**­­­CPSC 2150: Week 11 Lab Midterm 2 Prep**

**Due: As indicated by submission link**

**Total Marks: 10**

**Instructions – PLEASE READ**

1. This work is to be done individually.
2. You should submit only one version via D2L. Check instructions from TA regarding what to submit (zip/just code). Code files must always be included.
3. Keep a copy of everything you submit in some online storage that is accessible by you only.

# EXERCISES

***All Problems must be solved and submitted during lab***

***ALL of STL Containers/Algorithms are allowed!***

1. Write the code for the following methods
   1. Write a function **bool IsMaxHeap(BinaryTree t)** that returns true if t is a max heap; false otherwise.

You may assume each node in the tree has pointers to its left and right children.

BinaryTree class has a data member named root that points to the root node.

bool IsMaxHeap(BinaryTree t){

return IsMaxHeap(t.root);

}

// assume each node contains a member named data that holds the data value inside that node

bool IsMaxHeap(Node\* n){

if(n == nullptr){

return false;

}

else if(n->left && n->right){

if(n->data > n->left->data && n->data > n->right->data){

return IsMaxHeap(n->left) && IsMaxHeap(n->right);

}

else{

return false;

}

}

else if(n->left){

if(n->data > n->left->data){

return IsMaxHeap(n->left);

}

else{

return false;

}

}

else if(n->right){

if(n->data > n->right->data){

return IsMaxHeap(n->right);

}

else{

return false;

}

}

else{

return true; // no left or right child

}

}

* 1. Write a function **bool Hash(LinkedList<string>\* hashTable, int hashTableSize, string f)** that returns true if f is successfully hashed into hashTable; false otherwise.

Use separate chaining to resolve collision.

Assume LinkedList class has methods such as AddFirst, AddLast, RemoveFirst, RemoveLast.

You may use the built-in hash function in your code.

bool Hash(LinkedList<string>\* hashTable, int hashTableSize, string f){

hash<string> hashFun;

int hashCode = hashFun(f);

int index = hashCode % hashTableSize;

hashTable[index].AddLast(f);

return true;

}

// ALTERNATE SOLUTION – has false return scenarios

// my linked list class has a member method *bool contains(string s)* that returns true if string s exists in the linked list; otherwise this method returns false.

bool Hash(LinkedList<string>\* hashTable, int hashTableSize, string f){

if(!hashTable || hashTableSize == 0){

return false;

}

hash<string> hashFun;

int hashCode = hashFun(f);

int index = hashCode % hashTableSize;

if(hashTable[index].contains(f)){

return false;

}

else{

hashTable[index].AddLast(f);

return true;

}

}

* 1. Write a function **int CountEvens(Graph g)** that traverses the graph using DFS and returns the number of vertices that have an even index.

Assume g has as its member:

* vector<int> vertices
* vector<vector<int>> adjacencyList

Each vertex is represented by a single integer (0-indexed).

MUST USE DFS to compute the count!

int CountEvens(Graph g){

int count = 0;

vector<bool> visited(g.vertices.size(), false);

for(int i = 0; i < g.vertices.size(); i++){

if( !visited[i] ){

DFS(g, i, visited, count);

}

}

return count;

}

void DFS(Graph &g, int vertex, vector<bool> &visited, int &count){

if(visited[vertex]){

return;

}

if(vertex % 2 == 0){

count++;

}

visited[vertex] = true;

for(int i = 0; i < g.adjacencyList[vertex].size(); i++){

if( !visited[g.adjacencyList[vertex][i]] ){

DFS(g, g.adjacencyList[vertex][i], visited, count);

}

}

return;

}