Finding Euler path for O (M)

Euler path - A path in the graph, passing through all its edges. Euler tour - is the Euler path is a cycle.

The challenge is to find an Euler path in **an undirected multigraph with loops** .

Algorithm

Please check whether there is an Euler path. Then we find all simple cycles and combine them into one - it will be an Euler tour. If the graph is that the Euler path is not a cycle, then add the missing edge, find an Euler tour, then remove the extra edge.

To test whether there is an Euler path, you need to use the following theorem. Euler cycle exists if and only if the degrees of all vertices are even. Eulerian path exists if and only if the number of vertices equals the odd powers of two (or zero if there is Euler cycle).

Also, of course, the graph must be sufficiently connected (ie, if we remove from it all isolated vertices, then should get a connected graph).

Search all the loops and will combine them a recursive procedure:

```
procedure FindEulerPath (V)
        1. iterate over all edges emanating from the vertex V;
                each such edge is removed from the graph, and
                FindEulerPath call from the second end of the rib;
        2. add a vertex V in response.
```

The complexity of this algorithm is obviously linear in the number of edges.

But the same algorithm can be written in **a non-recursive** version:

```
stack St;
in St put any vertex (top of page);
while St is not empty
        Let V - value on top of St;
        if the power (V) = 0, then
                add V to the answer;
                remove V from the top of St;
        otherwise
                find any edge going from V;
                remove it from the graph;
                the second end of the rib put in St;
```

It is easy to verify the equivalence of these two forms of the algorithm. However, the second shape is obviously faster running, the code will not be greater.

The problem of the domino

We present here a classic problem in the Euler cycle - the problem of dominoes.

Means N dominoes are known at both ends of dominoes recorded one number (typically 1 to 6, but in this case is not important). You want to publish all the dominoes in a row so that any two adjacent dominoes numbers written on their common side match. Dominoes are allowed to turn.

Reformulate the problem. Let the number recorded on donimoshkah - top graph, and the dominoes - edge of the graph (each domino with numbers (a, b) - this edge (a, b), and (b, a)). Then our problem **reduces to** the problem of finding **an Euler path** in this graph.

Implementation

The following program searches for and displays Euler tour or path in the graph, or displays -1 if it does not exist.

First, the program checks the degree of vertices if the vertices with odd degree is not, then the graph has an Euler tour if there are two vertices with odd degree, then the graph has an Euler path only (no Euler cycle), if such heights greater than 2, then graph no Euler cycle or Euler path. To find an Euler path (not a cycle), proceed as follows: if V1 and V2 - this two vertices of odd degree, then just add an edge (V1, V2), in the resulting graph will find an Euler tour (he obviously will exist), and then remove from the response of "fictitious" edge (V1, V2). Euler tour will look exactly as described above (non-recursive version), and at the same time at the end of this algorithm check was connected graph or not (if the graph is not connected, then at the end of the algorithm in the graph remain some ribs, and in this case we need to print -1). Finally, the program takes into account that in the graph can be isolated vertices.

```
int main () {

    int n;
    vector <vector <int>> g (n, vector <int> (n));
    ... read the graph in the adjacency matrix ...

    vector <int> deg (n);
    for (int i = 0; i <n; + + i)
```

```
        for (int j = 0; j <n; + + j)
                deg [i] + = g [i] [j];

int first = 0;
while (! deg [first]) + + first;

int v1 = -1, v2 = -1;
bool bad = false;
for (int i = 0; i <n; + + i)
        if (deg [i] & 1)
                if (v1 == -1)
                        v1 = i;
                else if (v2 == -1)
                        v2 = i;
                else
                        bad = true;

if (v1! = -1)
        + + G [v1] [v2], + + g [v2] [v1];

stack <int> st;
st.push (first);
vector <int> res;
while (! st.empty ())
{
        int v = st.top ();
        int i;
        for (i = 0; i <n; + + i)
                if (g [v] [i])
                        break;
        if (i == n)
        {
                res.push_back (v);
                st.pop ();
        }
        else
        {
                - G [v] [i];
                - G [i] [v];
                st.push (i);
```

```
            }
        }

        if (v1! = -1)
            for (size_t i = 0; i +1 <res.size (); + + i)
                if (res [i] == v1 && res [i +1] == v2 | | res [i] == v2 && res [i +1] ==
v1)
                {
                    vector <int> res2;
                    for (size_t j = i +1; j <res.size (); + + j)
                        res2.push_back (res [j]);
                    for (size_t j = 1; j <= i; + + j)
                        res2.push_back (res [j]);
                    res = res2;
                    break;
                }

        for (int i = 0; i <n; + + i)
            for (int j = 0; j <n; + + j)
                if (g [i] [j])
                    bad = true;

        if (bad)
            puts ("-1");
        else
            for (size_t i = 0; i <res.size (); + + i)
                printf ("% d", res [i] +1);

}
```