PROBLEM

An Euler Tour of a _connected_ directed graph G(V, E) is a cycle that traverses each edge of G exactly once although it may visit a vertex more than once.

I give an Algorithm to check whether a given graph contains an Euler tour and if it does the algorithm will give the Euler tour also.

APPROACH

We check whether a graph contains an Euler tour or not by checking the condition in-degree(v)=out-degree(v) for all the vertices v in the graph. If the condition holds for all the vertices in the graph then we can say that it has an Euler tour.

We start with a node and continue as long as we can move along the edges. The only restriction is to use each edge exactly once. Since a graph with Euler Tour will have in-degree(v)=out-degree(v) for all vertices in the graph, we eventually reach the starting node. Since the selection of the next edge during our traversal is arbitrary, we may reach the start node without actually covering all the edges.

So, we see if there are any edges that are not yet covered and start once again with a node connected to the uncovered edge and continue as in the previous case and in the same way we reach the start node at the end.

In this way we will get a set of cycles formed by disjoint sets of edges. Since our graph is a connected graph, each cycle will have at least a node common with another cycle. So, we can merge the cycles using the common nodes in the cycles and get a large cycle covering all the edges exactly once.

Example: Consider the following graph, which has an Euler tour.

Example: Consider the following graph, which has an Euler tour.

---

1_connected-A graph is connected if you can get from any node to any other by following a sequence of edges; in the case of a directed graph you are allowed to go the wrong way along an arrow.
Applying my approach we will get the following cycles:

\[ a \rightarrow b \rightarrow d \rightarrow c \rightarrow a \]
\[ b \rightarrow c \rightarrow e \rightarrow b \]

Merging the two cycles using the common node that is ‘b’, we get the following cycle, which is an Euler tour for the graph:

\[ a \rightarrow b \rightarrow c \rightarrow e \rightarrow b \rightarrow d \rightarrow c \rightarrow a \]

We can write the algorithm using my approach as follows:

**ALGORITHM**

**Procedure MAIN(**A[1…n]**))
Begin
   /*Program to find whether a given graph contains an Euler tour or not. If it contains this program will give the Euler tour. *A[1..n] is an array of pointers containing the adjacency lists for all the nodes named 1..n */
   if(!CHECK(**A[1…n]**)) then{
      return("No Euler Tour in the given graph");
   } else EULER( **A[1…n]**);
endMAIN;

**Procedure CHECK(**A[1…N]**))
Begin
   /* Procedure to check whether a graph contains an Euler tour or not*/
   i←1;
   while(i<=n) do {
      s←A[i];
      while(s!=NULL) {
         out[i]←out[i]+1;
         in[s→value]← in[s→value] + 1;
         s←s→next;
      }
      i++;
   } for i←1 to n step by 1 do {
      if (in[i] != out[i]) then return (FALSE);
   }
   return(TRUE);
end CHECK;
Procedure EULER(*a[1..N])
Begin
/* Procedure to generate the Euler tour in the given graph. A linked list L is used to store
the nodes and start is the pointer pointing to the start node in the linked list. ADD adds an
edge to the linked list and ADD_REMOVE adds an edge to the linked list and removes
the edge from the adjacency list. */
position$\leftarrow$start;
ptr$\leftarrow$start;
k$\leftarrow$1;
j$\leftarrow$1;
ADD(position, j);
i$\leftarrow$ A[j$\rightarrow$value;
while (k != n) {

/* Finding a cycle arbitrarily */
while (j!=i) {
    ADD_REMOVE(position, i);
    position$\leftarrow$position$\rightarrow$next;
    i$\leftarrow$ A[i$\rightarrow$value;
}
/* Adding the edge towards the first node at which we have started to
complete the cycle*/
ADD_REMOVE(position, i);

/*Updating the variables for finding another cycle in the next iteration*/
s$\leftarrow$ptr;
while ((A[k]=NULL) or (k != s$\rightarrow$value)) do {
    if (A[k]=NULL) then k$\leftarrow$k+1;
    else s$\leftarrow$s$\rightarrow$next;
}
ptr$\leftarrow$s;
position$\leftarrow$s;
j$\leftarrow$k;
i$\leftarrow$ A[j$\rightarrow$value;
}
/* Printing the Euler tour */
s$\leftarrow$start;
while(s!=NULL) do {
    PRINT(s$\rightarrow$value);
    s$\leftarrow$s$\rightarrow$next;
}
end EULER;
PROOF OF THE ALGORITHM

ClaimA: A connected directed graph $G(V, E)$ has an Euler path if it satisfies the following predicate (say $C$).

“\(\text{In-degree}(v)=\text{out-degree}(v)\) for all but at most two vertices, $v \in V$. One of which has in-degree one greater than its out-degree and the other has in-degree one less than its out-degree”.

If it has in-degree=out-degree for all the nodes then the path will be a cycle covering all the edges. If it has two nodes with unequal degrees then the path will be from the node with in-degree one less than its out-degree to the node with in-degree one greater than its out-degree.

Basis: $E$ is empty. That is the case when the number of edges, $n=0$. The above claim is true for this base case as there are no edges in the graph then surely there is an Euler path!

Induction Step: Assume the induction hypothesis that any connected directed graph $G(V, E)$ with number of edges less than or equal to $n$, which satisfies the predicate ‘$C$’ has an Euler path. The Euler path will be from the node with in-degree-1=out-degree to the node with in-degree=out-degree+1. If $G$ has in-degree=out-degree for all the nodes then the path will be a cycle.

Consider any connected directed graph $G_1(V_1, E_1)$ with $n+1$ edges that satisfies the predicate C. Here we have two cases. $G_1$ can have zero or two nodes with degrees unequal.

Case1: If $G_1$ has zero nodes with unequal degrees:

Here all the nodes in the graph $G_1$ has in-degree=out-degree. Now remove any edge $e$ ($v_e, v_s$) from the graph. The in-degree of $v_e$ becomes one greater than its out-degree and in-degree of $v_s$ becomes one less than its out-degree. The new graph will be connected as we have other edges connected to the nodes $v_e$ and $v_s$ in addition to the edge $e$. We have two nodes here after removal of edge $e$ with unequal degrees. The newly formed graph after the removal of edge $e$ has $n$ edges and it satisfies the predicate C. So $G_1$ without edge $e$ has an Euler path from $v_s$ to $v_e$ according to our assumption.

The graph $G_1(V_1, E_1-e)$ (after the removal of the edge $e$) has an Euler path from $v_s$ to $v_e$. Now consider the graph $G_1(V_1, E_1)$. All the edges excluding the edge $e$ form a path from $v_s$ to $v_e$. We can add the edge $e$ to the path at the end node and the path becomes a cycle as we are adding an edge from the end node to the start node of the path. This cycle covers all the edges including the edge $e$ and hence it is an Euler path (also Euler tour). Thus $G_1$ has an Euler path.

So, any graph $G$ with $n+1$ edges that satisfies the predicate $C$ (with all nodes having equal degrees) has an Euler path. This path is actually a cycle that covers all the edges.
Case 2: If \( G_1 \) has two nodes with unequal degrees:

Let \( v_s \) and \( v_e \) be the two nodes, which have their degrees unequal. Let \( v_e \) has its in-degree one greater than its out-degree and \( v_s \) has its out-degree one greater than its in-degree.

Now, remove an edge from the graph \( G_1 \) so that the graph still satisfies the predicate \( C \). We can remove an edge, \( e_s \) from \( v_s \) to some node, \( v_p \) in the graph. Now, in-degree of \( v_s \) becomes equal to its out-degree and in-degree of \( v_p \) becomes one less than its out-degree.

The Resulting graph may be connected or disconnected after the removal of the edge.

If the Resulting graph is connected:

If we remove \( e_s \), \( v_p \) and \( v_e \) are the only nodes, which have their degrees unequal. So there will be only two nodes with unequal degrees. Hence the new graph \( G_1 \) (\( V_1, E_1-\{e_s\} \)), will satisfy the predicate \( C \).

Now, the new graph \( G_1 \) is connected and has \( n \) edges and satisfies the predicate \( C \). So, according to our induction hypothesis we can say that the graph \( G_1 \) has an Euler path from \( v_p \) to \( v_e \) (after the removal of \( e_s \)).

The graph \( G_1 \) (\( V_1, E_1-e_s \)) has an Euler path from \( v_p \) to \( v_e \). Now, consider the graph \( G_1 \) (\( V_1, E_1 \)). We know that all the edges in \( G_1 \) excluding \( e_s \) form a path from \( v_p \) to \( v_e \). We can add the edge \( e_s \) to the starting node \( v_p \) of the path. Then the new path will be from \( v_s \) to \( v_e \). Since this path covers all the edges including \( e_s \) in the graph \( G_1 \), it is the Euler path of the graph \( G_1 \).

If the Resulting graph is disconnected:

The resulting graph will contain two connected components.

The nodes \( v_s \) and \( v_p \) will be in different components. Let the component containing \( v_s \) be \( G_2 \) and the component containing \( v_p \) be \( G_3 \). The two components will have number of edges less than \( n \).

The component \( G_2 \) will have in-degree=out-degree for all the nodes. According to our assumption we can say that the edges of the graph \( G_2 \) will form a path (actually a cycle) since it satisfies the predicate \( C \). The component \( G_3 \) will have two nodes (\( v_p \) and \( v_e \)) with degrees unequal and hence satisfies the predicate \( C \). Since the number of edges in \( G_3 \) is less than \( n \), based on our assumption we can say that \( G_3 \) has an Euler path and is from \( v_p \) to \( v_e \).

Now consider the graph \( G_1 \) (\( V_1, E_1 \)). We have an edge from any node in \( G_2 \) to a node in \( G_3 \). Since we have a cycle in \( G_2 \) covering all the edges, we can say that we have a path from \( v_s \) to \( v_e \) with all edges in \( G_2 \) covered. We have a path from \( v_p \) to \( v_e \) with all the edges in \( G_3 \) covered. In addition to the edges in both paths we have edge \( e_s \) from \( v_s \) to \( v_p \).
in the graph $G_1$. This edge connects both paths and result in long path with all the edges in graph $G_1$ covered. This path is nothing but the Euler path since it covers all the edges in the graph $G_1$. Thus $G_1$ has an Euler path.

So, any graph $G$ with $n+1$ edges that satisfies the predicate $C$ (with exactly two nodes having unequal degrees) has an Euler path.

This completes the induction step and completes the proof by mathematical induction.

**ClaimB:** A connected directed graph $G (V, E)$ has an Euler tour if $\text{in-degree}(v) = \text{out-degree}(v)$ for all nodes, $v$ in the graph.

Consider any graph, $G (V, E)$ that has $\text{in-degree}(v)=\text{out-degree}(v)$ for all the nodes $v$ in the graph.

Now remove any edge $e$ from the graph. Let the removed edge be $e_k$ from some node $v_e$ to some other node $v_s$. The in-degree of $v_e$ becomes one greater than its out-degree and the in-degree of $v_s$ becomes one less than its out-degree. Let the new graph be $G_1$. It is a connected graph since we have other edges connecting the nodes $v_s$ and $v_e$. The graph $G_1$ satisfies the predicate $C$ defined in the previous claim. So, according to claimA $G_1$ has an Euler path and is from the node $v_s$ to the node $v_e$.

Considering the graph $G$ again, we found that the edges in the graph $G$ excluding the edge $e_k$ form a path, $P$ from $v_s$ to $v_e$. Add this edge $e_k$ to the path $P$. Now it becomes a cycle and this cycle covers all the edges in the graph exactly once (as we know that the path $P$ covers all the edges except $e_k$ exactly once). This cycle is nothing but the Euler tour as it is a cycle and covers all the edges in the graph $G$.

Thus the claimB is TRUE.

**ClaimC:** If a connected directed graph $G (V, E)$ has an Euler tour then $\text{in-degree}(v)=\text{out-degree}(v)$ for all nodes $v$ in the graph $G$.

Any cycle must leave a node exactly as many times as it enters it. Since Euler tour covers all the edges exactly once and doesn’t leave any edge, $\text{in-degree}$ of a node is nothing but the number of times the Euler cycle enters the node and $\text{out-degree}$ of a node is nothing but the number of times the Euler cycle leaves the node.

Thus, $\text{in-degree}(v)=\text{out-degree}(v)$ for all $v \in V$

Thus, if a graph $G(V,E)$ has an Euler tour then $\text{in-degree}(v)=\text{out-degree}(v)$ for all the nodes, $v \in V$. 
**ClaimD:** A connected directed graph $G (V, E)$ has an Euler tour if and only if $\text{in-degree}(v) = \text{out-degree}(v)$ for all nodes $v$ in the graph.

By looking at the proofs for claimB and claimC, we can say that the above claim is $TRUE$.

In my algorithm, I first check whether the given graph contains an Euler tour or not using the claimC. Since the graph has an Euler tour we have in-degree=out-degree for all the nodes in the graph.

I start with an arbitrary node and continue adding the edges to the linked list. Since in-degree=out-degree for all the nodes whenever we enter a node there will be an edge leaving that node and hence we can reach another node. Thus addition of edges is stopped only when we reach the node at which we started. This set of edges traversed form a cycle. We start with another node whose all edges are not covered yet. Similar to the procedure above we will get another cycle. So, we can construct a set of cycles with disjoint sets of edges each according to my approach.

Since the graph $G$ is connected we have one of the following.

1. There are few edges, which are not in any of the cycles and each edge connects one cycle with other.

2. There will be at least a node in each cycle, which also exists at least once in other cycle.

As there are no edges left after constructing the cycles, there will be at least a node in one cycle, which also exists at least once in another cycle. We can merge the two cycles using the common node in the two cycles. In this way we can merge all the cycles and get an Euler Tour, which covers all the edges exactly once.

In my algorithm, I start constructing the second cycle from a node within the already formed cycle whose all edges are not yet covered. This ends being a cycle inserted within the existing cycle. This insertion of a new cycle within the existing cycle ends when there are no edges left to cover. As we are inserting cycles within the existing cycle at the common node the final list will be a cycle covering all the edges of the node.

Thus my algorithm constructs an Euler Tour of the given graph if one exists.
TIME COMPLEXITY

**CHECK( )**: The time complexity of the procedure CHECK( ) is $O(n(E))$.

The inner while loop in the CHECK is executed exactly equal to the number of edges in the graph and hence it takes time in $O(n(E))$.

The for loop in the CHECK is executed exactly equal to the number of vertices in the graph and hence it takes time in $O(n(V))$.

Hence, Time complexity of CHECK( ) $\in \Theta(n(E)+n(V))$

$\in \Theta(n(E))$.

**EULER( )**: Time complexity of the procedure EULER( ) is $O(n(E))$.

*Consider the first inner while loop:*

The body of the first inner while loop is executed at most $n(E)$ times because in each iteration we remove an edge after inserting it into the cycle and there are $n(E)$ edges in the graph. So, the time taken by the first inner while loop is in $O(n(E))$.

*Consider the second inner while loop:*

Similarly, the body of the second inner while loop is also executed at most $n(E)$ times because either we consider an edge or an adjacency list. It is executed at least $n(V)$ times and at most $n(E)$ times. So, the time taken by the second inner while loop is in $O(n(E) + n(V))$ which is equal to $O(n(E))$.

Time Complexity of the procedure EULER( ) $\in O(n(E)+n(V))$.

$\in O(n(E))$.

**MAIN( )**:

Time Complexity of MAIN( ) $\in O(n(E)+n(E))$.

$\in O(n(E))$.