CS29003 Algorithms Laboratory

Assignment No: 11

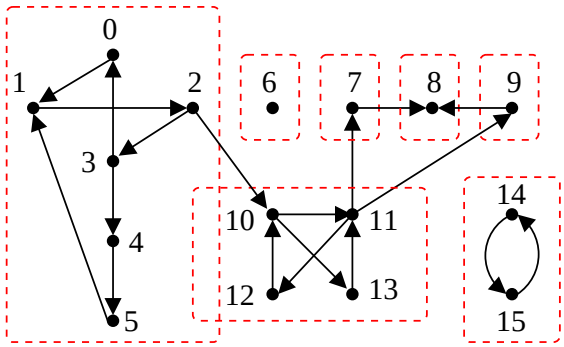
Last date of submission: 09–November–2017

A group of n scientists are on a secret mission. Everyday, the mission manager sends some of the scientists to the work site for collecting secret data. These data are communicated to the remaining scientists by special communication devices which allow anonymous transmissions only. For secrecy, scientists are not given the communication addresses of all other scientists. When a scientist receives a new message, (s)he forwards it to the contacts known to him/her. The mission manager must ensure that the secret data are passed on to every scientist, and wants to achieve this goal by sending to the work site as few scientists as possible.

The interacting capabilities of the scientists are represented by a directed graph $G = (V, E)$. The vertices are the scientists $V = \{0, 1, 2, \ldots, n-1\}$. If Scientist i knows the communication address of Scientist j, then there is a directed edge from Vertex i to Vertex j. The graph is not necessarily symmetric, that is, the presence of an edge (i, j) does not imply the presence of the edge (j, i). Since communications are anonymous, a recipient does not obtain the communication address of the sender from the message transcripts.

Two scientists i and j are connected to each other if there is a directed path from i to j and there is also a directed path from j to i, in the interaction graph G. A mutually connected subgroup of scientists is a subset $W \subseteq V$ such that for every two different $i, j \in W$, Scientists i and j are connected to each other. We want the mutually connected subgroups to be as large as possible. Clearly, sending to the work site one scientist from each such subgroup suffices to ensure that the secret data are eventually passed on to everybody.

As an example, consider the following interaction graph. The maximal mutually connected subgroups are $\{0, 1, 2, 3, 4, 5\}$, $\{6\}$, $\{7\}$, $\{8\}$, $\{9\}$, $\{10, 11, 12, 13\}$, and $\{14, 15\}$. Notice that although Scientists 7–13 can be reached from Scientists 0–5, the converse does not hold, and so there are multiple subgroups containing these scientists. Every isolated vertex (like 6) must be a subgroup by itself. But the converse is again not necessarily true, as exemplified by Vertices 7–9.



Part 1: Write a function *gengraph* to prepare the interaction graph $G = (V, E)$ as follows. The function first reads the number n of vertices and the number m of edges in the graph, from the user. Assume that the user ensures $0 \leqslant m \leqslant n(n-1)$. The user then supplies m edges (u_i, v_i) with each $u_i, v_i \in V$.

The graph is stored in the adjacency-list form. Each node in the adjacency list needs to store the vertex number of a (directed) neighbor. The lists should be kept sorted to avoid duplicate insertions of edges.

The graph is an array of n headers. Each header contains a pointer to the adjacency list of that vertex. In addition, the header may contain some extra information for use in Parts 2 and 3. For example, Part 2 may require you to store the following information in each header: a serial number, a low number, and a flag. The meanings of these items are explained in Part 2.

Also, write a function *prngraph* to print the adjacency lists of all the vertices (see Sample Output).

Part 2: A modified version of depth-first search (DFS) helps one identify the maximal mutually connected subgroups. The DFS gives a serial number to each vertex indicating the order in which the vertices are visited. Initially, all serial numbers are -1, a value standing for *unvisited*. The low number to be stored in

a vertex is the smallest serial number of the vertices in its subgroup, that can be reached by DFS tree edges and backward edges. A stack S of vertices is also maintained. A flag stored in each vertex indicates whether the vertex currently resides in the stack. A pseudocode of DFS(u) is given below.

```
Push u to S, and set the present-in-the-stack flag for u.
Give the next serial number to u.
Initialize the low number of u to its serial number.
For each directed neighbor v of u, repeat:
    If v is unvisited, do:
        Call DFS on v recursively.
        Set low(u) = min(low(u), low(v)).
    Else if (u,v) is a backward edge and v is present in the stack, do:
        Set low(u) = min(low(u), serialno(v)).
If serialno(u) is equal to low(u), repeat:      /* A new subgroup is located */
    Set v to be the top of the stack. Print v.
    Pop v from the top of the stack, and reset the present-in-the-stack flag for v.
    If v is equal to u, the subgroup is completely generated.
```

The DFS may start from any vertex. Multiple invocations of the DFS may be necessary (earlier invocations may leave certain vertices unvisited). Write a function *findsubgroups* to compute the maximal mutually connected subgroups. The function should make the outermost calls of the DFS procedure outlined above.

Part 3: The determination of the mutually connected subgroups does not still solve the minimization problem of the mission manager. The example above has seven maximal mutually connected subgroups, whereas sending only three scientists to the work site suffices: one of Scientists 0–5, Scientist 6, and one of Scientists 14–15. Propose and implement an efficient algorithm to solve this minimization problem. Write a function *mincount* to solve this part. The function prints which are the essential subgroups (a member of each of which must be sent to the site), and returns the count of the essential subgroups.

The *main*() function

- Call *gengraph* to read the graph G from the user. Call *prngraph* to print the graph.
- Call *findsubgroups* to print all maximal mutually connected subgroups.
- Call *mincount* to identify the essential subgroups, and to return the count of such subgroups.

Sample output

```
n = 16
m = 19
 0 1     1 2     2 3     3 0     3 4     4 5     5 1     7 8     9 8
11 7    11 9     2 10   10 11   11 12   12 10   10 13   13 11   14 15
15 14

+++ The input graph
0  ->  1
1  ->  2
2  ->  3 10
3  ->  0  4
4  ->  5
5  ->  1
6  ->
7  ->  8
8  ->
9  ->  8
10 -> 11 13
11 ->  7  9 12
12 -> 10
13 -> 11
14 -> 15
15 -> 14
```

```
+++ The subgroups are:
--- Subgroup  0:  8
--- Subgroup  1:  7
--- Subgroup  2:  9
--- Subgroup  3: 13 12 11 10
--- Subgroup  4:  5  4  3  2  1  0
--- Subgroup  5:  6
--- Subgroup  6: 15 14

+++ The essential subgroups are:
--- Subgroup  4 is essential
--- Subgroup  5 is essential
--- Subgroup  6 is essential
    Count = 3
```

Submit a single C/C++ source file. Do not use global/static variables.