SC1007 Data Structures and Algorithms **2022/23 Semester 2**

Lab 6: Graph Representation

School of Computer Science and Engineering *Nanyang Technological University*

Q1 Write a function adjM2adjL() to convert an adjacency matrix to an adjacency list. The structure of a graph is given below.

```
enum GraphType {ADJ_MATRIX , ADJ_LIST}; // Types of Graph Representation

typedef struct _listnode
{
    int vertex;
  struct _listnode *next;
} ListNode;

union GraphForm{
    int **matrix;
    ListNode **list;
};

typedef struct _graph{
    int V;
    int E;
    enum GraphType type;
    union GraphForm adj;
}Graph;
```

for [v][i], create a node for every vertex v and nodes for every i visited in a linked list form

```
for v loop
   create listnode v
   for i loop
        if [v][i] = 1
           create listnode i and traverse v linked list. Add
           listnode i to the end
```

```
for v loop
  temp = v
  traverse once
  while not null
      get value of node as i, update [v][i] =1
```

the vertices are named from 1 to $|V|$.

The function prototype is given as follows:

```
void adjM2adjL(Graph *g);
```

Q2 Write a function adjL2adjM() to convert an adjacency list to an adjacency matrix. Please reuse the work down in Q1. The function prototype is given as follows:

```
void adjL2adjM(Graph *g);
```

Q3 The degree of a vertex v of a graph is the number of edges incident on v. Write a function calDegreeV() to compute vertex degrees using adjacent lists and using adjacency matrix. Please reuse the work done in Q1 and Q2.

```
void calDegreeV(Graph g, int *degreeV)
```

if type LL , go down LL and count

if type matrix for [v][i], go down i++ and count

Q4 Write a function BFS() to do a breadth first search from a input vertex v and print out the visited vertices in the order of visiting. The labels of v are from 1 to $|V|$. The algorithm will visit the neighbor nodes in ascending order. The function prototype is given as follows:

queue and dequeue

```
void BFS(Graph g, int v)
```

Remark Please make sure that your program will not be crashed by continuously converting between two representation forms and the degree of vertices is correctly computed in every conversion.

```
void BFS(Graph g, int v){
    // Mark all the vertices as not visited
    bool visited[MAX_VERTICES];
    for (int i = 0; i < g->V; i++) {
        visited[i] = false;
    }

    // Create a queue for BFS

    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    // Mark the current node as visited and enqueue it

    visited[s] = true;
    queue[rear++] = s;

    while (front != rear) {
        // Dequeue a vertex from queue and print it

        s = queue[front++];
        printf("%d ", s);

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it

        for (int adjacent = 0; adjacent < g->V;
             adjacent++) {
            if (g->adj[s][adjacent] && !visited[adjacent]) {
                visited[adjacent] = true;
                queue[rear++] = adjacent;
            }
        }
    }
}
```