**Question 2: Perfect Complete Graph**

Divyansh Mathur, 210357

1. **CONDITION 1: For a directed graph to be perfect complete graph every two node should have exactly one edge between every pair of distinct vertices**

**CONDITION 2: For any three vertices a, b, c, if (a, b) and (b, c) are directed edges, then (a, c) is present in the graph.**

1. **To prove:** If directed graph is a Perfect Complete Graph then between any pair of vertices, there is at most one edge, and for all k ∈ {0, 1, . . . , n − 1}, there exist a vertex v in the graph, such that Outdegree(v) = k

Consider this node,

Where k edges leaves this node (outdegree) and g edges enter this node (indegree). Total edges entering + leaving this node = n-1 since perfect complete graph condition 1. Therefore g=n-1-k

Assertion: Any two nodes can’t have same outdegree.

Proof: Lets assume two nodes have same outdegree=k, name that A and B:

A circular object with arrows pointing to the center

Description automatically generatedA circular object with arrows pointing to the center

Description automatically generated

C

B

A circular object with arrows pointing to the center

Description automatically generated

A

There is a edge between B and A. Assume B points to A.

Now take another node C which directs to B. There will be total n-1-k such nodes.

Now using condition 2 of perfect complete graph, this C node also points to A, So total indegree of A due to B and Cs=1+n-k-1 ( 1 from B and 1 from n-k-1 Cs)

Minimum possible indegree of A= n-k

Maximum possible outdegree of A=k-1 (since total edges from a node=n-1 condition 1)

But this contradicts our assumption that A had outdegree =k.

Hence proved

**ii.) To prove:** If in a directed graph, between any pair of vertices, there is at most one edge, and for all k ∈ {0, 1, . . . , n − 1}, there exist a vertex v in the graph, such that Outdegree(v) = k then the graph is perfect complete graph.

According to the given statement there are n nodes everyone has different outdegree from 0 to n-1.

The node with outdegree n-1 has n-1 edges implies it points to all other (n-1) nodes,

Similarly the node with outdegree =n-2 points to all nodes except itself and except the above node {because the above node has already edge to this node and it is given that any two nodes can have at most 1 edge between them}

….

The node with 0 outdegree donot point to any node but it has indegree=n-1 because of above analysis.

From above analysis the following statement can be stated:

**Statement X: If oudegree(b) > outdegree(a), then** **there is an edge between a and b where b points to a and  
if b points to a then outdegree(b)> outdegree(a)**

Therefore, a node is pointed by all nodes whose outdegree is greater than the node’s outdegree. And it points to all nodes whose outdegree less than the node's outdegree. Hence total edges at each node =n-1

any 2 nodes taken at a time are connected. (condition 1 of perfect complete graph)

Assume any three vertices a, b, c, such that (a, b) and (b, c) are directed edges i.e. a points to b and b points to c. This implies that Outdegree(a)> Outdegree(b) > Outdegree(c)

Outdegree(a)> Outdegree(c), using statement X, there is an edge between a and c where a points to c. (condition 2 of perfect complete graph).

Hence Proved.

1. **Adjacency matrix:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sending nodes | 0 | 1 | . | . | n-1 |
| Receiving node |
| 0 |  |  |  |  |  |
| 1 |  |  |  |  |  |
| . |  |  |  |  |  |
| . |  |  |  |  |  |
| n-1 |  |  |  |  |  |

We can make use of an observation that each node is having an edge with each other node. Therefore if we see, adj[i][j] and adj[j][i] exactly one of them should be 1 and other should be 0. Now, we see the outdegree for each node and store them in another array say Out[n].

Then According to a.) part this array should have all values from 0 to n-1. Sort this array and compare with whole no. from 0 to n-1

*Pseudocode:*

**Is\_perfect\_complete\_graph{**

if(check(matrix)==false){

    return false;

}else{

    for(i from 0 to n-1){

        out[i]=0;

        for(j from 0 to n-1){

            out[i]=out[i]+matrix[i][j];

        }

    }

    sort(out.begin, out.end);

    for(i from 0 to n-1){

        if(i!=out[i]) return false;

    }

    return true;

}

}

**check(matrix**){

  for(i from 0 t0 n-1){

        for(j from 0 to n-1){

            if(matrix[i][j]+matrix[j][i]!=1) return false;

        }

    }

    return true;

}

**Time Complexity:**

In check function we are visiting each node one time. This take O(n2) time. Further our algorithm sorts the n element array. This may take O(nlogn) time worst case also we are seeing each element of out array takes O(n) time. So our overall time complexity =O(n2).