ENEE641 : Mathematical Foundations of Computer Engineering - Programming Assignment

---

Naman Gupta
(116949330)

1 Algorithm

The objective is to print an Euler Tour for a given undirected, connected, simple graph G(V, E) if one exists. V is the vertex set contains all the vertices in the graph and E is the edge set containing all the edges in the graph.

1.1 isEulerTour()

An Euler Tour exists in undirected, simple graph only if the graph is connected and every vertex has even degree. In code 116949330.c, function ***isEulerTour()*** checks that an euler tour exists in the given un-directed, connected and simple graph. The algorithm is pretty straight forward as the data structure used here is linked list. It checks the number of nodes in every list, i.e., checks the no. of edges connected with every vertex. If the no. of edges are even then the it returns 1 and if for any vertex the no. of edges are not even, the function stops there and returns 0.

In the worst case, that is when an Euler Tour exists, the function takes O(|V||E|) runtime as every vertex is taken into account and from that vertex every edge is taken twice because of vertex pair to form an edge.

1.2 getEulerCircuit()

After we get a flag=1 that means Euler Tour exists, we move to the Hierholzer's Algorithm as implemented in function ***getEulerCircuit()***.

- The Hierholzer's algorithm chooses any vertex v randomly (we're not doing that) and starts following the trail of edges until it reaches the vertex v again. If the edge set E is not empty (unexplored edges) and the algorithm reaches the vertex v, it checks that if any edge is still remaining to explore that are connected to vertex v. If yes then it keeps on following the trail until it reaches again to vertex v.
- After exploring all the edges connected to vertex v, we pick a vertex u that is in the current euler tour but has adjacent edges which aren't explored yet. The algorithm loops over the unexplored edges and connect the current euler tour to the previous euler tour. Check if the edge set E is empty (no unexplored edges) or not.
- Repeat the above step unless the edge set E is empty.

The Hierholzer's Algorithm runs until it exhausts the edge set E, therefore the runtime for the algorithm is O(|E|).

2 Output

I have used doubly linked list as the data structure to store the adjacency list. Let's understand what the code does. ***getEulerCircuit()***.

1. Starting from vertex 1 the edge trail is followed until it reaches 1. The edges that are explored in the graph are popped and stored in the current tour list (curr_path) as the algorithm traverses over the graph. (Note : Doubly linked list is used to store the current tour and final euler tour). When no more edges are remaining for that are adjacent to vertex 1, then the vertex one is stored in eulerTour list.
2. Now, a vertex is popped from the current tour list and above step is repeated until all the adjacent edges to the new current vertex are exhausted. Again as the edges are explored they are added in the current tour list.
3. The above step is repeated until all the nodes are popped from the current tour list and stored in eulerTour list. The final eulerTour list generated is the Euler Tour of the graph

Since, we used doubly linked list, the popping process involves adding and deleting a node. Adding a node in linked list takes O(1) runtime and deleting a node from linked list takes O(|V|) runtime in worst case. This makes our algorithm O(|V||E|) runtime in total to get an Euler Tour. Let's understand the algorithm using in1.txt file. The adjacency list is stored in linked list in the following form

1 -> 2 -> 3
2 -> 1 -> 4 -> 5 -> 6
3 -> 1 -> 4
4 -> 2 -> 3 -> 5
5 -> 2 -> 4 -> 6
6 -> 2 -> 5

Our algorithm starts from Vertex 1 counting no. of nodes in the list, i.e., adjacent edges to vertex 1 make by vertex pair (1,2) and (1,3). There are two edges adjacent to vertex 1. Similarly, 4 edges adjacent to vertex 2, 2 edges adjacent to vertex 3. But 3 edges adjacent to vertex 4 which is not the even no. of edges. So our algorithm function ***isEulerTour()*** return 0 which means no Euler Tour exists. The output is printed in A1.txt as

0

The algorithm takes 0.000175 seconds but it varies every time we run the code. Every vertex takes 3 operations, 1 for for loop, 1 for calling next vertex from graph and 1 for checking the no. of edges. Every edge takes 3 C commands, 1 for iterating over adjacent edges, 1 for calling the other vertex adjacent to same node and 1 for increments the counter which counts the no. of edges adjacent to every vertex. In B1.txt we can find the no. of C commands are used for each vertex, each edge, maximum no. of operations charged to any single vertex, single edge and total no. of C operations used by the algorithm as

Total no. of operations to Vertex 1 : 3
Total no. of operations to Edge(1,2) : 3
Total no. of operations to Edge(1,3) : 3
Total no. of operations to Vertex 2 : 3
Total no. of operations to Edge(2,1) : 3
Total no. of operations to Edge(2,4) : 3
Total no. of operations to Edge(2,5) : 3
Total no. of operations to Edge(2,6) : 3
Total no. of operations to Vertex 3 : 3
Total no. of operations to Edge(3,1) : 3
Total no. of operations to Edge(3,4) : 3
Total no. of operations to Vertex 4 : 3
Total no. of operations to Edge(4,2) : 3
Total no. of operations to Edge(4,3) : 3
Total no. of operations to Edge(4,5) : 3
Total no. of operations to Vertex 5 : 0
Total no. of operations to Vertex 6 : 0
Total no. of operations to Vertex 7 : 0
Total no. of operations to Vertex 8 : 0
Maximum number of operations charged to any single vertex is : 3
Maximum number of operations charged to any single edge is : 3
Total number of operations is : 233