CS29002 Algorithms Laboratory

Assignment No: 11

Last date of submission: 26–October–2016

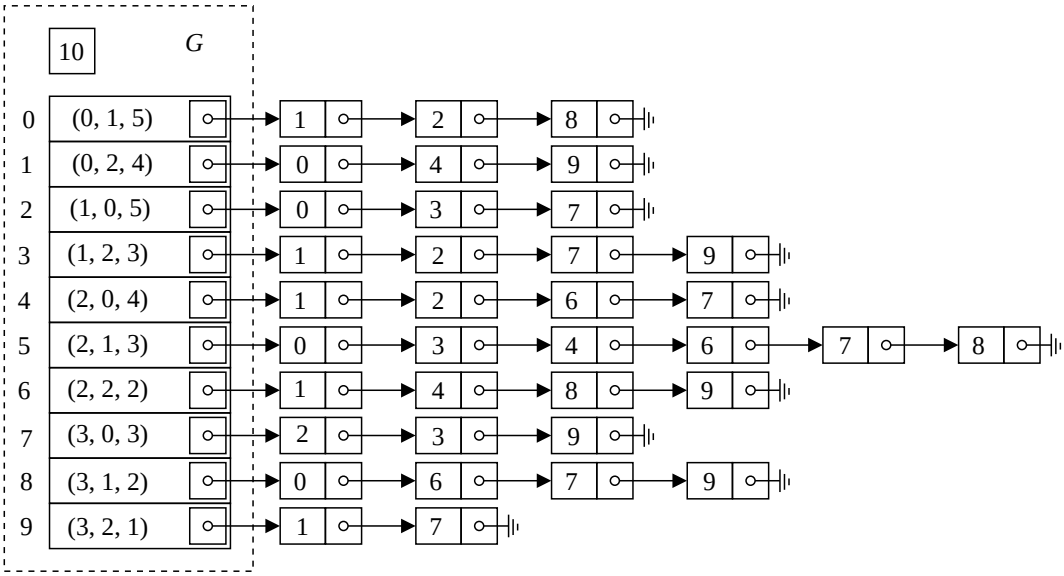
You have three glasses of capacities C_1, C_2, C_3 (positive integers). There is water of quantities a_1, a_2, a_3 (non-negative integers) initially in the glasses. You want the glasses to finally contain water of quantities b_1, b_2, b_3 (non-negative integers). Assume that $a_1 + a_2 + a_3 = b_1 + b_2 + b_3 = T$ (say). You do not have any measuring device. All you can do is to take some non-empty Glass i and start pouring its content into a non-full Glass j ($j \neq i$). You stop when either Glass i becomes empty or Glass j becomes full, whichever happens earlier. You are required to find out whether such a sequence of moves can change the glass contents from (a_1, a_2, a_3) to (b_1, b_2, b_3). In this assignment, you formulate this as a graph-theoretic problem, and solve it by a graph-traversal algorithm.

Part 1: To start with, we construct a directed graph $G = (V, E)$ that stores the information about the moves mentioned above. Let the contents of the three glasses after some moves be (u_1, u_2, u_3). We call this a valid configuration if the following three conditions are satisfied:

1. Each u_i is an integer in the range $0 \leqslant u_i \leqslant C_i$.
2. $u_1 + u_2 + u_3 = T$.
3. For at least one $i \in \{1, 2, 3\}$, we have $u_i = 0$ or $u_i = C_i$.

The initial configuration (a_1, a_2, a_3) need not be valid, but assume that the final configuration (b_1, b_2, b_3) is valid. The vertices of G stand for all valid configurations along with the initial configuration. You add a directed edge from configuration (u_1, u_2, u_3) to configuration (v_1, v_2, v_3) if a single move can change the glass contents from (u_1, u_2, u_3) to (v_1, v_2, v_3).

Represent the digraph G in the adjacency-list format. Let n be the number of vertices in G. The vertices are indexed as $0, 1, 2, \ldots, n-1$. Each node stores a configuration and a pointer to the linked list storing the indices of the neighbors. The following figure illustrates such a representation. This corresponds to $(C_1, C_2, C_3) = (3, 2, 5)$, and $(a_1, a_2, a_3) = (2, 1, 3)$.



Write a function *gengraph*$(a_1, a_2, a_3, C_1, C_2, C_3)$ that returns G in the format mentioned above. Notice that the graph does not depend on the final configuration (b_1, b_2, b_3). It is desirable to list the vertices in some sorted order of the configurations (u_1, u_2, u_3), like the lexicographic ordering of the triples. Do not use any sorting function. Generate the valid configurations in the sorted order. If the initial configuration (a_1, a_2, a_3) is not valid, insert it in the correct position after the complete list of valid configurations is generated. It is not mandatory to keep the adjacency lists sorted with respect to the indices of the neighbor.

Part 2: Write a function *reachable*(G,s,t) that determines and returns the decision whether Vertex t can be reached from Vertex s by following the directed edges of G. Here, s and t are indices in the range $0,1,2,\ldots,n-1$, where $n=|V|$. These indices correspond to the the nodes storing respectively the initial configuration (a_1,a_2,a_3) and the final configuration (b_1,b_2,b_3). Write a recursive function to perform DFS traversal in G. The function *reachable* would call this DFS function with the indices s and t passed along with other necessary parameters. The DFS traversal starts at Vertex s. The search succeeds if and only if Vertex t is ever reached during the traversal.

Part 3: Suppose that the call of *reachable* returns *true*. You then want to compute a path from s to t. Write a function *prnpath*(G,s,t) to do this. This can be done by a second DFS traversal. One possibility is to build the DFS tree using parent pointers during the traversal. Once t is reached, you follow the t-to-s pointers, and print the desired s,t path in the reverse order of the vertices encountered by the parent pointer traversals.

Another possibility is to keep track of the current DFS path. When you are at a vertex u, the traversal defines a unique path from s to u. If $u=t$, print this path. Add an additional array parameter to your DFS function in order to pass this path. Also pass the length of the current path.

G may contain multiple s,t paths (given that t is reachable from s). Your DFS traversal discovers one of these paths, depending upon the sequence of vertices explored from each visited vertex. Print whichever s,t path your DFS traversal discovers. DFS traversals have a tendency of discovering long paths, whereas BFS traversals produce shortest paths. But you do not need to worry about the length of the path you print.

The *main*() function

- The user first enters (C_1,C_2,C_3), (a_1,a_2,a_3), and (b_1,b_2,b_3).
- Call *gengraph* to obtain the graph G. Print the adjacency information of the returned graph G in a format as shown in the sample output.
- Determine the indices s and t of the initial and the final configurations in the vertex list of G.
- Call *reachable*, and print whether t can be reached from s.
- If t is unreachable from s, exit. Otherwise, call *prnpath* to print an s,t path, and exit.

Sample output

```
+++ Capacities : ( 3, 2, 5)
+++ Start      : ( 2, 1, 3)
+++ End        : ( 1, 0, 5)

+++ Graph created
    n = 10
    Node  0 : ( 0, 1, 5) ->  1  2  8
    Node  1 : ( 0, 2, 4) ->  0  4  9
    Node  2 : ( 1, 0, 5) ->  0  3  7
    Node  3 : ( 1, 2, 3) ->  1  2  7  9
    Node  4 : ( 2, 0, 4) ->  1  2  6  7
    Node  5 : ( 2, 1, 3) ->  0  3  4  6  7  8
    Node  6 : ( 2, 2, 2) ->  1  4  8  9
    Node  7 : ( 3, 0, 3) ->  2  3  9
    Node  8 : ( 3, 1, 2) ->  0  6  7  9
    Node  9 : ( 3, 2, 1) ->  1  7

+++ Source (s) : 5
+++ Target (t) : 2

+++ Sequence of moves exists

+++ A sequence is:
    (2,1,3) -> (0,1,5) -> (0,2,4) -> (2,0,4) -> (1,0,5)
```

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