EECS 560 Fall 2021 Final Exam

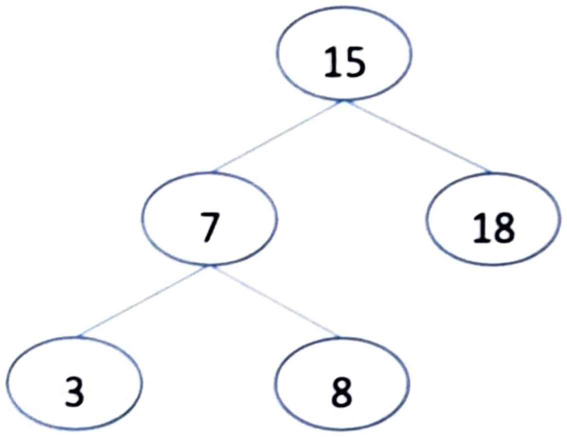
Name: ______________________ **KU ID:**______________________

Time: Dec 16th, 2021 7:30AM-10:00AM. The exam is closed-book and closed-notes.

(1) Describe two essential requirements for hash function design. (10pts)

(2) Recall AVL tree.
What is the AVL property? (2pts)

Draw the resulted AVL tree after inserting 10 into the following AVL tree. (8pts, include all steps to earn partial credits)



(3) Write the pseudocode of the Dijkstra's algorithm for finding the shortest path from an edge-weighted graph that contains no negative edge. Let the source node be s, and your algorithm should find the shortest path between s and all other nodes in the graph. (10pts)

(4) Prove that the Dijkstra's algorithm is correct when assuming no negative edge (7pts). And give a counter example where the algorithm does not work if the graph contains negative edges (3pts).

(5) Write the pseudocode (NOT the C++ code; it doesn't need to be runnable) for the forest-based implementation of disjoint sets with union-by-size and path compression (8pts). Argue that the union($r1, r2$) operation can be done in $O(1)$ time (where $r1, r2$ are the roots of the trees to be merged.) (2pts).

VARIABLE DEFINITIONS: // define the variables (e.g., arrays) need to be used here

METHOD IMPLEMENTATIONS:

init(S): // initialize the disjoint set; where S contains all elements to be stored

find(x): // find the root of an arbitrary element x

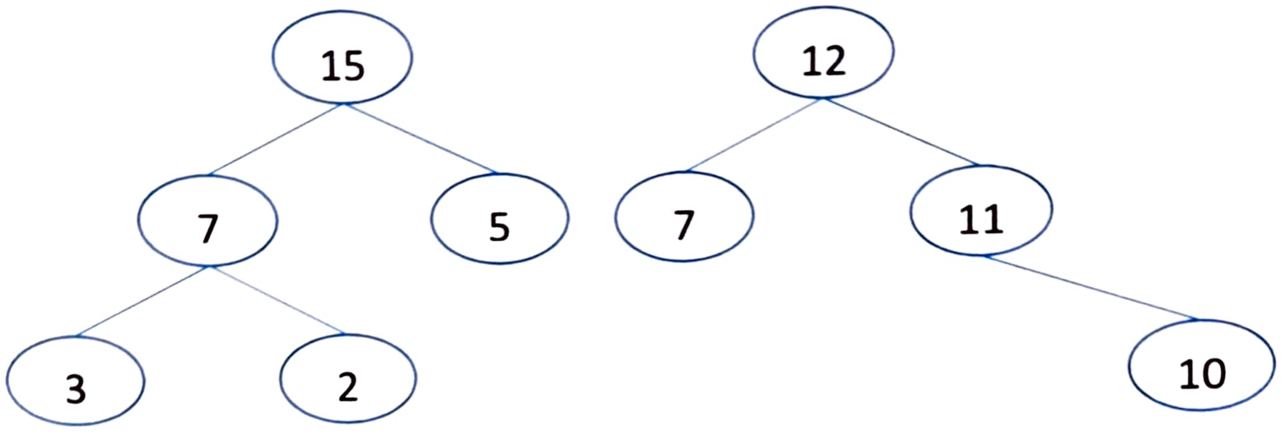
union$(r1, r2)$ // merge the two trees whose roots are $r1, r2$, respectively

union(x, y) // merge the two trees that contains x, y, respectively; x, y are arbitrary elements

Time complexity analysis of **union**$(r1, r2)$:

(6) Continuing from the above question, prove that the find(x) operation is in $O(\log n)$ with union-by-size. Note that x is an arbitrary element contained in the disjoint set. (10pts)

(7) Recall the leftist heap. What is the definition of "null path length"? (2pts) Draw the resulted leftist heap from merging the following two leftist heaps. Note that both heaps are max heaps. The left/right children are indicated by the nodes' relative positions w.r.t the parent. For example, in the first heap, 7 is the left child of 15, and 5 is the right child of 15. In the second heap, 10 is the right child of 11. Show all steps to earn partial credits. (8pts).



(8) Recall the adjacency matrix and adjacency list implementation of the graph data structure. What are the time/space complexity of both implementations w.r.t the following aspects: storage required, adding a new node, adding a new edge, deciding whether an edge exists between two nodes, and enumerating all neighbors of a given node. Use $|V|$ for the number of vertices in the graph, $|E|$ for the number of edges in the graph, and d for the average degree of the graph. Your answer should NOT contain variables other than $|V|$, $|E|$ or $|d|$. (10pts)

STORAGE REQUIRED:

Adjacency matrix:
Adjacency list:

ADDING A NEW NODE:

Adjacency matrix:
Adjacency list:

ADDING A NEW EDGE:

Adjacency matrix:
Adjacency list:

DECIDING WHETHER AN EDGE EXISTS BETWEEN TWO NODES:

Adjacency matrix:
Adjacency list:

ENUMERATING ALL NEIGHBORS OF A GIVEN NODE:

Adjacency matrix:
Adjacency list:

(9) What are the "Big Five" defined in the context of C++ object interface? (10pts)

(10) Write the pseudocode for depth-first search (DFS) and bread-first search (BFS) algorithms. Let s be the source node and g be the graph. Your algorithms should print out the lists of visited nodes. (5pts for DFS, 5pts for BFS)

DFS(s, g)

BFS(s, g)

(II) Write the C++ code (expected to be runnable, while minor glitches in syntax are acceptable) for the top(), enqueue(), dequeue() operations and the destructor function of a queue data structure. The data structure is implemented using singly linked list. Note that your implementation should not cause any memory leak nor leaving any uncollected memory blocks upon the termination of the program. (20pts)

Hint:

- Read the comments in the code carefully.
- When a new element is added, you should use new to allocate memory space for it.
- When destructing the data structure, you should make sure the above allocated memory is collected

DEFINITIONS:

```
#include <iostream>
#include <cstddef>

struct NodeType       {
      int data;           // assuming only store integer type
      NodeType *next;     // pointer to the next element
};

class QueueList       {
  public:
      QueueList()       {      // constructor function
            head = tail = nullptr;
            queue_size = 0;

      }
      ~QueueList();            // destructor function
      NodeType & top(void);
      void enqueue(const NodeType & d);
      void dequeue(void);
  private:
      NodeType *head;          // it does point to real data
      NodeType *tail;          // it does point to real data
      int queue_size;          // the number of elements in the queue
};
```

IMPLEMANTATIONS:

// returns the first element in the queue; print an error message if the queue is empty

```
NodeType & QueueList::top(void)      {




}
```

// add the element d into the queue

```
void QueueList::enqueue(const NodeType & d)  {




}
```

//remove the first element from the queue; print an error message if the queue is empty

```
void QueueList::dequeue(void) {










}
```

// the destructor function

```
QueueList::~QueueList()   {










}
```