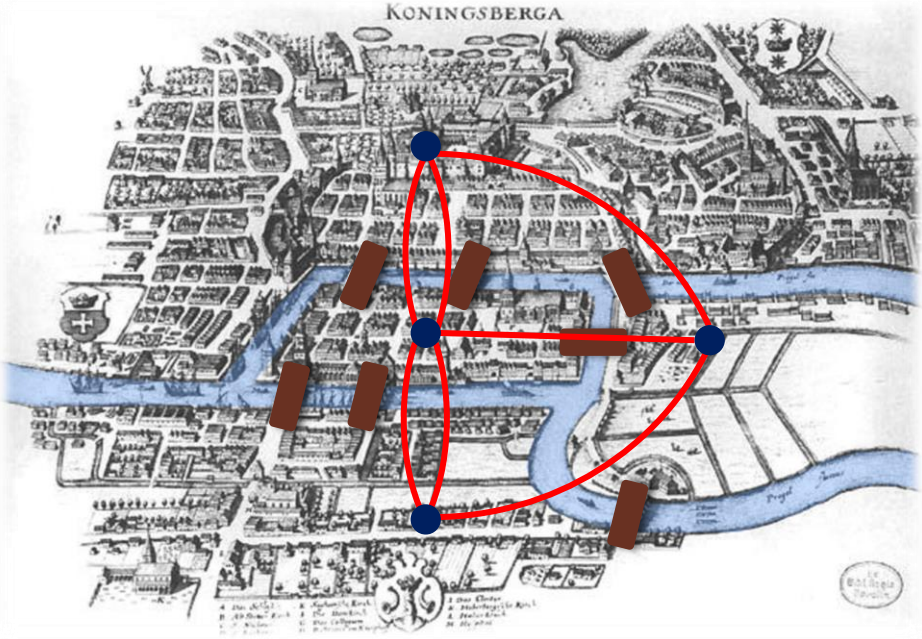
TAKING THE EULER TOUR



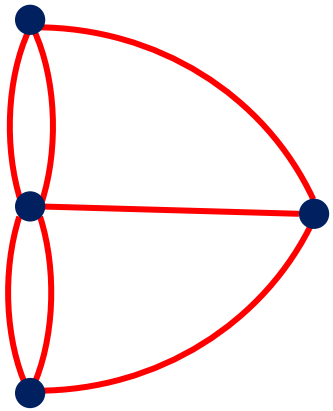
Königsberg
1736

Leonhard Euler
1707 - 1783

Is there a walk in the graph that visits every edge exactly once?

Euler Trail: A walk in a finite graph which visits **every edge** exactly **once**.

Euler Cycle: An Euler trail which starts and ends on the same vertex.

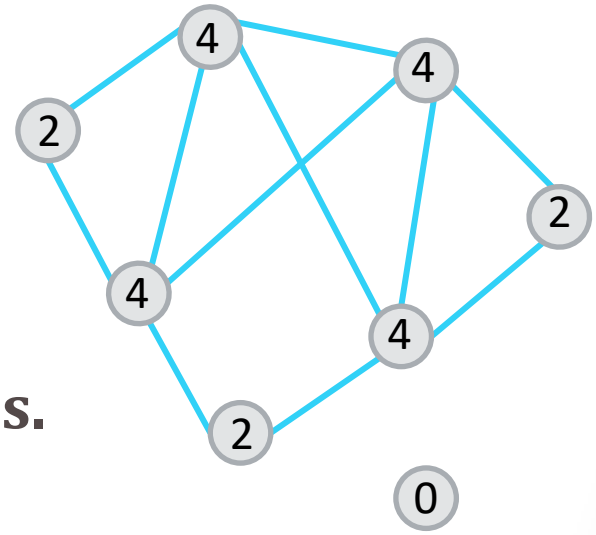


Given a graph G,

Does G has an Euler Cycle?

If there is an Euler Cycle then:

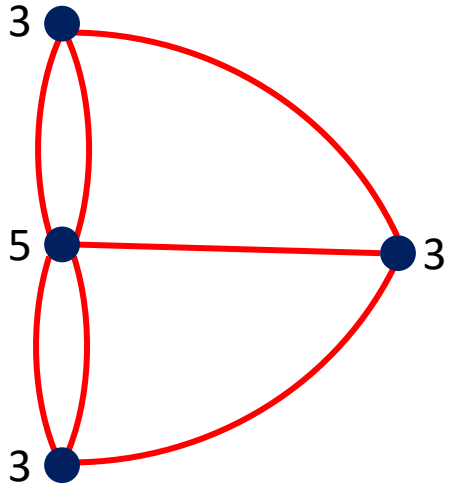
1. **G is connected*.**
2. **All vertices have even degrees.**



*all vertices with degree>0 are connected

Given a graph G,

Does G has an Euler Cycle?



No Euler Cycle in Königsberg!

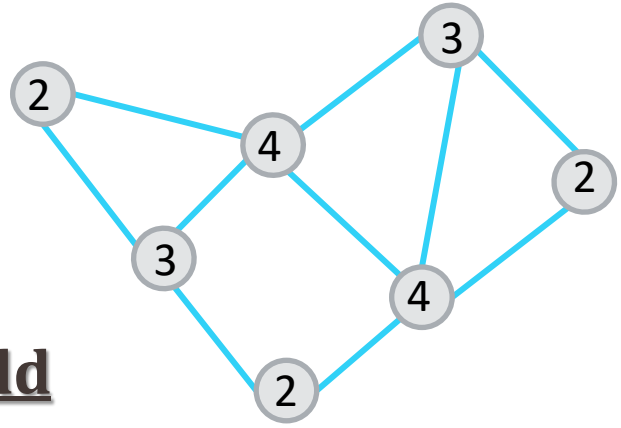
Maybe an Euler Trail?...

Given a graph G,

Does G has an Euler Trail?

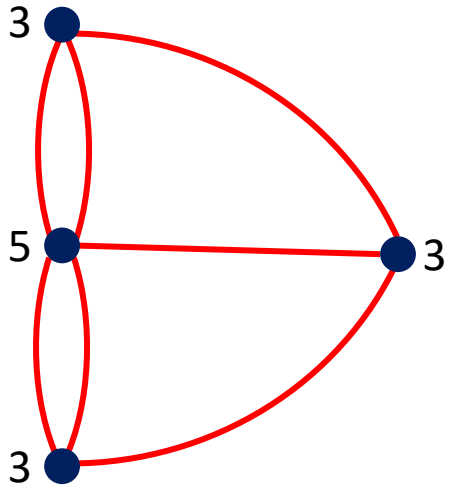
If there is an Euler Trail then:

1. **G is connected*.**
2. **At most two vertices have odd degrees.**

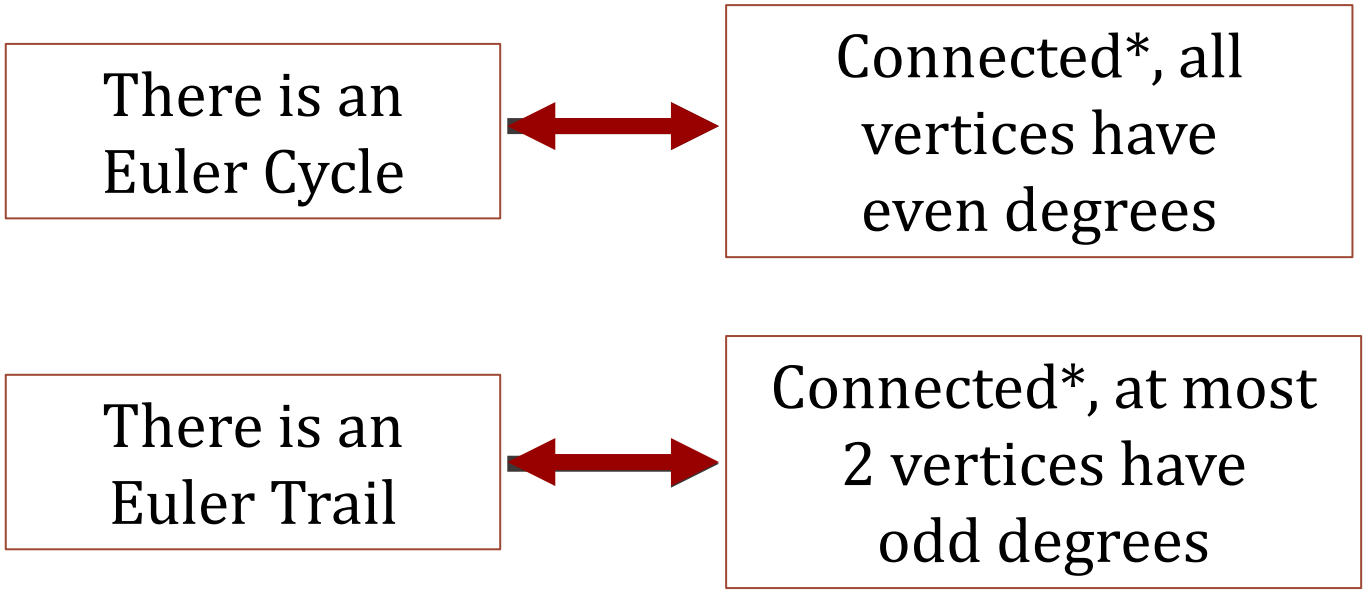


Given a graph G,

Does G has an Euler Trail?



No Euler Trail in Königsberg!


There is an Euler Cycle
Connected*, all vertices have even degrees
There is an Euler Trail
Connected*, at most 2 vertices have odd degrees

Summary:

Given a graph G,

- There is an Euler cycle in G if and only if G is connected and all vertices have even degrees.
- There is an Euler trail in G if and only if G is connected and at most 2 vertices have odd degrees.

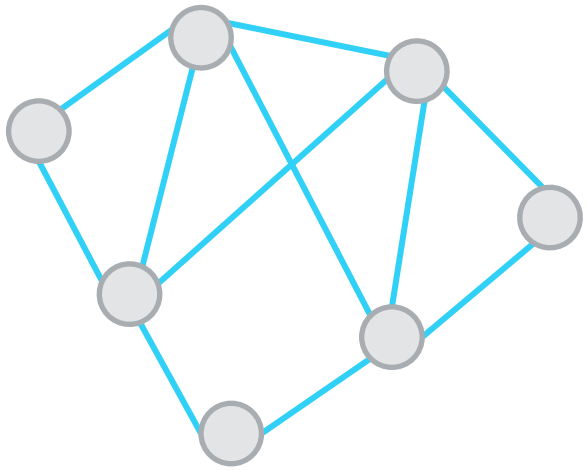
So, it's very easy to check if G has an Euler cycle or trail!

How can we find an Euler cycle?

Hierholzer ‘s Algorithm*(1873)

Preperations:

- Check if G is connected
- Check if all vertices have even degrees
- Mark all edges “unvisited”
- start at any vertex



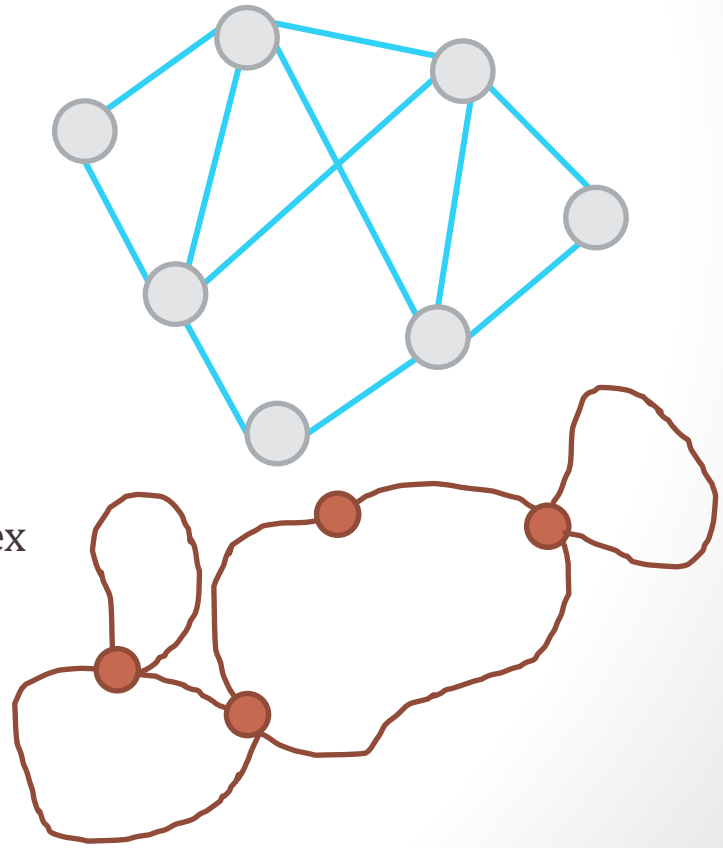
How can we find an Euler cycle?

Hierholzer 's Algorithm*(1873)

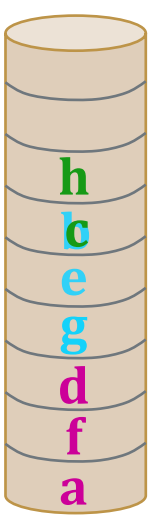
Main loop:

- **if** current vertex has an **unvisited** edge:
  use it to get to the next vertex
  and mark the edge "**visited**"

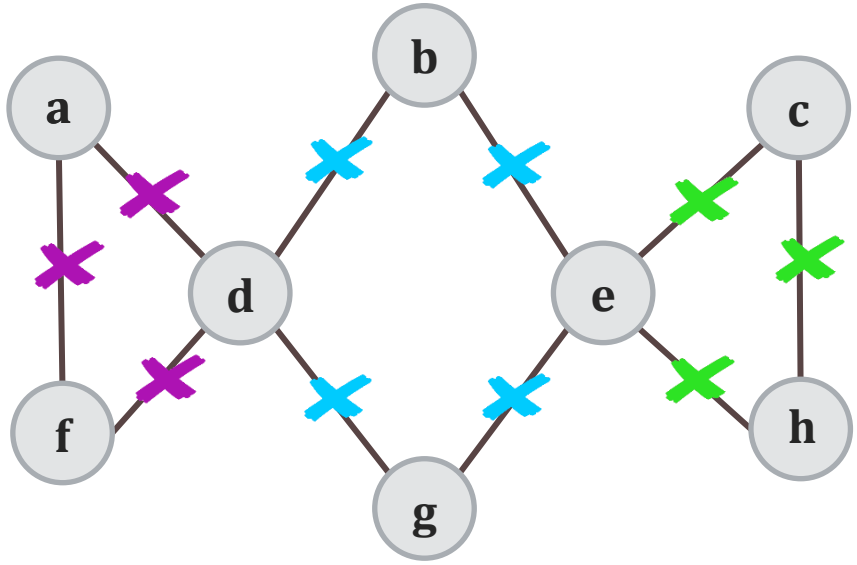
- **else** (a cycle is closed but maybe some edged are still unvisited)
  backtrack to a previously visited vertex that still has an unvisited edge.



Implementation (cycle)



Stack



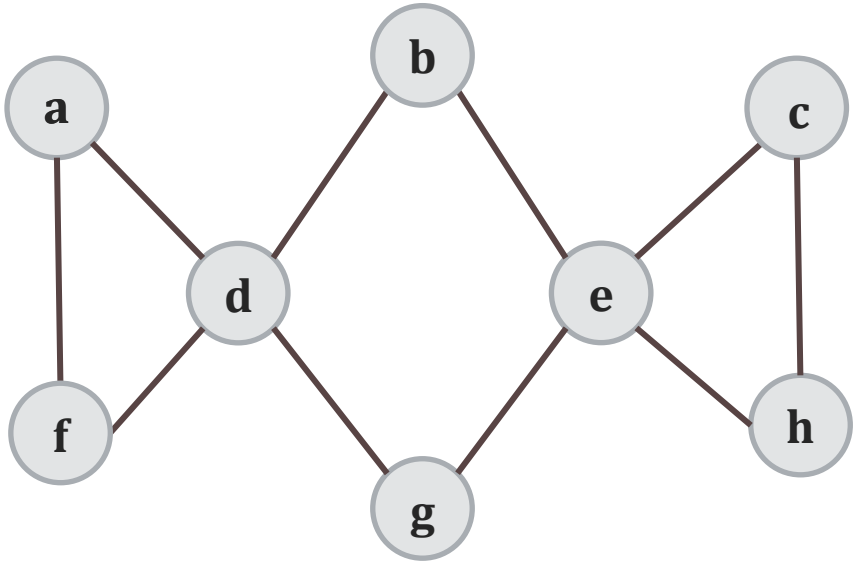
Output: **a d b e h c e g d f a**

```
Stack S;

curr = a;

loop:
   if there is unvisited edge e(curr, v):
        push(curr, S);
        mark e "visited";
        curr=v;
    else:
        output (curr);
        if S is not empty:
              curr = pop(S);
        else:  break;
```

Implementation (cycle)



a f d a

d g e b d

e c h e

```
Stack S;

curr = a;

loop:
   if there is unvisited edge e(curr, v):
        push(curr, S);
        mark e "visited";
        curr=v;
    else:
        output (curr);
        if S is not empty:
              curr = pop(S);
        else:  break;
```

Running time:

$O(|V|+|E|)$

What about directed graphs?

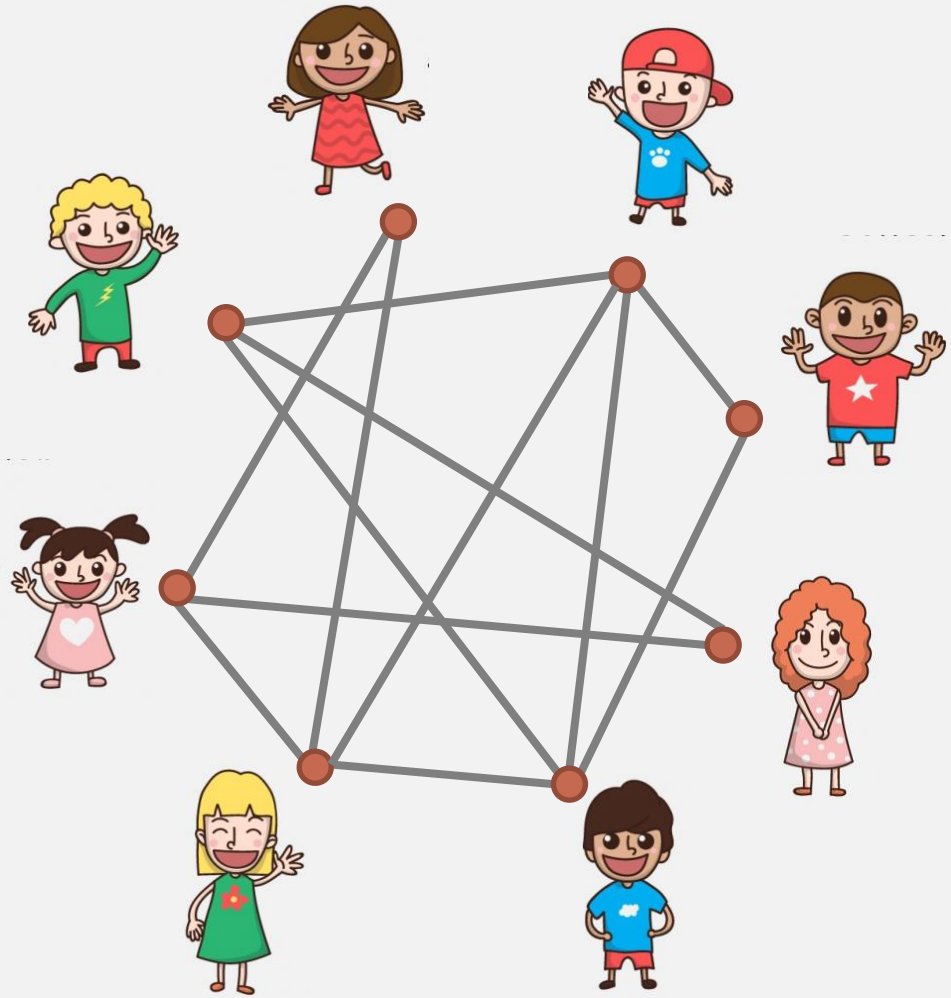
Graph G has an Euler cycle if and only if:

1. G is strongly connected
2. For every vertex, the in-degree and out-degree is equal.

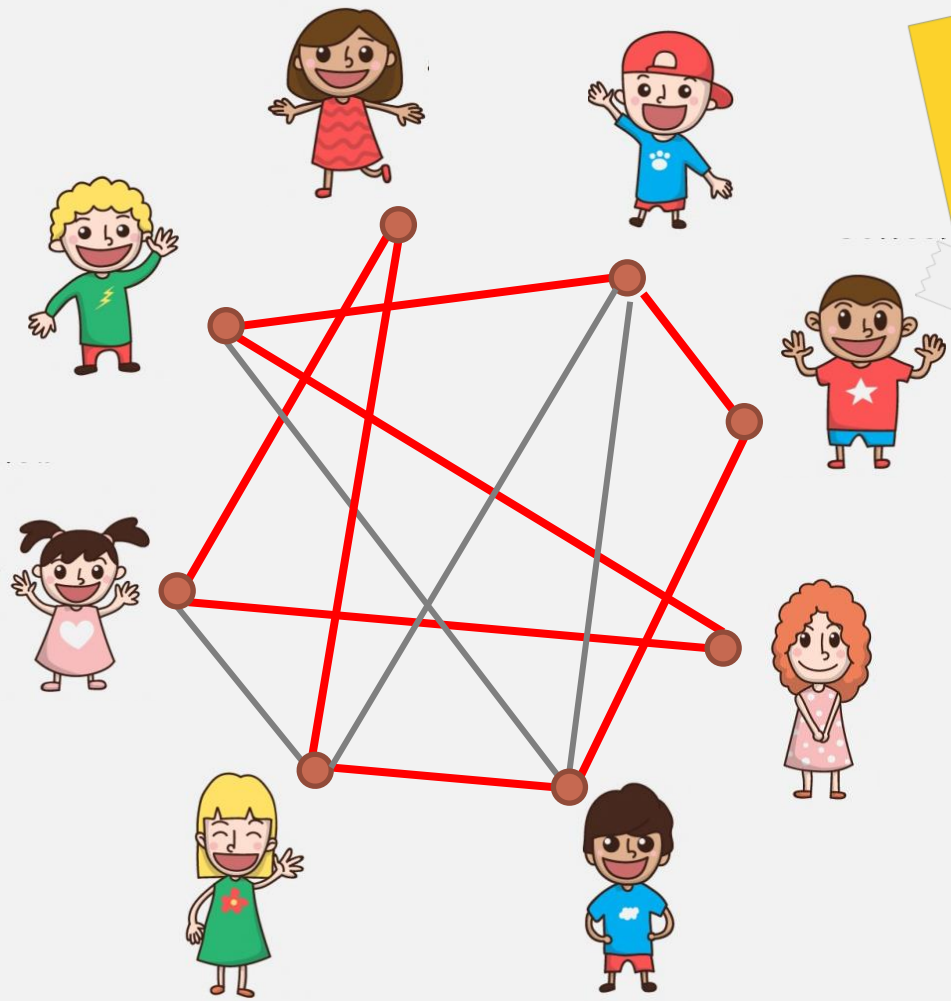
Algorithm is the same,

don't forget to reverse the output :)

Example:



Example: Hamiltonian cycle/path



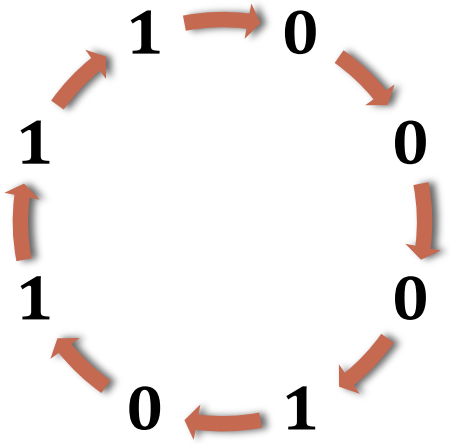
NP-COMPLETE

Example:

Task: Build a circle of 2^3 bits in which every 3-bits sequence occurs exactly once.

3 bits sequences:

000
001
010
011
100
101
110
111



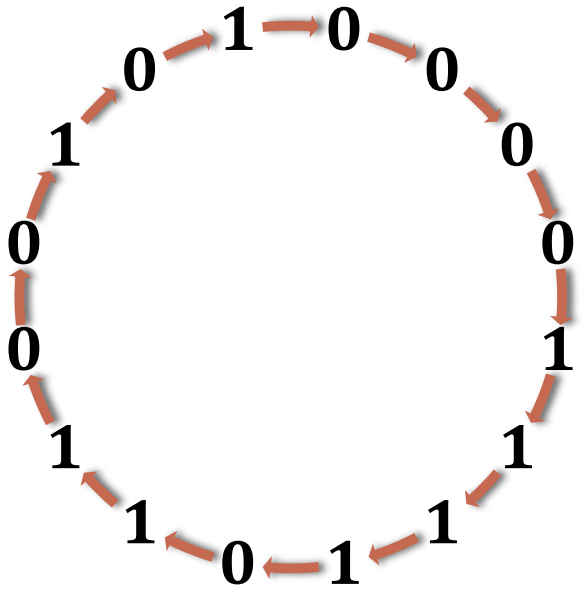
circle of 2^3 bits

Example:

Task: Build a circle of 2^4 bits in which every 4-bits sequence occurs exactly once.

4 bits sequences:

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111



circle of 2^4 bits

Example:

Task: Build a circle of 2^K bits in which every K-bits sequence occurs exactly once.

Hints:

- Euler cycle (of course…)
- The graph has 2^{K-1} vertices

THANK YOU ☺