Anuj Sampat
asampat@bu.edu

CS 566 Programming Assignment 3

Problem

To implement the STRONGLY-CONNECTED-COMPONENT algorithm using the Depth-First-Search method for graph traversal.

Solution

The STRONGLY-CONNECTED-COMPONENT algorithm was implemented using the C language.

Data Structures Used:

VERTEX and ADJ_VERTEX:

```
typedef struct adjVertex_t
{
    VERTEX * graphVertex;
    struct adjVertex_t * next;
} ADJ_VERTEX;
```

The **ADJ_VERTEX** structure maintains a list of pointers to graph vertices that are adjacent to a vertex.

- 'graphVertex' is a pointer to an adjacent graph VERTEX.
- 'next' gives the next adjacent graph vertex in the list.

```
typedef struct vertex_t
{
    int vertexNum;
    int color, discoverTime, finishTime;
    struct vertex_t * parent;
    struct vertex_t * discoveredComponent;
    ADJ_VERTEX * adjacencyList;
} VERTEX;
```

The **VERTEX** structure represents a single vertex in the graph.

- 'vertexNum' is the input vertex number for the vertex. This must be a value ranging from 0 to MAX_VERTICES (100000).

- 'color', 'discoverTime' and 'finishTime' are various bookkeeping parameters as used in the DEPTH-FIRST-SEARCH algorithm.

- 'parent' points to the strongly connected parent VERTEX of the current VERTEX.

- 'discoveredComponent' points to the strongly connected child VERTEX of the current VERTEX.

- 'discoveredComponent' and 'parent' are used to maintain linked lists of strongly connected components linked with the current VERTEX.

- 'adjacencyList' represents the adjacency list for the current VERTEX. It is a list of ADJ_VERTEX structures.

GRAPH:

```
typedef struct graph_t
{
    int dfsTime;
    int numVertices;
    VERTEX * vertexList[MAX_VERTICES + 1];
    VERTEX * sortedVertexList[MAX_VERTICES + 1];
    VERTEX ** dfsVertexList;
    int currSortIndex;
} GRAPH;
```

- 'dfsTime' is a time counter used for keeping track of vertex discovery and finish times.

- 'numVertices' is the total number of vertices in the graph. This must not be more than MAX_VERTICES + 1 (100001).

- 'vertexList' is a list of pointers to the vertices in the graph.

- 'sortedVertexList' is a list that stores pointers to vertices discovered during a DEPTH-FIRST-SEARCH on the graph. This list stores pointers to vertices in the increasing order of the finish time of each vertex during the search. This list is traversed in reverse order during a DEPTH-FIRST-SEARCH on the transpose of the graph for computing the strongly connected components.

- 'dfsVertexList' points to the list of graph vertices on which the DEPTH-FIRST-SEARCH is performed. It points to 'vertexList' during the DEPTH-FIRST-SEARCH on the graph G and to 'sortedVertexList' during the DEPTH-FIRST-SEARCH on the transpose graph G'.

- 'currentSortIndex' is a bookkeeping parameter that is used to populate 'sortedVertexList'.

Functions Used:

```
static GRAPH * initializeGraph(void)
```

- This function will allocate, initialize and return a pointer to a GRAPH data structure.

```
static GRAPH * freeGraph(GRAPH * currGraph)
```

– This function will free the memory allocated for the graph 'currGraph' and its vertices and return a NULL pointer.

```
static VERTEX * createGraphVertex(int vertexNum)
```

- This function will allocate, initialize and return a pointer to a VERTEX representing 'vertexNum'.

```
static int createGraphEdge(GRAPH * currGraph, int srcVertexNum, int destVertexNum)
```

- This function will create the vertices 'srcVertexNum' and 'destVertexNum' in the graph 'currGraph' if they don't already exist and connect them via the graph adjacency list. The vertices are inserted in ascending order of the vertex number to facilitate similar traversal of the adjacency list during a DEPTH-FIRST-SEARCH. Returns 0 on success, -1 otherwise.

```
static void depthFirstVisit(GRAPH * currGraph, VERTEX * currVertex) and static void depthFirstSearch(GRAPH * currGraph)
```

- These functions are an implementation of the DEPTH-FIRST-SEARCH algorithm. The depthFirstVisit() method will populate the 'sortedVertexList' in 'currGraph' in increasing order of the vertex finish times. The 'sortedVertexList' is traversed in reverse during a DEPTH-FIRST-SEARCH on the graph transpose.

```
static void depthFirstSearchTranspose(GRAPH * transposeGraph)
```

- This function will do a depth first search of the graph pointed to by 'transposeGraph' in the reverse order of the vertices pointed to by 'dfsVertexList'.

```
static GRAPH * getTranspose(GRAPH * currGraph)
```

- This function will return a pointer to a GRAPH structure that represents a transpose of 'currGraph'.

```
static void stronglyConnectedComponents(GRAPH * currGraph)
```

- This function finds the strongly connected components in 'currGraph' using the STRONGLY-CONNECTED-COMPONENTS algorithm. It will result in the printing of the strongly connected components to screen and to an output file.

```
int main(int argc, char * argv[])
```

- The main function to read the input adjacency list and run the strongly connected components algorithm.

Files Provided:

p3_Sampat_Anuj.c: The STRONGLY-CONNECTED-COMPONETS C implementation.

p3_Sampat_Anuj.out: The C executable.

Compiling, Execution And Output:

The C file was compiled using gcc version 3.4.6 on a Red Hat Linux 3.4.6-9 system using the command:

$ cc p3_Sampat_Anuj.c -o p3_Sampat_Anuj.out

The executable file can be run using the command:

$ p3_Sampat_Anuj.out <input file path> <outputFilePath>

The no <inputFilePath> is given, the program will look for the inputfile 'p3_input.txt' in the current working directory. If no <outputFilePath> is specified, the program will write the output to the file 'p3_output.txt' in the current working directory.