CSCE 221 Cover Page
Programming Assignment # 5

FirstName: Eric LastName: Gonzalez

UIN: 322002089

User Name: egoftcp

E-mail address: egoftcp@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more in the Aggie Honor System Office http://aggiehonor.tamu.edu/

Type of sources	
People	
Web pages (provide URL)	stackoverflow.com
Printed material	
Other Sources	CSCE 221 slides

I certify that I have listed all the sources that I used to develop the solutions/code to the submitted work.
"*On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.*"

Your Name (signature) Eric Gonzalez Date 04/17/2015

Program Description

This program is designed to create a series of disjoint set data structures represented as vectors of doubly linked lists and perform the operations MakeSet, Union, and FindSet on them. The heuristics of path compression and union by rank are used to improve performance.

Purpose of the Assignment

The purpose of this assignment was to teach us how to make use of the disjoint set data structure and implement its three basic operations: MakeSet, Union, and FindSet. This assignment will help prepare us for the final project in class.

Data Structures Description

For the DListNode class, the only modifications made were to define the getter and setter functions not already defined. For the DisjointSet class, modifications/implementations made include: the constructor, the destructor, the getnodeLocator() function, the overloaded ostream operator, MakeSet(), Union(), and FindSet().

The MakeSet function takes an integer (“key”) and a value (“node”) for that key and creates a new DListNode based on those parameters. Then, it sets the representative and trailer pointers to point to themselves, and stores the new node into the vector nodeLocator based on the index just before the key.

The FindSet function based on a node will return the representative of that node. When based on a nodeKey, it will search the nodeLocator vector, find the key and return the representative of that node.

The Union function takes in two nodes and sets them equal to two new local DListNodes by using the FindSet function. Then, it checks the list size of both nodes and decides whether the shorter list should be appended to the longer list. To do this, the trailer of the longer list has a “next” pointer that points to the representative of the shorter list. The representative of the shorter list then uses a “previous” pointer to point to the trailer of the longer list, connecting them. Then, it sets the trailer of the longer list to point to the trailer of the shorter list, and sets the trailer of the shorter list to NULL. Finally, it goes through the shorter list and sets each node’s representative point to the representative of the first node, and finishes by updating the list size.

Runtime Analysis

Worst Case: MakeSet() = O(1)

Best Case: MakeSet() = O(1)

Worst Case: FindSet() = O(n)

Best Case: FindSet() = O(1)

Worst Case: Union() = O(n)

Best Case: Union() = (log(n))

Instructions to Compile and Run

Run the following in the program directory:

```
make clean

make

./main
```

Logical Exceptions

The numbering of the sets in the ouput is wrong.

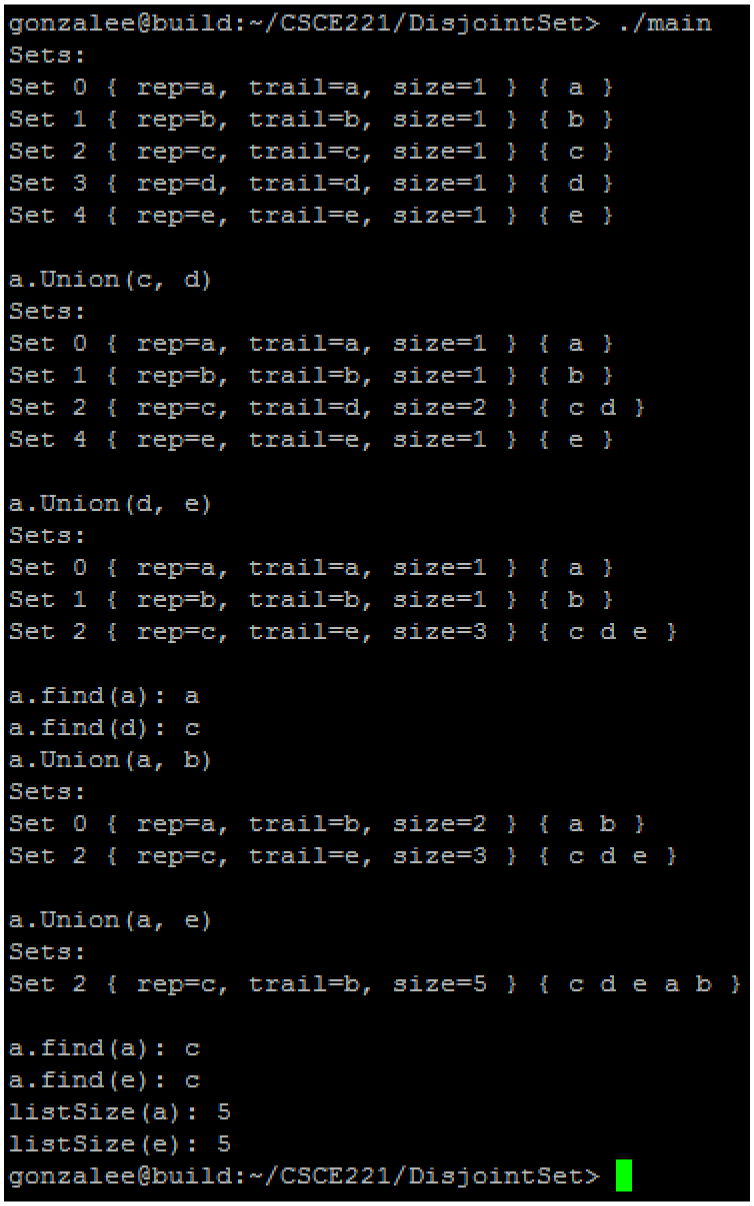
C++ Object Oriented or Generic Programming Features

-Templates

-Auto

Testing Results

Output:



```
gonzalee@build:~/CSCE221/DisjointSet> ./main
Sets:
Set 0 { rep=a, trail=a, size=1 } { a }
Set 1 { rep=b, trail=b, size=1 } { b }
Set 2 { rep=c, trail=c, size=1 } { c }
Set 3 { rep=d, trail=d, size=1 } { d }
Set 4 { rep=e, trail=e, size=1 } { e }

a.Union(c, d)
Sets:
Set 0 { rep=a, trail=a, size=1 } { a }
Set 1 { rep=b, trail=b, size=1 } { b }
Set 2 { rep=c, trail=d, size=2 } { c d }
Set 4 { rep=e, trail=e, size=1 } { e }

a.Union(d, e)
Sets:
Set 0 { rep=a, trail=a, size=1 } { a }
Set 1 { rep=b, trail=b, size=1 } { b }
Set 2 { rep=c, trail=e, size=3 } { c d e }

a.find(a): a
a.find(d): c
a.Union(a, b)
Sets:
Set 0 { rep=a, trail=b, size=2 } { a b }
Set 2 { rep=c, trail=e, size=3 } { c d e }

a.Union(a, e)
Sets:
Set 2 { rep=c, trail=b, size=5 } { c d e a b }

a.find(a): c
a.find(e): c
listSize(a): 5
listSize(e): 5
gonzalee@build:~/CSCE221/DisjointSet>
```