Eulerian Cycle (2A)

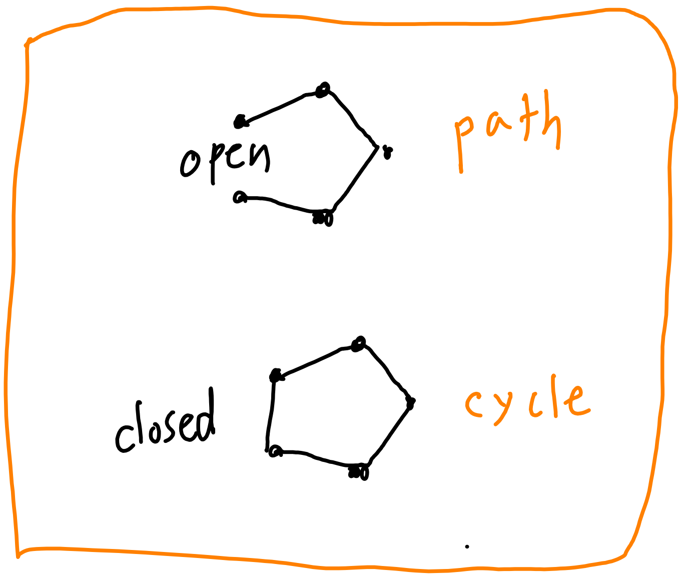
Walk : vertices may repeat, edges may repeat (closed or open)

Trail: vertices may repeat, edges cannot repeat (open)

circuit : vertices my repeat, edges cannot repeat (closed)

path : vertices cannot repeat, edges cannot repeat (open)

cycle : vertices cannot repeat, edges cannot repeat (closed)



E path

E cycle

1) Undirected Graphs

2) Directed Graphs

Node

Vertex, Vertices

정점

Edge, Edges

간선

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Euler Cycle

visits every <u>edge</u> exactly <u>once</u>

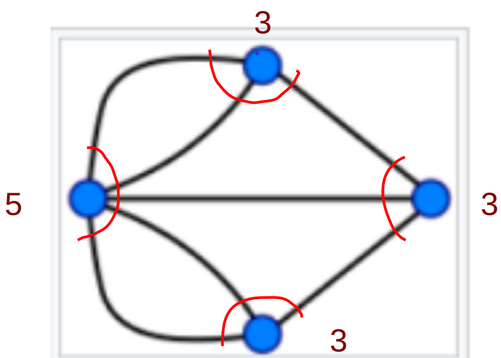
a necessary condition
for the existence of Eulerian cycles is
that all vertices in the graph have an **even** degree

that connected graphs with **all** vertices of **even** degree
have an Eulerian circuit.

https://en.wikipedia.org/wiki/Eulerian_path

Euler Cycle

degree 차수

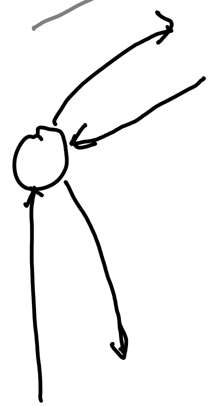


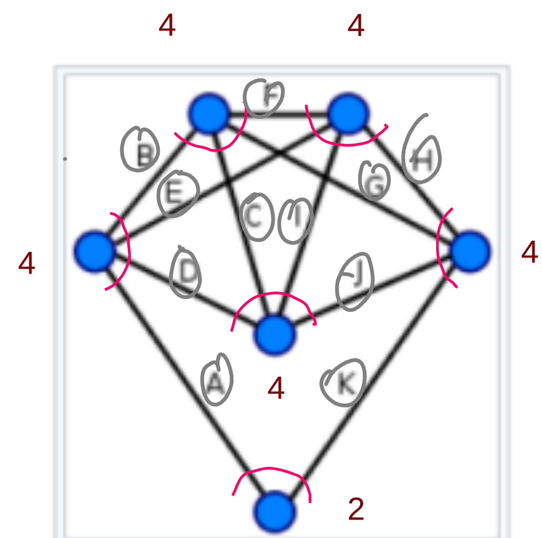
The Königsberg Bridges multigraph. This multigraph is not Eulerian, therefore, a solution does not exist.

All odd degree vertices

6개 vertices

11개 edge
A, ..., K





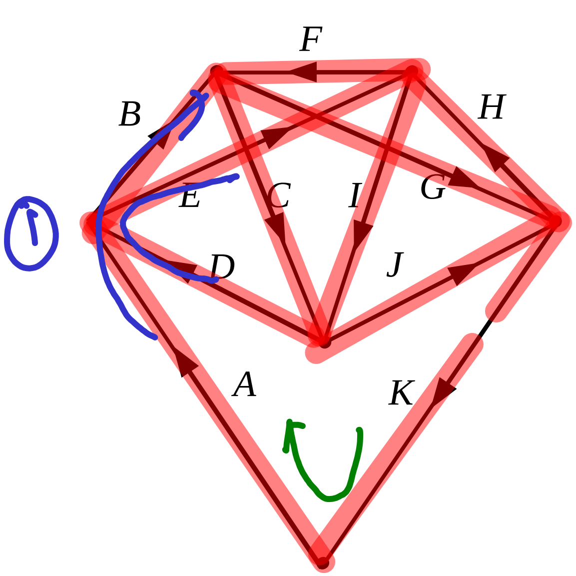
Every vertex of this graph has an even degree. Therefore, this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle.

All even degree vertices

https://en.wikipedia.org/wiki/Eulerian_path

Euler Cycle

1 11
A B C D E F G H I J K



2번

ABCDEFGHIJK
1 2 3 4 5 6 7 8 9 10 11

Eulerian Cycle

Cycle: Vs = Vt
start, terminal

Path : Vs ----> Vt
Vs != Vt

E. Cycle O

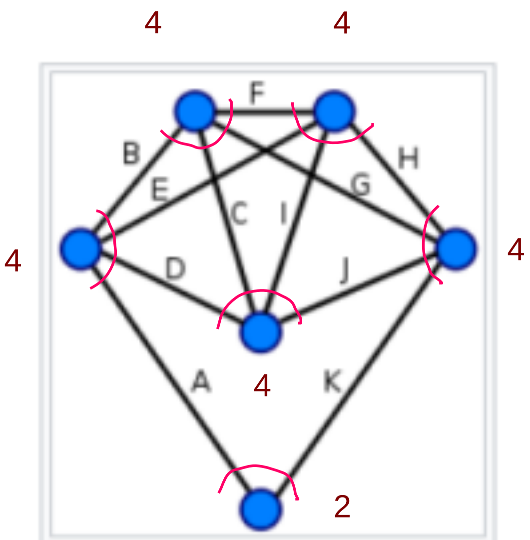
H cycle X

en.wikipedia.org

An undirected graph has an **Eulerian <u>cycle</u>**
if and only if every vertex has **even degree**,
and all of its vertices with nonzero degree
belong to a single connected component.

An undirected graph can be
decomposed into **edge-disjoint cycles**
if and only if all of its vertices have even degree.

So, a graph has an Eulerian cycle
if and only if it can be decomposed
into **edge-disjoint cycles**
and its nonzero-degree vertices
belong to a single connected component.

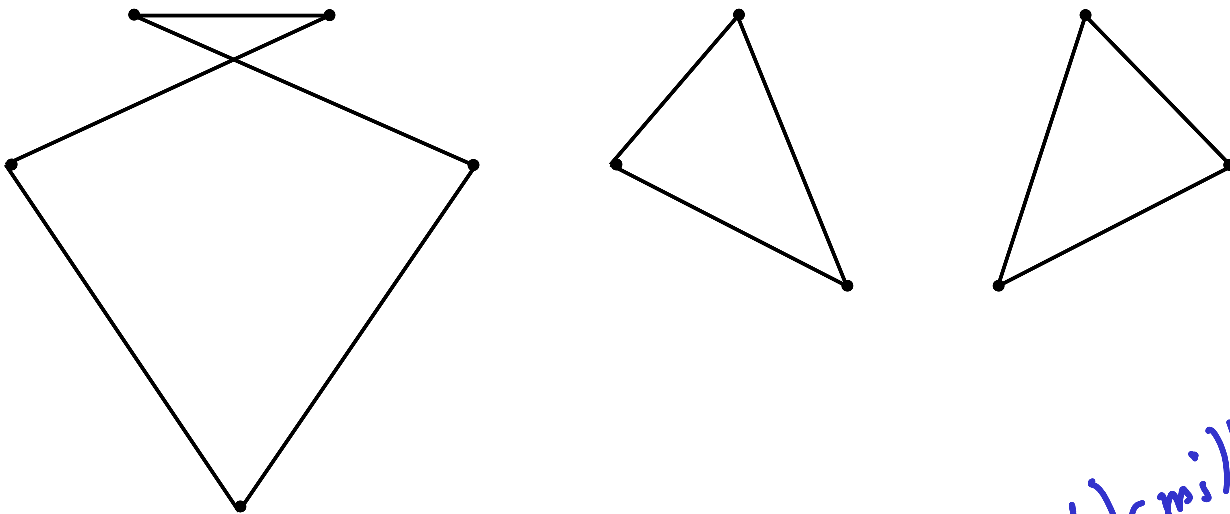


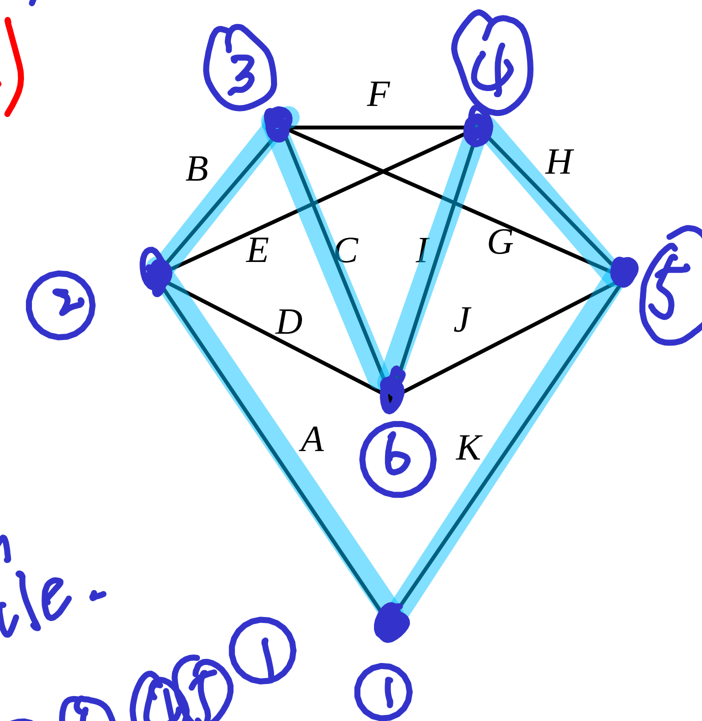
Every vertex of this graph has an even degree. Therefore, this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle.

https://en.wikipedia.org/wiki/Eulerian_path

Edge Disjoint Cycle Decomposition

H. cycle (O)
E cycle (X)



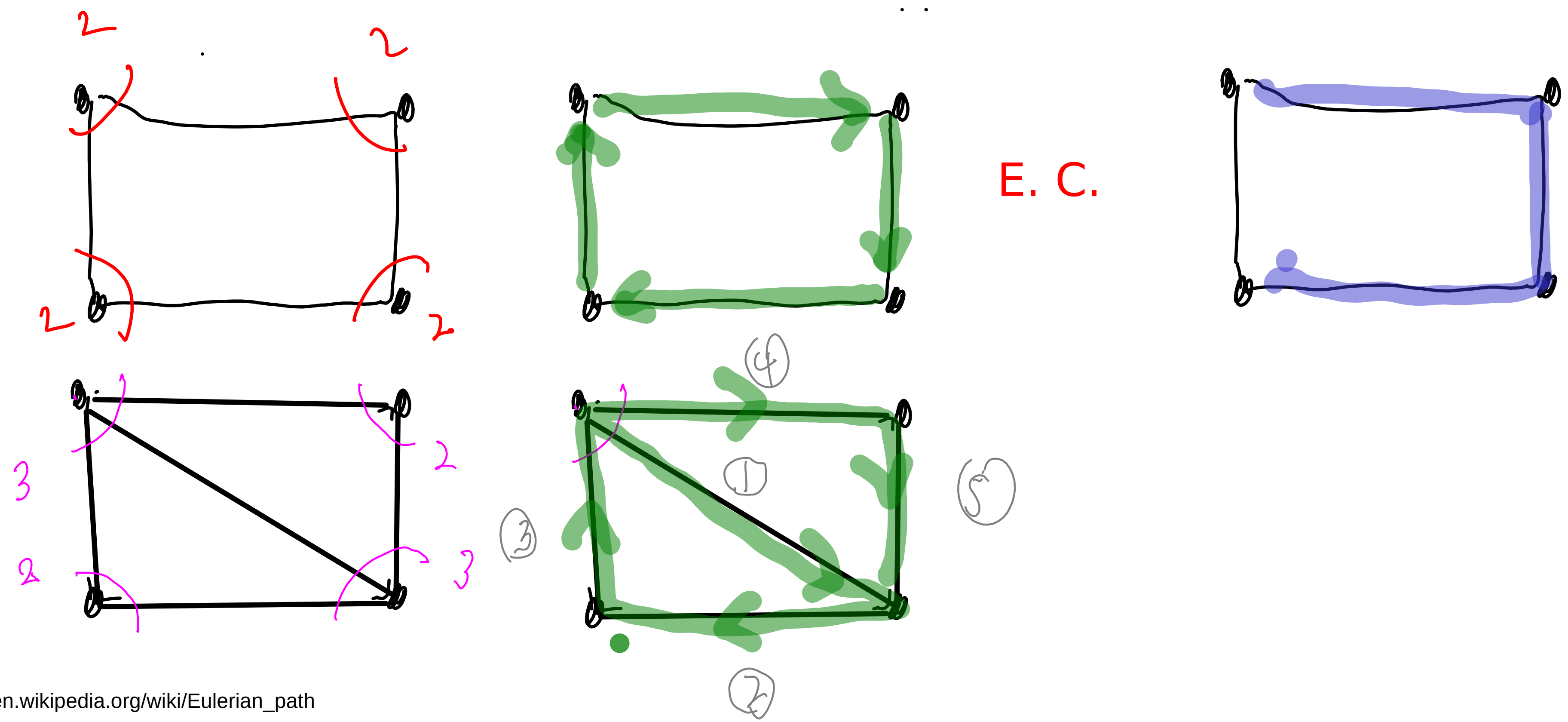


Hamilton cycle.

1 2 3 6 4 5 1

Eulerian Paths of Undirected Graphs

An undirected graph has an **Eulerian path**
if and only if exactly **zero** or **two vertices** have **odd degree**,
and all of its vertices with nonzero degree
belong to a single connected component.



https://en.wikipedia.org/wiki/Eulerian_path

Eulerian Cycles of Directed Graphs

A directed graph has an **Eulerian <u>cycle</u>**
if and only if every vertex has **equal in degree** and **out degree**,
and all of its vertices with nonzero degree
belong to a single strongly connected component.

Equivalently, a directed graph has an Eulerian cycle
if and only if it can be decomposed
into **edge-disjoint directed cycles**
and all of its vertices with nonzero degree
belong to a single strongly connected component.

https://en.wikipedia.org/wiki/Eulerian_path

Eulerian Paths of Directed Graphs

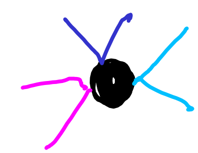
A directed graph has an **Eulerian path**
if and only if **at most one** vertex has (out-degree) − (in-degree) = 1,
at most one vertex has (in-degree) − (out-degree) = 1,
every other vertex has equal in-degree and out-degree,
and all of its vertices with nonzero degree belong to a single connected component of the underlying undirected graph.

https://en.wikipedia.org/wiki/Eulerian_path

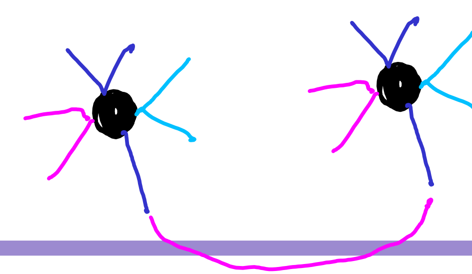
Seven Bridges of Königsberg

The problem was to devise a walk through the city that would cross each of those bridges once and only once.

Euler Cycles (X)
Euler Path (X)



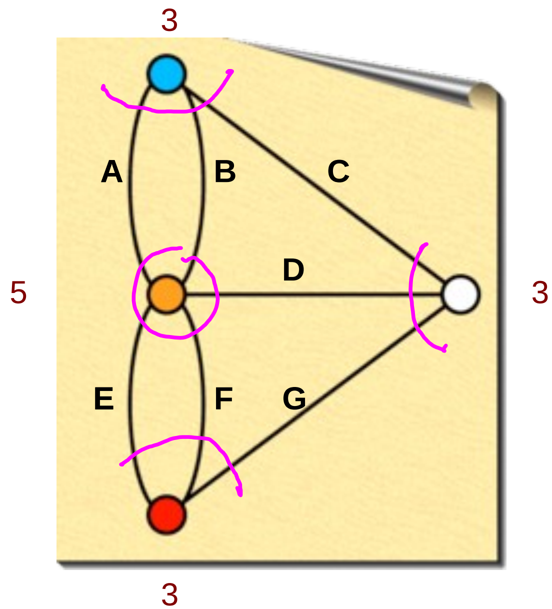
Euler Cycle	All even degree vertices = 0 odd degree vertices
Euler Path	Only 0 or 2 odd degree vertices



https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

Seven Bridges of Königsberg

7Bridges Problem

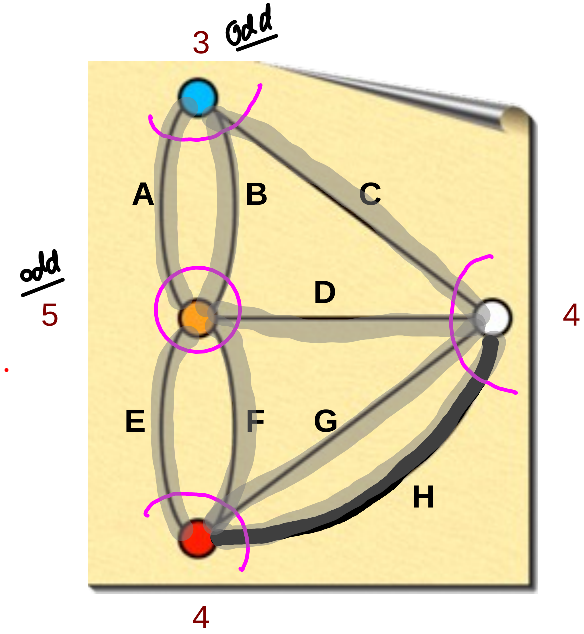


E Cycle (X)
E Path (X)

All - Odd Deg - Vertices

AEHGFDCB

8Bridges Problem



Eulerian Path

AEHGFDCB

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

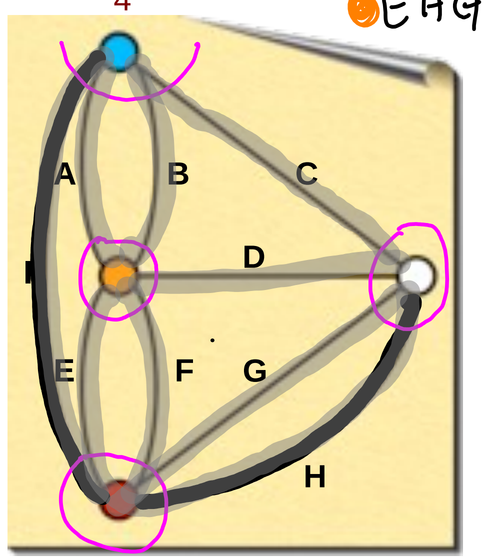
Seven Bridges of Königsberg

9 Bridges Problem

EHGFDCBAI

4

odd
5



4

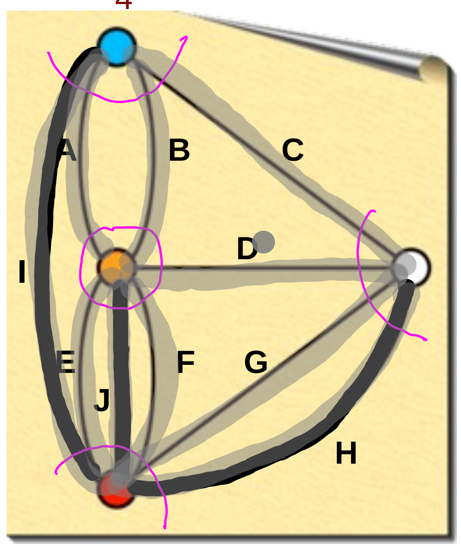
5
odd

AEHGFDCBJI

10 Bridges Problem

4

6



4

6

Eulerian Path

EHGFDCBAI

Eulerian Cycle

AEHGFDCBJI

https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg

References

[1] http://en.wikipedia.org/
[2]

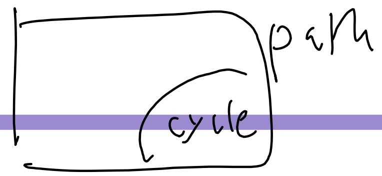
Hamiltonian Cycle (3A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice and Octave.

Hamiltonian Cycles

path | cycle

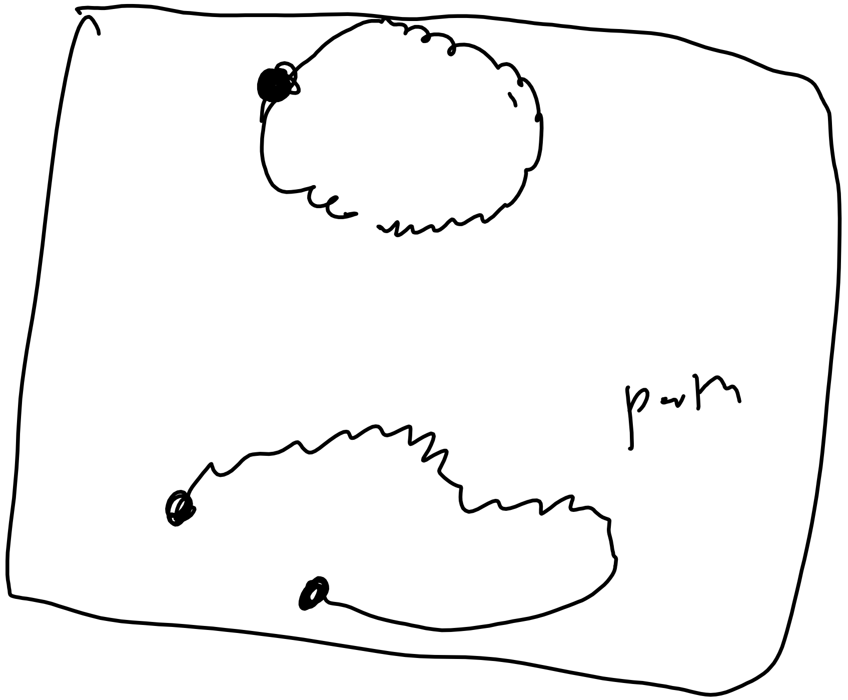


edge

A Hamiltonian path is a path
in an undirected or directed graph
that visits **each vertex** exactly **once**.

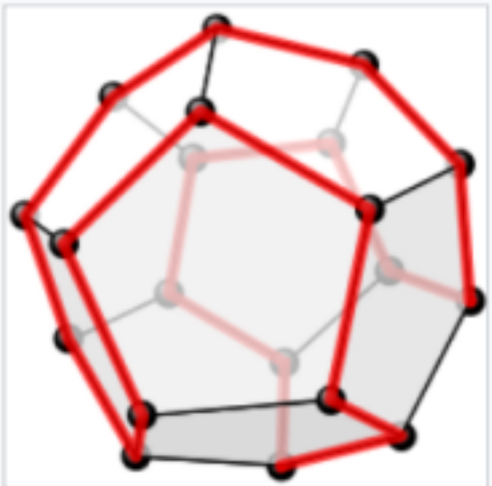
A Hamiltonian cycle is
a Hamiltonian path that is a cycle.

the Hamiltonian path problem is NP-complete.

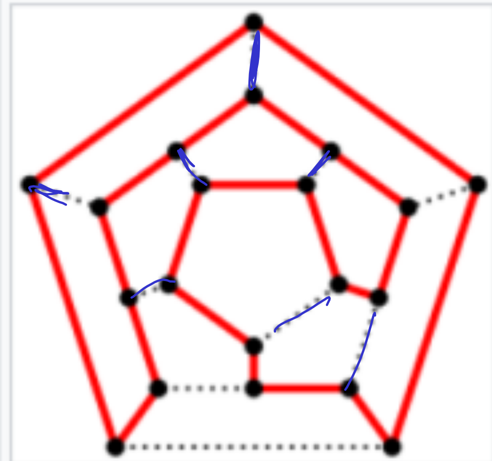


https://en.wikipedia.org/wiki/Hamiltonian_path

Hamiltonian Cycles



One possible Hamiltonian cycle through every vertex of a dodecahedron is shown in red – like all platonic solids, the dodecahedron is Hamiltonian



The above as a two-dimensional planar graph

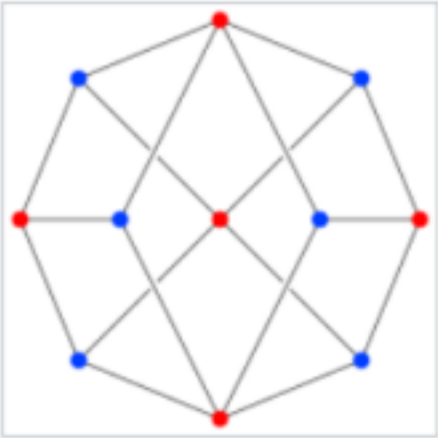
Euler Cycle X

빠진 edge

Hamilton cycle

https://en.wikipedia.org/wiki/Hamiltonian_path

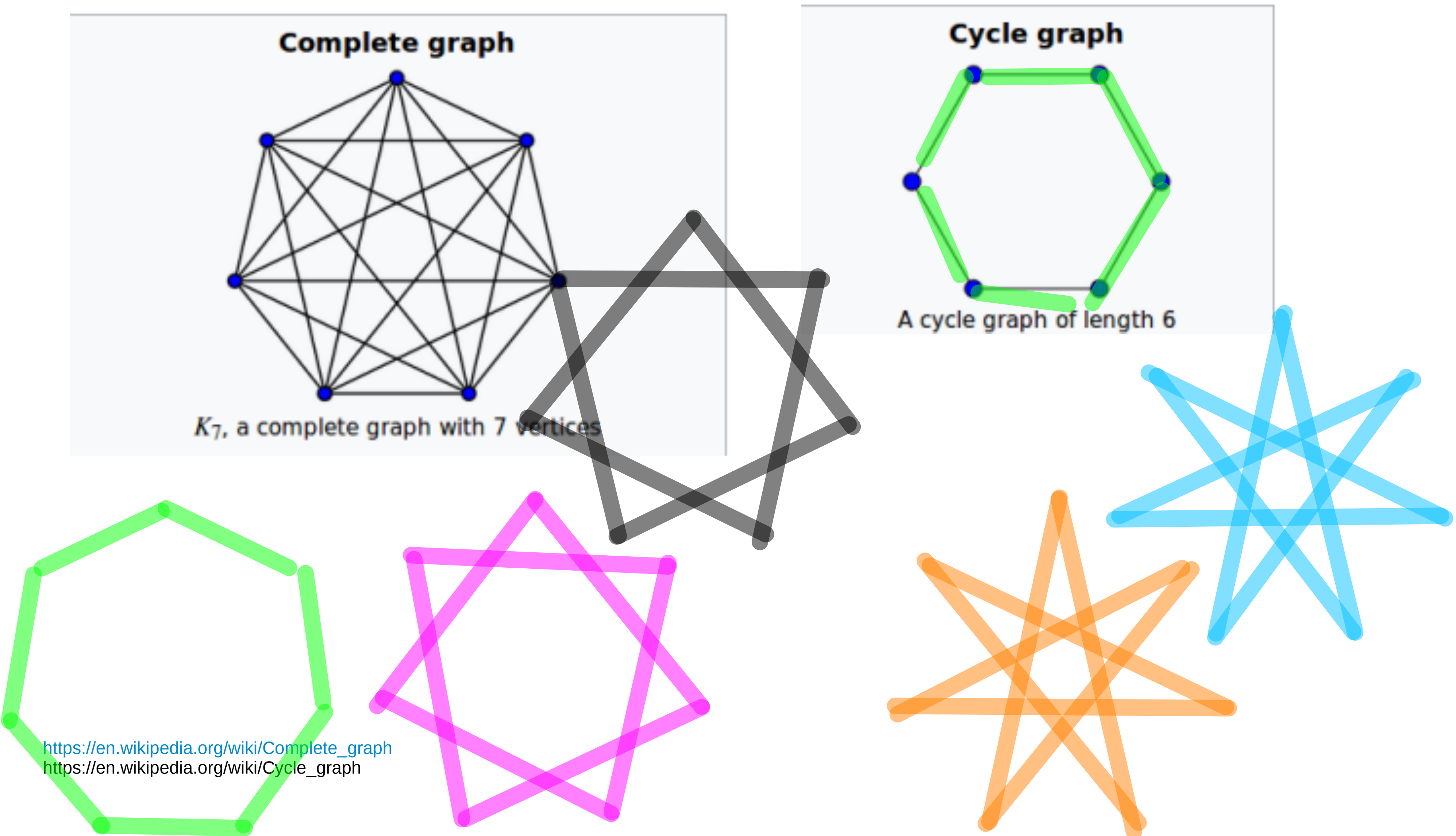
Hamiltonian Cycles



The Herschel graph is the smallest possible polyhedral graph that does not have a Hamiltonian cycle.

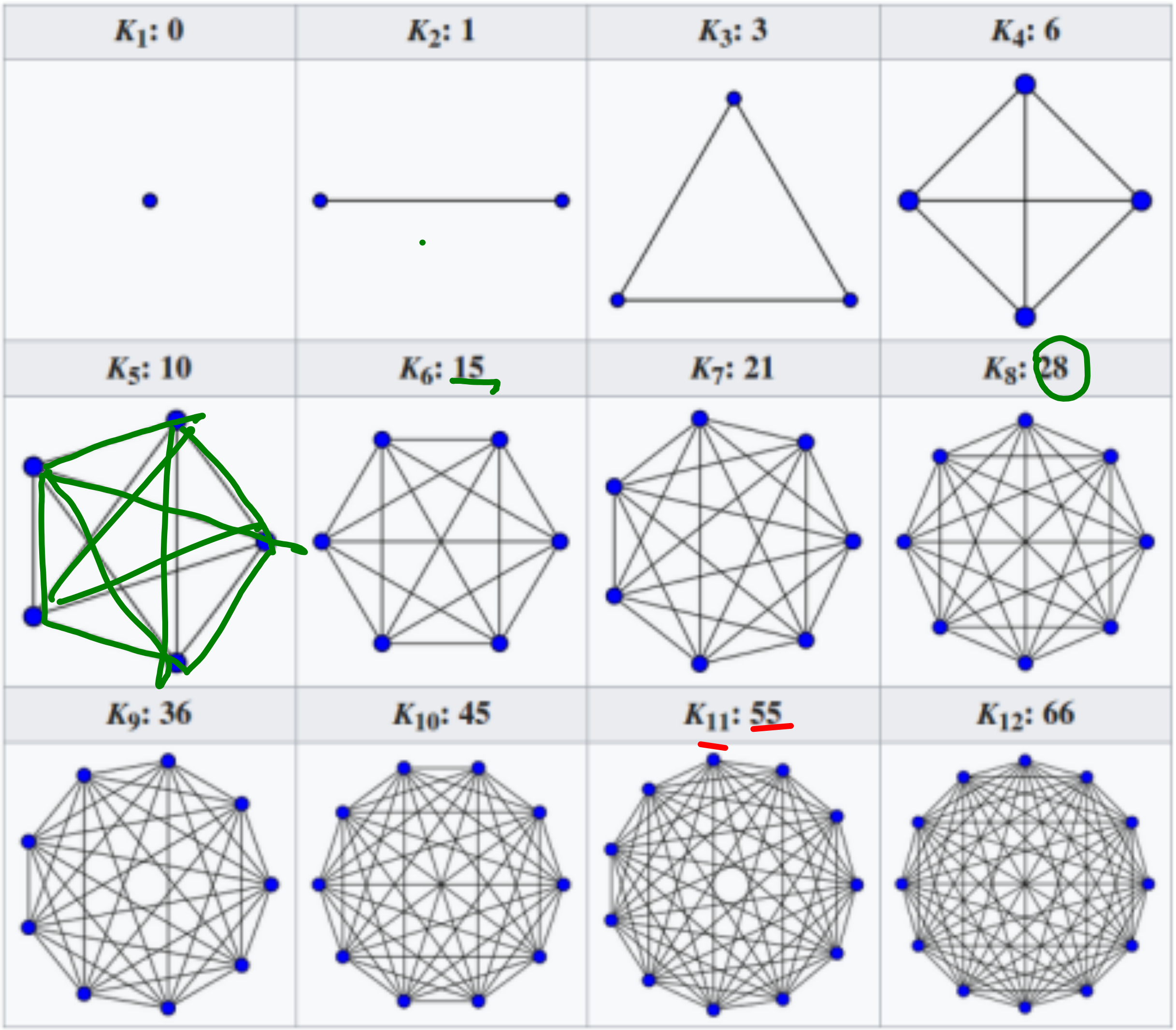
https://en.wikipedia.org/wiki/Hamiltonian_path

Complete Graphs and Cycle Graphs



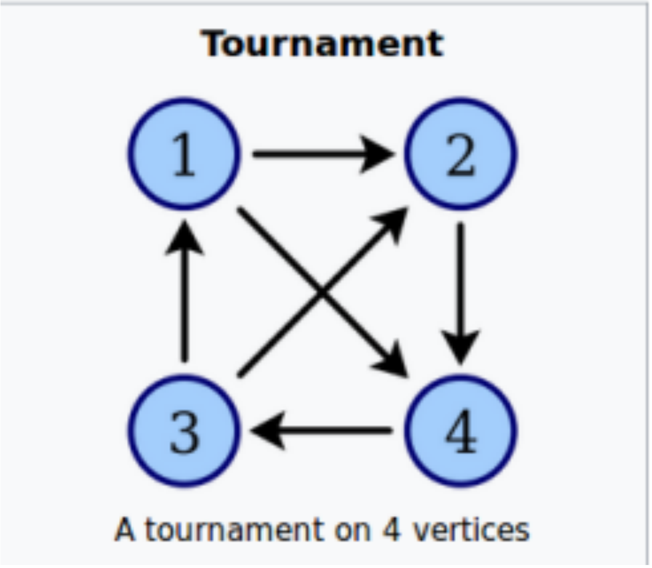
https://en.wikipedia.org/wiki/Complete_graph
https://en.wikipedia.org/wiki/Cycle_graph

Complete Graphs

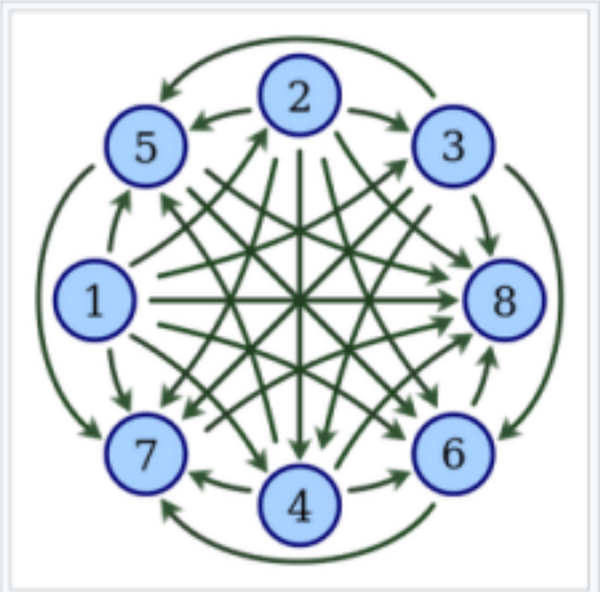


https://en.wikipedia.org/wiki/Complete_graph

Tournament Graphs



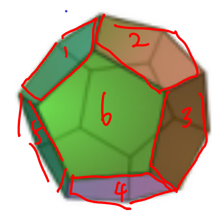
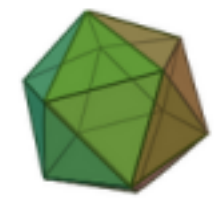
A tournament on 4 vertices



A transitive tournament on 8 vertices.

https://en.wikipedia.org/wiki/Tournament_(graph_theory

Platonic Solid Graphs

Tetrahedron	Cube	Octahedron	Dodecahedron	Icosahedron
Four faces	Six faces	Eight faces	Twelve faces	Twenty faces
				
(Animation)	(Animation)	(Animation)	(Animation)	(Animation)
(3D model)	(3D model)	(3D model)	(3D model)	(3D model)

https://en.wikipedia.org/wiki/Platonic_solid

Hamiltonian Cycles – Properties (1)

$\text{H.C} \;\underset{\not\leftarrow}{\rightarrow}\; \text{H.P}$

Any **Hamiltonian cycle** can be converted
to a **Hamiltonian path** by removing one of its edges,

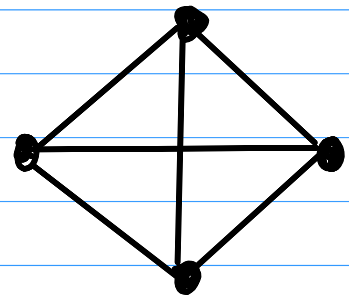
but a **Hamiltonian path** can be extended to
Hamiltonian cycle only if its endpoints are adjacent.

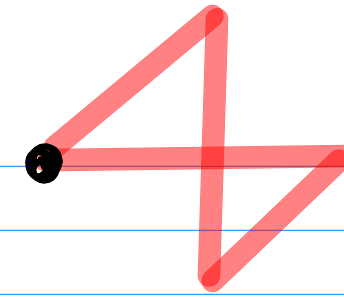
All **Hamiltonian graphs** are **biconnected**, but a
biconnected graph need not be Hamiltonian

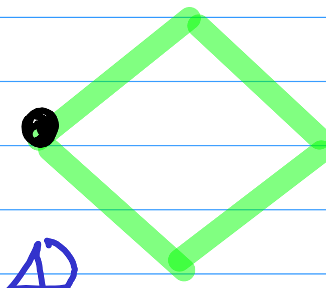
Hamiltonian Graphs $\underset{\not\leftarrow}{\rightarrow}$ Biconnected Graphs

https://en.wikipedia.org/wiki/Hamiltonian_path

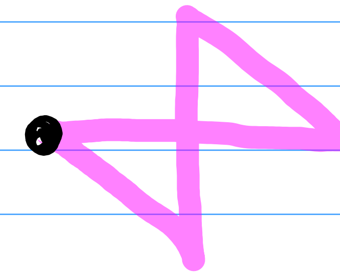
Complete Graph K_4

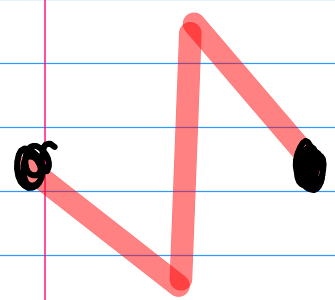


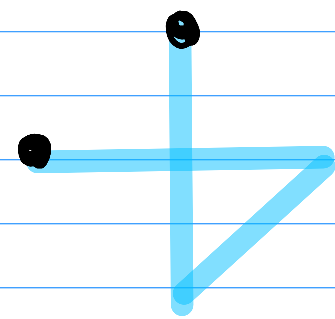


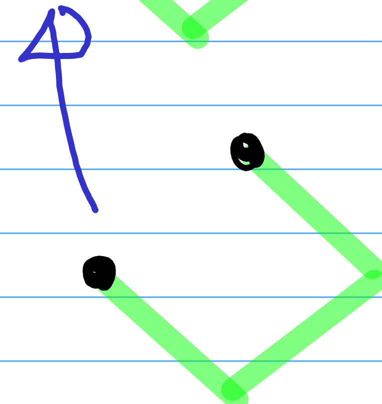


H.C.



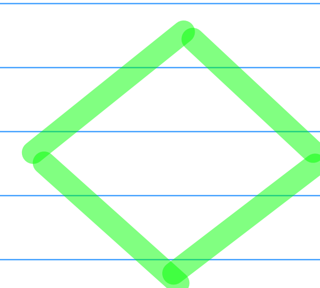




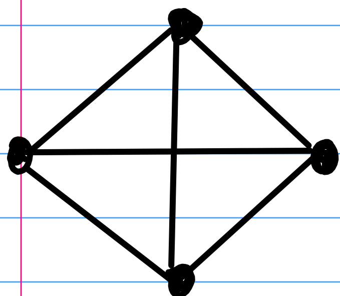


H. P

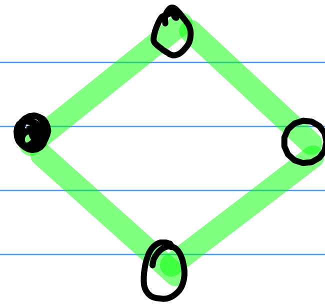
<u>Complete graphs</u>



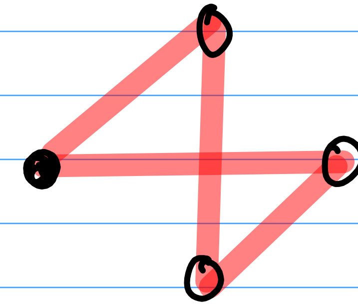
K4



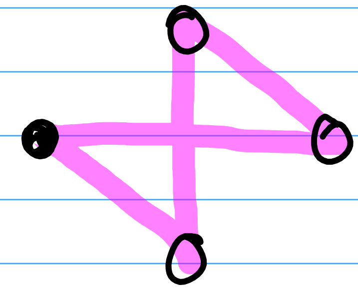
H. C

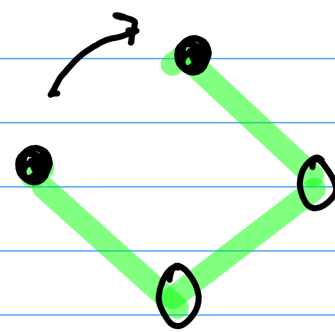


H. C

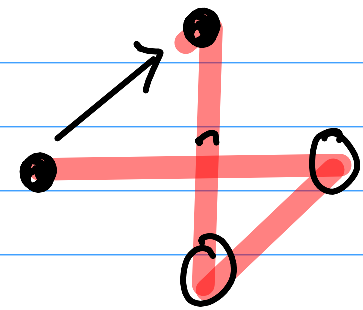


H. C

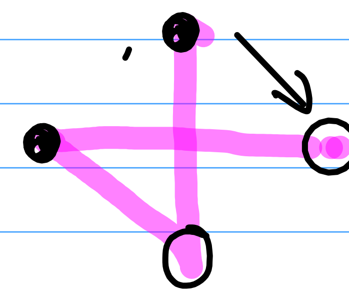




H. P

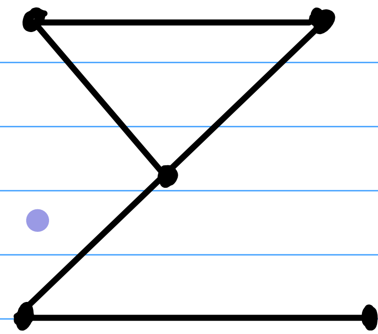


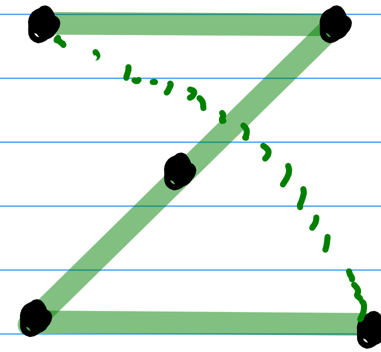
H. P



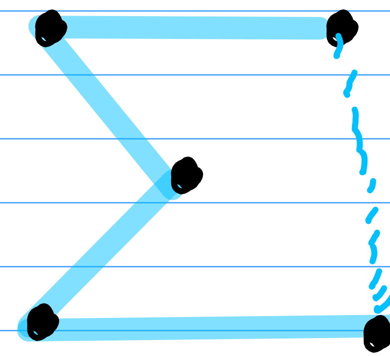
H. P

Hamitonian ~~Cycle~~ or Path





H.P 1



H.P 2

Hamiltonian Cycles – Properties (2)

E - Cycle
edge
H - cycle
vertex

An **Eulerian graph** G :
a **connected** graph in which every **vertex** has **even degree**

An **Eulerian graph** G necessarily has an **Euler path**,
a closed walk passing through each **edge** of G exactly **once**.

This **Eulerian path** corresponds to a **Hamiltonian cycle** in the **line graph** L(G), so the line graph of every **Eulerian graph** is **Hamiltonian**.

Line graphs may have other Hamiltonian cycles that do not correspond to Euler paths.

The **line graph** L(G) of every **Hamiltonian graph** G is itself **Hamiltonian**, regardless of whether the graph G is **Eulerian**.

https://en.wikipedia.org/wiki/Hamiltonian_path

(Dual Graph.)

$G = (V, E)$

$L(G) = (E, V)$

In the mathematical discipline of graph theory, the line graph of an undirected graph G is another graph L(G) that represents the adjacencies between edges of G.

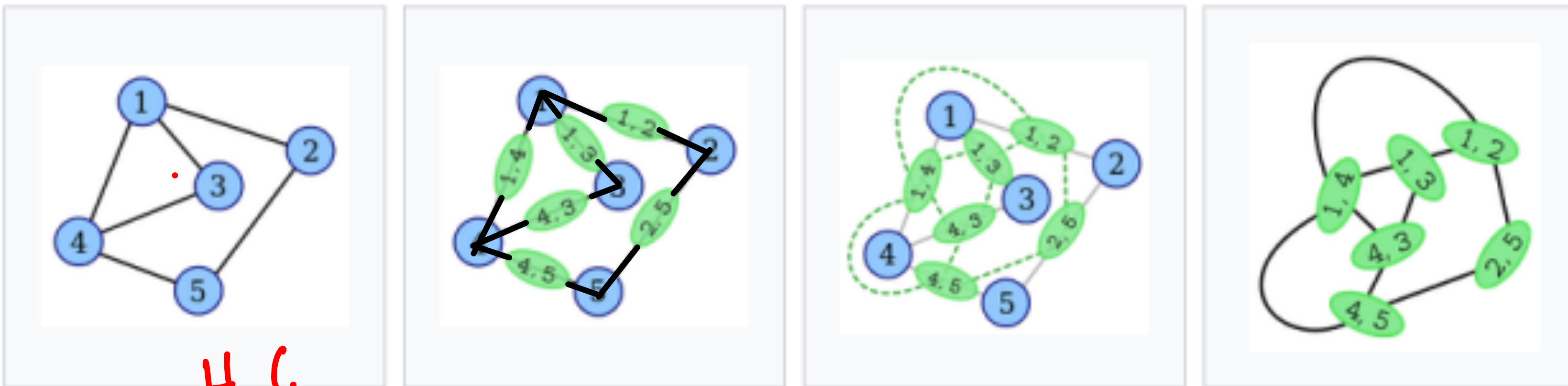
Given a graph G, its line graph L(G) is a graph such that

- each **vertex** of L(G) represents an **edge** of G; and
- two vertices of L(G) are **adjacent** if and only if their corresponding edges share a **common endpoint** ("are incident") in G.

That is, it is the **intersection graph** of the edges of G, representing each edge by the set of its two endpoints.

https://en.wikipedia.org/wiki/Line_graph

Line Graphs Examples



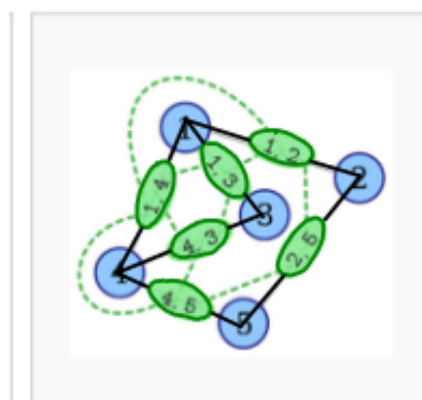
H.C.

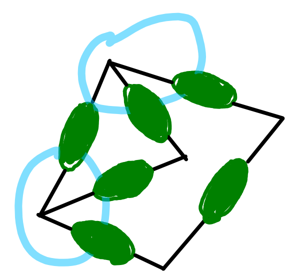
Graph G

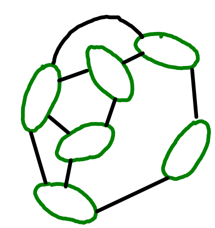
Vertices in L(G) constructed from edges in G

Added edges in L(G)

The line graph L(G)







E. C

https://en.wikipedia.org/wiki/Line_graph

References

[1] http://en.wikipedia.org/
[2]

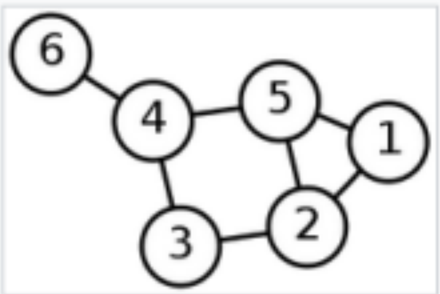
Shortest Path Problem (4A)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

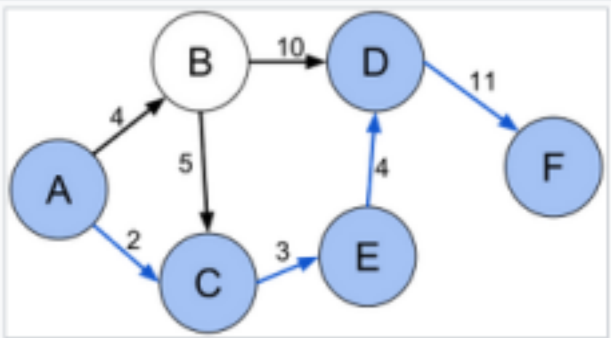
This document was produced by using LibreOffice and Octave.

Shortest Path Problem

the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.



(6, 4, 5, 1) and (6, 4, 3, 2, 1) are both paths between vertices 6 and 1



Shortest path (A, C, E, D, F) between vertices A and F in the weighted directed graph

https://en.wikipedia.org/wiki/Shortest_path_problem

Types of Shortest Path Problems

The **single-pair shortest path problem:**
to find shortest paths from a **source** vertex v to a **destination** vertex w in a graph

The **single-source shortest path problem:**
to find shortest paths from a **source** vertex v to **all** other vertices in the graph.

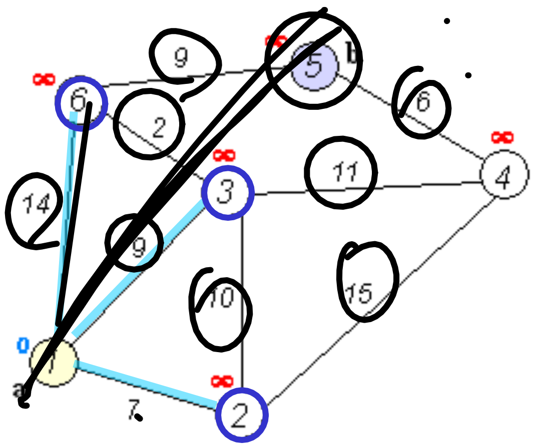
The **single-destination shortest path problem:**
to find shortest paths from **all** vertices in the directed graph to a single **destination** vertex v. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.

The **all-pairs shortest path problem:**
to find shortest paths between every **pair** of vertices v, v' in the graph.

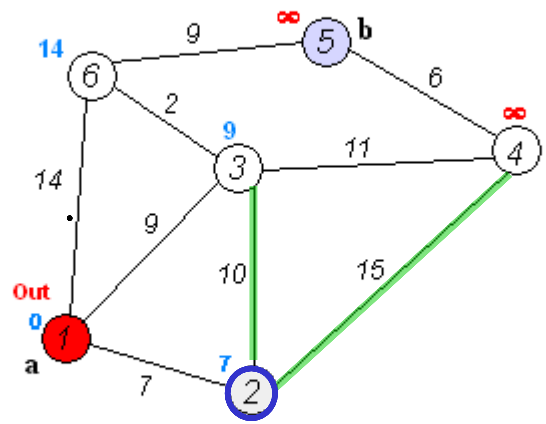
https://en.wikipedia.org/wiki/Shortest_path_problem

Dijkstra's Algorithm Example Summary

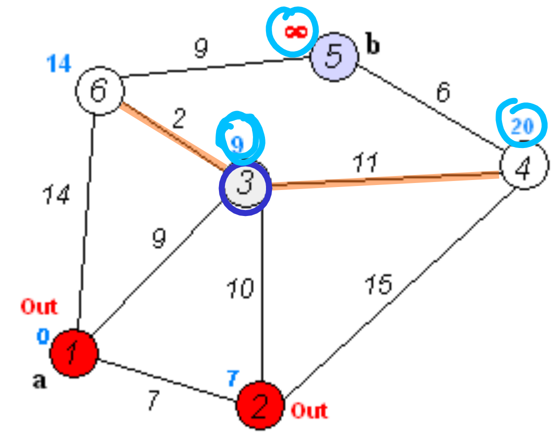
U={1,2,3,4,5,6,7,8}

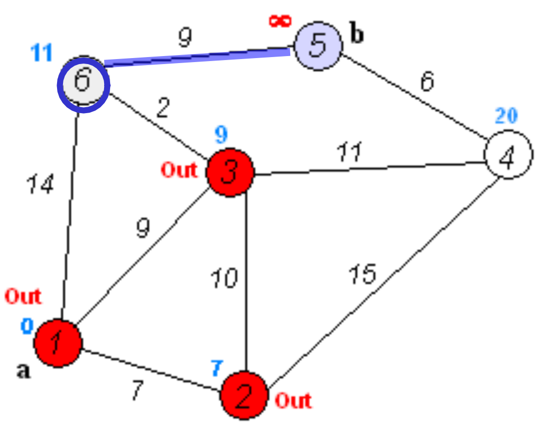


U={2,3,4,5,6,7,8}



U={3,4,5,6,7,8}

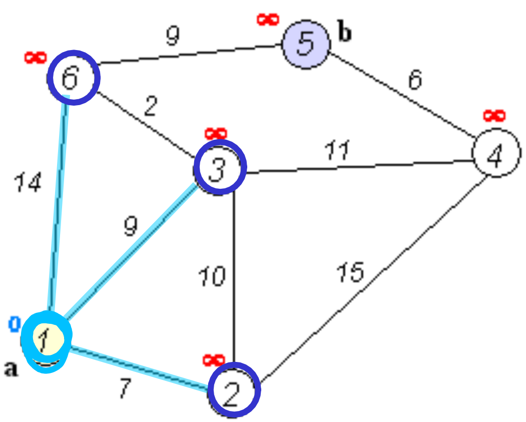




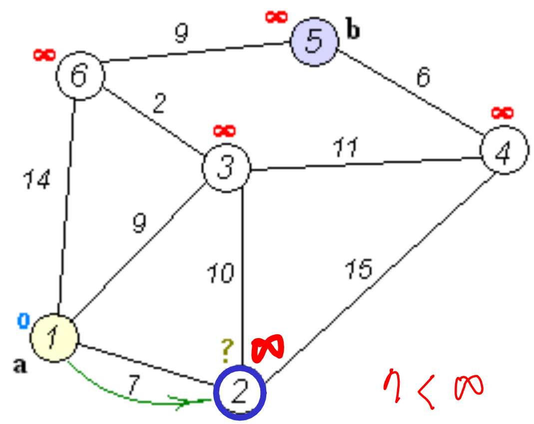
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

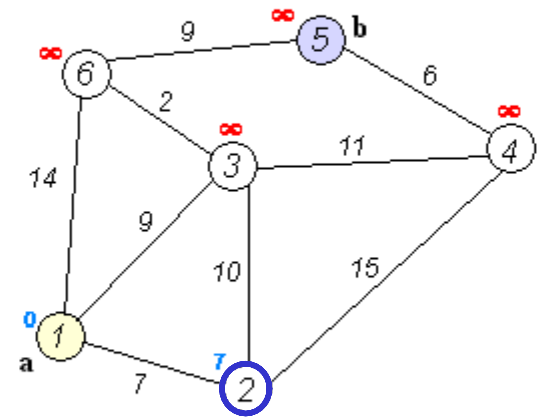
U={4,5,6,7,8}

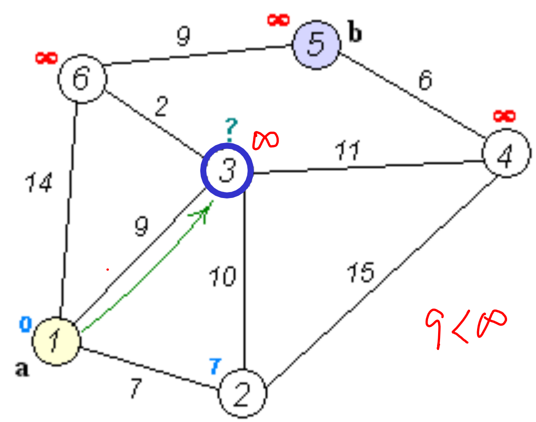
Dijkstra's Algorithm Example (1)

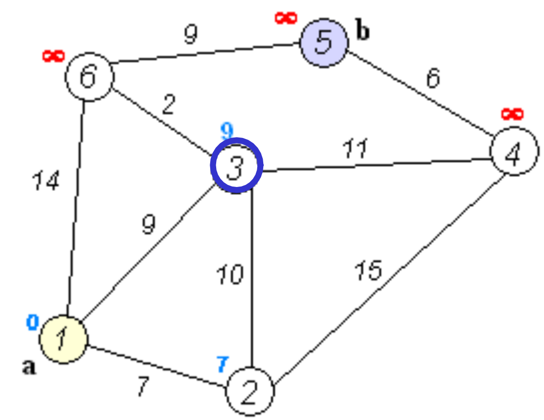


U={1,2,3,4,5,6}
C= 1
N={2,3,4}





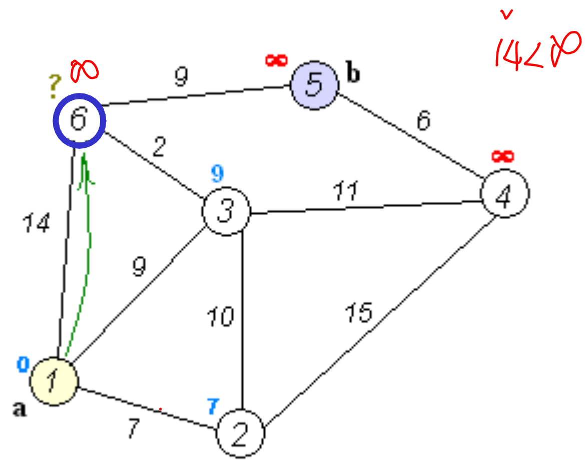


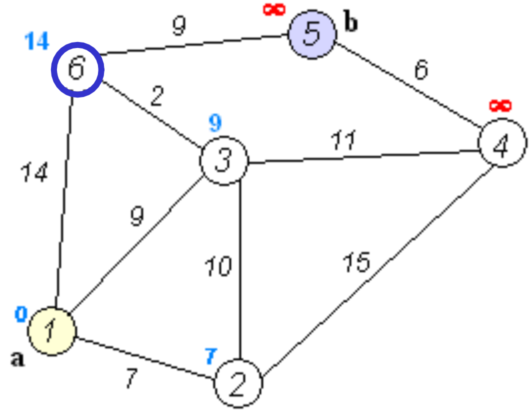


https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

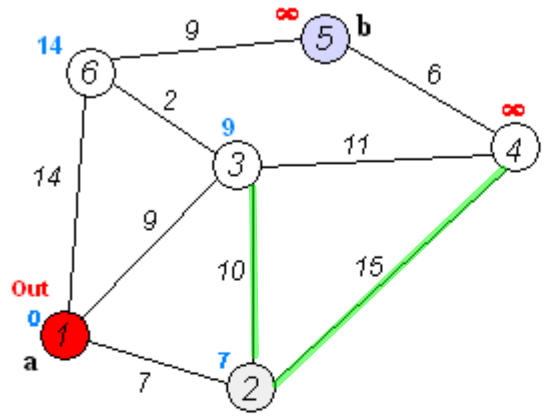
Dijkstra's Algorithm Example (2)

U={1,2,3,4,5,6}
C= 1
N={2,3,4}



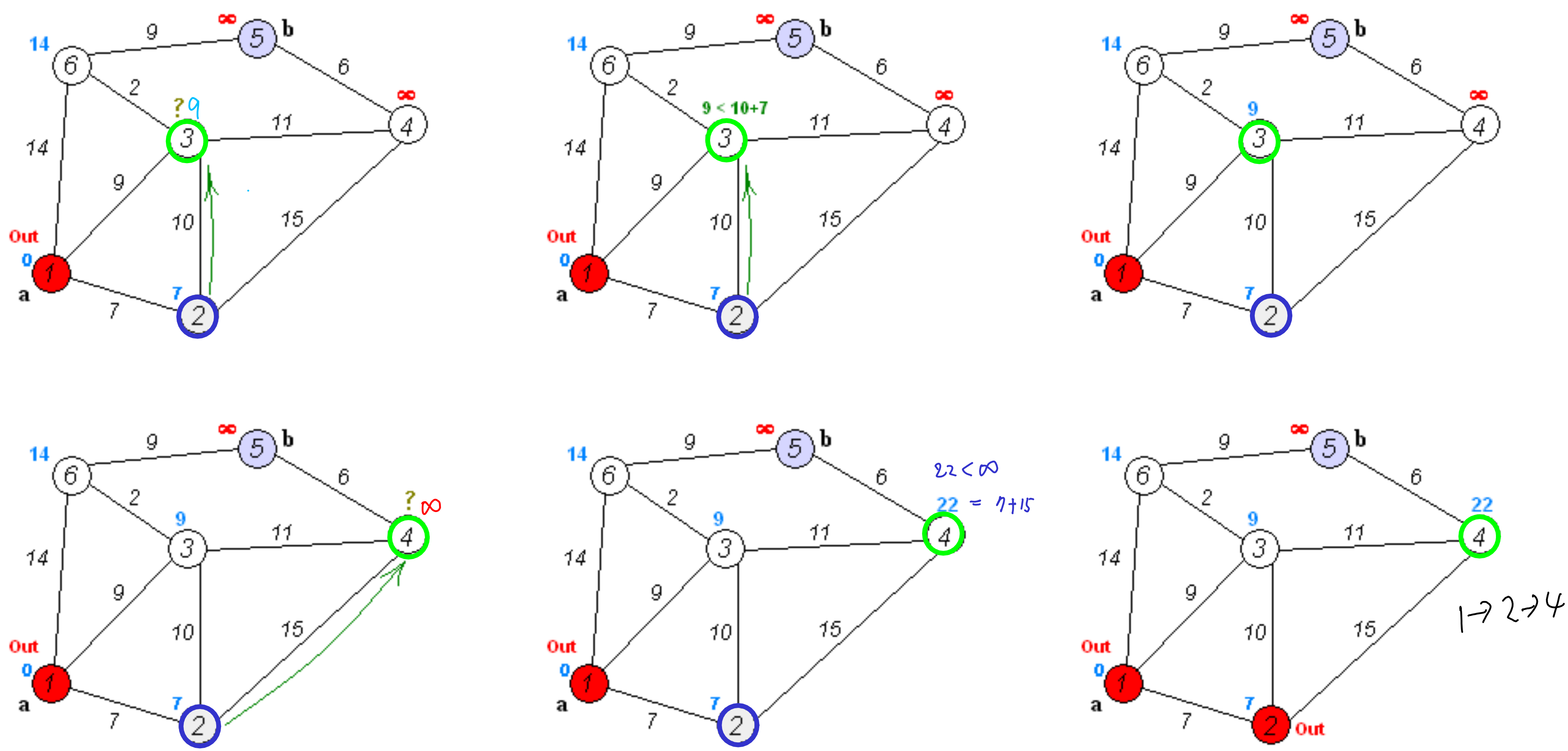


7 9 ∞ ∞ 14
U={2,3,4,5,6}
C=2 (min=7)
N={3,4}



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

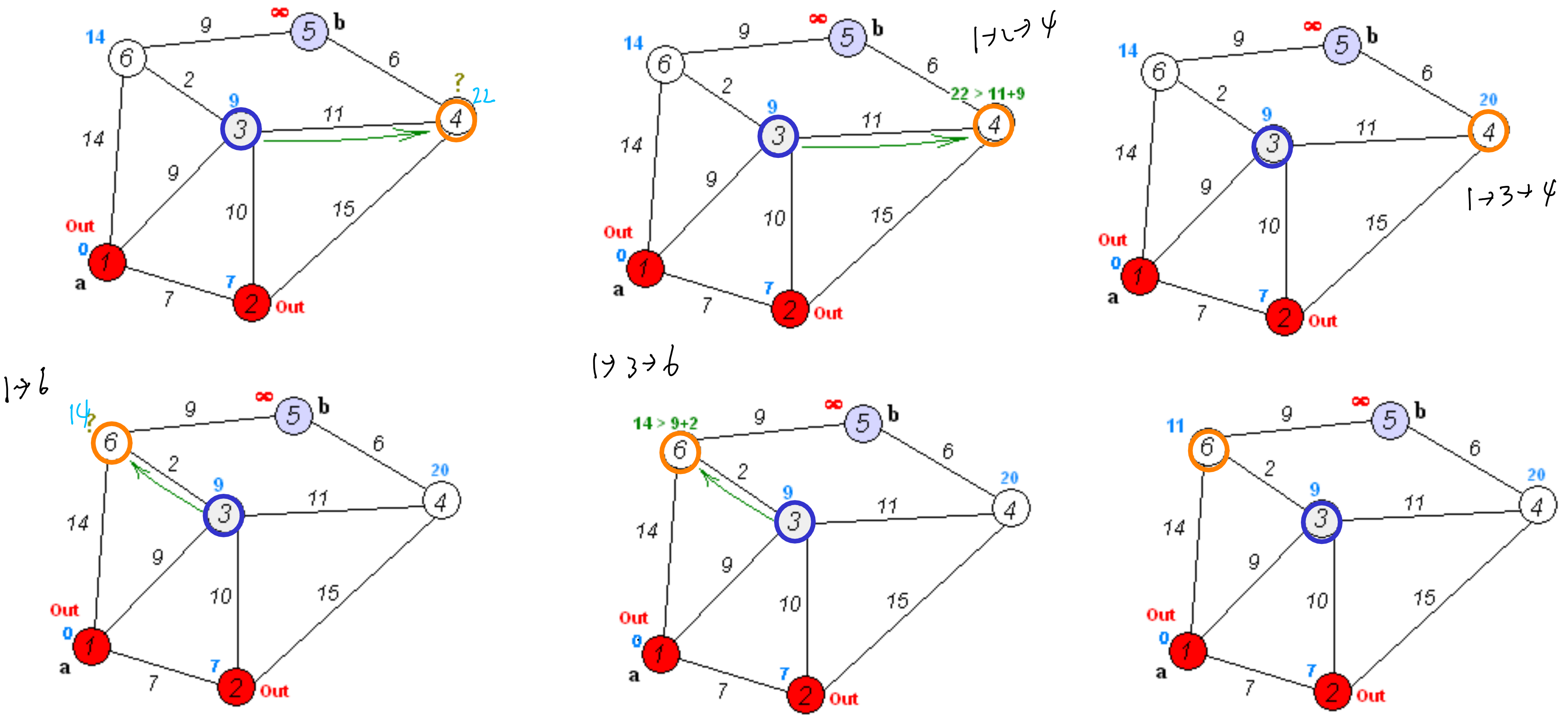
Dijkstra's Algorithm Example (3)



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

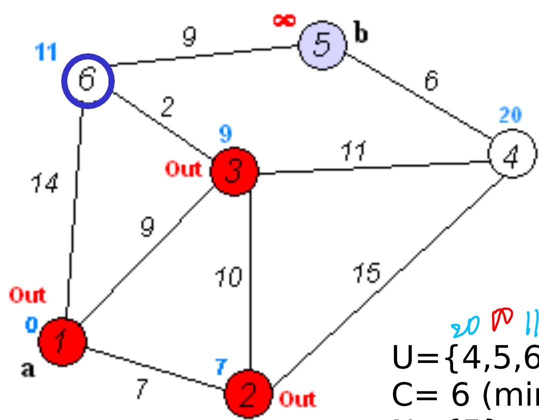
9 22 ∞ 14
U={3,4,5,6}
C= 3 (min=9)
N={4,6}

Dijkstra's Algorithm Example (4)

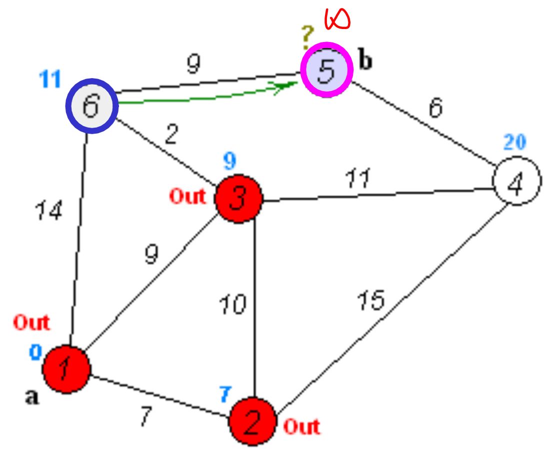


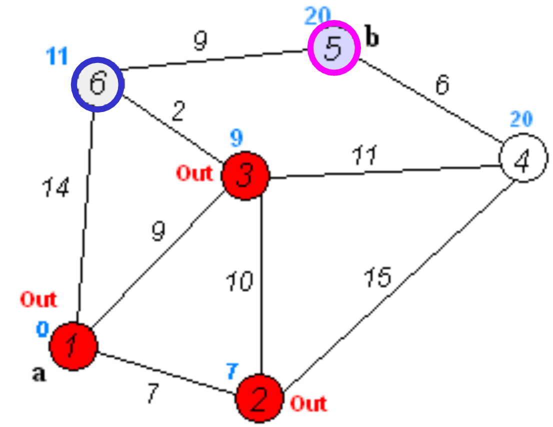
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

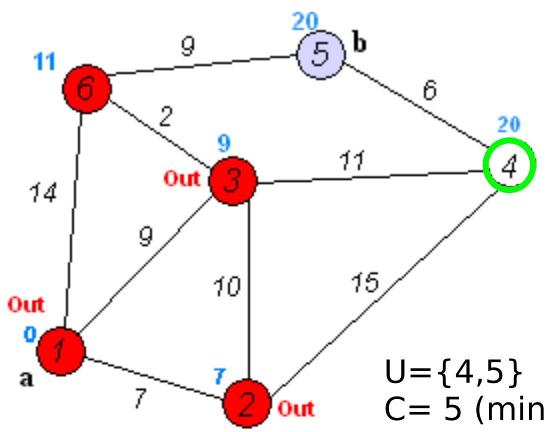
Dijkstra’s Algorithm Example (5)



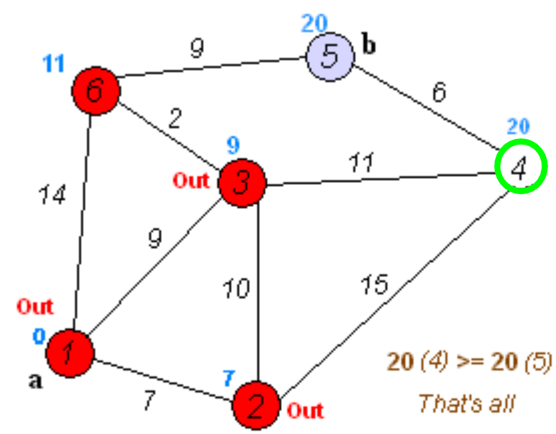
20 ∞ 11
U={4,5,6}
C= 6 (min=11)
N={5}





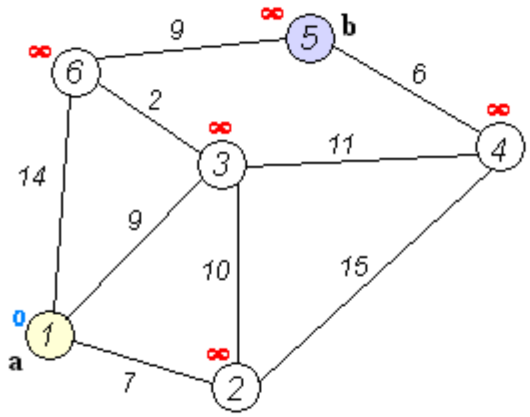


U={4,5}
C= 5 (min=20)
N={4}



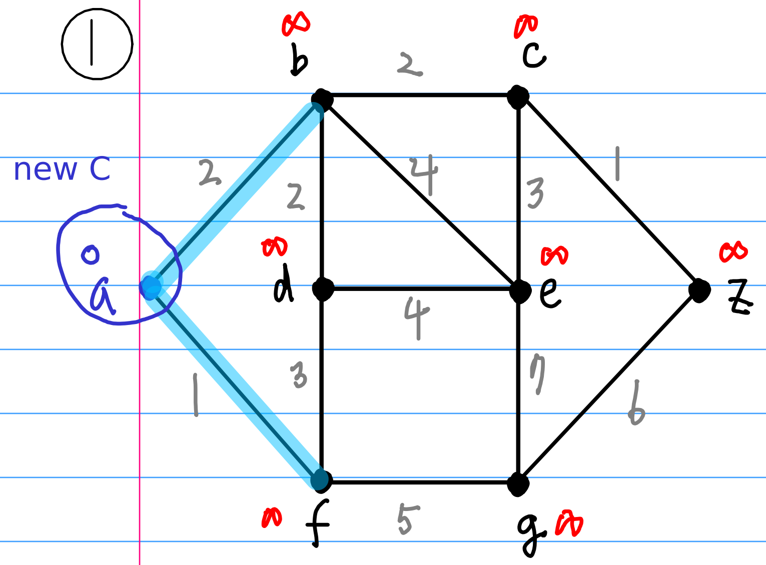
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Hamiltonian Cycles



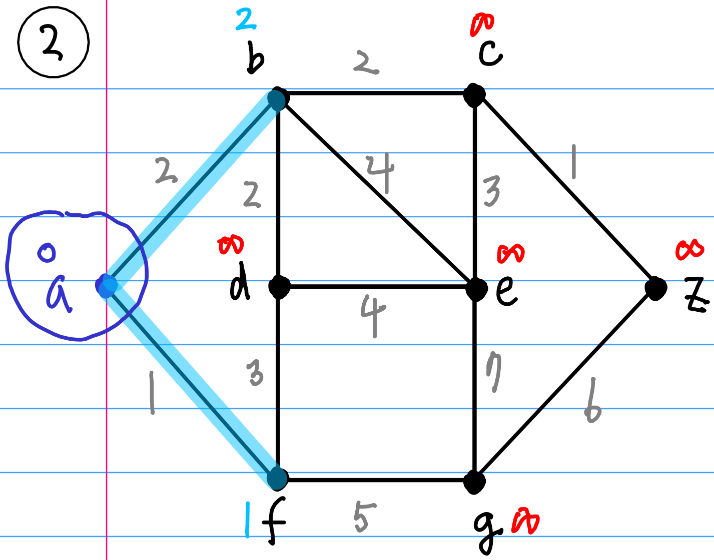
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

①



a → z shortest path?

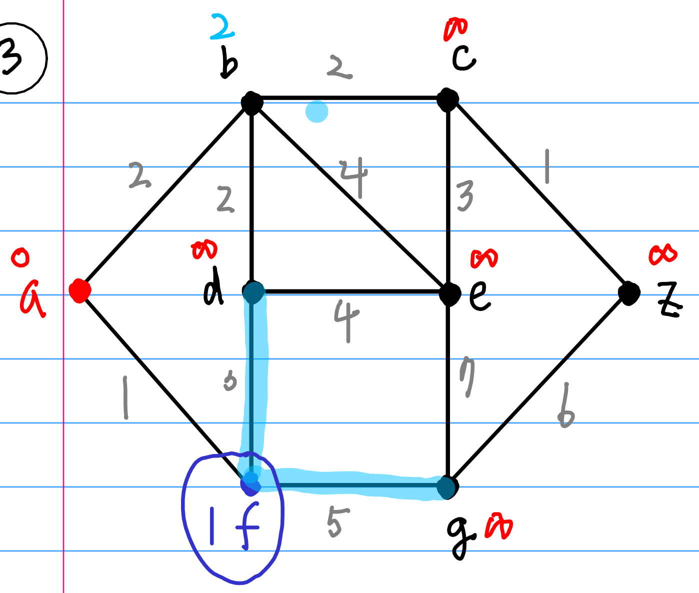
U={a,b,c,d,e,f,g,z}
C=a (min=0)
N={b,f}

② 

$a \to z$ shortest path?

U={a,b,c,d,e,f,g,z}
C=a (min=0)
N={b,f}

③



1 2 ∞ ∞ ∞ ∞ ∞

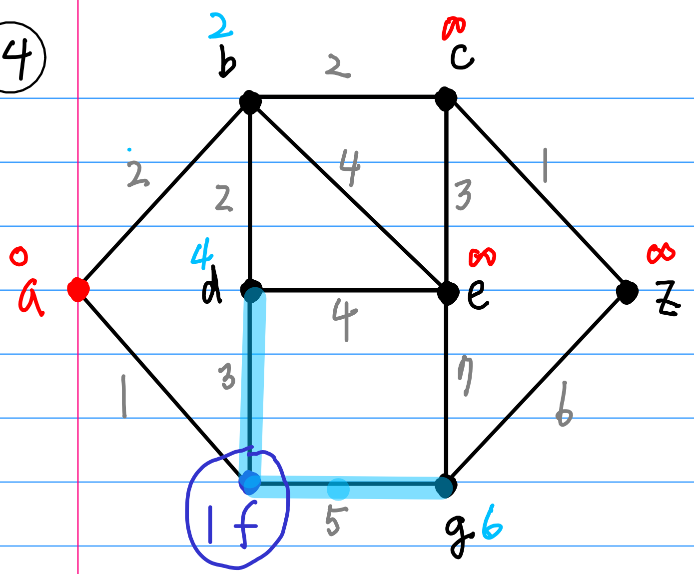
U={f,b,c,d,e,g,z}

C=f (min=1)

N={d,g}

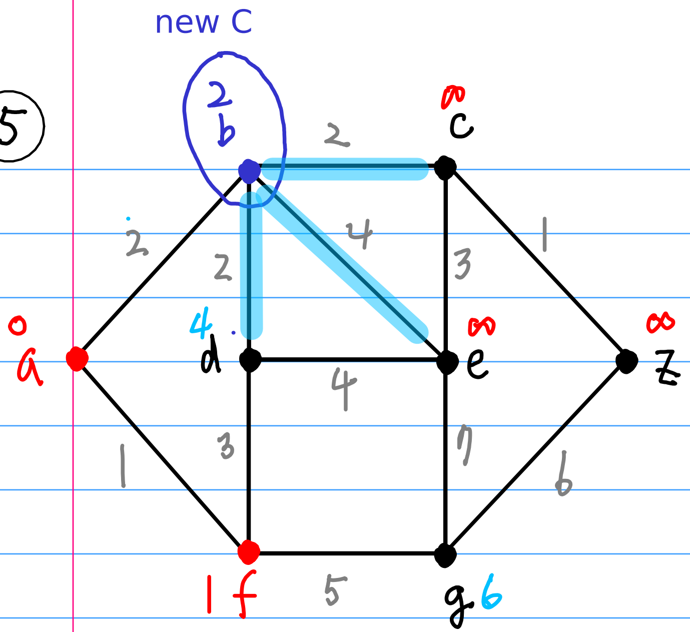
new C

④



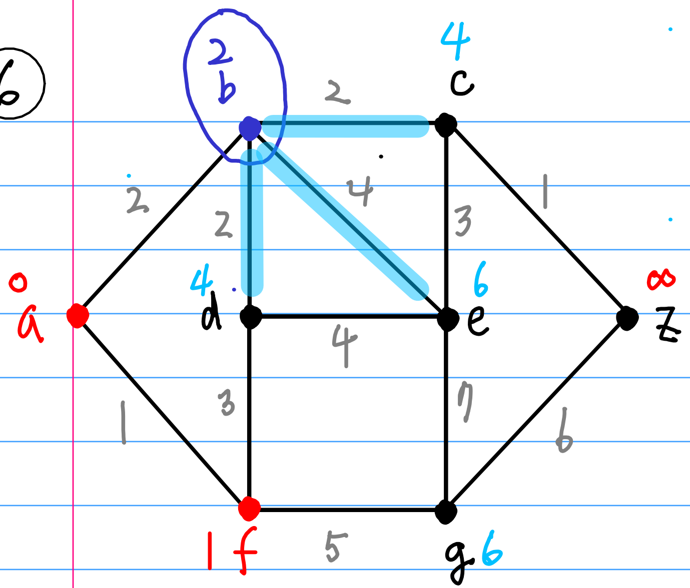
U={f,b,c,d,e,g,z}
C=f (min=1)
N={d,g}

⑤

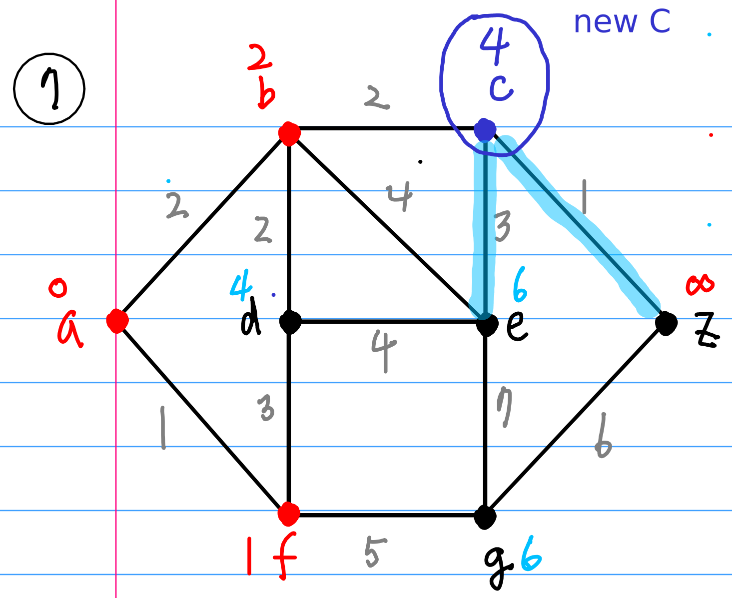


3 ∞ 4 ∞ 6 ∞
U={b,c,d,e,g,z}
C=b (min=2)
N={c,d,e}

⑥



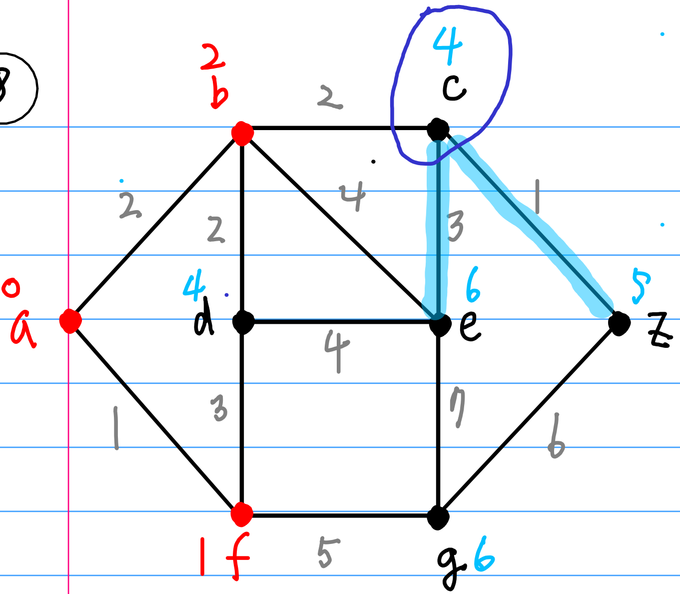
U={b,c,d,e,g,z}
C=b (min=2)
N={c,d,e}



4 4 6 6 ∞

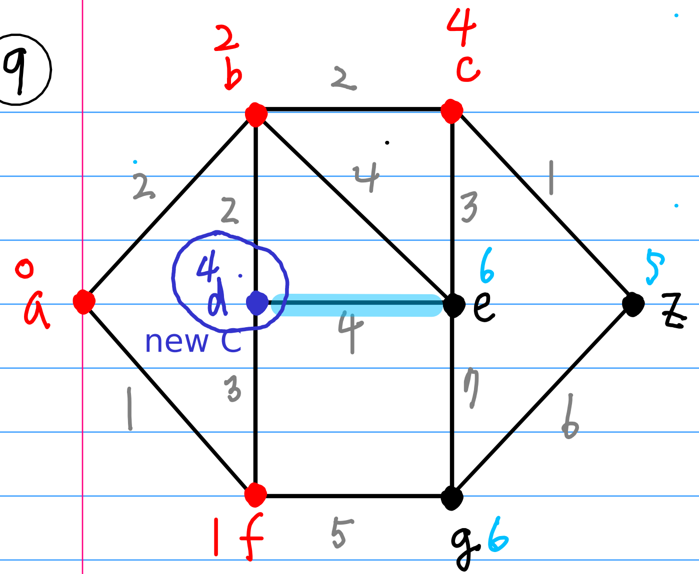
U={c,d,e,g,z}
C=c (min=4)
N={e,z}

8



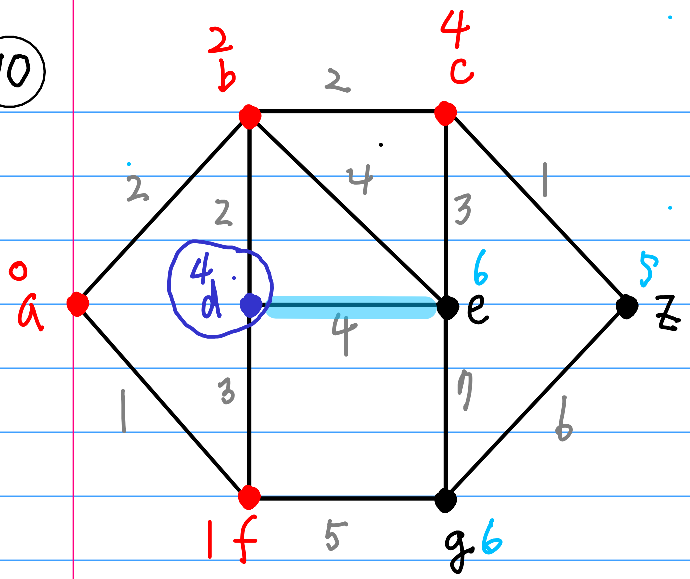
U={c,d,e,g,z}
C=c (min=4)
N={e,z}

⑨



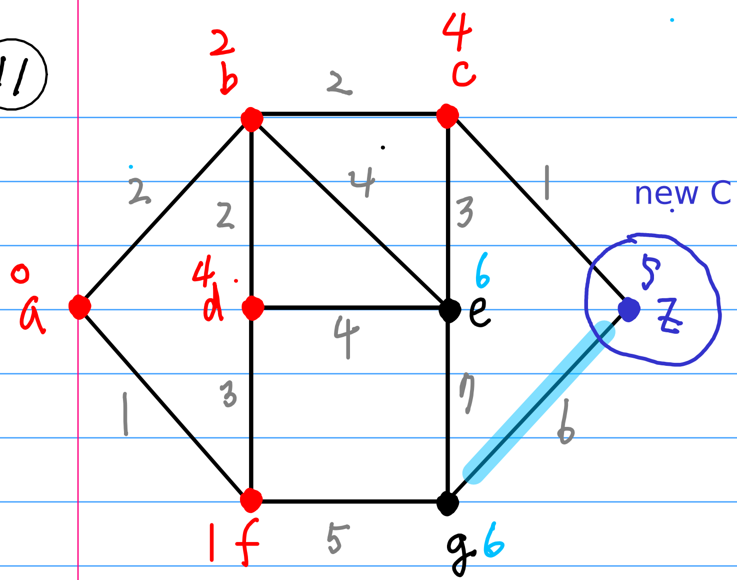
4 6 6 5
U={d,e,g,z}
C=d (min=4)
N={e}

10



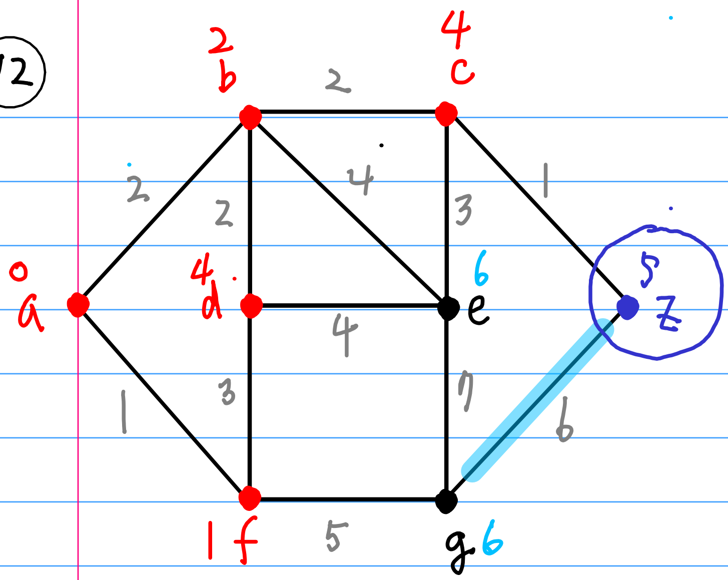
U={d,e,g,z}
C=d (min=4)
N={e}

11



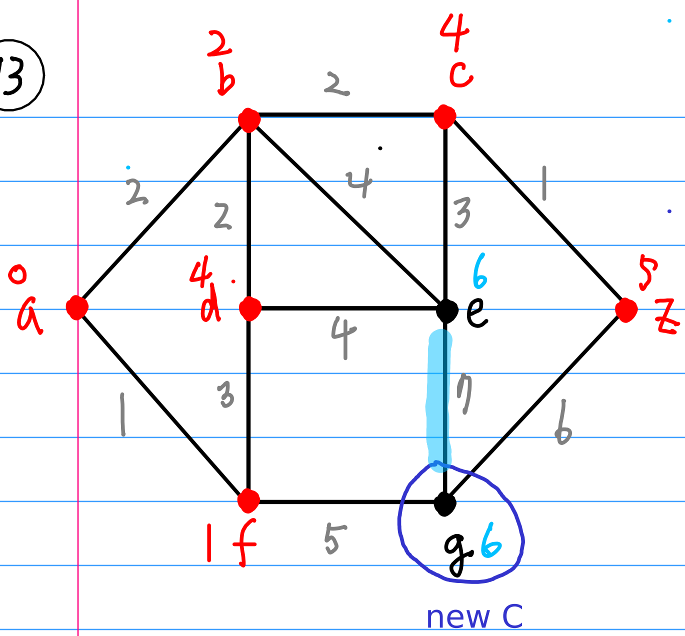
6 6 5
U={e,g,z}
C=z (min=5)
N={g}

12



U={e,g}
C=z (min=5)
N={g}

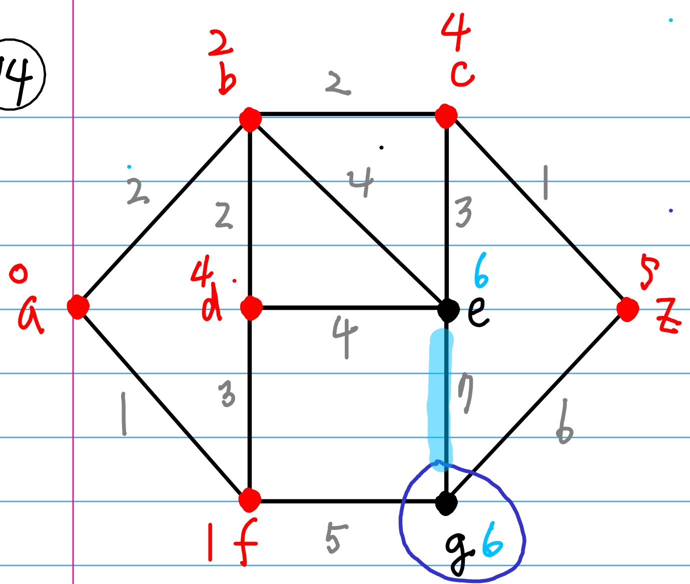
13



6 6
U={e,g}
C=e (min=6)
N={g}

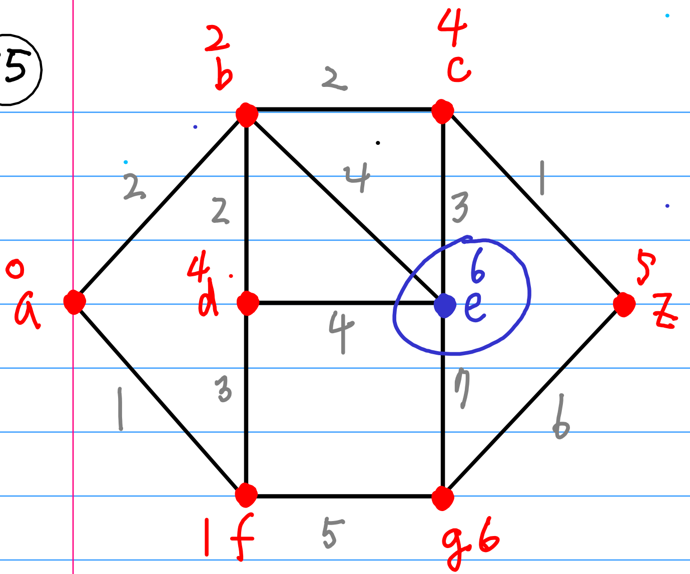
stop!

(14)



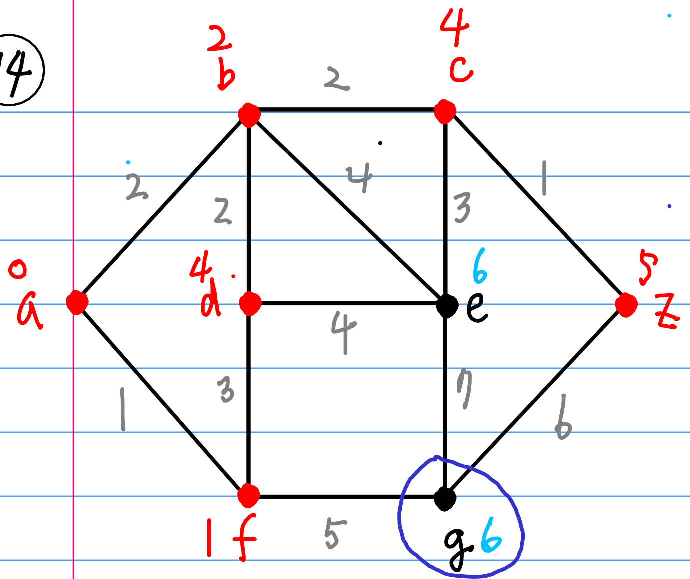
6 6
U={e,g}
C=e (min=6)
N={g}

(15)



6 6
U={e,g}
C=e (min=6)
N={g}

14



6 6
U={e,g}
C=e (min=6)
N={g}

Dijkstra’s Algorithm

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Mark all nodes **unvisited**. Create a set of all the unvisited nodes called the **unvisited set**.

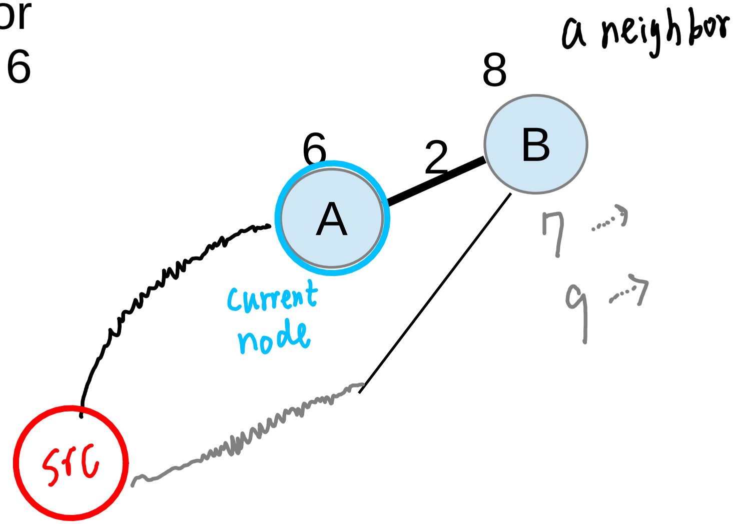
2. Assign to every node a **tentative** distance value: set it to **zero** for our initial node and to **infinity** for all other nodes. Set the initial node as **current**.

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Dijkstra's Algorithm

3. For the current node, consider all of its **unvisited neighbors** and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.

For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.



https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Dijkstra’s Algorithm

4. When we are done considering all of the neighbors of the current node, mark the current node as **visited** and remove it from the the **unvisited set**. A **visited node** will never be checked again.

5. Move to the **next unvisited node** with the smallest tentative distances and repeat the above steps which check neighbors and mark visited.

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Dijkstra’s Algorithm

①

6. If the **destination** node has been marked **visited** (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is **infinity** (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

7. Otherwise, select the **unvisited** node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Dijkstra’s Algorithm

```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:            // Initialization
        dist[v] ← INFINITY                 // Unknown distance from source to v
        prev[v] ← UNDEFINED                // Previous node in optimal path from source
        add v to Q                         // All nodes initially in Q (unvisited nodes)

     dist[source] ← 0                      // Distance from source to source

     while Q is not empty:
        u ← vertex in Q with min dist[u]   // Node with the least distance
                                           // will be selected first
         remove u from Q

         for each neighbor v of u:         // where v is still in Q.
             alt ← dist[u] + length(u, v)
             if alt < dist[v]:             // A shorter path to v has been found
                 dist[v] ← alt
                 prev[v] ← u

     return dist[], prev[]
```

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

Hamiltonian Cycles

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#/media/File:Dijkstra_Animation.gif

References

[1] http://en.wikipedia.org/
[2]