Submission

```
5206267 Luo, Zheng                              3785/4 AEROAH

Submissions:-

S 0     Thu Apr 22 21:46:42 2021        5206267 mon15b ass2 -1:-2

Tue Apr 27 10:27:59 2021                ## wagner.orchestra.cse.unsw.EDU.AU ##
```

Listing

cp: cannot stat ‘/home/cs2521/21T1.work/ass2/mon15b/5206267/!dryrun_record’: No such file or directory

CentralityMeasures.c

```c
// Centrality Measures ADT interface
// COMP2521 Assignment 2
// Written by Zheng Luo (z5206267@ad.unsw.edu.au) on April/2021

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "CentralityMeasures.h"
#include "FloydWarshall.h"
#include "Graph.h"

static EdgeValues initiateEdgeValueStruct(Graph g);
static double numberOfEdgePasses(int edgeSrc, int edgeDes,
                                                                ShortestPaths sps, EdgeValues evs);
/**
 * Finds  the  edge  betweenness  centrality  for each edge in the given
 * graph and returns the results in a  EdgeValues  structure.  The  edge
 * betweenness centrality of a non-existant edge should be set to -1.0.
 */
EdgeValues edgeBetweennessCentrality(Graph g) {
        // Implement the framework for EdgeValues.
        EdgeValues evs = initiateEdgeValueStruct(g);
        // Find the shortest pair for all nodes.
        ShortestPaths sps = FloydWarshall(g);

        // Calculate the number of shortest paths through current edge
        // Which means the number of appearance of current edge in sps.next
        // Determine the edge betweenness in evs.values,
        // by looping thro the 2d array.
        for (Vertex i = 0; i < evs.numNodes; i++) {
                for (Vertex j = 0; j < evs.numNodes; j++) {
                        // Determine the path
                        // if there are adjacent and a path exist.
                        if (GraphIsAdjacent(g, i, j) == true && sps.next[i][j] != -1) {
                                evs.values[i][j] = numberOfEdgePasses(i, j, sps, evs);
                        }
                }
        }

        // Free all memories related to FloydWarshall.
        freeShortestPaths(sps);

        return evs;
}

// This function implement the framework for EdgeValues,
// allocate and assign memories for evs.numNodes and evs.values,
// and return EdgeValues evs at the end.
static EdgeValues initiateEdgeValueStruct(Graph g) {
        // Allocate spaces for edgevalues evs.
        EdgeValues evs;
        evs.numNodes = GraphNumVertices(g);
        evs.values = malloc(evs.numNodes * sizeof(double *));
        for (int i = 0; i < evs.numNodes; i++) {
                evs.values[i] = malloc(evs.numNodes * sizeof(double));
        }
        for (int i = 0; i < evs.numNodes; i++) {
                for (int j = 0; j < evs.numNodes; j++) {
                        evs.values[i][j] = -1.0;
                }
        }
        return evs;
}

// This function searchs for number of edge passed thro the current edge,
// and returns the number of passes as double.
static double numberOfEdgePasses(int edgeSrc, int edgeDes,
                                                                ShortestPaths sps, EdgeValues evs) {
        double counterEdgePasses = 0.0;
        for (int i = 0; i < evs.numNodes; i++) {
```

```
72                  for (int j = 0; j < evs.numNodes; j++) {
73                          int a = i;
74                          int b = j;
75                          // Keep searching until there is no path.
76                          while (sps.next[a][b] != -1) {
77                                  int k = sps.next[a][b];
78                                  if ((a == edgeSrc && k == edgeDes)) {
79                                          counterEdgePasses++;
80                                  }
81                                  a = k;
82                          }
83                  }
84          }
85          return counterEdgePasses;
86  }
87
88  /**
89   * Prints  the  values in the given EdgeValues structure to stdout. This
90   * function is purely for debugging purposes and will NOT be marked.
91   */
92  void showEdgeValues(EdgeValues evs) {
93
94  }
95
96  /**
97   * Frees all memory associated with the given EdgeValues  structure.  We
98   * will call this function during testing, so you must implement it.
99   */
100 void freeEdgeValues(EdgeValues evs) {
101         for (Vertex i = 0; i < evs.numNodes; i++) {
102                 free(evs.values[i]);
103         }
104         free(evs.values);
105
106 }
107
108
```

FloydWarshall.c

```c
// Floyd Warshall ADT interface
// COMP2521 Assignment 2
// Written by Zheng Luo (z5206267@ad.unsw.edu.au) on April/2021

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "FloydWarshall.h"
#include "Graph.h"

static ShortestPaths initiateSpsStruct(Graph g);
static Vertex findNext(ShortestPaths sps, int src, int dest);
/**
 * Finds all shortest paths between all pairs of nodes.
 *
 * The  function  returns  a 'ShortestPaths' structure with the required
 * information:
 * - the number of vertices in the graph
 * - distance matrix
 * - matrix of intermediates (see description above)
 */
ShortestPaths FloydWarshall(Graph g) {
        // Implement the framework for ShortestPaths.
        ShortestPaths sps = initiateSpsStruct(g);

        // First, fill in the value of dist[v][v] itself = 0.
        for (Vertex v = 0; v < sps.numNodes; v++) {
                sps.dist[v][v] = 0;
        }
        // Second, fill in the neighbour distance.
        for (Vertex v = 0; v < sps.numNodes; v++) {
                AdjList ListOutIncident = GraphOutIncident(g, v);
                while (ListOutIncident != NULL) {
                        sps.dist[v][ListOutIncident->v] = ListOutIncident->weight;
                        // Assume in next: node 1 to node 2 will have a next of node 2.
                        sps.next[v][ListOutIncident->v] = ListOutIncident->v;

                        ListOutIncident = ListOutIncident->next;
                }
        }
        // Last step, search the shortest path between inter-vertices.
        for (Vertex k = 0; k < sps.numNodes; k++) {
                for (Vertex i = 0; i < sps.numNodes; i++) {
                        for (Vertex j = 0; j < sps.numNodes; j++) {
                                if (sps.dist[i][j] > sps.dist[i][k] + sps.dist[k][j] &&
                                        sps.dist[i][k] + sps.dist[k][j] > 0) {
                                        sps.dist[i][j] = sps.dist[i][k] + sps.dist[k][j];
                                        sps.next[i][j] = findNext(sps, i, k);
                                }
                        }
                }
        }
        return sps;
}

// The function initiateSpsStruct takes Graph g as argument,
// initialise and allocate memories for dist and next in sps,
// and return ShortestPaths sps.
static ShortestPaths initiateSpsStruct(Graph g) {
        ShortestPaths sps;
        sps.numNodes = GraphNumVertices(g);

        // Implement sps.dist:
        // An 2d array which shows shortest distance between any two vertices.
        sps.dist = malloc(sps.numNodes * sizeof(Vertex *));
        // Implement sps.next:
        // An 2d array which shows next vertex from given vertex to des.
        sps.next = malloc(sps.numNodes * sizeof(Vertex *));
        for (Vertex v = 0; v < sps.numNodes; v++) {
                sps.dist[v] = malloc(sps.numNodes * sizeof(Vertex));
```

```
72                  sps.next[v] = malloc(sps.numNodes * sizeof(Vertex));
73          }
74          // Set the distance between all as infinity.
75          // Set the next array fill with -1.
76          for (Vertex i = 0; i < sps.numNodes; i++) {
77                  for (Vertex j = 0; j < sps.numNodes; j++) {
78                          sps.dist[i][j] = INFINITY;
79                          sps.next[i][j] = -1;
80                  }
81          }
82          return sps;
83  }
84
85  // This findNext is a recusive function to find dest from src,
86  // and return the dest as Vertex when found.
87  static Vertex findNext(ShortestPaths sps, Vertex src, Vertex dest) {
88          if (sps.next[src][dest] == dest) {
89                  return dest;
90          }
91          return findNext(sps, src, sps.next[src][dest]);
92  }
93
94
95  /**
96   * This  function  is  for  you to print out the ShortestPaths structure
97   * while you are debugging/testing your implementation.
98   *
99   * We will not call this function during testing, so you may  print  out
100  * the  given  ShortestPaths  structure in whatever format you want. You
101  * may choose not to implement this function.
102  */
103 void showShortestPaths(ShortestPaths sps) {
104         for (Vertex i = 0; i < sps.numNodes; i++) {
105                 for (Vertex j = 0; j < sps.numNodes; j++) {
106                         if (sps.dist[i][j] != INFINITY) {
107                                 printf("From %d to %d has the shortest distance of %d\n",
108                                 i, j, sps.dist[i][j]);
109                         }
110
111                 }
112         }
113         for (Vertex i = 0; i < sps.numNodes; i++) {
114                 for (Vertex j = 0; j < sps.numNodes; j++) {
115                         if (sps.next[i][j] != -1) {
116                                 printf("From %d to %d has the next vertex of %d\n",
117                                 i, j, sps.next[i][j]);
118                         }
119
120                 }
121         }
122 }
123
124 /**
125  * Frees  all  memory associated with the given ShortestPaths structure.
126  * We will call this function during testing, so you must implement it.
127  */
128 void freeShortestPaths(ShortestPaths sps) {
129         // Free rows for both dist and next first.
130         for (Vertex i = 0; i < sps.numNodes; i++) {
131                 free(sps.dist[i]);
132                 free(sps.next[i]);
133         }
134         // Free dist and next itself.
135         free(sps.dist);
136         free(sps.next);
137 }
138
```

GirvanNewman.c

```c
// Girvan-Newman Algorithm for community discovery
// COMP2521 Assignment 2
// Written by Zheng Luo (z5206267@ad.unsw.edu.au) on April/2021

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#include "CentralityMeasures.h"
#include "GirvanNewman.h"
#include "Graph.h"

#define HEAD -1

static Dendrogram newDendrogram(int v);
static int calculateComponentSeparation(Graph g, Vertex *componentOf,
                                                                        int numOfNodes);
static void bfsSearch(Graph g, Vertex *componentOf, Vertex v, int componentId);
static Dendrogram treeSearchAndInsert(Dendrogram d, Vertex searchValue,
                                                                Vertex src, Vertex dest);
static void storingParentVertex(Vertex *componentOf, Vertex *parentOf,
                                                        int numOfNodes, Vertex src, Vertex dest);

/*
 * Generates a Dendrogram for the given graph g using the Girvan-Newman
 * algorithm.
 *
 * The function returns a 'Dendrogram' structure.
 */
Dendrogram GirvanNewman(Graph g) {
        // 1. Calculate the edge betweenness of all edges in the network.
        EdgeValues evs = edgeBetweennessCentrality(g);

        // Initiate certain memory for pointer Dendrogram d.
        Dendrogram d = newDendrogram(HEAD);
        // Initiate an array of vertex to store component catogory.
        Vertex *componentOf = malloc(evs.numNodes * sizeof(Vertex));
        // Initiate an array to store the imformation (vertex) of its parent.
        Vertex *parentOf = malloc(evs.numNodes * sizeof(Vertex));
        for (Vertex i = 0; i < evs.numNodes; i++) {
                parentOf[i] = -1;
        }
        int componentIdPrev = 1, componentId = 0;
        Vertex src = -1, dest = -1;


        // 4. Repeat Steps 2 and 3 until no edges remain.
        while (GraphNumVertices(g) != 0) {
                // 3. Recalculate the edge betweenness
                // of all edges affected by the removal.
                EdgeValues evs = edgeBetweennessCentrality(g);

                // 2. Remove the edge(s) with the highest edge betweenness.
                // Find the highest edge betweenness first.
                double max = -1;
                for (Vertex i = 0; i < evs.numNodes; i++) {
                        for (Vertex j = 0; j < evs.numNodes; j++) {
                                if (evs.values[i][j] > max) {
                                        max = evs.values[i][j];
                                        src = i;
                                        dest = j;
                                }
                        }
                }
                // Exit the while loop if no edge betweenness > 0
                if (max == -1) {
                        break;
                }

                // Remove selected edges.
                GraphRemoveEdge(g, src, dest);
```

```
    72
    73
                        ^
+ ============================================================================ +
+ Move the code related to counting the number of components to helper functions +
+ ============================================================================ +
    74                  // Algorithm to assign vertices to connected component.
    75                  // e.g. componentOf[v] = 1, v is vertex, and 1 means first component.
    76                  if (componentId != 0) {
    77                          componentIdPrev = componentId;
    78                  }
    79                  componentId = calculateComponentSeparation(g, componentOf,
    80                                                                                  evs.numNodes);
    81                  // If number of components did not increased after edge removal,
    82                  // then remove the edge with same betweenness until different.
    83                  while (componentId == componentIdPrev) {
    84                          for (Vertex i = 0; i < evs.numNodes; i++) {
    85                                  for (Vertex j = 0; j < evs.numNodes; j++) {
    86                                          if (evs.values[i][j] >= max && i != src && j != dest) {
    87                                                  GraphRemoveEdge(g, i, j);
    88                                                  src = i;
    89                                                  dest = j;
    90                                          }
    91                                  }
    92                          }
    93                          componentId = calculateComponentSeparation(g, componentOf,
    94                                                                                  evs.numNodes);
    95                  }
    96
    97                  // Insert src and dest into required location.
    98                  if (parentOf[src] == HEAD) {
    99                          d->left = newDendrogram(src);
   100                          d->right = newDendrogram(dest);
   101                  }
   102                  else {
   103                          d = treeSearchAndInsert(d, parentOf[src], src, dest);
   104                  }
   105                  // Exit the loop if the number of components are enough.
   106                  if (componentId == evs.numNodes) {
   107                          break;
   108                  }
   109
   110                  // Update the parents of vertices.
   111                  storingParentVertex(componentOf, parentOf, evs.numNodes, src, dest);
   112          }
   113
   114          free(componentOf);
   115          free(parentOf);
   116
   117          return d;
   118  }
   119
   120  // Allocate memories for DNode in dendrogram,
   121  // newDendrogram takes one input v,
   122  // used to assign the vertex value in dendrogram,
   123  // this function returns the new dendrogram.
   124  static Dendrogram newDendrogram(int v) {
   125          Dendrogram new = malloc(sizeof(DNode));
   126          new->vertex = v;
   127          new->left = NULL;
   128          new->right = NULL;
   129          return new;
   130  }
   131
   132  // This function searchs number of components and their contents in a graph,
   133  // and update into the array componentOf.
   134  // This function returns the number of components in the graph as int type.
   135  // The componentOf[] can be illustrated thro an example below:
   136  // 0->1->2->3->4->5->6->7
   137  // the link break between 3 and 4,
   138  // 0    1       2       3       4       5       6       7
```

```
// 0    0       0       0       1       1       1       1
// This array componentOf indicates the component of its index belongs to.
static int calculateComponentSeparation(Graph g, Vertex *componentOf,
                                                                        int numOfNodes) {
        for (int v = 0; v < numOfNodes; v++) {
                componentOf[v] = -1;
        }
        int componentId = 0;
        for (int i = 0; i < numOfNodes; i++) {
                if (componentOf[i] == -1) {
                        bfsSearch(g, componentOf, i, componentId);
                        componentId++;
                }
        }
        return componentId;
}

// The bfsSearch function is a
// void type sub-function for calculateComponentSeparation.
// This function search for links/edges between vertices,
// and categorised them into different components,
// a component is consider to be no incoming or outcoming edges
// with other vertices not in the component.
static void bfsSearch(Graph g, Vertex *componentOf,
                                        Vertex v, int componentId) {
        componentOf[v] = componentId;
        AdjList listOfOutgoing = GraphOutIncident(g, v);
        AdjList listOfIncoming = GraphInIncident(g, v);
        while (listOfOutgoing != NULL) {
                if (componentOf[listOfOutgoing->v] == -1) {
                        bfsSearch(g, componentOf, listOfOutgoing->v,
                        componentId);
                }
                listOfOutgoing = listOfOutgoing->next;
        }
        while (listOfIncoming != NULL) {
                if (componentOf[listOfIncoming->v] == -1) {
                        bfsSearch(g, componentOf, listOfIncoming->v,
                        componentId);
                }
                listOfIncoming = listOfIncoming->next;
        }
}

// treeSearchAndInsert takes in a searchValue,
// and search for this value in the Dendrogram d.
// Insert src into the left and dest into the right at found Dendrogram.
// Return Dendrogram d regardless of found or not,
static Dendrogram treeSearchAndInsert(Dendrogram d, Vertex searchValue,
                                                                Vertex src, Vertex dest) {
        if (d == NULL) {
                return d;
        }
        if (d->vertex == searchValue) {
                d->left = newDendrogram(src);
                d->right = newDendrogram(dest);
                return d;
        }
        d->left = treeSearchAndInsert(d->left, searchValue, src, dest);
        d->right = treeSearchAndInsert(d->right, searchValue, src, dest);
        return d;
}

// This a void type function, which stores the parent of each index,
// and used to locate the proper inserting position for
// function treeSearchAndInsert
// The storingParentVertex function can be explain in the same example:
// 0->1->2->3->4->5->6->7
// the link break between 3 and 4,
// 0    1       2       3       4       5       6       7
// -1   -1      -1      -1      -1      -1      -1      -1
```

```
210  // This array parentOf indicates the parent of its index belongs to.
211  // At the moment is -1 (HEAD).
212  // If the link then break between 1 and 2,
213  // 0    1       2       3       4       5       6       7
214  // 3    3       3       3       4       4       4       4
215  // And the function is purposely designed to be 1 step slower.
216  static void storingParentVertex(Vertex *componentOf, Vertex *parentOf,
217  int numOfNodes, Vertex src, Vertex dest) {
218          int srcComponent = componentOf[src];
219          int destComponent = componentOf[dest];
220          for (int i = 0; i < numOfNodes; i++) {
221                  if (componentOf[i] == srcComponent) {
222                          parentOf[i] = src;
223                  }
224                  else if (componentOf[i] == destComponent) {
225                          parentOf[i] = dest;
226                  }
227          }
228  }
229
230  /**
231   * Frees all memory associated with the given Dendrogram  structure.  We
232   * will call this function during testing, so you must implement it.
233   */
234  void freeDendrogram(Dendrogram d) {
235          free(d);
236  }
237
ls: cannot access !dryrun_record: No such file or directory
[ not a regular file ]
gcc -Wall -Werror -g -o testFloydWarshall testFloydWarshall.c FloydWarshall.c Graph.c GraphRead.c
```

```
----------------------------------------
```

```
** FloydWarshall - Compiles OK
```

```
----------------------------------------
```

```
gcc -Wall -Werror -g -o testCentralityMeasures testCentralityMeasures.c CentralityMeasures.c FloydWarshall.c Graph.c
GraphRead.c
```

```
----------------------------------------
```

```
** CentralityMeasures - Compiles OK
```

```
----------------------------------------
```

```
gcc -Wall -Werror -g -o testGirvanNewman testGirvanNewman.c GirvanNewman.c CentralityMeasures.c FloydWarshall.c Graph.c
GraphRead.c BSTree.c
```

```
----------------------------------------
```

```
** GirvanNewman - Compiles OK
```

```
----------------------------------------
```

Tests

```
** Test 1: FloydWarshall (given graph)
--------------------------------
** Test passed
--------------------------------
** Test 2: FloydWarshall (given graph)
--------------------------------
** Test passed
--------------------------------
** Test 3: FloydWarshall (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 4: FloydWarshall (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 5: FloydWarshall (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 6: FloydWarshall (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 7: CentralityMeasures (given graph)
--------------------------------
** Test passed
--------------------------------
** Test 8: CentralityMeasures (given graph)
--------------------------------
** Test passed
--------------------------------
** Test 9: CentralityMeasures (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 10: CentralityMeasures (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 11: CentralityMeasures (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 12: CentralityMeasures (new graph)
--------------------------------
** Test passed
--------------------------------
** Test 13: GirvanNewman (given graph)
--------------------------------
** Test passed
--------------------------------
** Test 14: GirvanNewman (given graph)
--------------------------------
** Test passed
--------------------------------
** Test 15: GirvanNewman (new graph)
--------------------------------
** Test failed (student's output on left, expected on right). Output difference:-
                                                        >    0: {0, 1, 2}
                                                        >    1: {0, 2}
                                                        >    1: {1} (leaf)
                                                        >    2: {0} (leaf)
                                                        >    2: {2} (leaf)

warning: core file may not match specified executable file.
** Stack trace from gdb:

warning: core file may not match specified executable file.
[New LWP 32236]
Core was generated by `./testGirvanNewman graphs/3.in'.
```

```
Program terminated with signal SIGXCPU, CPU time limit exceeded.
#0  0x00005576e55876c2 in ?? ()
#1  0x00007fff25772f80 in ?? ()
#2  0x00005576e55dc590 in ?? ()
#3  0x0000000000000003 in ?? ()
#4  0x00005576e55dc7d0 in ?? ()
#5  0x0000000000000003 in ?? ()
#6  0x00005576e55dc570 in ?? ()
#7  0x0000000100000000 in ?? ()
#8  0x00005576e55dc7f0 in ?? ()
#9  0x00005576e55dc810 in ?? ()
#10 0x0000000100000000 in ?? ()
#11 0x0000000300000003 in ?? ()
#12 0x4000000000000000 in ?? ()
#13 0x00000001000000c2 in ?? ()
#14 0x0000000100000001 in ?? ()
#15 0x0000000300000001 in ?? ()
#16 0x00005576e55dc4f0 in ?? ()
#17 0x00007fff25772ea0 in ?? ()
#18 0x00005576e5587441 in ?? ()
#19 0x00007fff25772f88 in ?? ()
#20 0x0000000200000000 in ?? ()
#21 0x00005576e55893e0 in ?? ()
#22 0x00005576e5587130 in ?? ()
#23 0x00007fff25772f80 in ?? ()
#24 0x00005576e55dc590 in ?? ()
#25 0x00005576e55893e0 in ?? ()
#26 0x00007fe68db9809b in ?? ()
#27 0x0000000000000000 in ?? ()
--------------------------------
** Test 16: GirvanNewman (new graph)
--------------------------------
** Test passed
--------------------------------
```

Assessment

```
!!perftab       ** PERFORMANCE ANALYSIS **

Test  1 (1)     FloydWarshall (given graph)  ..  ..  !!PASSed
Test  2 (1)     FloydWarshall (given graph)  ..  ..  !!PASSed
Test  3 (0.75)  FloydWarshall (new graph) .  ..  ..  !!PASSed
Test  4 (0.75)  FloydWarshall (new graph) .  ..  ..  !!PASSed
Test  5 (0.75)  FloydWarshall (new graph) .  ..  ..  !!PASSed
Test  6 (0.75)  FloydWarshall (new graph) .  ..  ..  !!PASSed
Test  7 (1)     CentralityMeasures (given graph) ..  !!PASSed
Test  8 (1)     CentralityMeasures (given graph) ..  !!PASSed
Test  9 (0.75)  CentralityMeasures (new graph)   ..  !!PASSed
Test 10 (0.75)  CentralityMeasures (new graph)   ..  !!PASSed
Test 11 (0.75)  CentralityMeasures (new graph)   ..  !!PASSed
Test 12 (0.75)  CentralityMeasures (new graph)   ..  !!PASSed
Test 13 (1.5)   GirvanNewman (given graph)   ..  ..  !!PASSed
Test 14 (1.5)   GirvanNewman (given graph)   ..  ..  !!PASSed
Test 15 (1.5)   GirvanNewman (new graph) ..  ..  ..  !!FAILed (-1.5)
Test 16 (1.5)   GirvanNewman (new graph) ..  ..  ..  !!PASSed

!!perfmark      ** TOTAL PERFORMANCE MARK:    14.5/16

 ^
 + ======================================================== +
 + Pretty good style :)                                     +
 + A minor issue is your function GivanNewman is a bit long. +
 + ======================================================== +
!!marktab       **  MARKER'S  ASSESSMENT  **

                    Style and Complexity  (4)   3.5

!!finalmark     **  FINAL  ASSIGNMENT  MARK:    18/20

5206267 Luo, Zheng                              3785/4 AEROAH


Marked by z5314098 on Fri May 14 00:36:16 2021



Marked by z5314098 on Fri May 14 00:54:23 2021
```