Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa
L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Assignment 3 - 3.75% (40 points)

Assignment topics: AVL trees, 2-4 trees, Hashing, Graph traversals

CSI2110/CSI2510 (Fall 2024)

Due: November 22, 11:59 PM (recommended date to finish November 19)

Late assignment policy: ***1min-24hs late are accepted with 30% off; no assignments accepted after 24hs late.***

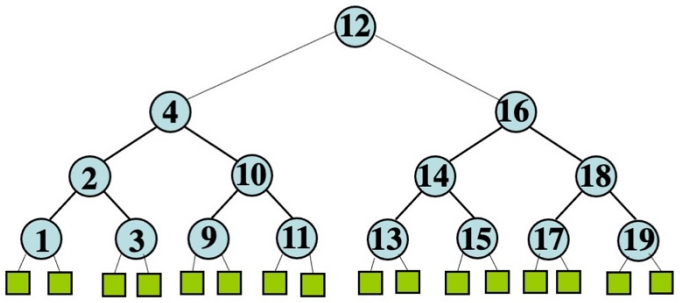
Submission instructions -
Please give the answer to each question entirely on its page, to facilitate marking.

You may draw by hand in paper and scan. You can add each scanned picture into the corresponding page of the word file.
Alternatively, you can print the handout and complete it by hand and scan.
Another possibility is that you can use a handout in pdf, and use a tablet to write on top of the given pdf.

Please join pages into a single pdf.
You only need to hand in pages 2-10.

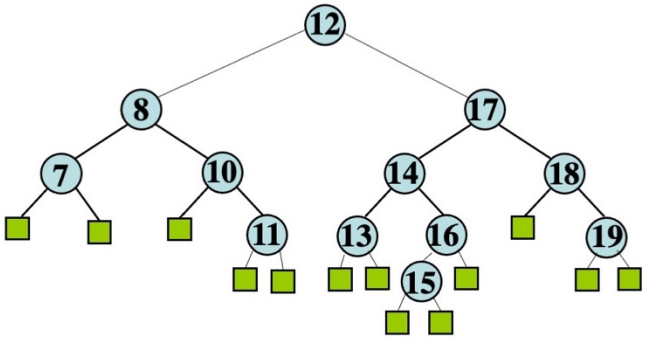
*Your name:*_____ *Your student number:* _____

Question 1 (4 points) Given the following AVL tree



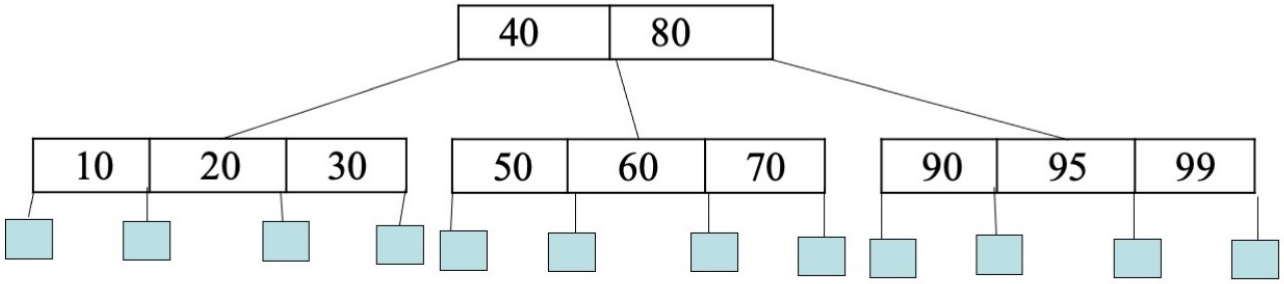
Do AVL insertions, rebalancing when necessary, for the following keys in this order: 6, 5, 7, 8 . Show the tree after each insertion (each insertion ends after rebalancing when applicable). You should show 4 drawings.

Question 2 (4 points)
Remove **7** from the following AVL tree. Show the tree after each rebalancing step.



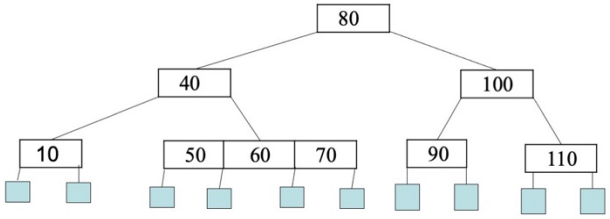
Question 3 (4 points)

Consider the following (2,4)-tree where the last level has dummy leaves.
Insert the following keys, and show the tree after each insertion: insert 15, insert 100 (within each insertion there can be one or more splits)

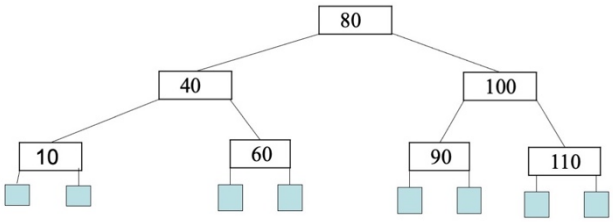


Question 4 (6=2+4 points)

(a) Delete 10 from the following 24-tree (show the tree after each split or trasnfer)



(b) Delete 10 from the following 24-tree : (show the tree after each split or trasnfer)



Question 5. [6=2+2+2 points]

Consider the following Hash table where insertions are done using the hash function

$$h(k) = k \bmod 11.$$

In this exercise you will insert the following keys 25, 3, 10, 32, 46, 58, 57, 26, 17, 65 in an empty Hash table of size 11, **mark the positions (underline/square/circle a key on the position)** that are probed for each insertion (in the given table show the Hash table after each insertion). After all insertions, what is the **average number of probes** (comparisons) needed to search for an existing key in this table? In each part of this question, collisions are resolved by different methods specified in the part.

Hint: Here is an example of how to display your simulation, if only the first 3 keys were inserted with linear probing: inserted 25, 3 and 22. Each row shows the table after the insertion of each key. For each insertion we underline the positions that were probed; since for 25 we probed 1 position, for 3 we probed 2 positions and for 10 we probed 1 position, the average number of probes is (1+2+1)/3 = 4/3=1.33...

0	1	2	3	4	5	6	7	8	9	10
			25							
			25	3						
			25	3						10

key	#probes
25	1
3	2
10	1
average # probes	1.33...

a) **[2 point]** Linear probing with h(k) = k mod 11

0	1	2	3	4	5	6	7	8	9	10
			25	3						10

key	#probes
25	1
3	2
10	1
32	
46	
58	
57	
26	
17	
65	

Average number of probes at the end =

b) **[2 points]** Quadratic probing with h(k) = k mod 11

0	1	2	3	4	5	6	7	8	9	10
			25	3						10

key	#probes
25	1
3	2
10	1
32	
46	
58	
57	
26	
17	
65	

Average number of probes at the end =

c) **[2 points]**
Double hashing with h(k) = k mod 11 and secondary hash function: d(k) = 7 – (k mod 7).
Double hashing is done using $h_j(k) = [h(k) + j \cdot d(k)] \bmod 11$.

0	1	2	3	4	5	6	7	8	9	10
			25				3			10

key	#probes
25	1
3	2
10	1
32	
46	
58	
57	
26	
17	
65	

Average number of probes at the end =

Question 6. [2 points] Consider a hash table that is constructed with quadratic probing:

$h_j(k) = [h(k) + j^2] \bmod 11$

with hash function is $h(k) = (3k + 1) \bmod 11$.

a) [0.5 point] Insert the keys **16** and then **27** in the following hash table and describe the process by indicating for each key the positions probed.

0	1	2	3	4	5	6	7	8	9	10
					5	9	2			

b) [0.5 point] Delete key **0** in the hash table below and describe the process by indicating positions probed.

0	1	2	3	4	5	6	7	8	9	10
	AVAILABLE	0			5	9	2			

c) [0.5 point] Search for key **11** in the hash table and describe the process by indicating positions probed.

0	1	2	3	4	5	6	7	8	9	10
	AVAILABLE	AVAILABLE			5	9	2			11

d) **[0.5 point]** Insert the key **7** in the hash table, describe the process by indicating positions probed.

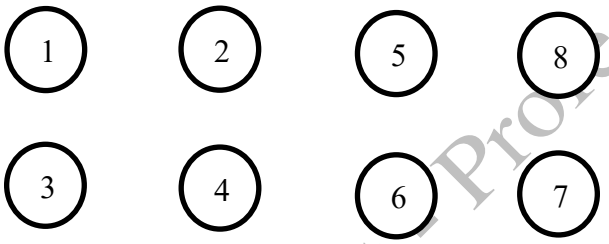
0	1	2	3	4	5	6	7	8	9	10
16	AVAILABLE	AVAILABLE			5	9	2			11

Question 7. **Graph traversals (14 points=2+2+5+5)**

Consider an undirected graph given by the following adjacency list representation

1: (1,2), (1,3), (1,4)
2: (2,1), (2,3) (2,4) (2,5)
3: (3,1), (3,2), (3,4)
4: (4,1), (4,2), (4,3), (4,6)
5: (5,2) (5,6), (5,7), (5,8)
6: (6,4), (6,5), (6,7)
7: (7,5), (7,6), (7,8)
8: (8,5), (8,7)

a) Draw the graph by displaying the edges on the diagram below:



b) Change the representation of the graph from adjacency lists to **adjacency matrix** and show the matrix.

c) Using the DFS algorithm in the Appendix, perform a depth-first search traversal on the given graph starting from node **1** and using the **adjacency lists** representation of the graph. The adjancency lists will influence the order in which the vertices are considered; for example, G.incidentEdges(1) will return the list : (1,2), (1,3), (1,4) so that the edges will be considered in this order.

List the vertices in the order they are visited, and list the edges in the order they are labelled by the algorithm, displaying their labels.

Vertices in order of visit:

Edges and labels in order of visit:

Please give the edges in the order they are labelled, display each edge in the direction of visit, and use the first letter of the label; for example - if a discovery edge was found coming from vertex b to a, the entry for this edge would be displayed "(b,a) D, "

d) Using the BFS algorithm in the Appendix, perform a breadth-first search traversal of the graph starting from node **1** and using the **adjacency lists** representation of the graph.

List the vertices in the order they are visited circling the groups of vertices that belong to each list L_0, L_1, L_2, etc. List the edges in the order they are labelled by the algorithm, displaying their labels. Please, use a similar format as suggested in question 2c.

Vertices in order of visit:

Edges and labels in order of visit:

Depth-first Search algorithm (DFS)

```
Algorithm DFS(G)
      Input graph G
      Output labeling of the edges
of G as discovery edges and back
edges

for all  u ∈ G.vertices()
  setLabel(u, UNEXPLORED)
for all  e ∈ G.edges()
  setLabel(e, UNEXPLORED)
for all  v ∈ G.vertices()
  if  getLabel(v) = UNEXPLORED
      DFS(G, v)
```

```
Algorithm DFS(G, v) Input graph G and a
start vertex v of G
      Output labeling of the edges of G
in the connected component of v as
discovery edges and back edges
      setLabel(v, VISITED)
for all  e ∈ G.incidentEdges(v)
   if  getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if getLabel(w) = UNEXPLORED
          setLabel(e, DISCOVERY)
          DFS(G, w)
      else
          setLabel(e, BACK)
```

Breadth-first Search algorithm (BFS)

```
Algorithm BFS(G)
      Input graph
      Output labeling of the edges
and partition of the vertices of G
for all  u ∈ G.vertices()
  setLabel(u, UNEXPLORED)
for all  e ∈ G.edges()
  setLabel(e, UNEXPLORED)
for all  v ∈ G.vertices()
  if  getLabel(v) = UNEXPLORED
      BFS(G, v)
```

```
Algorithm BFS(G, s)
      L0 ← new empty sequence
L0.insertLast(s)
setLabel(s, VISITED)
i ← 0
while  ! Li.isEmpty()
  Li+1 ← new empty sequence
  for all  v ∈ Li.elements()
    for all  e ∈ G.incidentEdges(v)
     if  getLabel(e) = UNEXPLORED
      w ← opposite(v,e)
      if  getLabel(w) = UNEXPLORED
        setLabel(e, DISCOVERY)
        setLabel(w, VISITED)
        Li+1.insertLast(w)
      else
        setLabel(e, CROSS)
  i ← i +1
```