

Indian Institute of Technology, Guwahati
Department of Computer Science and Engineering
Data Structure Lab (CS210)

Assignment: 6

Date: 29th September, 2016.

Total Marks: 30 (lab assignments) + 20 (offline assignments)

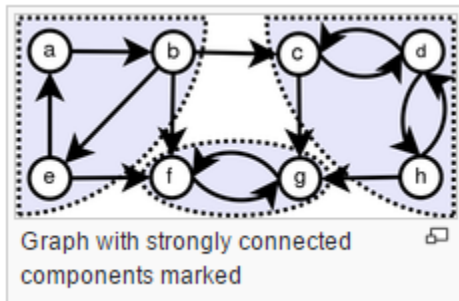
Deadline of Offline Assignment Submission: 5th October, 2016.

Lab Assignments:

1. Let $G = (V, E)$ be a **directed** graph represented as **Adjacency list**. For your reference, a sample code to create Adjacency List is given at the end of the assignment. However, you can choose your own adjacency list representation.
 - a. Write a function that will **transpose** a directed graph G . A transpose of a directed graph G is another directed graph G^T on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G .
 - b. Write down the **depth-first-search (DFS)** traversal algorithm of a graph. Output would be the DFS traversal of the graph vertices.
 - c. Write a function to find all **strongly connected components (SCCs)** in a directed graph. You **MUST** reuse the functions written for 1.a and 1.b in your solution.

(10 + 10 + 10 = 30)

Consider the following graph as your input graph.



Offline Assignments:

2. Write an $O(V + E)$ time algorithm to compute the component graph (i.e., graphs of SCCs) of directed graph $G = (V, E)$. Make sure that there is at most one edge between two vertices in the component graph. For this problem, input is a directed graph. Reuse your solutions for Problem 1 to get the SCCs of the input graph. The component graph is defined as follows:

(10)

Let S_1, S_2, \dots, S_k be the SCCs of G . The graph of SCCs is G^{SCC}

- Vertices are S_1, S_2, \dots, S_k
- There is an edge (S_i, S_j) if there is some $u \in S_i$ and $v \in S_j$ such that (u, v) is an edge in G .

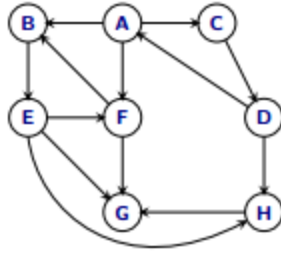


Figure: Graph G

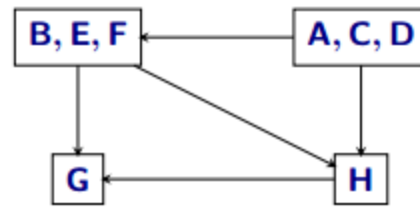


Figure: Graph of SCCs G^{SCC}

3. Suppose that the undirected graph is represented as adjacency matrix. Write Prim's Minimum Spanning Tree(MST) algorithm for this case that runs in $O(V^2)$ time.

(10)

Submission Instructions:

- Name of every file should be $\langle \text{Roll_No} \rangle_ \langle \text{Question_No} \rangle .c$. eg. 150101086_1.c, 150101086_2.c and so on.
- Name of the folder you upload should be $\langle \text{Roll_No} \rangle$ eg. 154101086.
- Please do not upload files like a.out, desktop.ini, etc. If there are 'N' questions in assignment, there must be exactly 'N' ".c" or ".cpp" files in the folder. Use only zip to compress the folder to be uploaded.
- You will be awarded ZERO mark if the above rules are violated.
- Please note that you will not be able to submit once the dead line (with 12 hours' late submission) is over.

Reference Code to Create Adjacency List:

// A C Program to demonstrate adjacency list representation of graphs

```
#include <stdio.h>
#include <stdlib.h>
// A structure to represent an adjacency list node
struct AdjListNode
{
    int dest;
    struct AdjListNode* next;
};
```

```

// A structure to represent an adjacency list
struct AdjList
{
    struct AdjListNode *head; // pointer to head node of list
};

// A structure to represent a graph. A graph is an array of adjacency lists.
// Size of array will be V (number of vertices in graph)
struct Graph
{
    int V;
    struct AdjList* array;
};

// A utility function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int dest)
{
    struct AdjListNode* newNode =
        (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// A utility function that creates a graph of V vertices
struct Graph* createGraph(int V)
{
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct Graph));
    graph->V = V;

    // Create an array of adjacency lists. Size of array will be V
    graph->array = (struct AdjList*) malloc(V * sizeof(struct AdjList));

    // Initialize each adjacency list as empty by making head as NULL
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;

    return graph;
}

// Adds an edge to an undirected graph
void addEdge(struct Graph* graph, int src, int dest)
{
    // Add an edge from src to dest. A new node is added to the adjacency
    // list of src. The node is added at the beginning
    struct AdjListNode* newNode = newAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;

    // Since graph is undirected, add an edge from dest to src also
    newNode = newAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}

```

```

// A utility function to print the adjacency list representation of graph
void printGraph(struct Graph* graph)
{
    int v;
    for (v = 0; v < graph->V; ++v)
    {
        struct AdjListNode* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl)
        {
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}

// Driver program to test above functions
int main()
{
    // create the graph given in above figure
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);

    // print the adjacency list representation of the above graph
    printGraph(graph);

    return 0;
}

```

Output:

```

Adjacency list of vertex 0
head -> 4-> 1
Adjacency list of vertex 1
head -> 4-> 3-> 2-> 0
Adjacency list of vertex 2
head -> 3-> 1
Adjacency list of vertex 3
head -> 4-> 2-> 1
Adjacency list of vertex 4
head -> 3-> 1-> 0

```