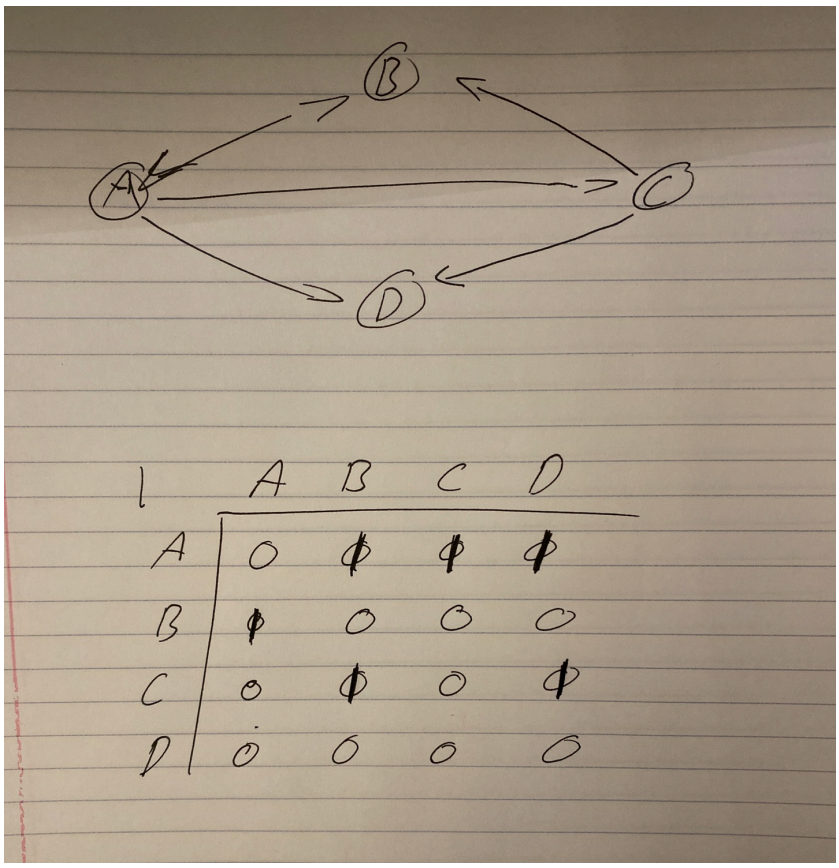


3. Here is an adjacency list representation of a directed graph where there are no weights assigned to the edges).



a) Draw a picture of the directed graph that has the above adjacency list representation **(5 points)**.

b) Another way to represent a graph is an adjacency matrix. Draw the adjacency matrix for this graph **(5 points)**.



4. Given a directed graph, how to check if there is a cycle in the graph?

(i) **describe the idea behind your algorithm in English (2 points);**

1. Set all the vertex flags to -1 and create a stack object
2. Visit the starting node, set its flag to 0 and push it into the stack
3. While the stack is not empty:
4. If all neighbors of the node are visited / are no more neighbors to visit pop from stack and set flag to 1
5. If there are neighbors, Check for cycle (flag == 0), return true if found
6. else move to the next unvisited neighbor, set the flag to 0 and push to the stack

(ii) provide pseudocode (5 points);

```
detectcycle(int startingNode)

    for (i = 0; i < numberOfVerticies; i++)
        vertex[i].cycleflag = -1
    end for

    stack = new Stack<Integer>()

    vertex[startingNode].cycleflag = 0
    stack.push(startingNode)

    while (stack is not empty) {
        neighbor = getUnvisited(stack.peek());

        if (no neighbors)
            w = stack.pop()
            vertex[w].cycleflag = 1
        end if

        else if (vertex[neighbor].cycleflag == 0)
            OUTPUT true; // There is a cycle
        end else if

        else
            vertex[neighbor].cycleflag = 0
            stack.push(neighbor)
        end else

    end while

    OUTPUT false; // there is no cycle
```

(iii) analyze its running time (3 points).

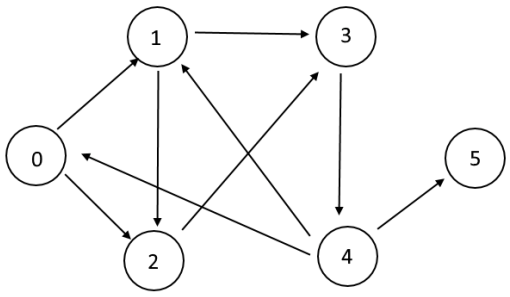
As this algorithm follows DFS, the time complexity would be O(|V|+|E|), but because we run it

for every vertex, the complexity is:

O(|V|) * O(|V| + |E|) —> O(|V|^2+|E||V|).

Section 2: Java Implementation

5. Create an adjacency list to represent the following graph in Java (30 points).



Note:
Find a file called Graph.java in assignment 8 folder.
Complete the main method to represent the given graph and print its adjacency list.
Test your method in the main method provided following the comments.

6. Following problem 5, implement problem 4 in Java to find cycles in the given graph (30 points).
Note:
Find a file called Graph.java in assignment 8 folder.
Complete the method of detectcycle().
Test your method in the main method provided following the comments.

TURN-IN CHECKLIST:

1. **Answers to Section 1 (.doc/.txt/.pdf), and to Section 2 (all your source Code (.java files)). Remember to include your name, the date, and the course number in comments near the beginning of your code/report.**

2. **Create a folder and name it 'FirstName_LastName_assignment_8'. In the newly created folder copy and paste your files (.doc/.txt/.java files). Then compress the folder, and push it to iLearn.**