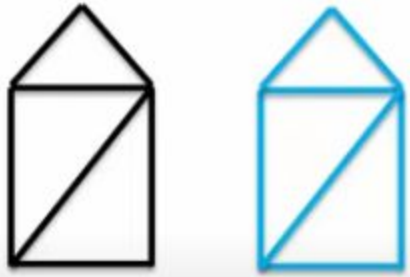
Graph Theory

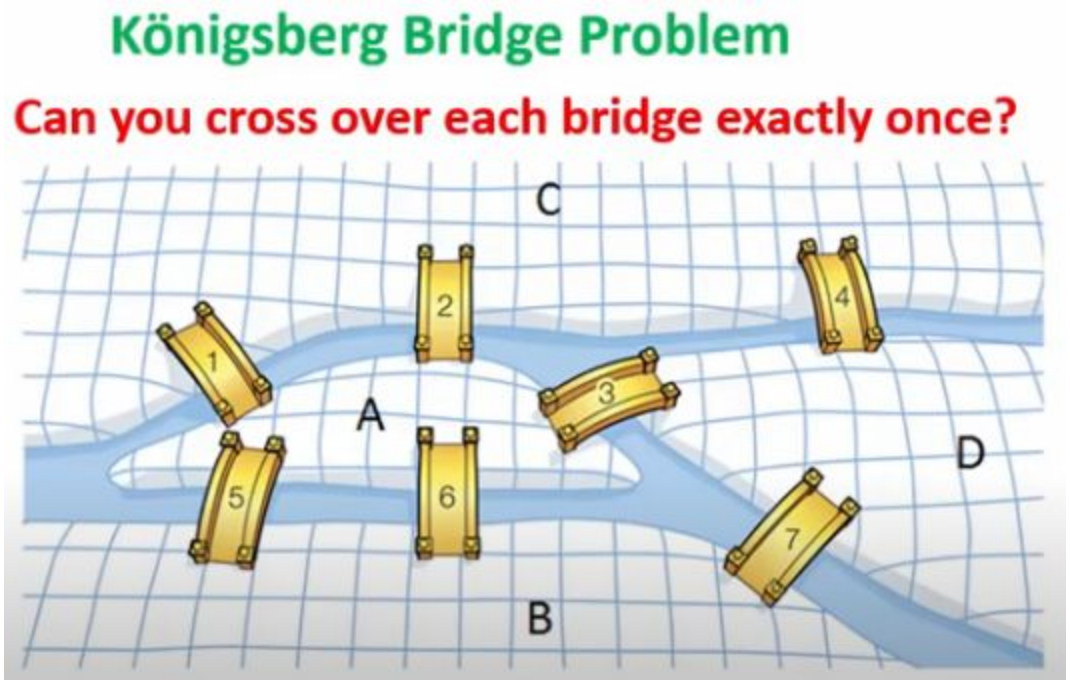
Can you draw these shapes?

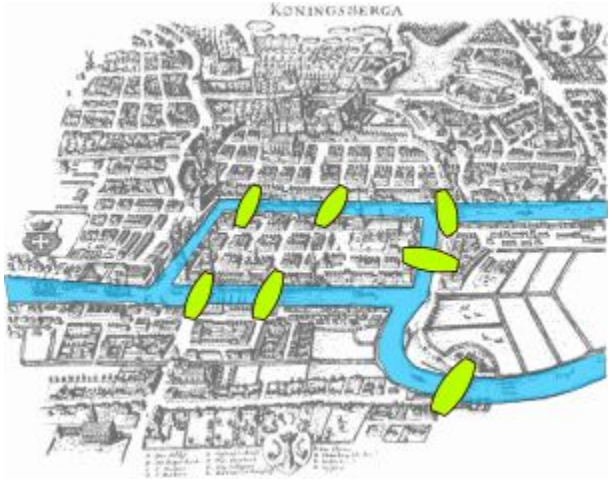


Impossible!

Königsberg Bridge Problem

Can you cross over each bridge exactly once?





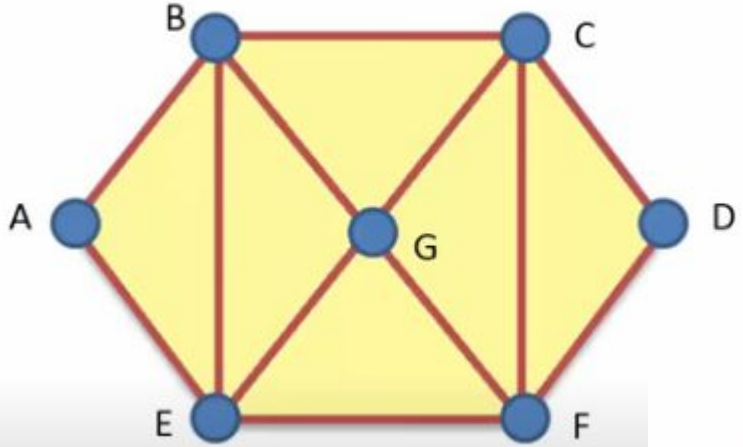
Map of Königsberg in Euler's time showing the actual layout of the seven bridges, highlighting the river Pregel and the bridges.

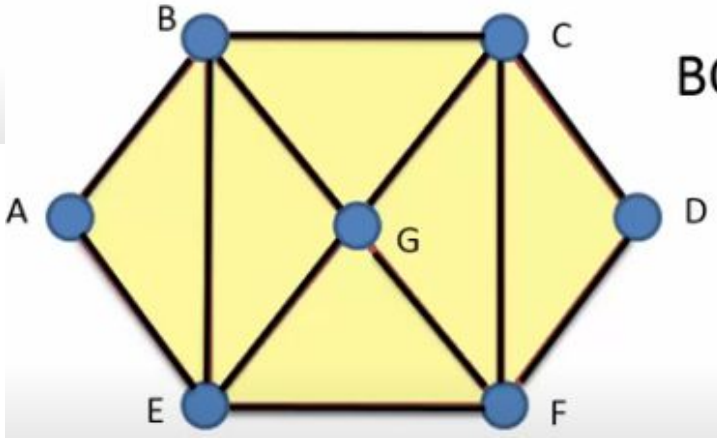
Euler first studied this question, these types of paths are named after him.

An **Euler path** is a path that uses every edge in a graph with no repeats.

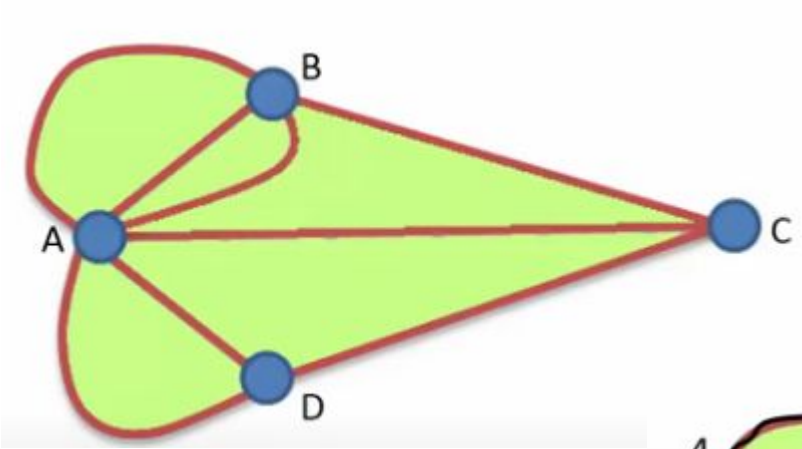
A network is traversable if you can trace the shape without lifting your pen and without going over a side more than once.

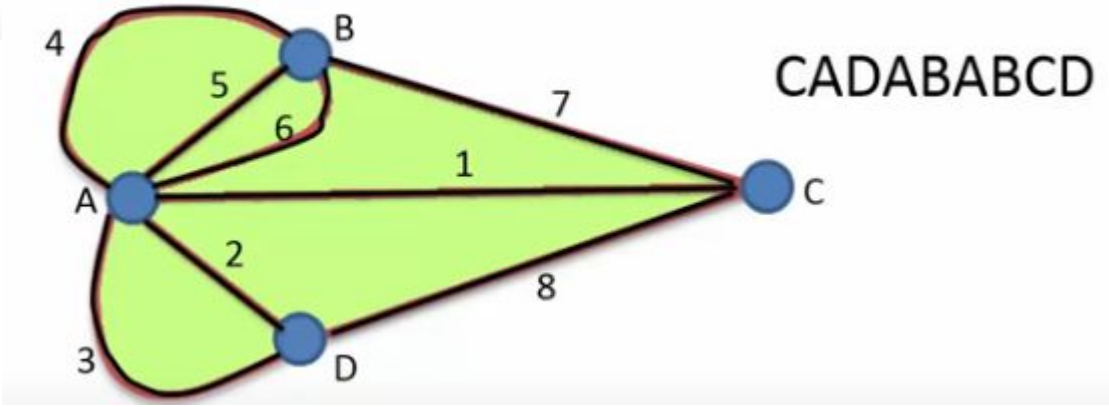
Is this shape traversable?





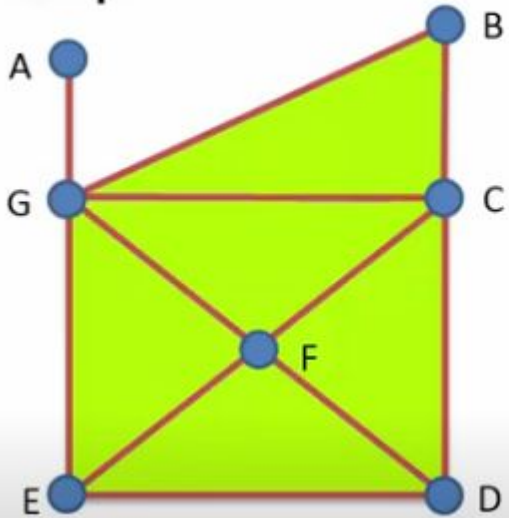
BCDFCGBAEFGEB

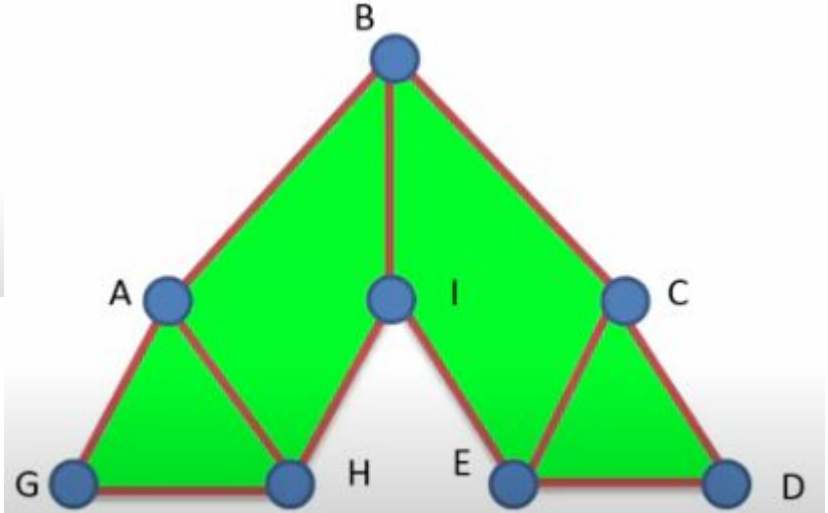




CADABABCD

This shape is not traversable.





For a shape to be traversable:

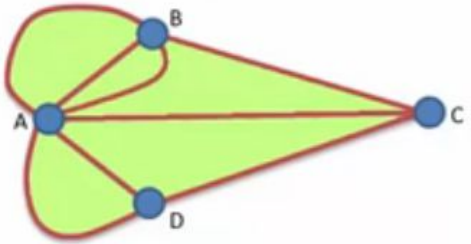
- all vertices are of even degree

or

- exactly two vertices are of odd order and the rest are even.

2 odd 2 even

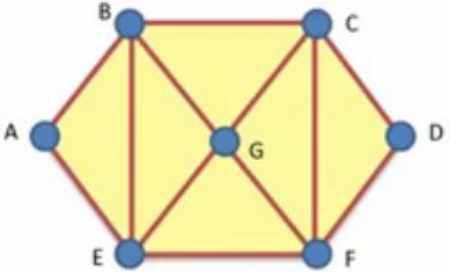
traversable



A even
B even
C odd
D odd

0 odd 7 even

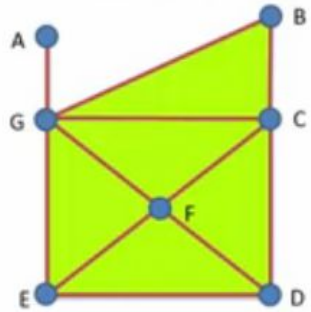
traversable



A even
B even C even
D even E even
F even G even

4 odd 3 even

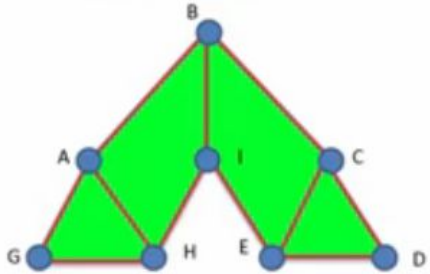
Not traversable



A odd
B even C even
D odd E odd
F even G odd

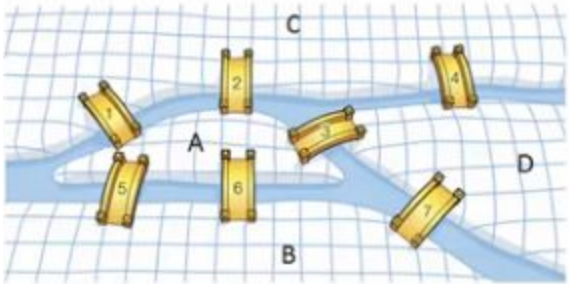
3 odd 6 even

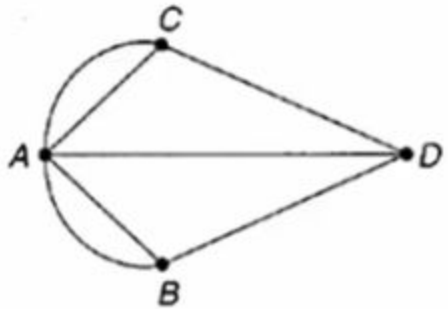
Not traversable



A odd
B odd C even
D odd E even
F even G odd
H even I odd

Can you cross over each bridge exactly once?





A odd
B odd
C odd
D odd

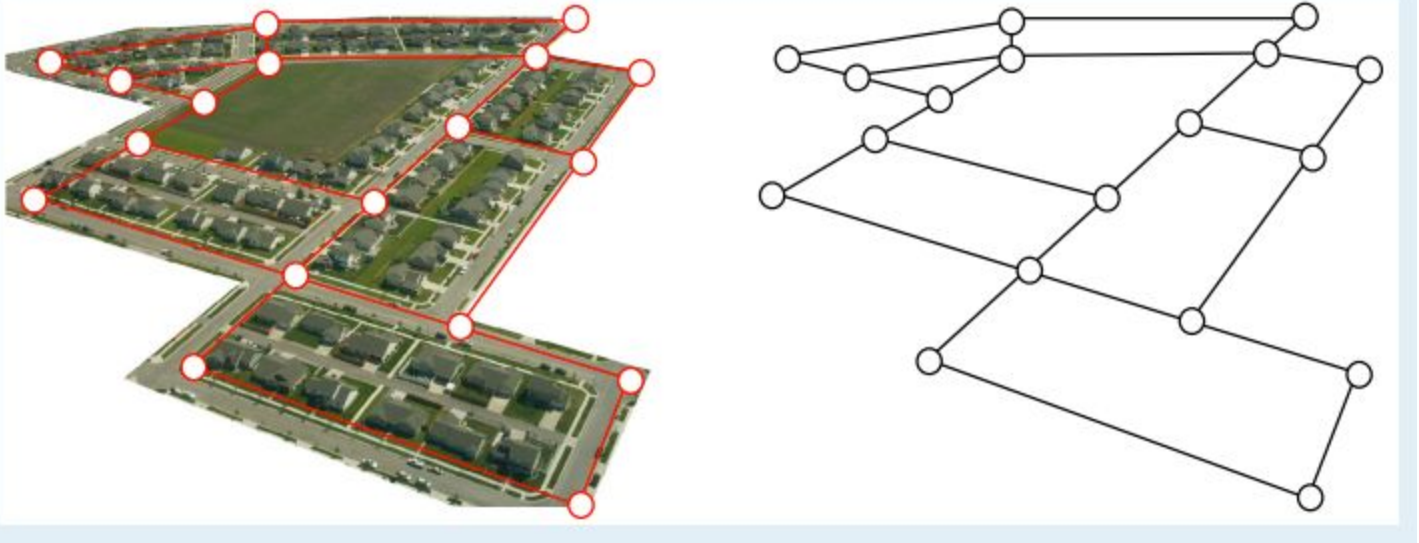
4 odd 0 even

Not traversable

Here is a portion of a housing development from Missoula, Montana. As part of her job, the development's lawn inspector has to walk down every street in the development making sure homeowners' landscaping conforms to the community requirements.

Naturally, she wants to minimize the amount of walking she has to do. Is it possible for her to walk down every street in this development without having to do any backtracking? While you might be able to answer that question just by looking at the picture for a while, it would be ideal to be able to answer the question for any picture regardless of its complexity.

To do that, we first need to simplify the picture into a form that is easier to work with. We can do that by drawing a simple line for each street. Where streets intersect, we will place a dot.

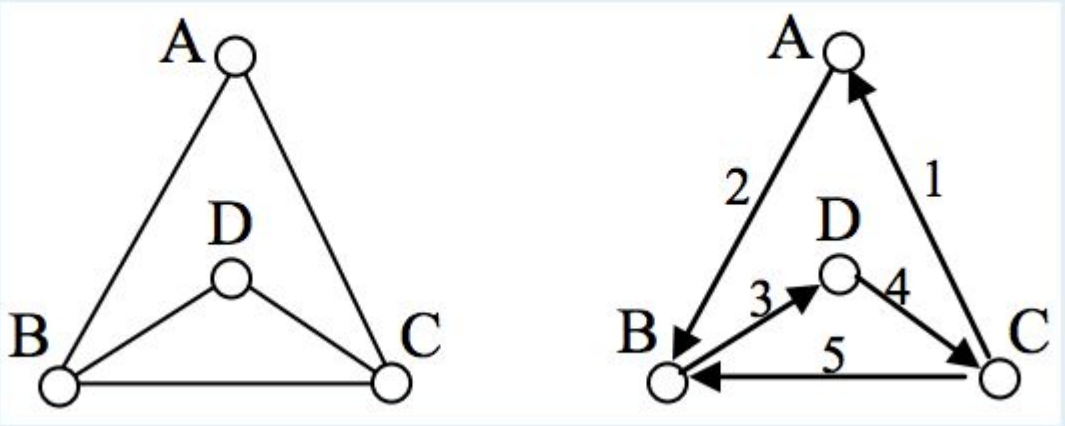


EULER PATH

In the first section, we created a graph of the Königsberg bridges and asked whether it was possible to walk across every bridge once. Because Euler first studied this question, these types of paths are named after him.

An **Euler path** is a path that uses every edge in a graph with no repeats. Being a path, it does not have to return to the starting vertex.

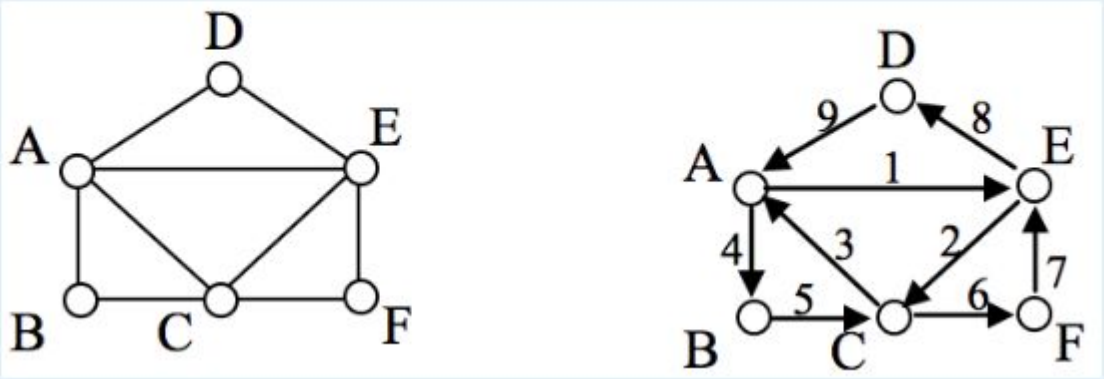
In the graph shown below, there are several Euler paths. One such path is CABDCB. The path is shown in arrows to the right, with the order of edges numbered.



EULER CIRCUIT

An **Euler circuit** is a circuit that uses every edge in a graph with no repeats. Being a circuit, it must start and end at the same vertex.

The graph below has several possible Euler circuits. Here's a couple, starting and ending at vertex A: ADEACEFCBA and AECABCFEDA. The second is shown in arrows.

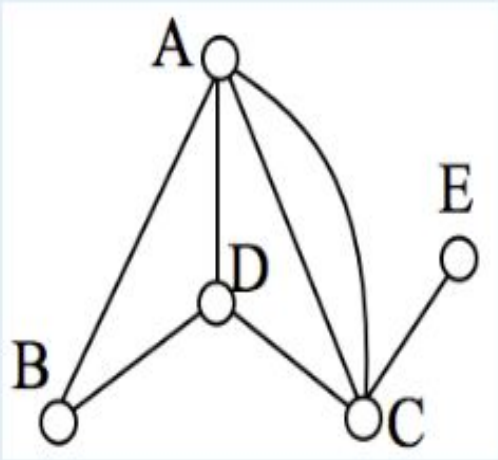


EULER'S PATH AND CIRCUIT THEOREMS

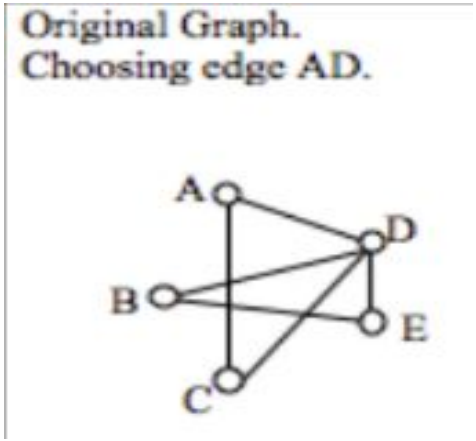
A graph will contain an Euler path if it contains at most two vertices of odd degree.

A graph will contain an Euler circuit if all vertices have even degree

In the graph below, vertices A and C have degree 4, since there are 4 edges leading into each vertex. B is degree 2, D is degree 3, and E is degree 1. This graph contains two vertices with odd degree (D and E) and three vertices with even degree (A, B, and C), so Euler's theorems tell us this graph has an Euler path, but not an Euler circuit.



Find an Euler Circuit on this graph starting at vertex A.

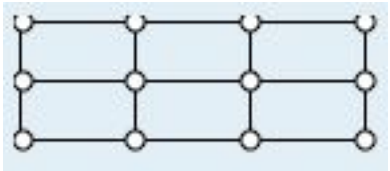


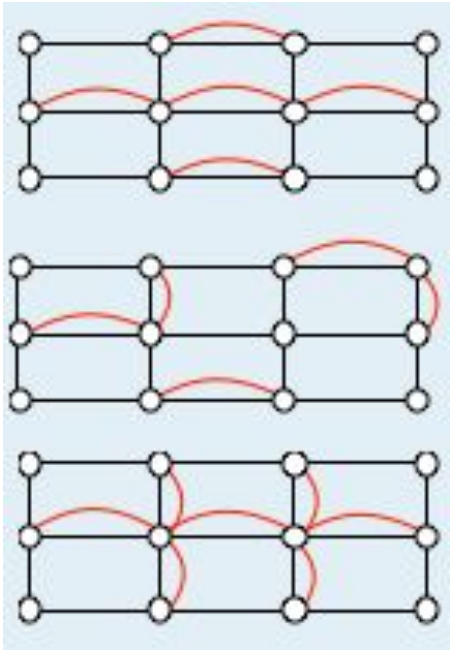
EULERIZATION

Eulerization is the process of adding edges to a graph to create an Euler circuit on a graph. To eulerize a graph, edges are duplicated to connect pairs of vertices with odd degree. Connecting two odd degree vertices increases the degree of each, giving them both even degree. When two odd degree vertices are not directly connected, we can duplicate all edges in a path connecting the two.

Note that we can only duplicate edges, not create edges where there wasn’t one before. Duplicating edges would mean walking or driving down a road twice, while creating an edge where there wasn’t one before is akin to installing a new road!

For the rectangular graph shown, three possible eulerizations are shown. Notice in each of these cases the vertices that started with odd degrees have even degrees after eulerization, allowing for an Euler circuit.

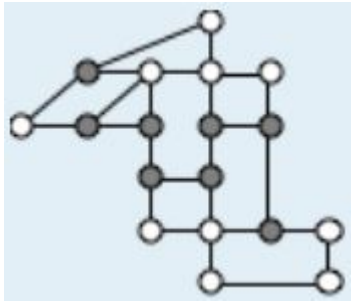


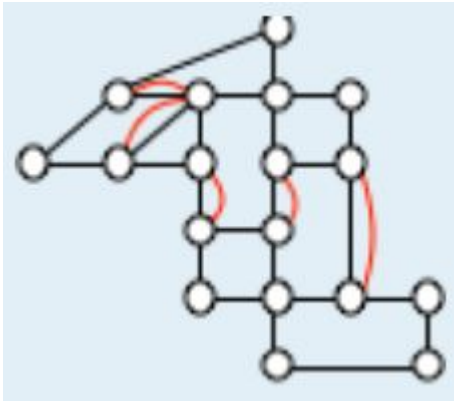


In the example above, you'll notice that the last eulerization required duplicating seven edges, while the first two only required duplicating five edges. If we were eulerizing the graph to find a walking path, we would want the eulerization with minimal duplications. If the edges had weights representing distances or costs, then we would want to select the eulerization with the minimal total added weight.

Chinese Postman Problem
The problem of finding the optimal eulerization is called the Chinese Postman Problem

he vertices with odd degree are shown highlighted. With eight vertices, we will always have to duplicate at least four edges. In this case, we need to duplicate five edges since two odd degree vertices are not directly connected. Without weights we can’t be certain this is the eulerization that minimizes walking distance, but it looks pretty good.





Hamiltonian Circuits

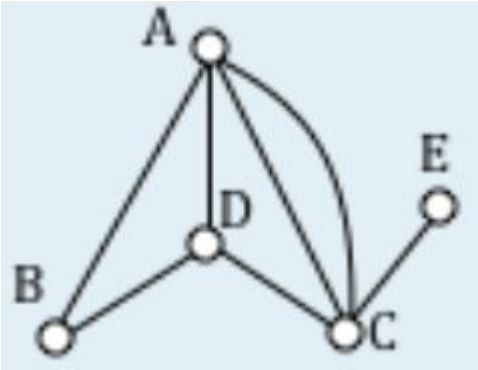
In the last section, we considered optimizing a walking route for a postal carrier. How is this different than the requirements of a package delivery driver? While the postal carrier needed to walk down every street (edge) to deliver the mail, the package delivery driver instead needs to visit every one of a set of delivery locations. Instead of looking for a circuit that covers every edge once, the package deliverer is interested in a circuit that visits every vertex once.

This is also called as **The Traveling Salesman Problem.**

HAMILTONIAN CIRCUITS AND PATHS

A **Hamiltonian circuit** is a circuit that visits every vertex once with no repeats. Being a circuit, it must start and end at the same vertex. A **Hamiltonian path** also visits every vertex once with no repeats, but does not have to start and end at the same vertex.

Does a Hamiltonian path or circuit exist on the graph below?



We can see that once we travel to vertex E there is no way to leave without returning to C, so there is no possibility of a Hamiltonian circuit. If we start at vertex E we can find several Hamiltonian paths, such as ECDAB and ECABD

This problem is called the **Traveling salesman problem** (TSP) because the question can be framed like this: Suppose a salesman needs to give sales pitches in four cities. He looks up the airfares between each city, and puts the costs in a graph. In what order should he travel to visit each city once then return home with the lowest cost?

To answer this question of how to find the lowest cost Hamiltonian circuit, we will consider some possible approaches.

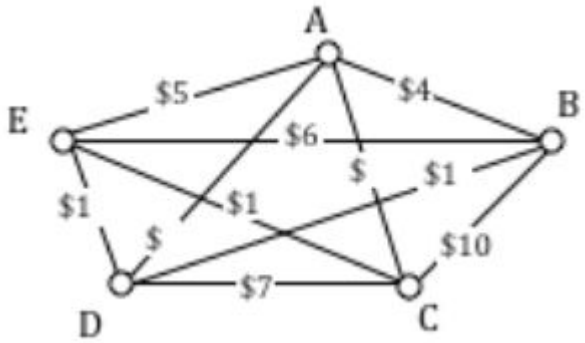
every vertex is connected to every other vertex. This is called a **complete graph.**

NUMBER OF POSSIBLE CIRCUITS

For *N* vertices in a complete graph, there will be $(n-1)! = (n-1)(n-2)(n-3)\ldots 3 \cdot 2 \cdot 1$ routes. Half of these are duplicates in reverse order, so there are $\frac{(n-1)!}{2}$ unique circuits.

Spanning Trees

A company requires reliable internet and phone connectivity between their five offices (named A, B, C, D, and E for simplicity) in New York, so they decide to lease dedicated lines from the phone company. The phone company will charge for each link made. The costs, in thousands of dollars per year, are shown in the graph.



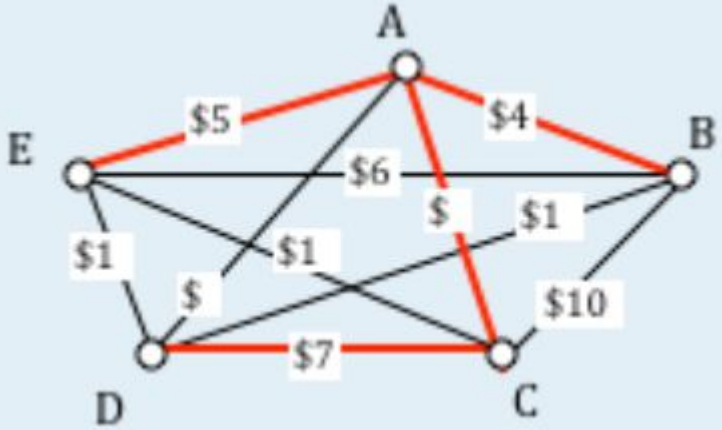
In this case, we don't need to find a circuit, or even a specific path; all we need to do is make sure we can make a call from any office to any other. In other words, we need to be sure there is a path from any vertex to any other vertex.

A **spanning tree** is a connected graph using all vertices in which there are no circuits.

In other words, there is a path from any vertex to any other vertex, but no circuits.

Using our phone line graph from above, begin adding edges:

AB	$4	OK
AE	$5	OK
BE	$6	reject – closes circuit ABEA
DC	$7	OK
AC	$8	OK



At this point we stop – every vertex is now connected, so we have formed a spanning tree with cost $24 thousand a year.

KRUSKAL'S ALGORITHM

1. Select the cheapest unused edge in the graph.
2. Repeat step 1, adding the cheapest unused edge, unless:
3. adding the edge would create a circuit
4. Repeat until a spanning tree is formed

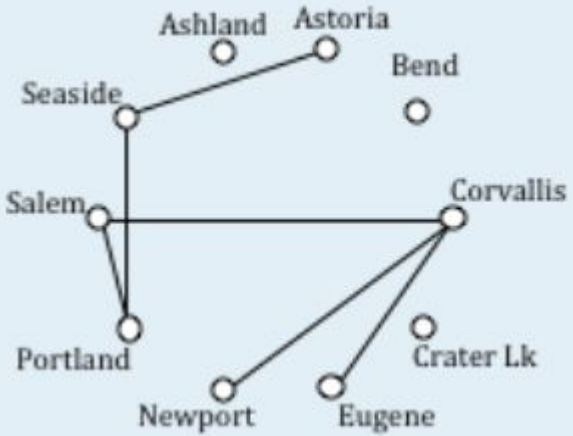
The power company needs to lay updated distribution lines connecting the ten cities below to the power grid. How can they minimize the amount of new line to lay?

	Ashland	Astoria	Bend	Corvallis	Crater Lake	Eugene	Newport	Portland	Salem	Seaside
Ashland	–	374	200	223	108	178	252	285	240	356
Astoria	374	–	255	166	433	199	135	95	136	17
Bend	200	255	–	128	277	128	180	160	131	247
Corvallis	223	166	128	–	430	47	52	84	40	155
Crater Lake	108	433	277	430	–	453	478	344	389	423
Eugene	178	199	128	47	453	–	91	110	64	181
Newport	252	135	180	52	478	91	–	114	83	117
Portland	285	95	160	84	344	110	114	–	47	78
Salem	240	136	131	40	389	64	83	47	–	118
Seaside	356	17	247	155	423	181	117	78	118	–

Using Kruskal's algorithm, we add edges from cheapest to most expensive, rejecting any that close a circuit. We stop when the graph is connected.

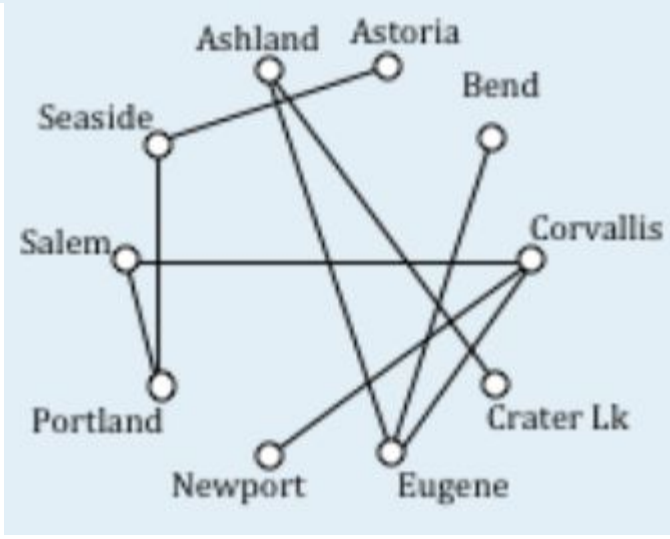
Seaside to Astoria	17 miles	Corvallis to Salem	40 miles
Portland to Salem	47 miles		
Corvallis to Eugene	47 miles		
Corvallis to Newport	52 miles		
Salem to Eugene	reject – closes circuit		
Portland to Seaside	78 miles		

The graph up to this point is shown below.

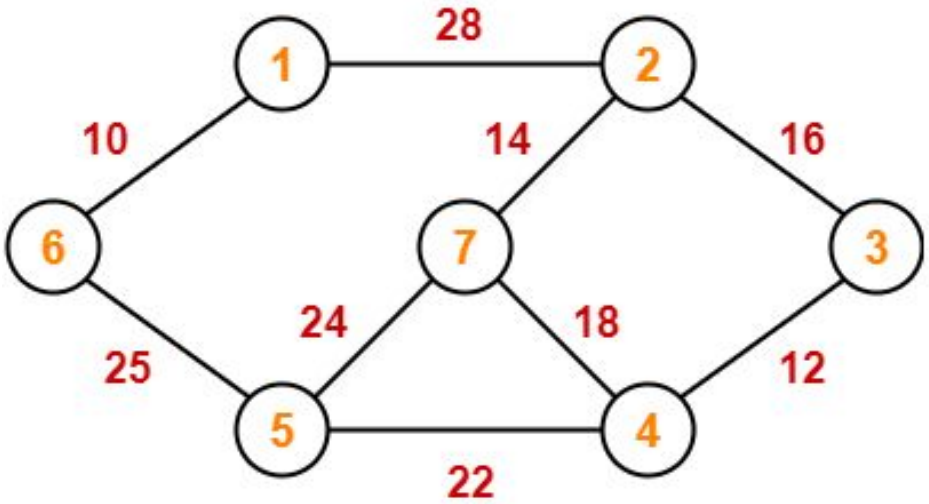


Newport to Salem	reject
Corvallis to Portland	reject
Eugene to Newport	reject
Portland to Astoria	reject
Ashland to Crater Lk	108 miles
Eugene to Portland	reject
Newport to Portland	reject
Newport to Seaside	reject
Salem to Seaside	reject
Bend to Eugene	128 miles

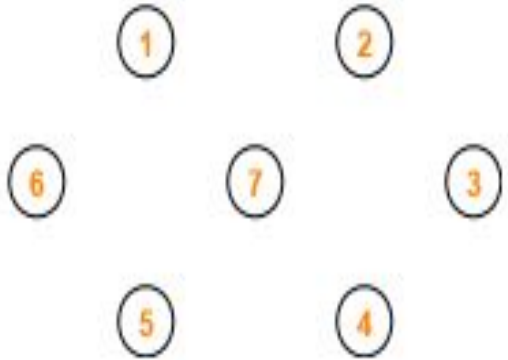
Bend to Salem	reject
Astoria to Newport	reject
Salem to Astoria	reject
Corvallis to Seaside	reject
Portland to Bend	reject
Astoria to Corvallis	reject
Eugene to Ashland	178 miles



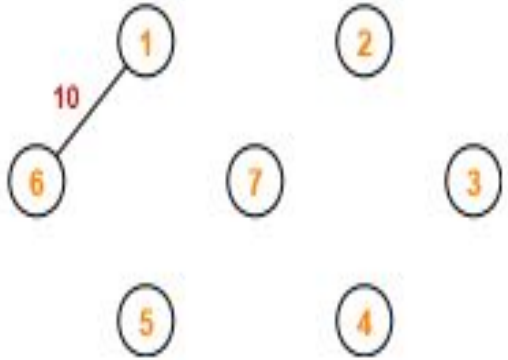
construct MST using Kruskal's Algorithm



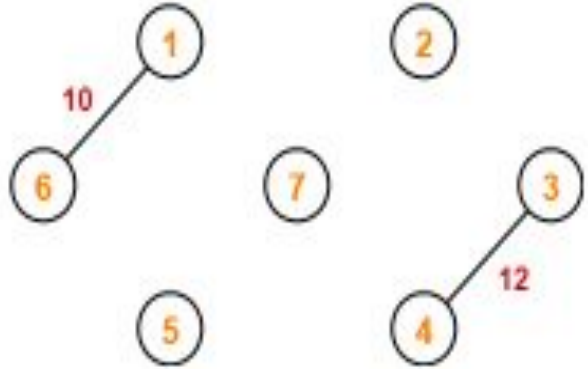
Step-01:



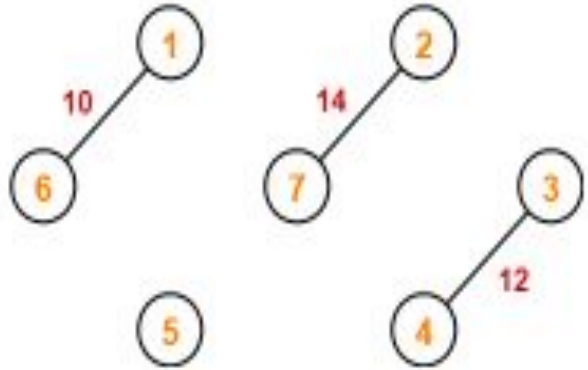
Step-02:



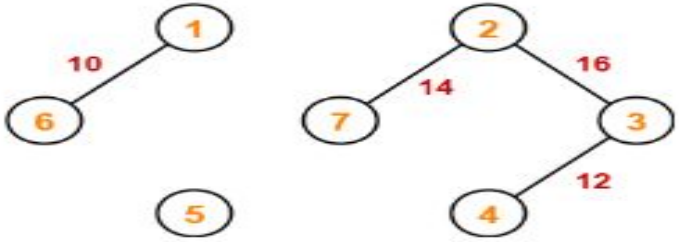
Step-03:



Step-04:

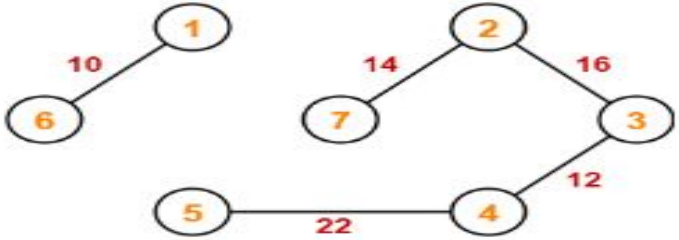


Step-05:



Since all the vertices have been connected / included in the MST, so we stop.

Step-06:



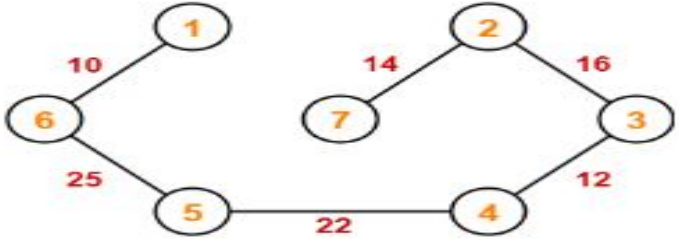
Weight of the MST

= Sum of all edge weights

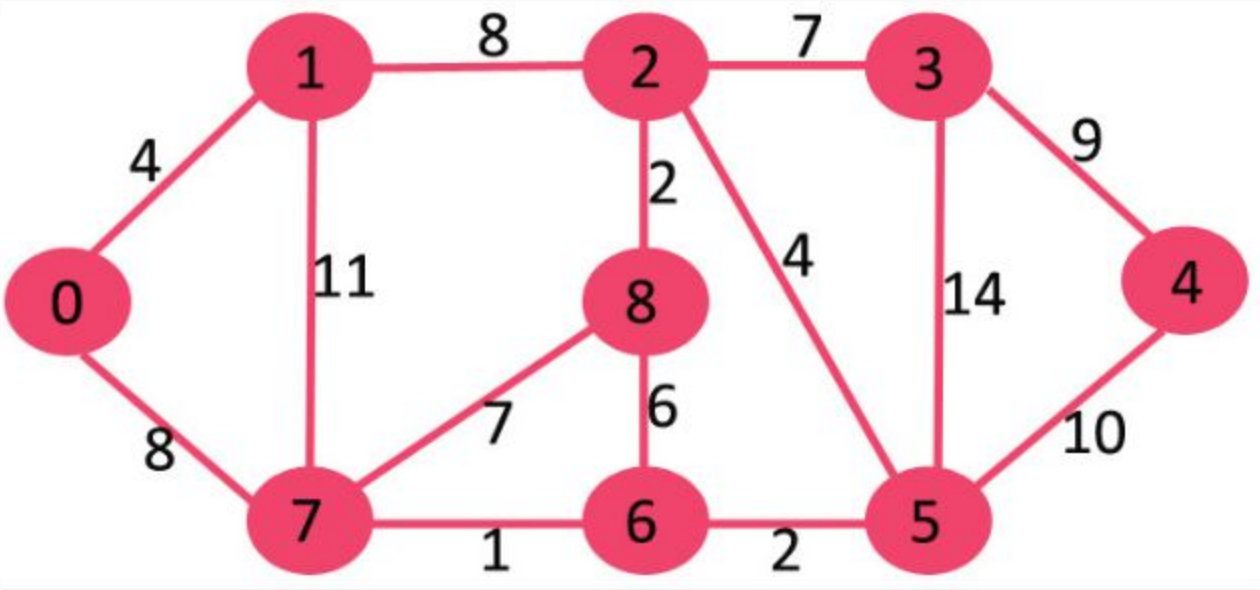
= 10 + 25 + 22 + 12 + 16 + 14

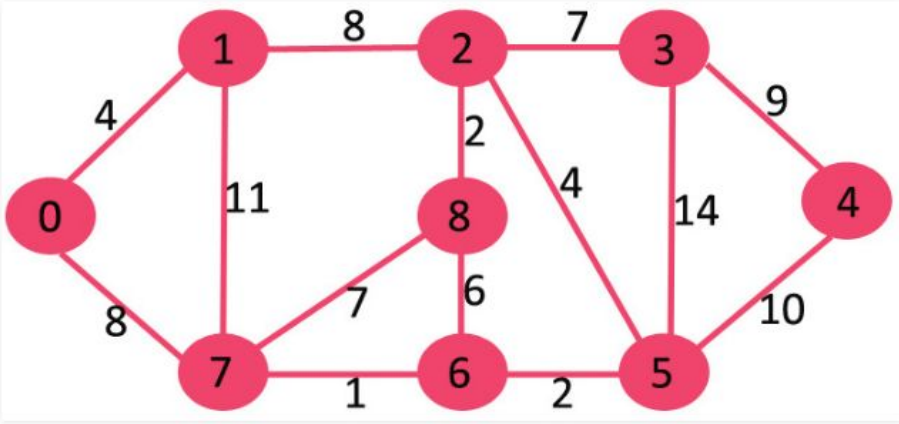
= 99 units

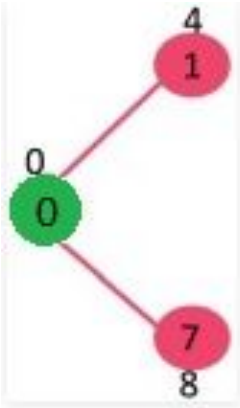
Step-07:

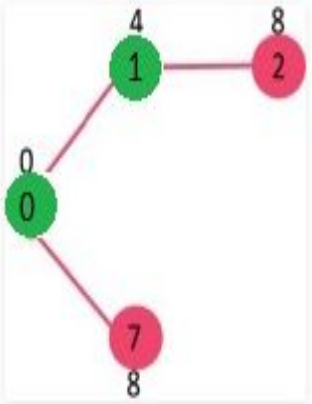


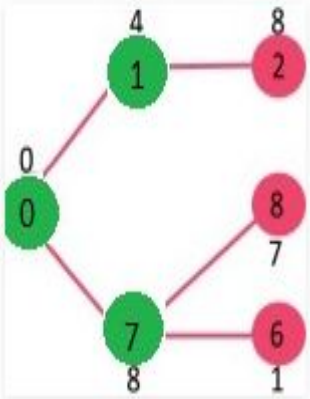
Prim’s Algorithm

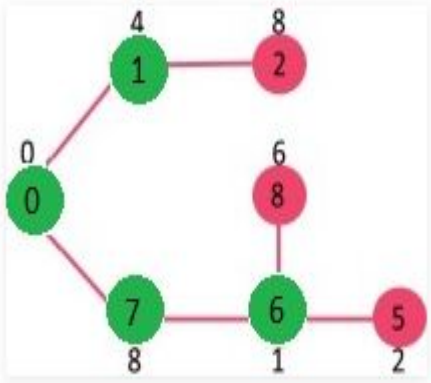


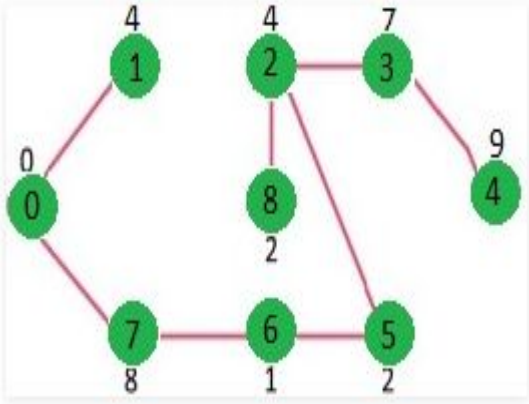

0
1
2
3
4
5
6
7
8
4
8
8
11
7
2
4
14
9
7
6
10
1
2


0
1
4
0
7
8

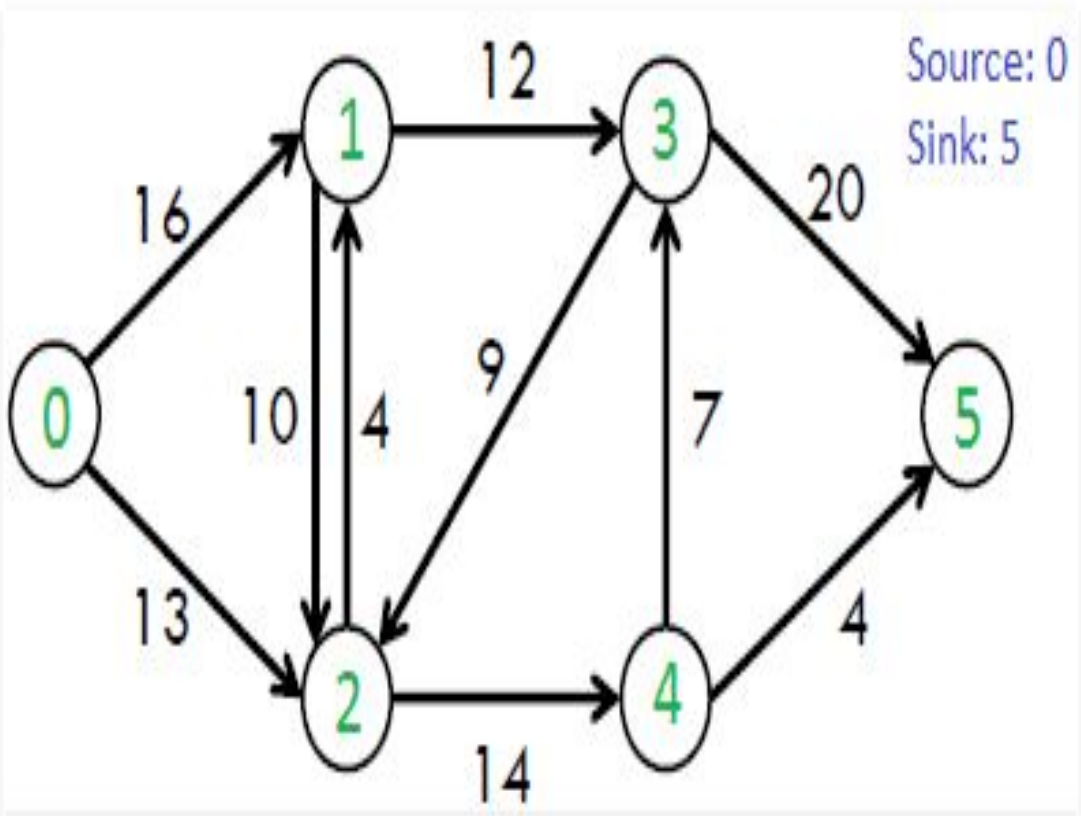

0
1
2
4
8
0
7
8


0
1
2
4
8
0
8
7
7
6
8
1

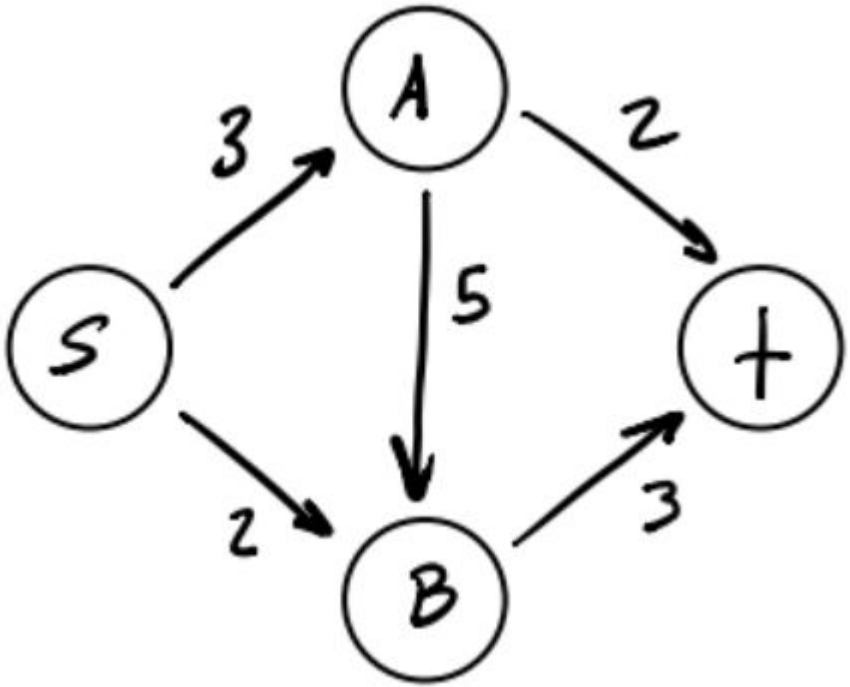

0
1
2
4
8
0
6
8
7
6
5
8
1
2

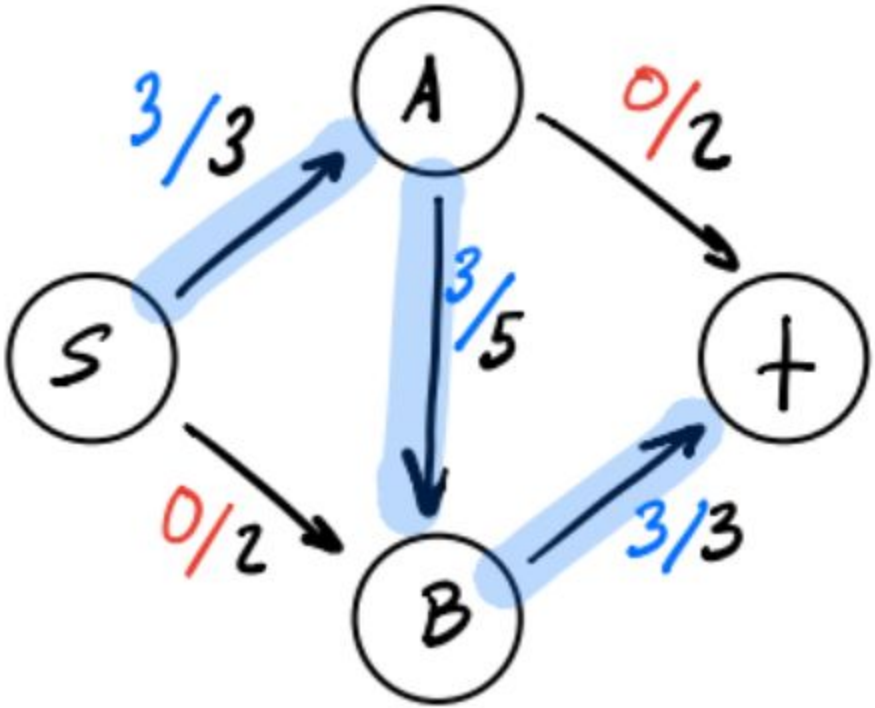

0
1
2
3
4
4
4
7
9
0
8
2
7
6
5
8
1
2

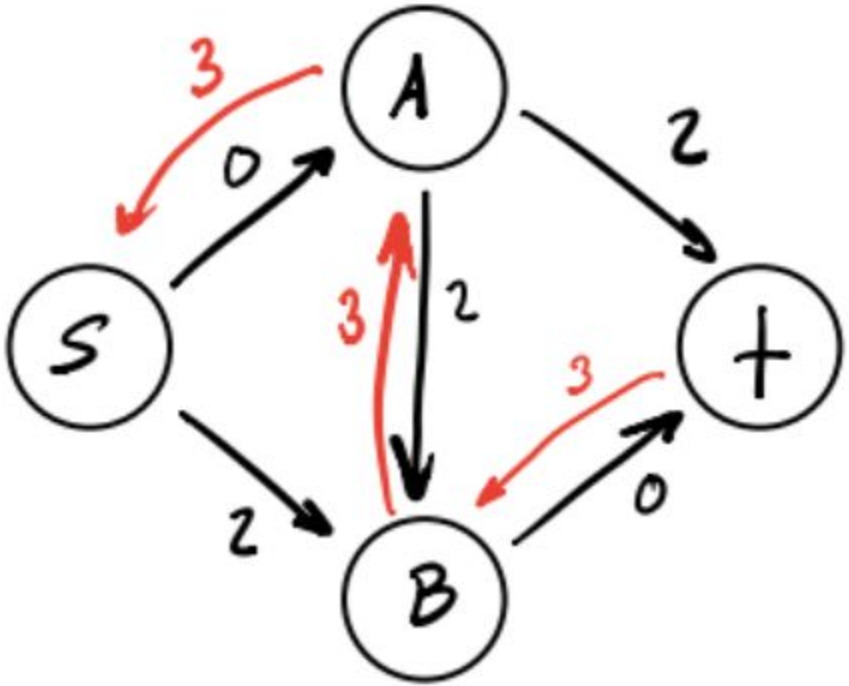
Ford Fulkerson Algorithm

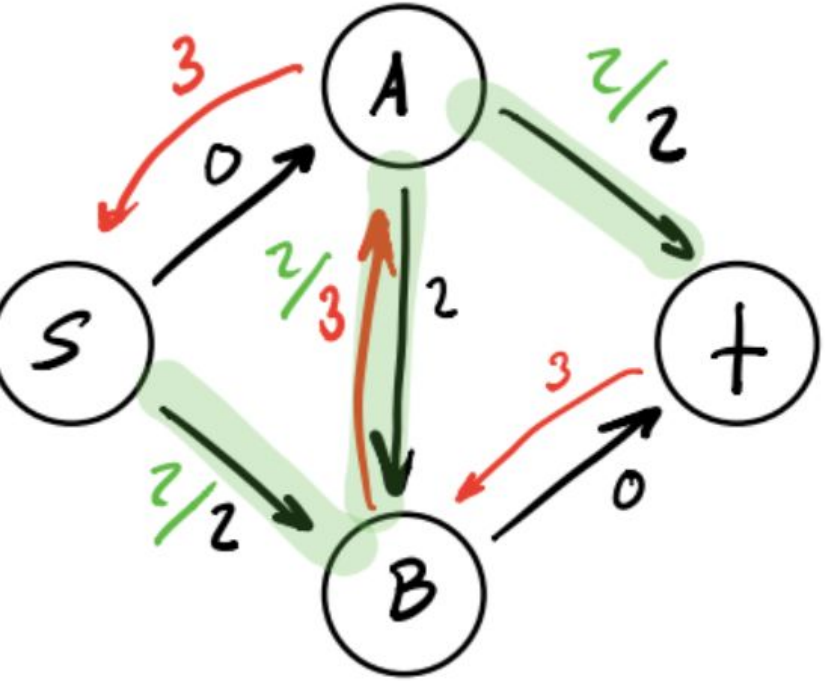


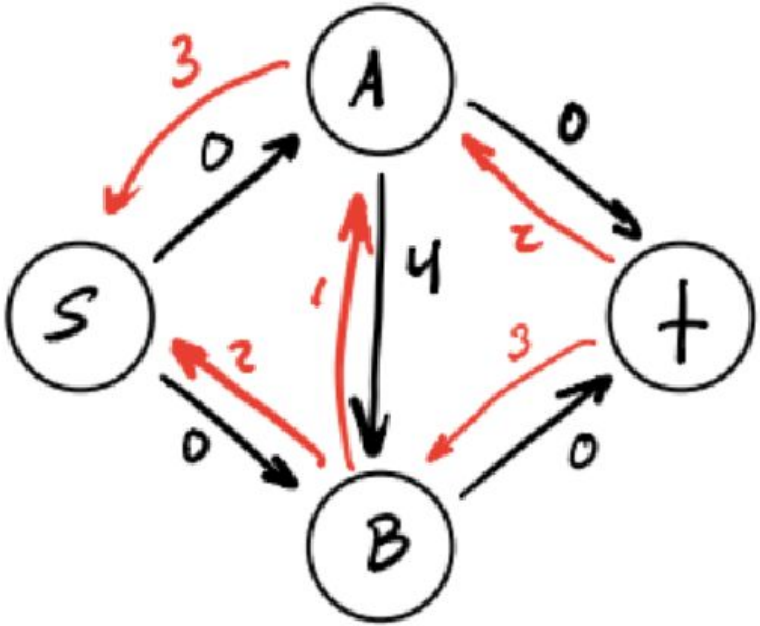
ford-fulkerson algorithm




A
S
t
B
3/3
0/2
3/5
0/2
3/3


A
S
t
B
3
0
2
3
2
3
2
0


3
A
2/2
0
S
2/3
2
t
3
2/2
0
B


3
A
0
0
S
1
4
2
t
2
3
0
0
B