CS29003 Algorithms Laboratory

Assignment No: 9

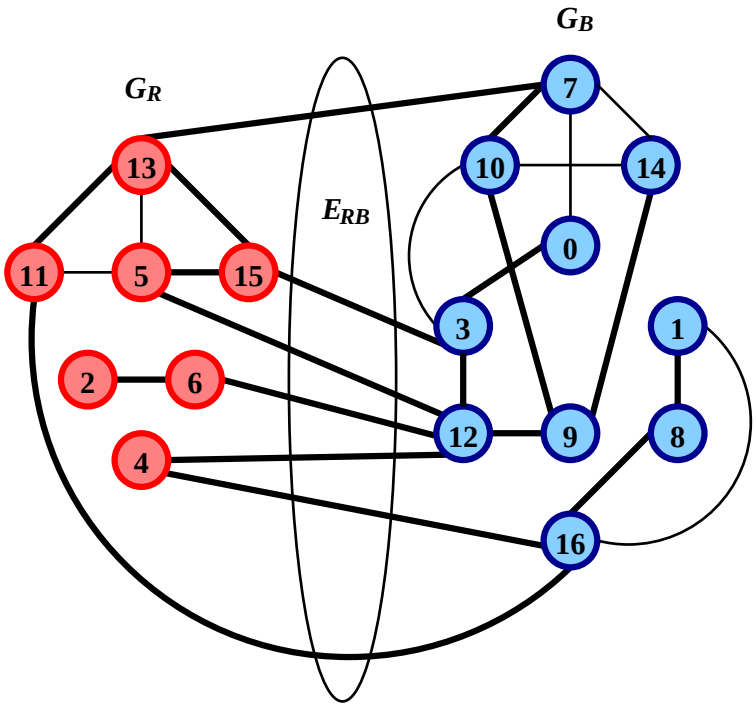
Date: 31–March–2020

Graph Traversals and Their Applications

Let $G = (V, E)$ be an undirected graph. Some vertices of G are red, and the others are blue. Let V_R denote the set of red vertices of G, and V_B the set of blue vertices of G. The vertex set V of G is the union of the two disjoint sets V_R and V_B. Likewise, the edge set E of G is the union of three mutually disjoint subsets: the set E_{RR} of edges with both endpoints red, the set E_{BB} of edges with both endpoints blue, and the set E_{RB} of edges with endpoints of two different colors. We have the two induced subgraphs: the red subgraph $G_R = (V_R, E_{RR})$, and the blue subgraph $G_B = (V_B, E_{BB})$. We will define a subgraph G_{RB} of G later (in Part 5).

A cycle in the red subgraph G_R is called a red cycle. Likewise, a cycle in the blue subgraph G_B is called a blue cycle. A red or a blue cycle is called *monochromatic*, since such a cycle consists of vertices of the same color. On the other hand, a cycle in G with vertices of both the colors is called *nonmonochromatic*. In this assignment, you write a program to identify the existence of monochromatic and nonmonochromatic cycles, and print some of them.

The following figure demonstrates such a graph. A red cycle in the graph is $(5, 11, 13, 15)$. A blue cycle in the graph is $(0, 3, 12, 9, 14, 7)$. These cycles are monochromatic. Two nonmonochromatic cycles in the graph are $(5, 12, 4, 16, 11, 13, 15)$ and $(13, 7, 10, 9, 12, 3, 15, 5)$. We assume that a cycle does not contain repeated vertices (except the first and the last ones). For a reason to be explained later, some of the edges of G are shown in bold. The two examples of nonmonochromatic cycles given above use only the bold edges.



Part 1: Storage of an undirected graph

Write a user-defined data type to store the following information about an undirected graph.

- The number of vertices in the graph (an integer).
- The colors of the vertices (an array of characters r and b).
- The vertex numbers (an array of integers): Let G have n vertices. These vertices are naturally numbered as $0, 1, 2, 3, \ldots, n-1$. However, the vertices of the monochromatic subgraphs cannot be so numbered. The red subgraph in the above example has the vertex set $\{2, 4, 5, 6, 11, 13, 15\}$. These vertices may be indexed as $0, 1, 2, 3, 4, 5, 6$, but their original numbers from G should be stored.

- The edges of the graph: Use the adjacency-list (not matrix) representation to store the edges. Since we are dealing with undirected graphs, for every undirected edge (u,v), v must appear in the adjacency list of u, and u in the adjacency list of v. There is no need to keep the adjacency lists sorted (with respect to the numbers of the neighboring vertices).

Part 2: Read and print a graph

Write a function *readgraph* to return a graph G constructed from user inputs. The user first enters the number n of vertices of G, followed by the colors of the vertices, and finally by the list of edges. Each edge is specified by a pair (u,v) (see the sample I/O section). It is the responsibility of the user to avoid entering the same edge multiple times. When the user is done with entering the edges, -1 is entered as u in order to indicate the end of the input session.

Write another function *prngraph* to print a graph in a format illustrated in the sample I/O.

Part 3: Get the red and the blue subgraphs

Write a function $getcolgraph(G, color)$ that, given the graph G and a $color$ (r or b), generates G_R or G_B as $color$ suggests, and returns this subgraph.

Part 4: DFS traversal in a graph, and detection of cycles

Write a recursive DFS function, and a multi-DFS function for an input graph. The traversal carries out two additional tasks. First, whenever a back edge is detected, the cycle causing this back edge is printed along with the colors of the vertices on the cycle (see Sample I/O). All cycles in a graph are not printed by a multi-DFS traversal. It suffices to print only the cycles corresponding to the back edges. Second, all the edges of the DFS forest are stored in the parent representation in an array. You also need to maintain the levels of the nodes in the DFS trees in order to identify the back edges. To be more precise, if u recursively calls DFS on v, you should set $parent[v] = u$, and $level[v] = level[u] + 1$. Of course, you additionally need to use a *visited* array. The parent array is to be returned by the multi-DFS function.

Part 5: Construct the graph G_{RB}

The parent array returned by multi-DFS on G_R identifies a set of edges $F_{RR} \subseteq E_{RR}$ defining the DFS forest in G_R. Likewise, we have the edges $F_{BB} \subseteq E_{BB}$ of the DFS forest in G_B. Write a function *getrbgraph* to create and return the graph $G_{RB} = (V, F_{RR} \cup F_{BB} \cup E_{RB})$. Pass, as arguments to this function, whatever you need for the construction of G_{RB}. The edges of G_{RB} are shown as bold in the figure on the first page. The existence of nonmonochromatic cycles in G is related to G_{RB} as follows:

Claim: *G contains a nonmonochromatic cycle if and only if G_{RB} contains a cycle.*

The *main*() function

- Call *readgraph* to construct the graph G from user inputs. Call *prngraph* to print G.
- Call *getcolgraph* twice with $color = r$ and $color = b$ to get G_R and G_B. Print the graphs.
- Invoke the multi-DFS function, once on G_R and once more on G_B, to print the red and the blue cycles detected by the traversals. As by-products, you also get the red and the blue forest edges returned in the parent arrays by the two calls of multi-DFS.
- Construct G_{RB} using *getrbgraph*. Print the graph.
- Call multi-DFS on G_{RB} to detect the existence of nonmonochromatic cycles in G. The returned parent array has no role for this call.

Theoretical Exercise (*not for submission*): Prove the claim given in Part 5.

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

Sample I/O

```
20
b r b b b b r b r b r r r b b b b r b b
0  2     0  10    0  15    0  19    1  5     1  16    2  7     2  9     3  12    4  5     4  10
6  17    6  18    6  19    7  13    9  16    10 15    13 15    13 18    13 19    16 17    18 19
-1

+++ Original graph (G)
    [b]   0 ->   2, 10, 15, 19
    [r]   1 ->   5, 16
    [b]   2 ->   0,  7,  9
    [b]   3 ->  12
    [b]   4 ->   5, 10
    [b]   5 ->   1,  4
    [r]   6 ->  17, 18, 19
    [b]   7 ->   2, 13
    [r]   8 ->
    [b]   9 ->   2, 16
    [r]  10 ->   0,  4, 15
    [r]  11 ->
    [r]  12 ->   3
    [b]  13 ->   7, 15, 18, 19
    [b]  14 ->
    [b]  15 ->   0, 10, 13
    [b]  16 ->   1,  9, 17
    [r]  17 ->   6, 16
    [b]  18 ->   6, 13, 19
    [b]  19 ->   0,  6, 13, 18

+++ Red subgraph (GR)
    [r]   1 ->
    [r]   6 ->  17
    [r]   8 ->
    [r]  10 ->
    [r]  11 ->
    [r]  12 ->
    [r]  17 ->   6

+++ Blue subgraph (GB)
    [b]   0 ->   2, 15, 19
    [b]   2 ->   0,  7,  9
    [b]   3 ->
    [b]   4 ->   5
    [b]   5 ->   4
    [b]   7 ->   2, 13
    [b]   9 ->   2, 16
    [b]  13 ->   7, 15, 18, 19
    [b]  14 ->
    [b]  15 ->   0, 13
    [b]  16 ->   9
    [b]  18 ->  13, 19
    [b]  19 ->   0, 13, 18

+++ Red cycles

+++ Blue cycles
    (15, 13, 7, 2, 0), Color: (b, b, b, b, b)
    (19, 18, 13, 7, 2, 0), Color: (b, b, b, b, b, b)
    (19, 18, 13), Color: (b, b, b)

+++ Nonmonochromatic graph (GRB)
    [b]   0 ->  10,  2
    [r]   1 ->  16,  5
    [b]   2 ->   9,  7,  0
    [b]   3 ->  12
    [b]   4 ->  10,  5
    [b]   5 ->   1,  4
    [r]   6 ->  19, 18, 17
    [b]   7 ->  13,  2
    [r]   8 ->
    [b]   9 ->  16,  2
    [r]  10 ->  15,  4,  0
    [r]  11 ->
    [r]  12 ->   3
    [b]  13 ->  18, 15,  7
    [b]  14 ->
```

```
    [b]  15 ->  10, 13
    [b]  16 ->  17,  1,  9
    [r]  17 ->  16,  6
    [b]  18 ->   6, 19, 13
    [b]  19 ->   6, 18

+++ Multi-color cycles
    (19, 6, 18), Color: (b, r, b)
    (4, 5, 1, 16, 17, 6, 18, 13, 15, 10), Color: (b, b, r, b, r, r, b, b, b, r)
    (7, 2, 9, 16, 17, 6, 18, 13), Color: (b, b, b, b, r, r, b, b)
    (2, 9, 16, 17, 6, 18, 13, 15, 10, 0), Color: (b, b, b, r, r, b, b, b, r, b)
```