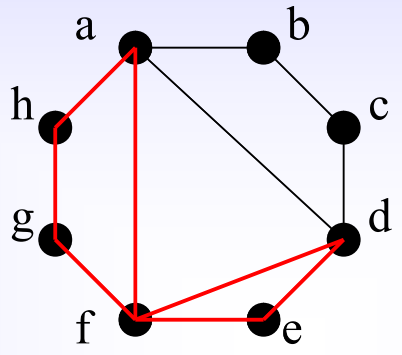
Cycle:

A sequence of consecutively linked edges whose starting vertex is the ending vertex, in which no edge can appear more than once. However, vertices can be repeated.

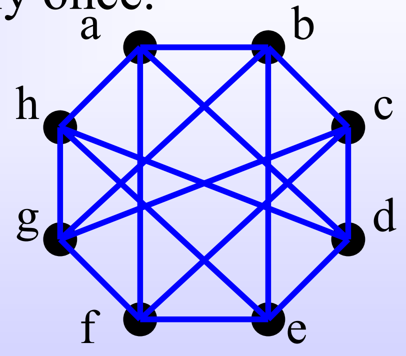
e.g.



C=a-h-g-f-e-d-f-a

Euler Cycle:

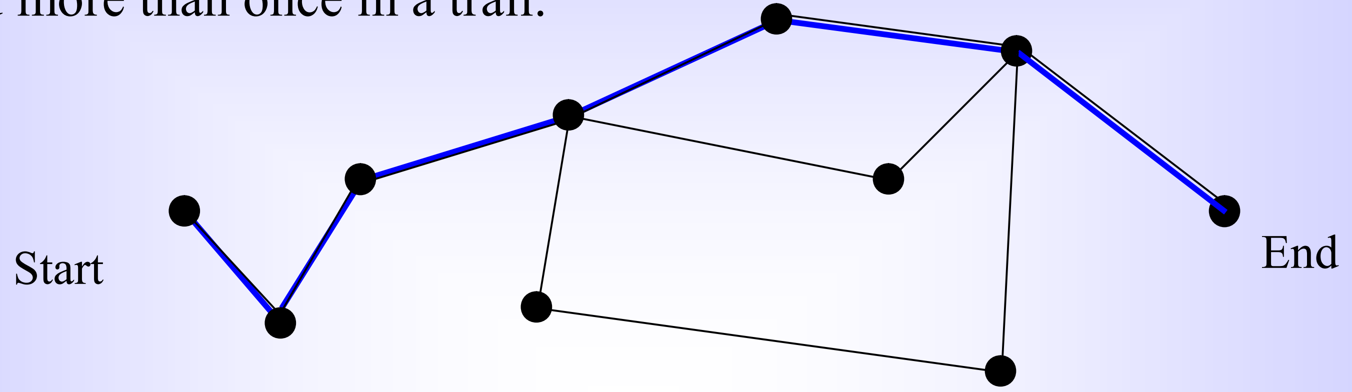
A path through a graph which starts and ends at the same vertex and includes every edge exactly once.



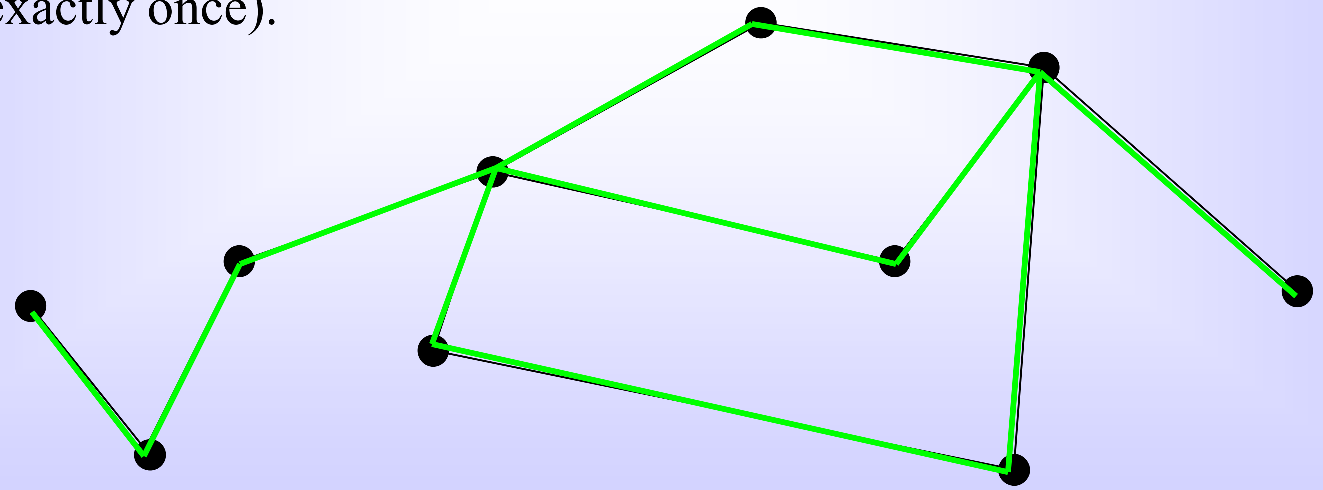
E = a-b-c-d-e-f-g-h-a-d-h-e-b-g-c- f-a

Trail: A sequence of consecutively linked edges in which no edge can appear more than once. $T = x_1 - x_2 - \ldots - x_n$. Unlike a path, a vertex can be visited more than once in a trail.

e.g.

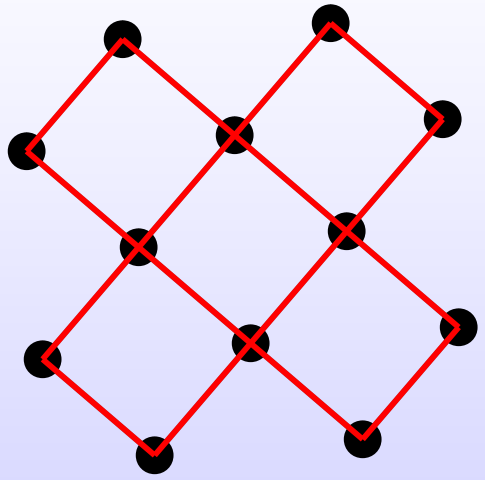


***Euler Trail*:-** A trail that contains all the edges in a graph (and visits each edge exactly once).



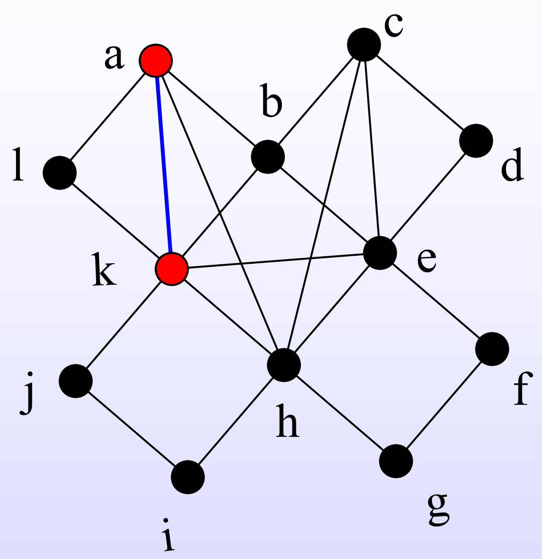
Theorem

- An undirected multigraph has an Euler cycle if and only if it is connected and has all vertices of even degree.



➢ This is a connected graph with all vertices of even degree, therefore it must have an Euler cycle.

Example



Multigraph H

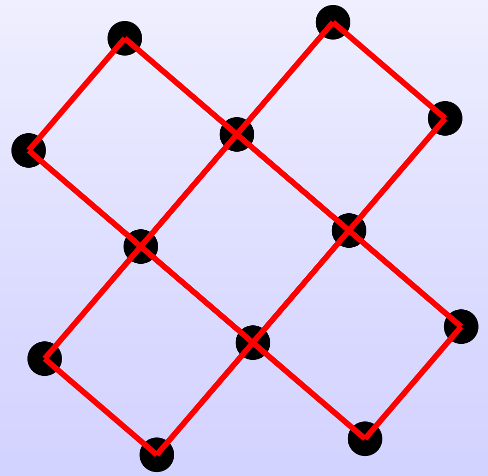
- Here the multigraph *H* does not have an Euler cycle because there are 2 odd degree vertices, namely vertex ‘k’ and vertex ‘a’.

- Therefore if and only if the edge (k , a) were completed then *H* would have an Euler Cycle.

Proof

An Euler cycle $\Longrightarrow$ all vertices are connected and have even degree

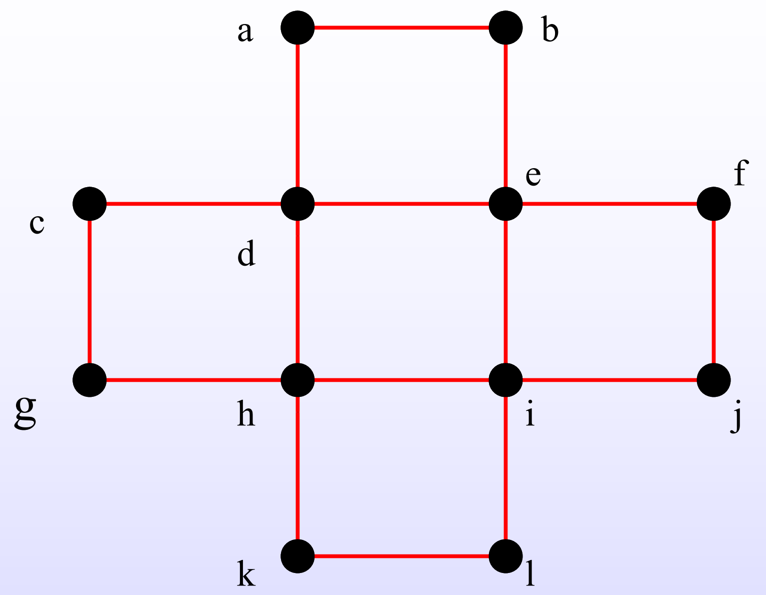
The Euler cycle connects all vertices, so the graph is connected. Every time the cycle enters a vertex it also exits, so the vertices have even degree in a whole complete cycle.



- This graph has an Euler cycle, therefore all the vertices are connected and have even degree.
- We call a connected graph with all even degrees an ***Eulerian Graph.***

Proof

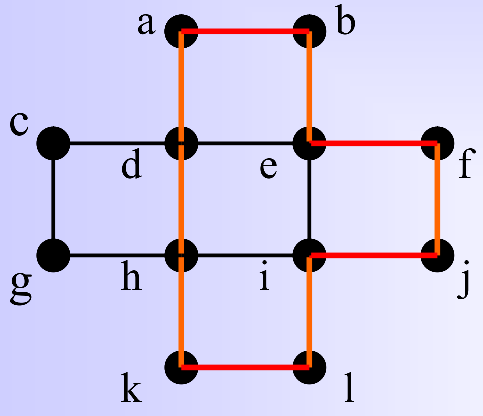
All even and connected $\Longrightarrow$ Euler cycle



- Pick any point on *G*. (I will start with a)
- Since all vertices have an even degree, we are not forced to stop until we complete the cycle. (get back to the starting point)

Proof cont'd

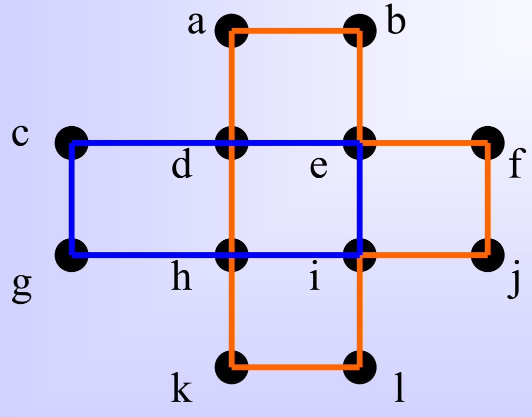
Path created by *C*



For example, let *C* equal the cycle created,

C = a-d-h-k-l-i-j-f-e-b-a

If there are any unused edges, and in this case there is, then create another cycle starting with any edge adjacent to *C*.



This is the cycle created by *D* in Blue

D = d-e-i-h-g-c-d

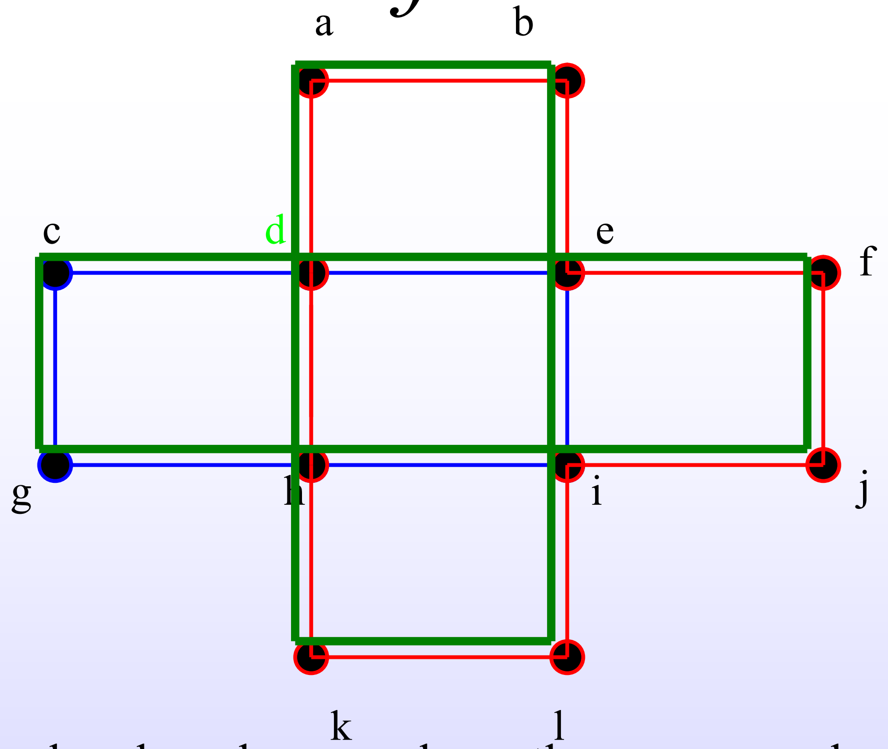
Proof cont'd

- Since *C* and *D* were originally connected, there must be at least one common vertex; pick one (point d)
- Build a new cycle *C*′, by putting *D* into *C* at the common point to get a larger cycle.

$$C' = D + C$$

C′ is presented on the next slide

Proof Cont'd

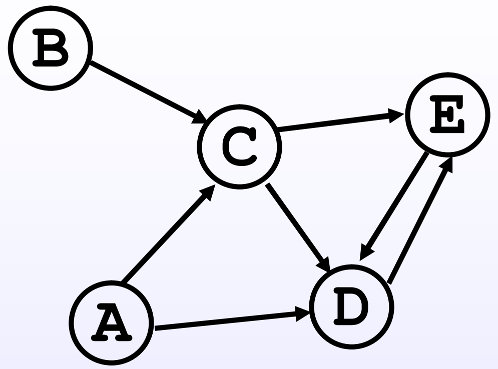


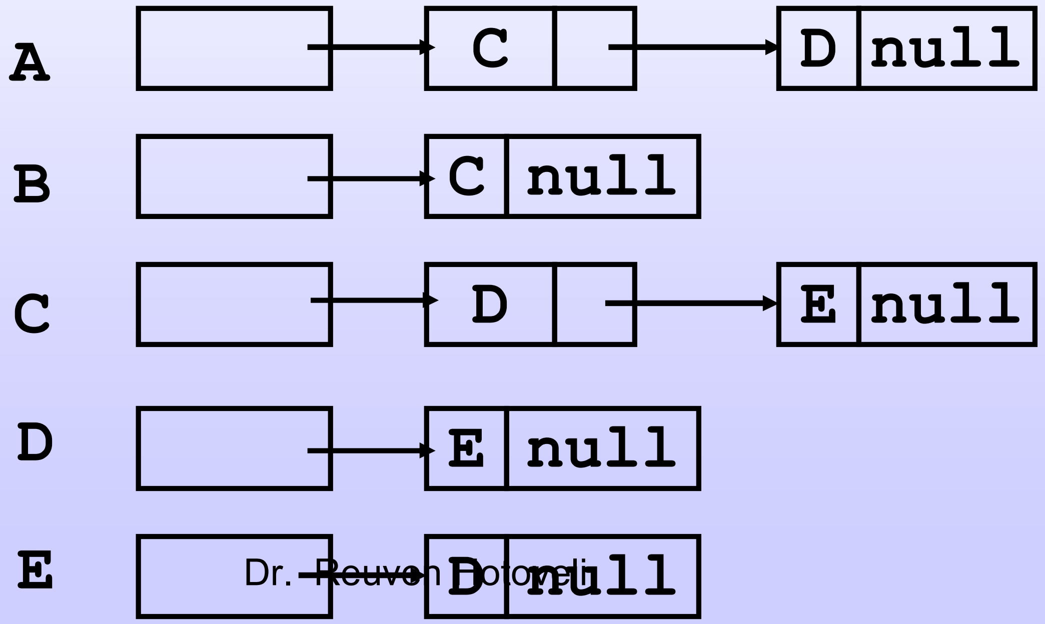
- All the edges have been used exactly once, so we have an Euler cycle.
- If all the edges were not used, we would simply pick one adjacent to C´ and repeat the process.

Graph representation (1)

- adjacency list:

  The nodes adjacent to one node are maintained by a linked list.

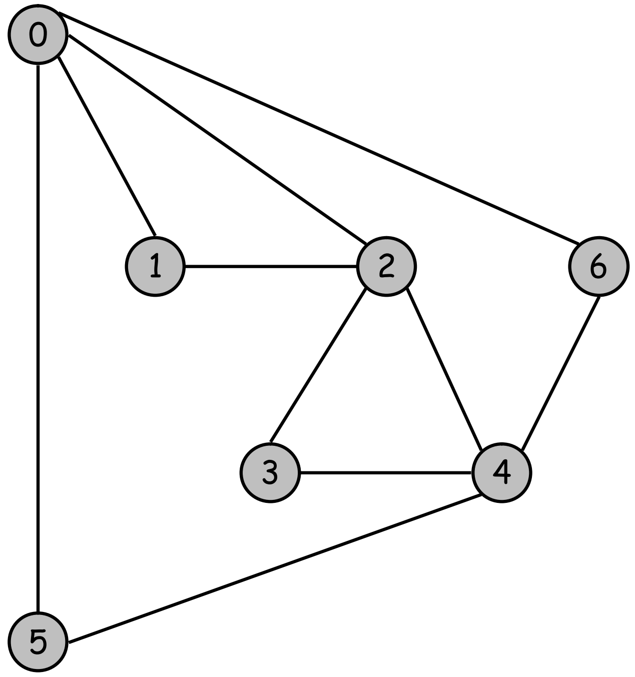




Euler Tour: Example

Adjacency list

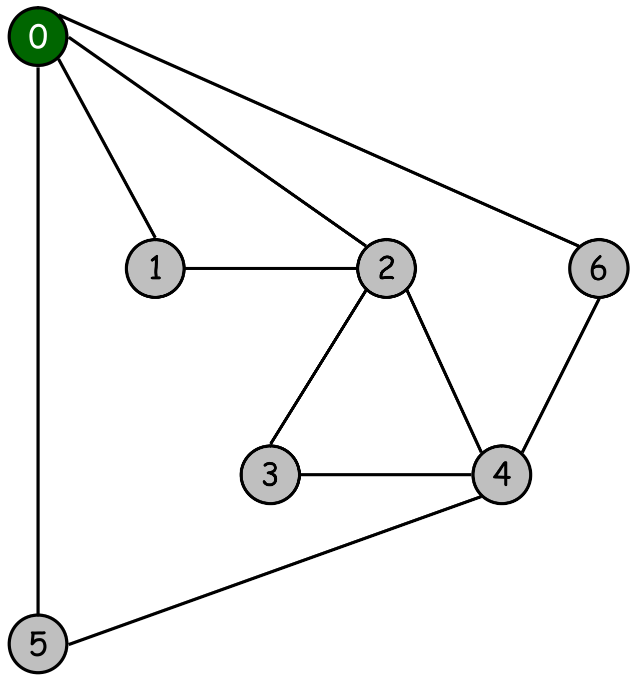
0:	1 2 5 6
1:	0 2
2:	0 3 4 1
3:	4 2
4:	6 5 3 2
5:	4 0
6:	4 0

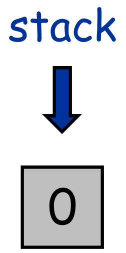


Euler Tour: Example

Adjacency list

0:	1 2 5 6
1:	0 2
2:	0 3 4 1
3:	4 2
4:	6 5 3 2
5:	4 0
6:	4 0



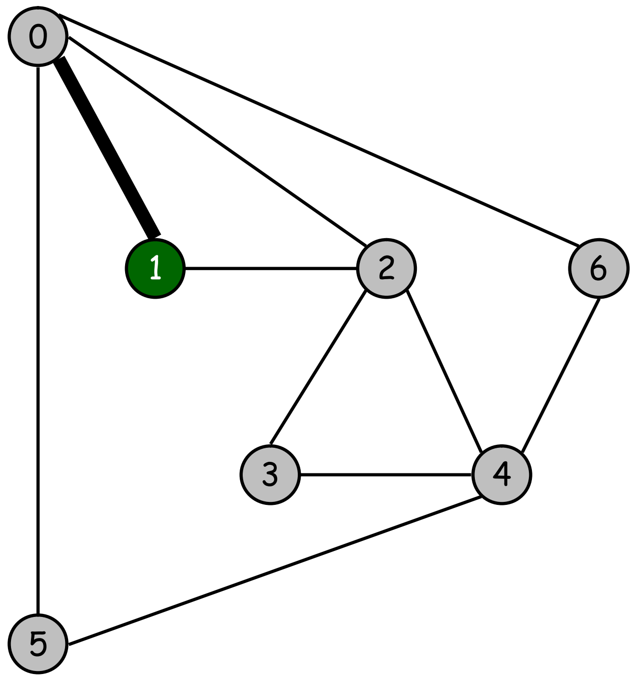


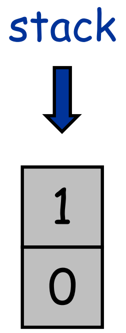
stdout

Euler Tour: Example

Adjacency list

0: ~~1~~ 2 5 6
1: ~~0~~ 2
2: 0 3 4 1
3: 4 2
4: 6 5 3 2
5: 4 0
6: 4 0



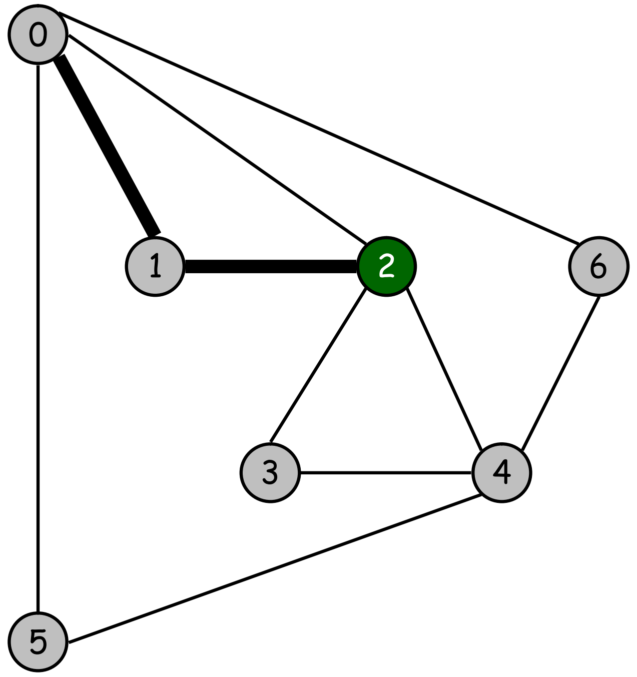


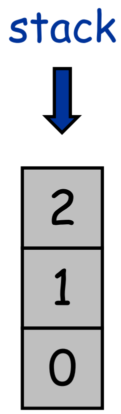
stdout

Euler Tour: Example

Adjacency list

0: ~~1~~ 2 5 6
1: ~~0~~ ~~2~~
2: 0 3 4 ~~1~~
3: 4 2
4: 6 5 3 2
5: 4 0
6: 4 0



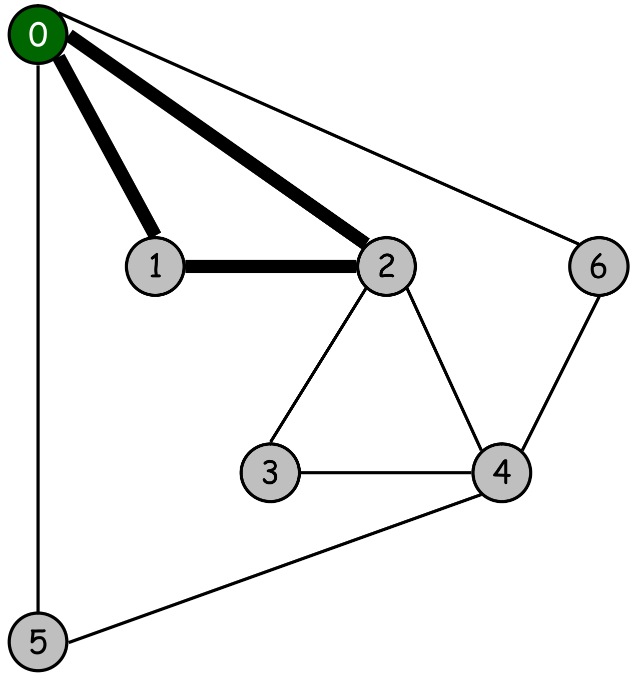


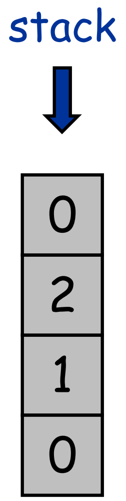
stdout

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ 5 6
1: ~~0~~ ~~2~~
2: ~~0~~ 3 4 ~~1~~
3: 4 2
4: 6 5 3 2
5: 4 0
6: 4 0



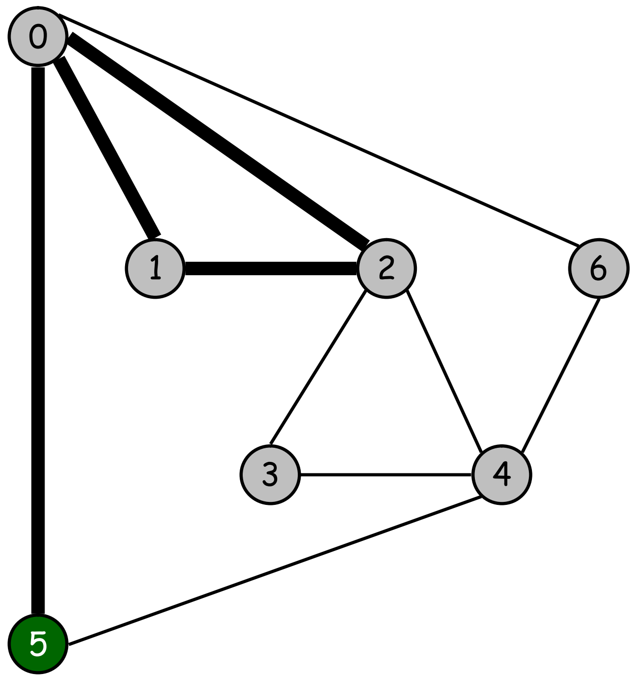


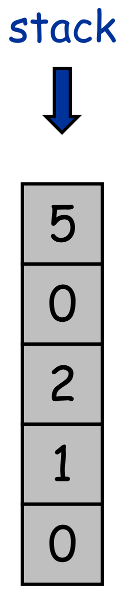
stdout

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ 6
1: ~~0~~ ~~2~~
2: ~~0~~ 3 4 ~~1~~
3: 4 2
4: 6 5 3 2
5: 4 ~~0~~
6: 4 0



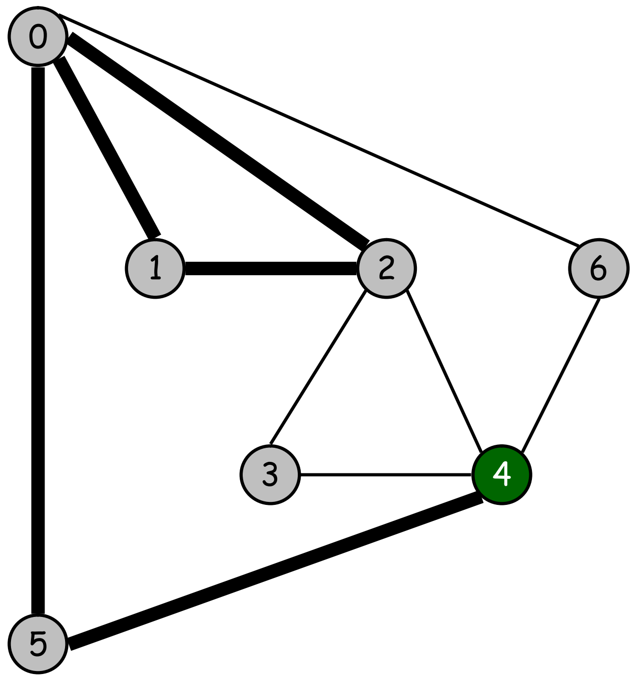


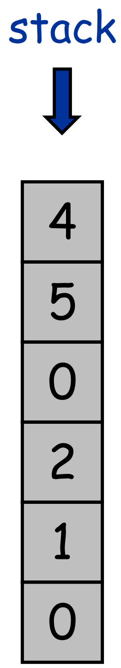
stdout

Euler Tour: Example

Adjacency list

0:	~~1~~	~~2~~	~~5~~	6
1:	~~0~~	~~2~~		
2:	~~0~~	3	4	~~1~~
3:	4	2		
4:	6	~~5~~	3	2
5:	~~4~~	~~0~~		
6:	4	0		



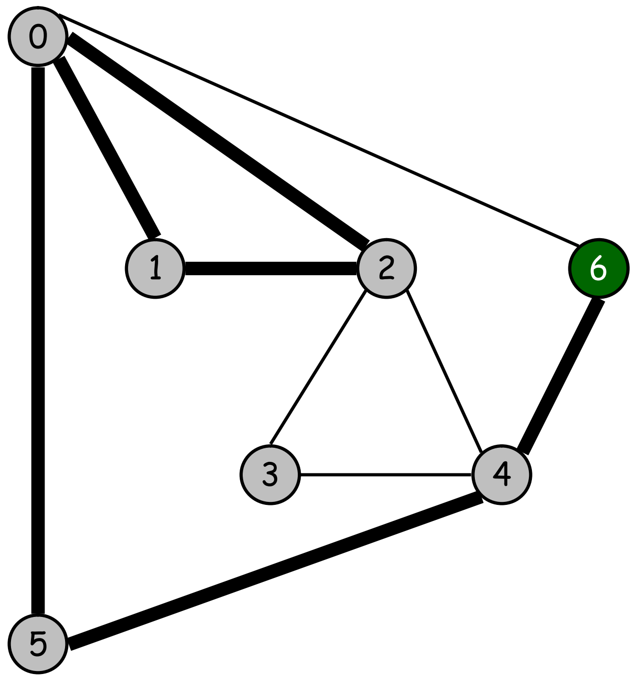


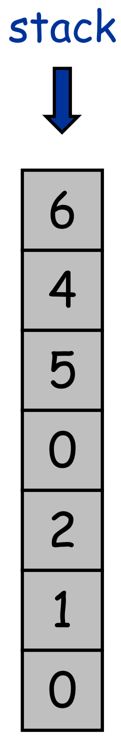
stdout

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ 6
1: ~~0~~ ~~2~~
2: ~~0~~ 3 4 ~~1~~
3: 4 2
4: ~~6~~ ~~5~~ 3 2
5: ~~4~~ ~~0~~
6: ~~4~~ 0



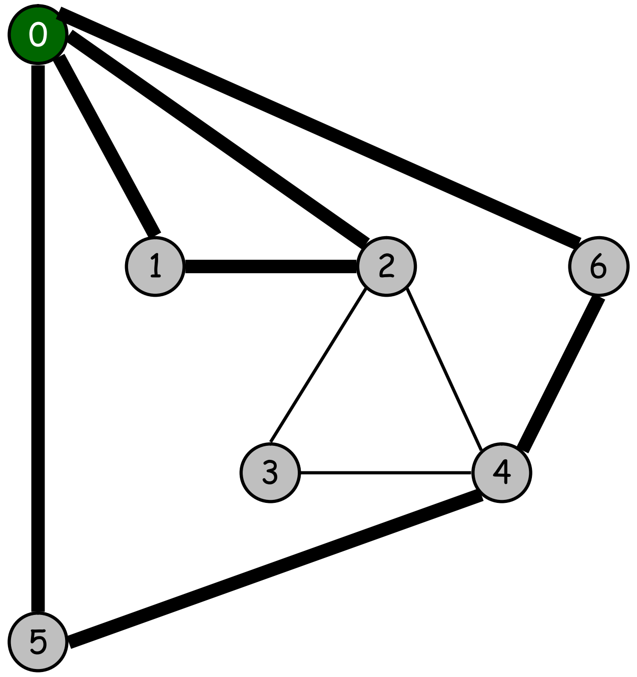


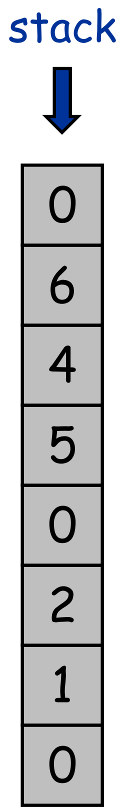
stdout

Euler Tour: Example

Adjacency list

```
0:  ~~1~~  ~~2~~  ~~5~~  ~~6~~
1:  ~~0~~  ~~2~~
2:  ~~0~~  3  4  ~~1~~
3:  4  2
4:  ~~6~~  ~~5~~  3  2
5:  ~~4~~  ~~0~~
6:  ~~4~~  ~~0~~
```



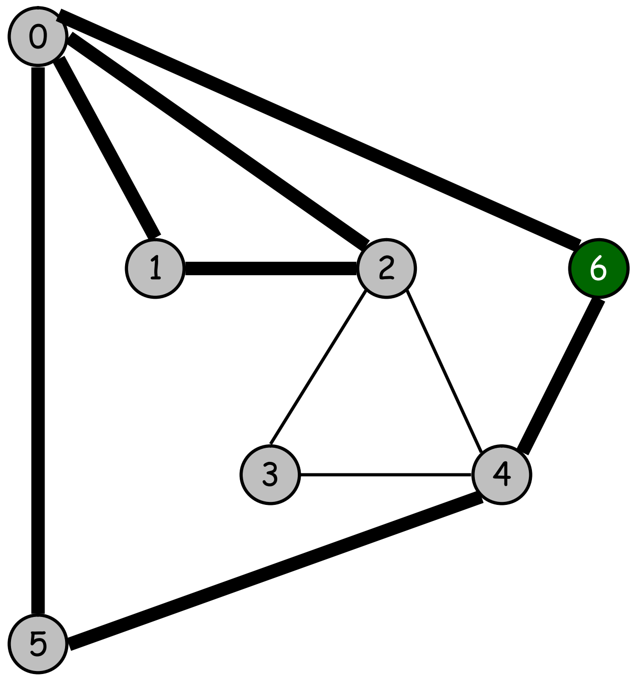


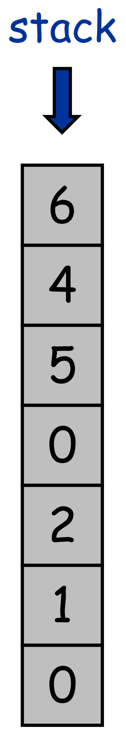
stdout

Euler Tour: Example

Adjacency list

```
0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ 3 4 ~~1~~
3: 4 2
4: ~~6~~ ~~5~~ 3 2
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~
```





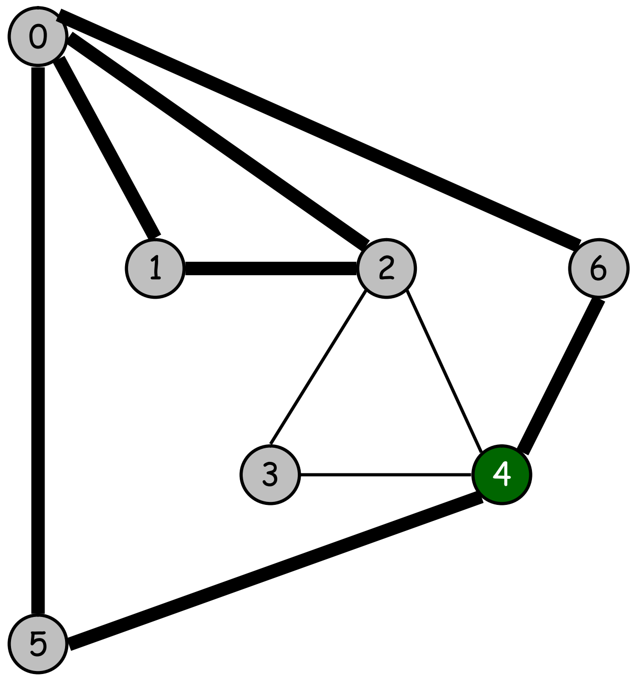
stdout

```
0
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ 3 4 ~~1~~
3: 4 2
4: ~~6~~ ~~5~~ 3 2
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~



stack

4
5
0
2
1
0

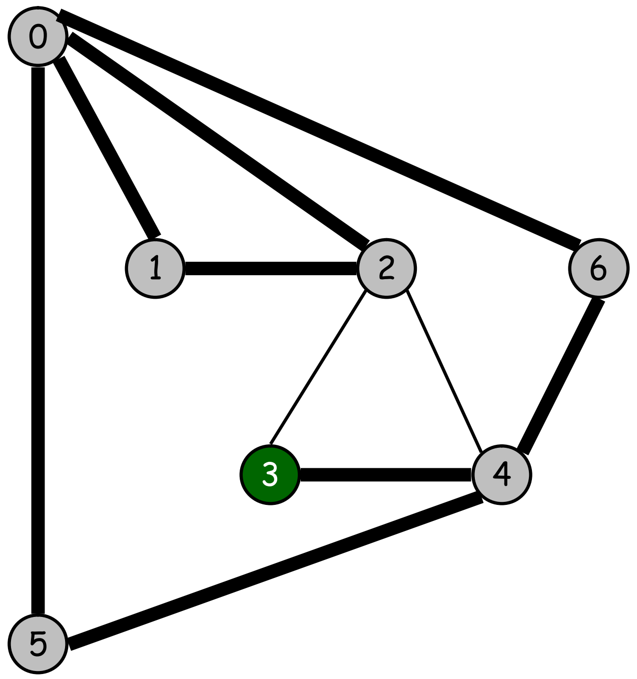
stdout

```
0 6
```

Euler Tour: Example

Adjacency list

```
0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ 3 4 ~~1~~
3: ~~4~~ 2
4: ~~6~~ ~~5~~ ~~3~~ 2
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~
```



stack

3
4
5
0
2
1
0

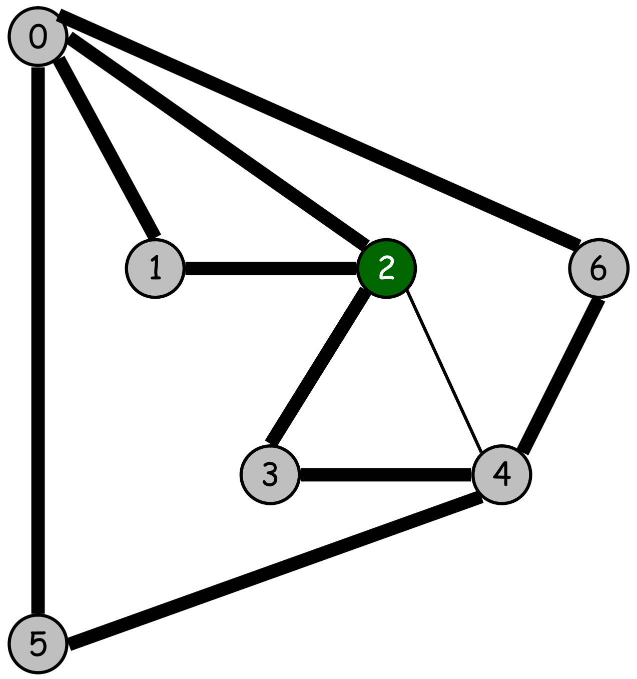
stdout

```
0 6
```

Euler Tour: Example

Adjacency list

```
0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ 4 ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ 2
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~
```



stack

2
3
4
5
0
2
1
0

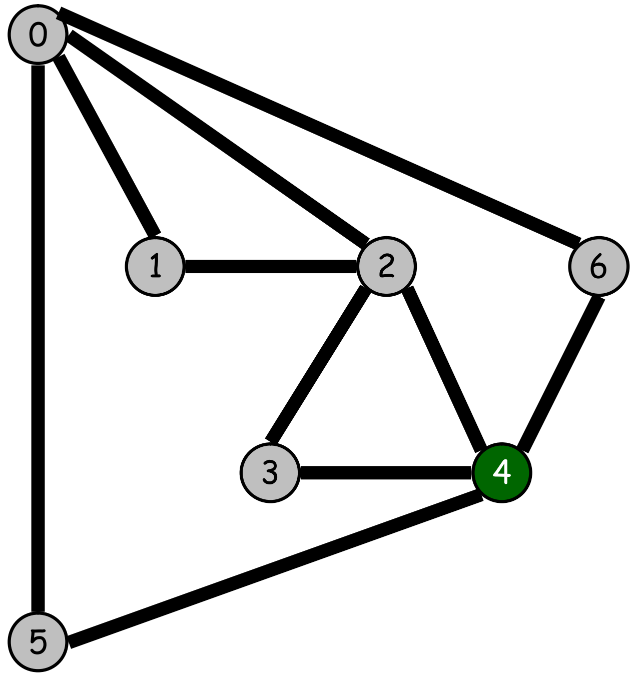
stdout

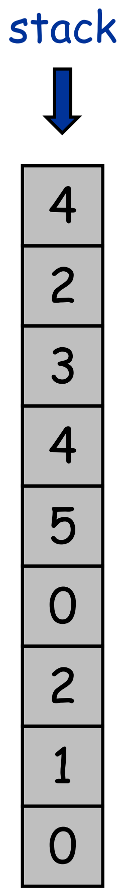
```
0 6
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~





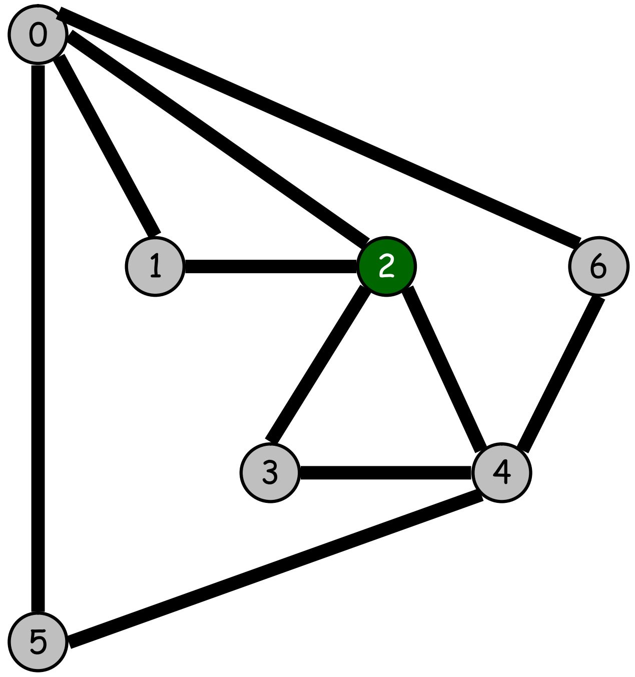
stdout

```
0 6
```

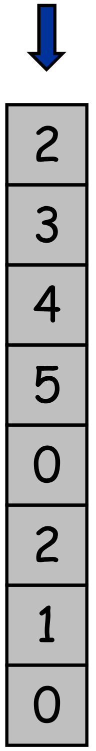
Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~



stack



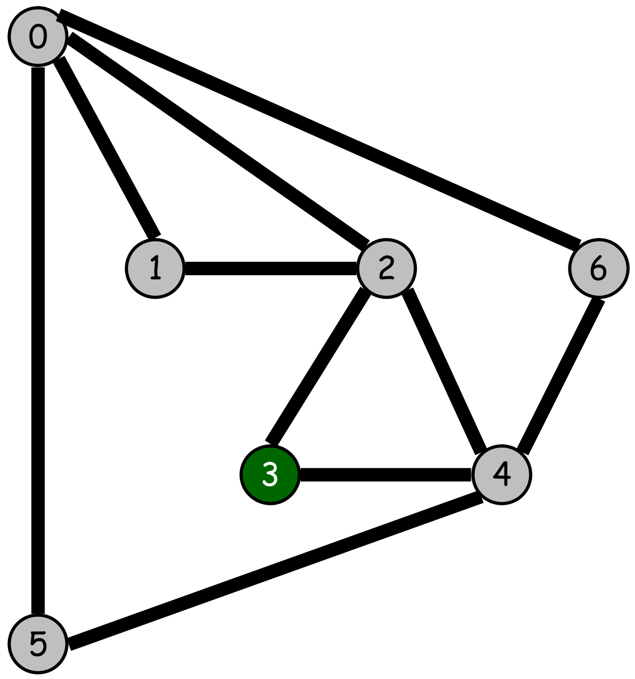
stdout

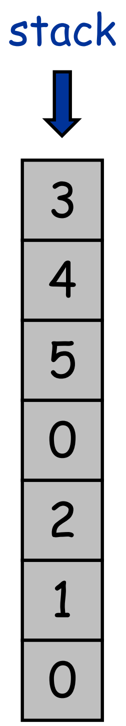
```
0 6 4
```

Euler Tour: Example

Adjacency list

```
0:  ~~1~~  ~~2~~  ~~5~~  ~~6~~
1:  ~~0~~  ~~2~~
2:  ~~0~~  ~~3~~  ~~4~~  ~~1~~
3:  ~~4~~  ~~2~~
4:  ~~6~~  ~~5~~  ~~3~~  ~~2~~
5:  ~~4~~  ~~0~~
6:  ~~4~~  ~~0~~
```





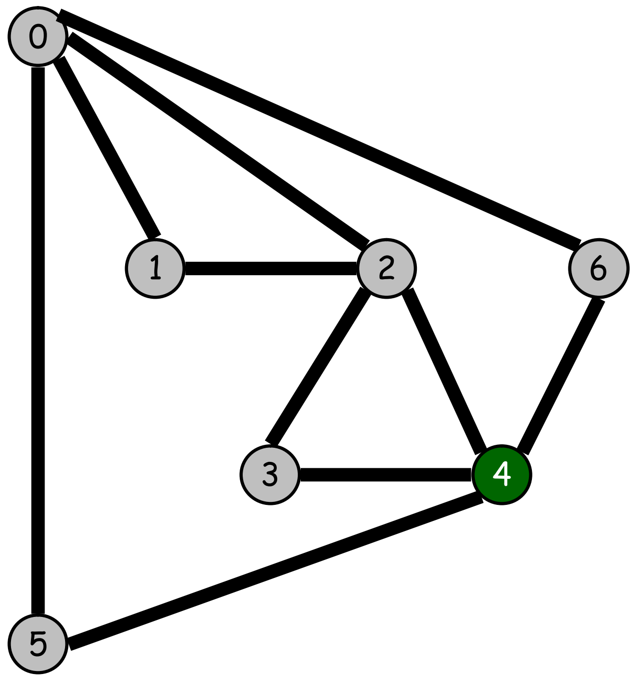
stdout

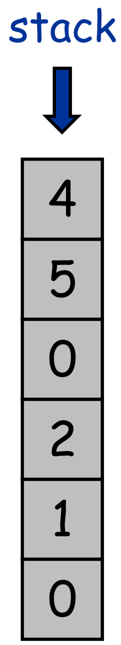
```
0 6 4 2
```

Euler Tour: Example

Adjacency list

```
0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~
```





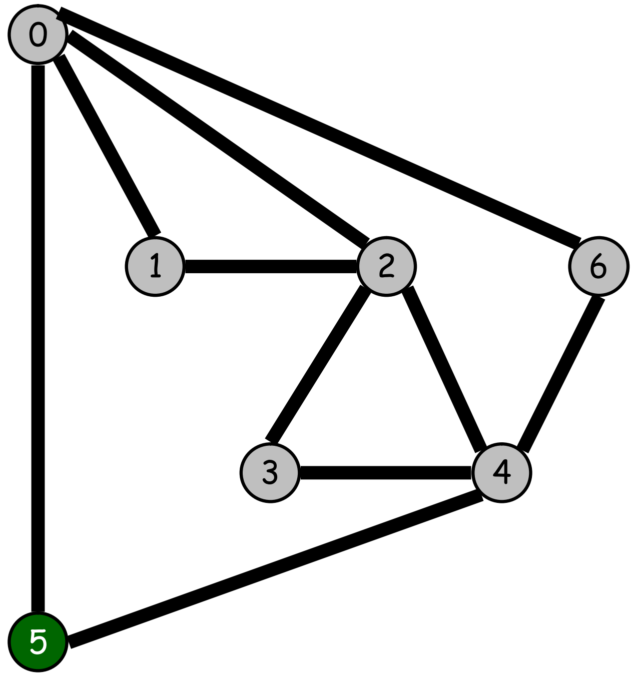
stdout

```
0 6 4 2 3
```

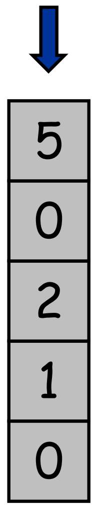
Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~



stack



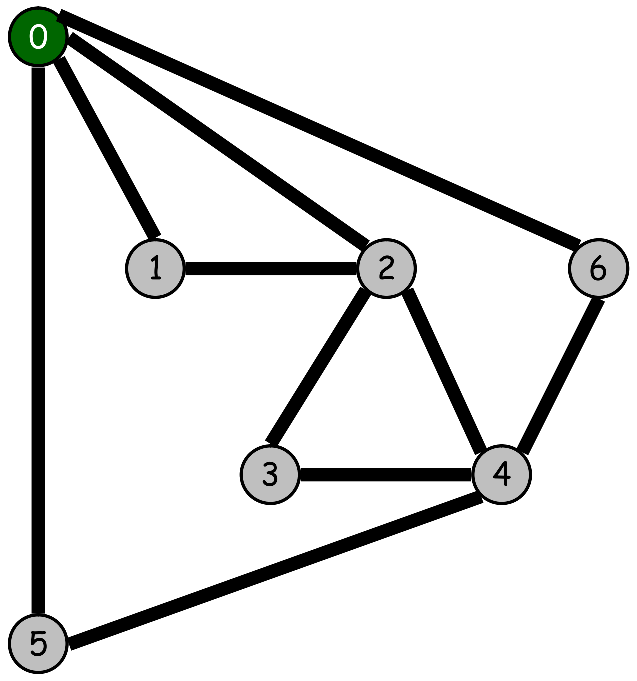
stdout

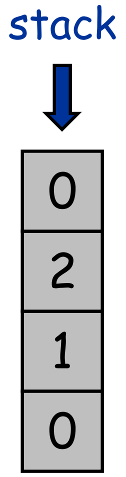
```
0 6 4 2 3 4
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~





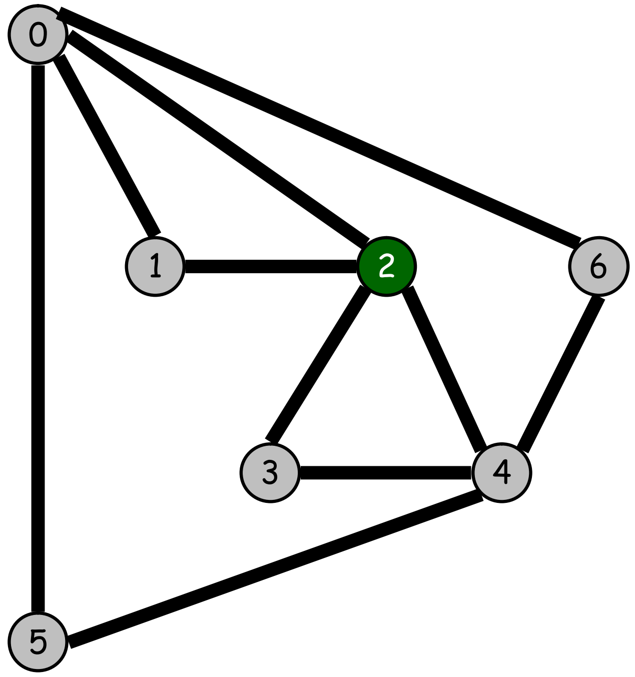
stdout

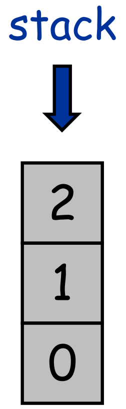
```
0 6 4 2 3 4 5
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~





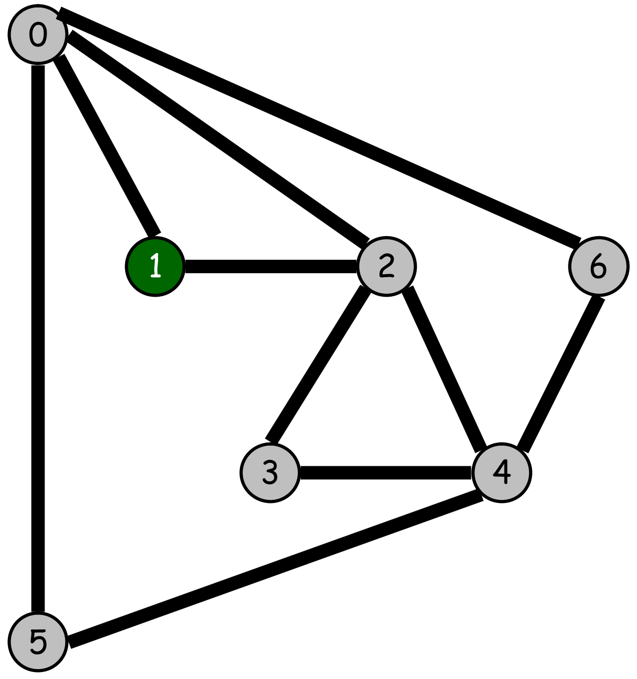
stdout

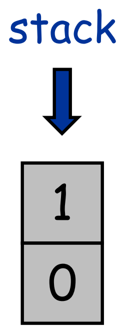
```
0 6 4 2 3 4 5 0
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~





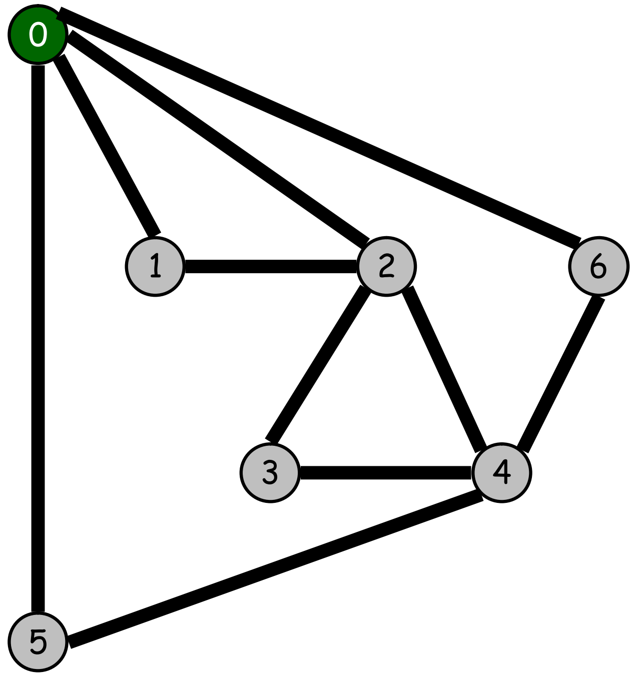
stdout

```
0 6 4 2 3 4 5 0 2
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~



stack

0

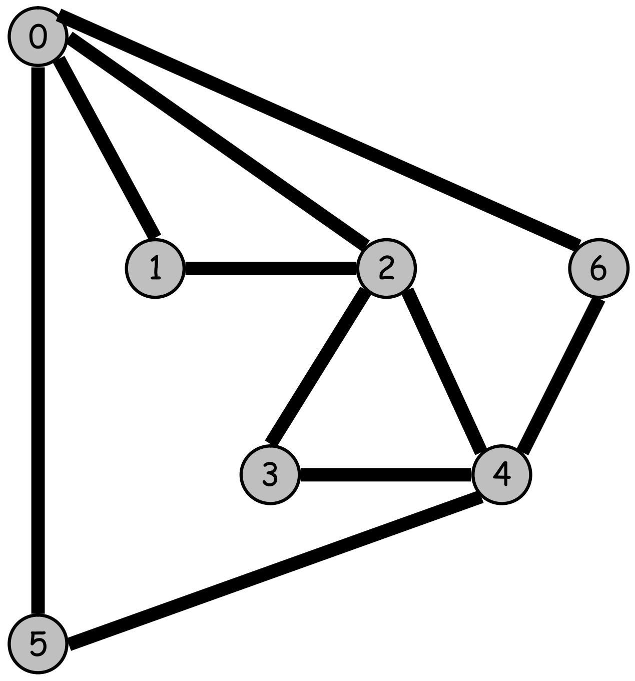
stdout

```
0 6 4 2 3 4 5 0 2 1
```

Euler Tour: Example

Adjacency list

0: ~~1~~ ~~2~~ ~~5~~ ~~6~~
1: ~~0~~ ~~2~~
2: ~~0~~ ~~3~~ ~~4~~ ~~1~~
3: ~~4~~ ~~2~~
4: ~~6~~ ~~5~~ ~~3~~ ~~2~~
5: ~~4~~ ~~0~~
6: ~~4~~ ~~0~~



stdout

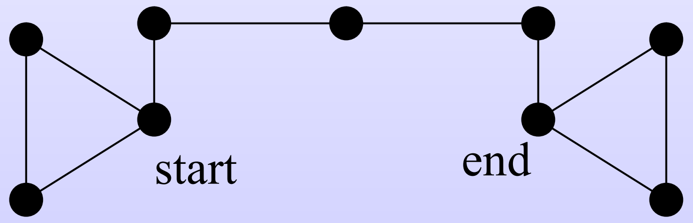
```
0 6 4 2 3 4 5 0 2 1 0
```

Corollary

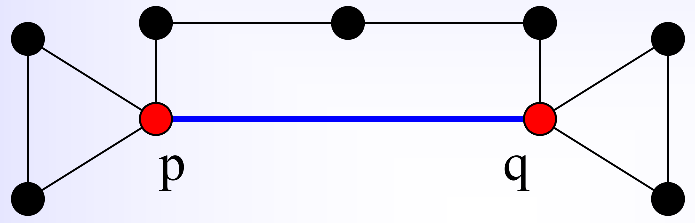
A multigraph has an Euler trail, but not an Euler cycle, if and only if it is connected and has exactly two vertices of odd degree.

Proof

- Suppose a multigraph *H* has an Euler trail *T*, but not an Euler cycle.
- The starting and ending vertices of *T* must have odd degree while all other vertices have an even degree (by the same reasoning that showed all vertices with even degree in an Euler cycle). Also the graph must be connected.

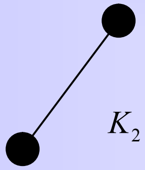


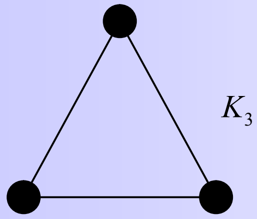
- On the other hand if a multigraph *H* is connected and has exactly two vertices, p and q, of odd degree, then to obtain a graph *H*′, add a supplementary edge (p,q) to *H*.

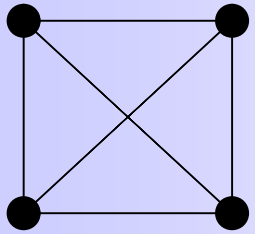


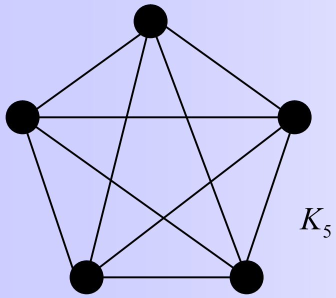
- *H*′ is now connected and has all vertices of even degree. Therefore by the Euler cycle theorem, *H*′ has an Euler cycle *C*.
- If you removed the edge (p,q) from *C*, this reduces the Euler cycle to an Euler trail that includes all of edges of *H*. Hence the corollary.

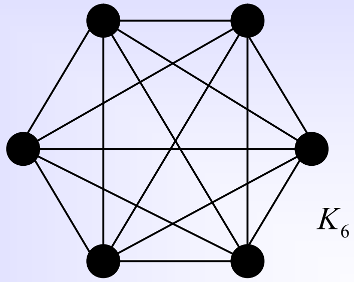
Examples

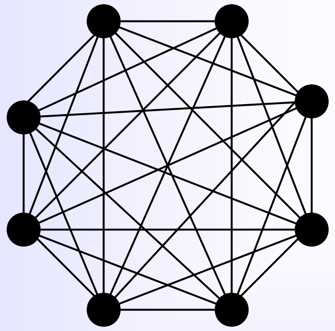

K_2


K_3


K_4


K_5


K_6


K_8

K_2: Euler trail – Yes- exactly two vertices of odd degree.

Euler cycle – No- vertices are not of even degree.

K_3 : Euler trail – No- vertices are not of odd degree.

Euler cycle – Yes- even degree vertices.

K_4 : Euler trail – No- more than two vertices of odd degree.

Euler cycle – No- there aren't any even degree vertices.

K_5 : Euler trail – No- there are no vertices of odd degree.

Euler cycle – Yes- cycle is connected and even degree.

K_6 : Euler trail – No- more than two vertices of odd degree.

Euler cycle – No- there aren't any even degree vertices.

.

K_8 : Euler trail – No- more than two vertices of odd degree.

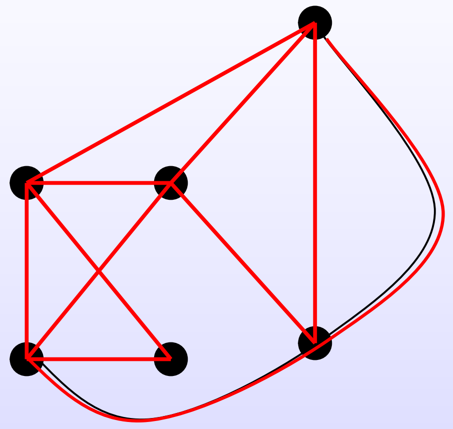
Euler cycle – No- there aren't any even degree vertices.

In general has K_n Euler cycle if n is an odd number.

Class Exercise

Give an undirected 12 edge graph, that has an Euler cycle.

Answer



-All vertices have even degree
-Is connected

Class Exercises

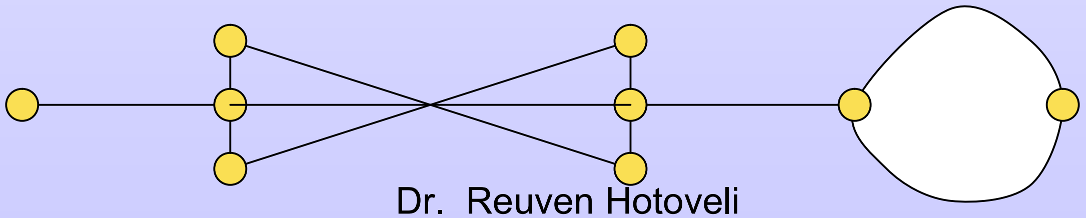
1. A) Can a graph with an Euler cycle have a bridge (an edge whose removal disconnects the graph)?

   B) Give an example of a 10-edge graph with an Euler trail that has a bridge.

Answers:

A) No. If you start on one side of the bridge and then you cross it the only way you can get back to where you started is to cross the bridge again. This means you have to travel the same edge twice and thus, a Euler cycle cannot exist.

B)



בנו גרף שקיים בו מסלול אוילר המקיים את התכונה הבאה:

– בבניית מסלול אוילר על-פי האלגוריתם לבניית מסלול אויילר, שלב ב' של האלגוריתם יכול להתבצע לפחות 3 פעמים.