CS29003 Algorithms Laboratory

Assignment No: 8

Last date of submission: 20–March–2018

Let $G = (V, E)$ be an undirected graph with $|V| = n$ and $|E| = m$. As usual, we take $V = \{0, 1, 2, \ldots, n-1\}$. Consider two subsets $V_1, V_2 \subseteq V$ of vertices with $|V_1| = n_1$ and $|V_2| = n_2$. The distance between two vertices $u, v \in V$, denoted $d(u, v)$, is the length of the shortest u,v-path if u and v are connected, or ∞ otherwise. Define the distance between V_1 and V_2 as

$$d(V_1, V_2) = \min\{d(u, v) \mid u \in V_1,\ v \in V_2\}.$$

If $V_1 \cap V_2 \neq \emptyset$, then $d(V_1, V_2) = 0$. This is an uninteresting case, so assume that $V_1 \cap V_2 = \emptyset$. This implies that $n_1 + n_2 \leqslant n$. Renumbering the vertices, we can always arrange that $V_1 = \{0, 1, 2, \ldots, n_1 - 1\}$ and $V_2 = \{n - n_2, n - n_2 + 1, \ldots, n - 2, n - 1\}$. In what follows, we assume that V_1 and V_2 are these sets.

Part 1: Graph Construction

Represent G in the adjacency-list format. Write a function **`readgraph`** to generate a graph from user inputs. The user enters n and m first. The user then enters m edges (pairs of endpoints). Assume that the user does not enter the same edge multiple times.

Write another function **`printgraph`** to print a graph in the format illustrated in the sample output.

Part 2: Modified BFS

The breadth-first traversal can be used to find shortest distances from the vertex from which the traversal begins. You need to keep track of the levels of visited vertices in the BFS queue. Since we are interested in $d(V_1, V_2)$, we start the traversal from vertices of V_1, and stop the traversal as soon as a vertex $v \in V_2$ is located (during an enqueue or dequeue operation). The level of the vertex v is returned. If the BFS terminates without ever visiting a vertex of V_2, the BFS function should return ∞.

Part 3: Computing $d(V_1, V_2)$ by Method 1

For all $u \in V_1 = \{0, 1, 2, \ldots, n_1 - 1\}$, call $\text{BFS}(G, u)$ as implemented in Part 2, and compute the minimum of the n_1 returned values. This is clearly the desired value of $d(V_1, V_2)$. Implement this algorithm in a function **`computedist1`** that runs in $O(n_1(n + m))$ time.

Part 4: Computing $d(V_1, V_2)$ by Method 2

Write an $O(n + m)$-time function **`computedist2`** to compute $d(V_1, V_2)$. Prepare a graph H from G, in which V_1 is contracted to a single vertex, and V_2 is contracted to another single vertex. This conversion should finish in $O(n + m)$ time. Now, make a single invocation of BFS on H.

The *main*() function

- Read the graph from user inputs. Print the graph. (See Part 1.)
- Call **`computedist1`** to display the n_1 values returned by the BFS calls, and their minimum (which is $d(V_1, V_2)$).
- Contract the graph as suggested in Part 4. Print the contracted graph H. Run BFS on H, and print the returned value.

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

Sample output

```
n = 12
m = 15
n1 = 3
n2 = 2
 3 10     9 7      7 10     5 1      2 6     11 6      9 6      7 3      4 8      8 10
 3 5      8 2      9 5      2 1      3 11

+++ The constructed graph
    0  ->
    1  ->  2,  5
    2  ->  1,  8,  6
    3  -> 11,  5,  7, 10
    4  ->  8
    5  ->  9,  3,  1
    6  ->  9, 11,  2
    7  ->  3, 10,  9
    8  ->  2, 10,  4
    9  ->  5,  6,  7
    10 ->  8,  7,  3
    11 ->  3,  6

+++ Method 1

    BFS(0) returns INFINITY
    BFS(1) returns 3
    BFS(2) returns 2

--- d(V1,V2) = 2

+++ Method 2

--- The contracted graph
    0  ->  8,  6,  5
    1  ->
    2  ->
    3  -> 11,  7,  5
    4  ->  8
    5  ->  0,  9,  3
    6  -> 11,  0,  9
    7  -> 11,  9,  3
    8  -> 11,  0,  4
    9  ->  7,  6,  5
    10 ->
    11 ->  8,  7,  6,  3

--- d(V1,V2) = 2
```