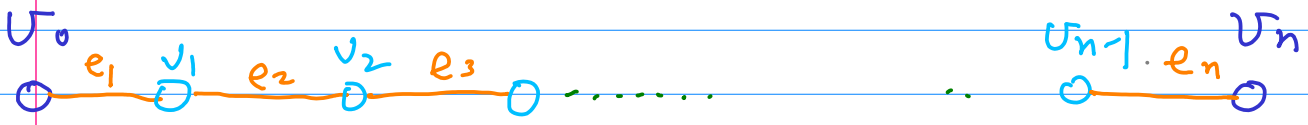
Graph (H1)

20150612

Path

$v_0, \cdots, v_n$: Vertices (정점)



node ○ : $n+1$ 개 교대로 alternatively

edge — : n 개

결합 (association)

e_i 는 v_i 에 결합되어있다

e_i 는 v_{i-1} 에 결합되어있다

Connected Graph

a vertex ⇝ another vertex

aka: a path

Graph (G) for any 2 vertices v, w

if $v \rightarrow w$ a path exists

then Graph G : "connected"

Subgraph of G

$G = (V, E)$

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

꼭지점 집합 (vertex set)

edge 집합

① $V \supseteq V'$

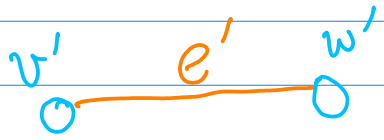
$E \supseteq E'$

② for every edge $e' \in E'$

associated vertices must be included

$v' \in V'$

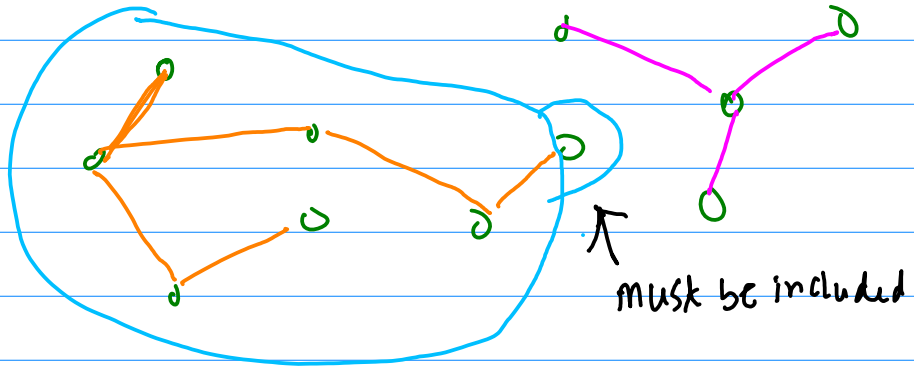
$w' \in V'$



Subgraph G' 부분그래프

if edge e is included in a subgraph G'

vertices v, w that are associated with e must be included in G'



원래 graph $G = (V, E)$ (V: 꼭지점 집합, E: edge 집합)

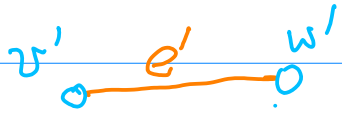
부분 graph $G' = (V', E')$

의 conditions

① $V' \subseteq V$

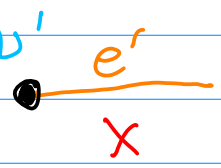
$E' \subseteq E$

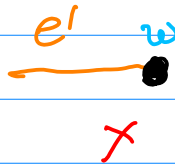
② E' 에 속한 모든 edge e' 에 대해서

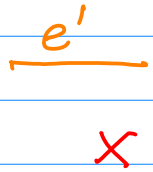


$v' \in V'$

$w' \in V'$

(

X 

X 

X)

$G = (V, E)$

$V = \{v_1, v_2\}$ $\quad V' \rightarrow \phi, \{v_1\}, \{v_2\}, \{v_1, v_2\}$

$E = \{e_1\}$ $\quad E' \rightarrow \phi, \{e_1\}$

$G_1 = (\{v_1\}, \phi)$

$G_2 = (\{v_2\}, \phi)$

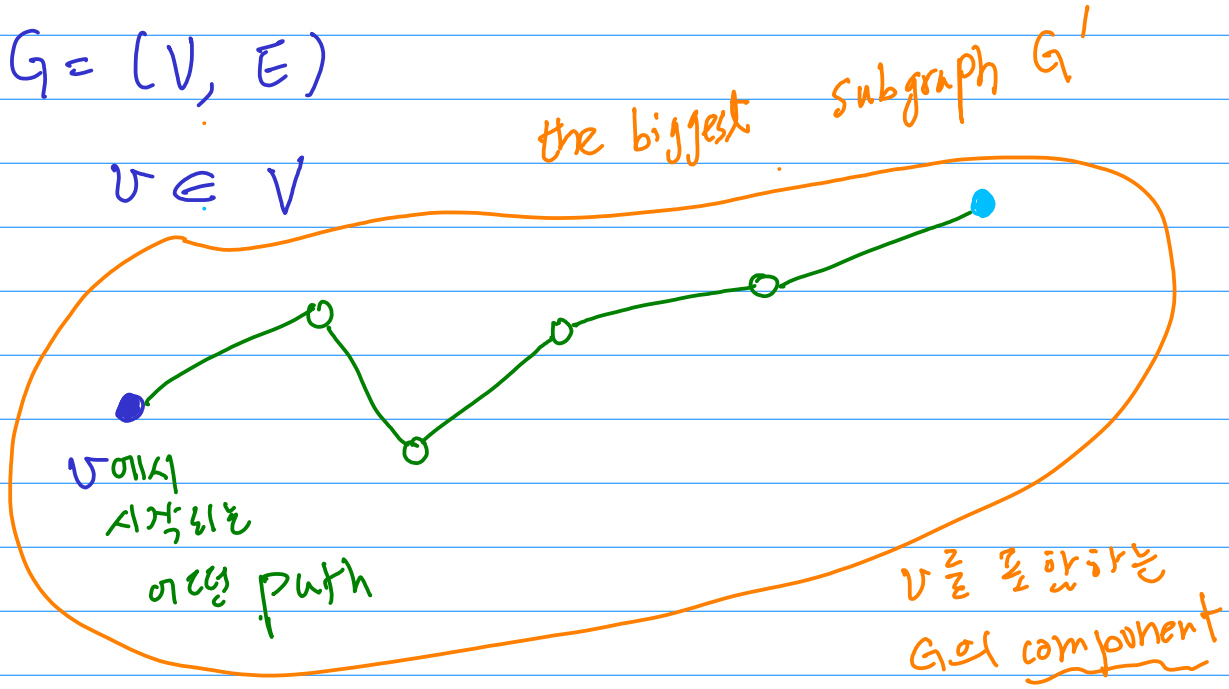
$G_3 = (\{v_1, v_2\}, \phi)$

$G_4 = (\{v_1, v_2\}, e_1)$

정의 8.2.11 G의 component

$G = (V, E)$

$v \in V$



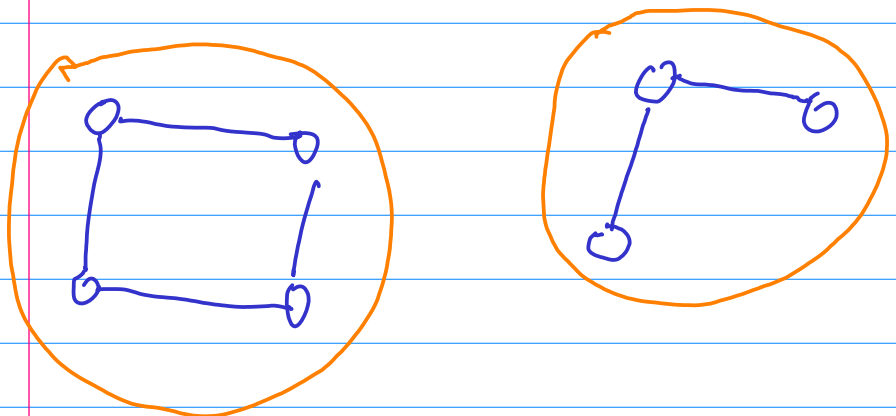
v에서 시작되는 어떤 경로에 포함된
G안의 모든 edge와 vertex로 구성된
G의 부분 graph G' → G의 component

a subgraph in which

any two vertices are connected to each other by paths

and which is connected to no additional vertices in the subgraph

maximally connected

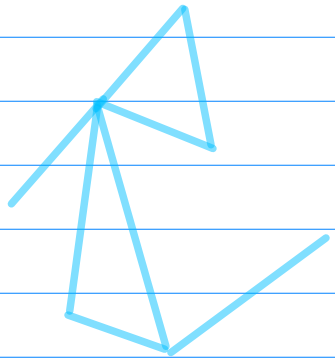


정의 8.2.11 Component 란?

graph G 에서 아무런 vertex v

v에서 시작되는 모든 경로에 포함되는
모든 edge 와 vertex 들로
구성된 부분 graph G' 이다

Let G be a graph and let v be a vertex in G. The subgraph of G consisting of all edges and vertices in G that are contained in some path beginning at v is called a component of G containing v.



1개 component

v에서 시작되는 어떤 경로 상에 있는 모든 edge와 vertex로 구성된 G의 subgraph

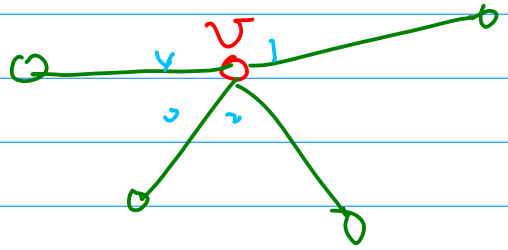
G의 subgraph인데 이 subgraph의 edge와 vertex들이
항상 v에서 시작되는 G의 어떤 경로 상에 있을 때

Connected graph ⟺ 1개 component

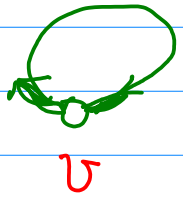
필요충분조건

degree of a vertex v $\delta(v)$

v에 결합된 Vertices의 갯수



$\delta(v) = 4$



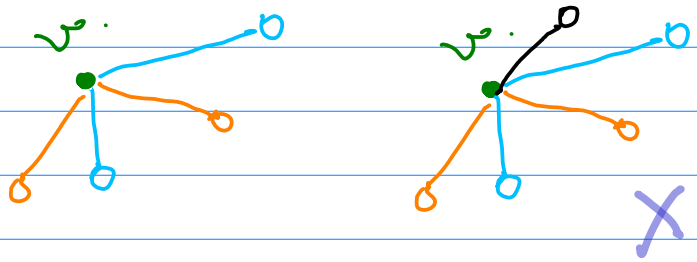
$\delta(v) = 2$

Theorem 8.2.17

Graph G has a Eulerian cycle

$\Rightarrow$ Graph G { connected
. { $\forall$ vertex v $\delta(v)$: even

$\delta(v)$: degree of a vertex v



Theorem 8.2.18

Graph G { connected
. { $\forall$ vertex v $\delta(v)$: even

$\Rightarrow$ Graph G has an Eulerian cycle

2-1 이산 8.2.17

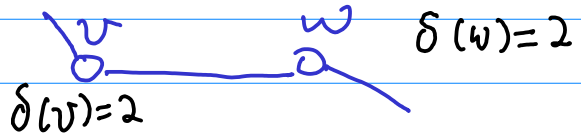
G has Euler cycle

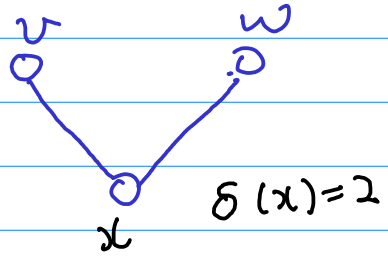
$\Rightarrow$ G : connected graph (1 component)

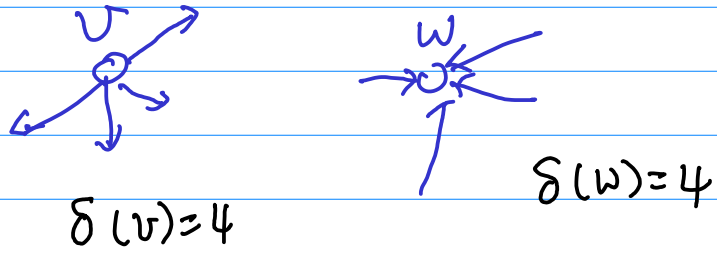
$\delta(v_i) = $ 짝수 for any vertex v_i

Goal any two vertices : v, w

Euler cycle







8.2.18

G : connected graph (1 component)

$\delta(v_i)$ = 짝수. for any vertex v_i

$\Rightarrow$ G는 Euler cycle을 가진다.

# of edge = n

$n=0$ no edge → one vertex

G has n edges

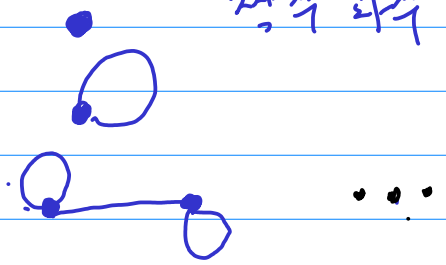
$k < n$
짝수과수정점
connected graph
} 가 Euler cycle을 가진다고 가정

n : # of edges

$n=0$

$n=1$

$n=2$



Graph G가 n개의 edge를 가진다 $n > k$

k개의 edge를 가지고
모든 Vertex가 even degree이고
connected graph $\Longrightarrow$
Euler Cycle을 가진다

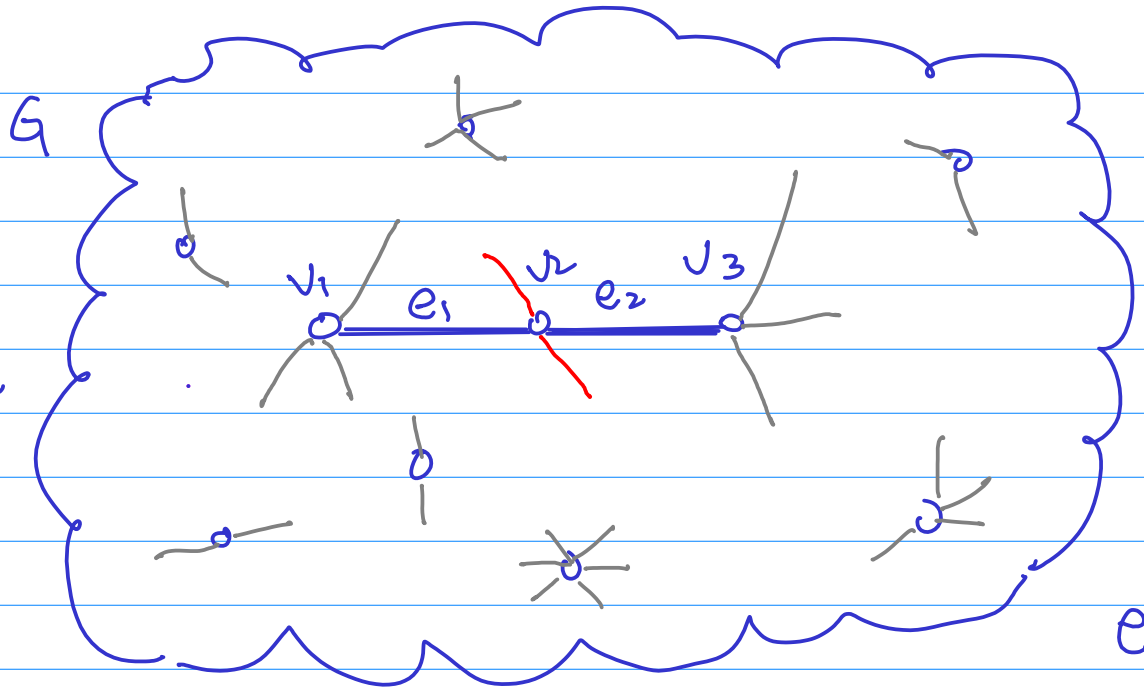
assumption

귀납법

n보다작은 짝수의

k개 edge graph에서
{even degree
{connected graph
$\Longrightarrow$ Euler cycle 있다

n개 edge graph에서
{even degree
{connected graph
$\Longrightarrow$ Euler cycle 있다



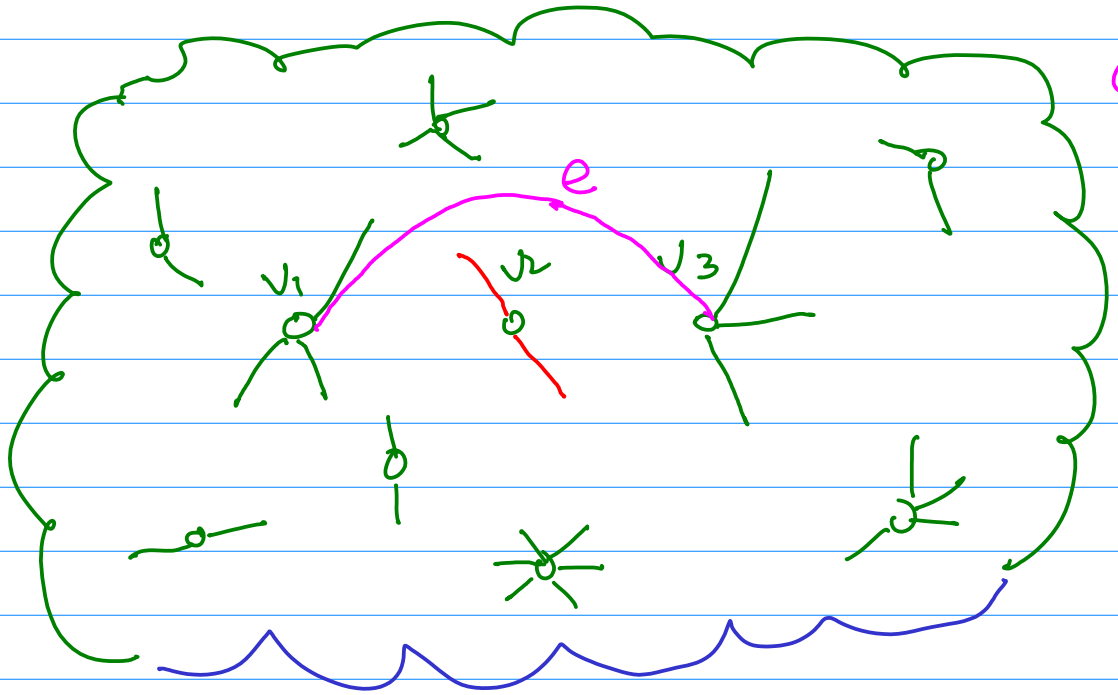
even degree

n no of edge

e_1, e_2

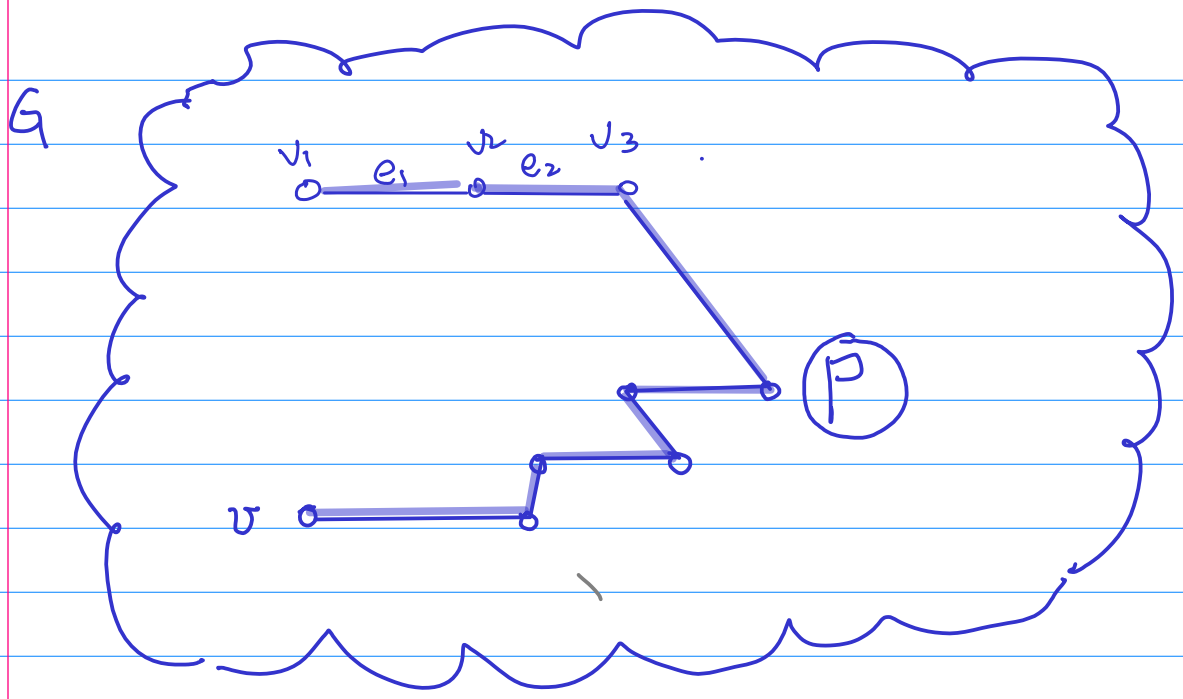
↓ replace

e

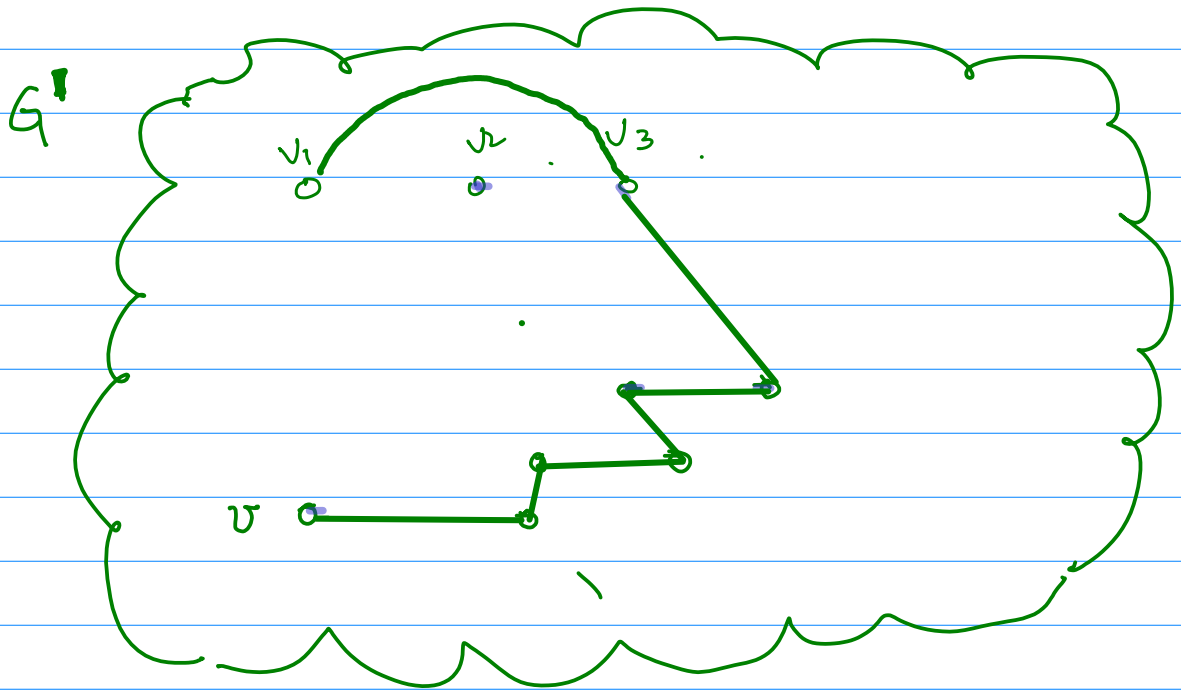


even degree

$n-1$ m edge

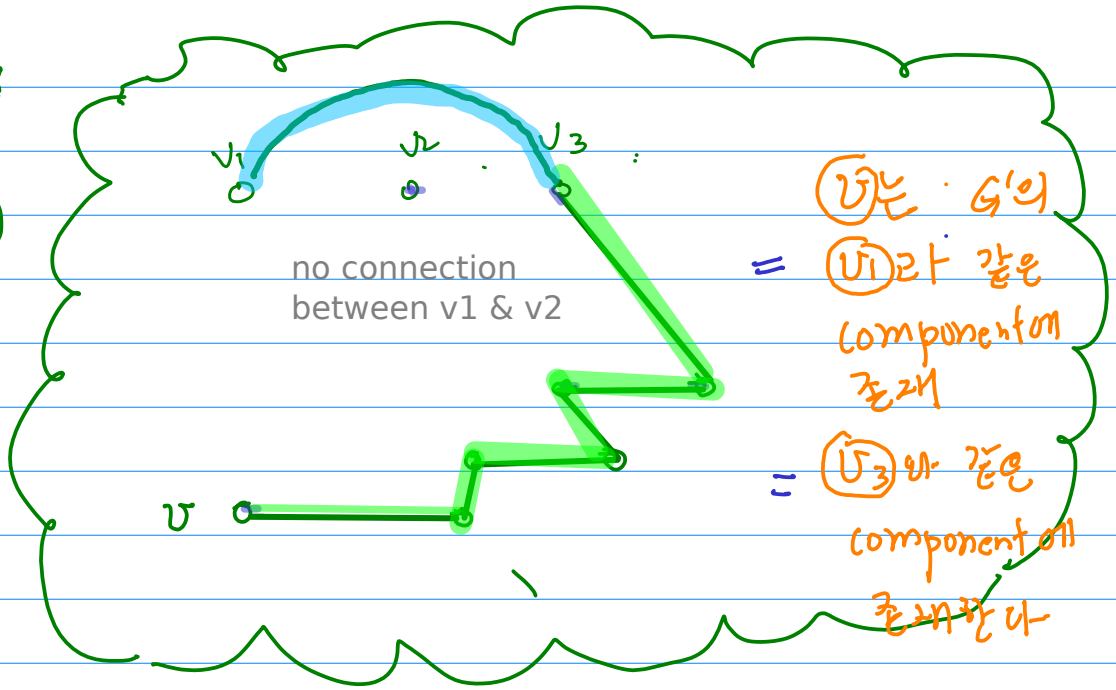


G : Connected $\Rightarrow$ ($v \rightarrow v_1$ path 존재함) (P)



p' = a part of P, whose vertex & node are in G'

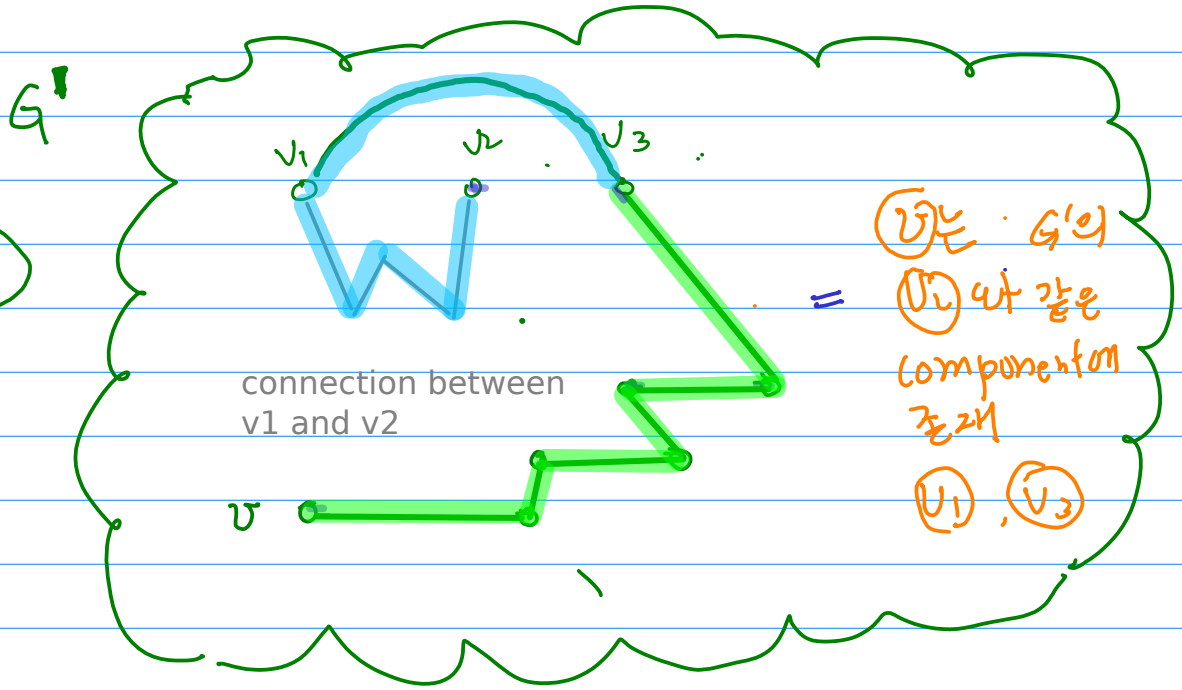
① $v \rightsquigarrow v_1$



v는 G'의 v1과 같은 component에 존재

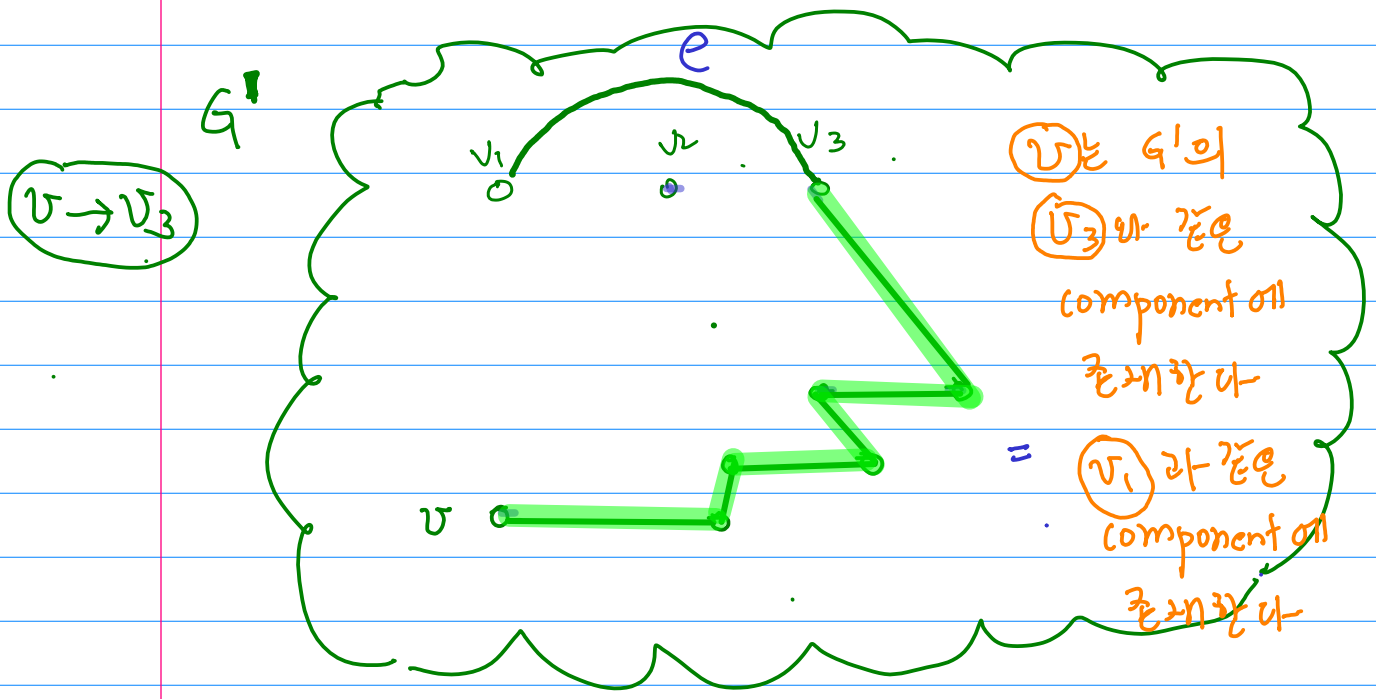
= v3와 같은 component에 존재한다

② $v \rightarrow v_2$



v는 G'의 v1과 같은 component에 존재

v1, v3



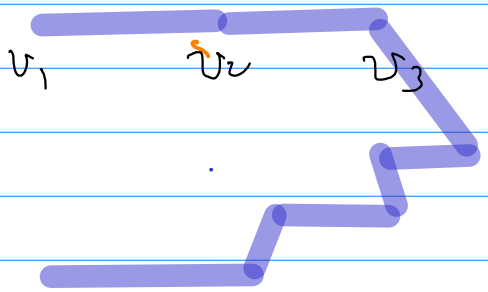
path P : graph G에서 $v \rightsquigarrow v_3 \to v_2 \to v_1$

path P' : graph G'에 남아있는 P의 부분 중
v에서 시작하는 path

3 cases
- $v \rightsquigarrow v_3$: P'가 v_3에서 끝나는 경우
- $v \rightsquigarrow v_2$: P'가 v_2에서 끝나는 경우
- $v \rightsquigarrow v_1$: P'가 v_1에서 끝나는 경우

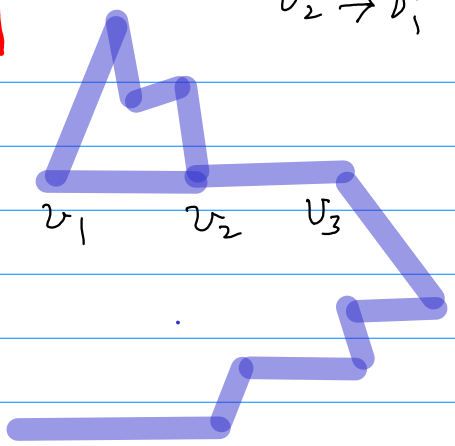
G

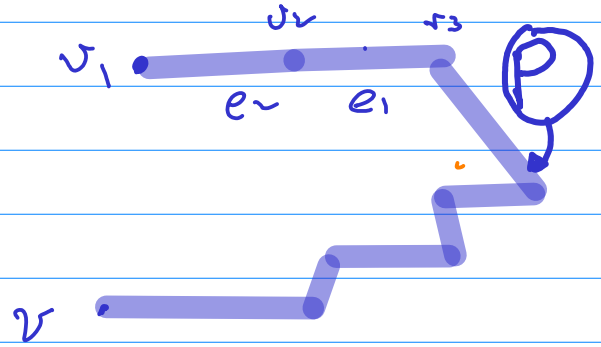
$v_2 \to v_1$ only one path

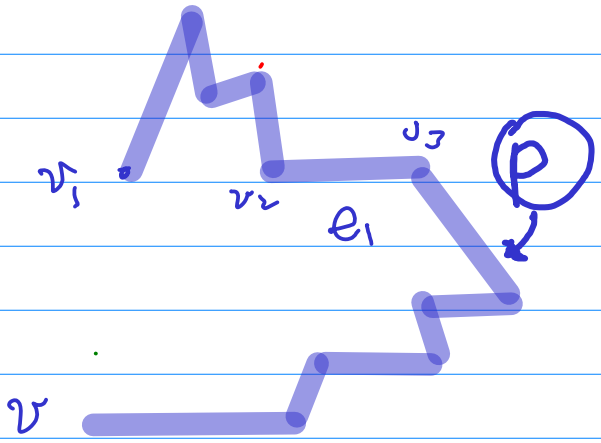


G

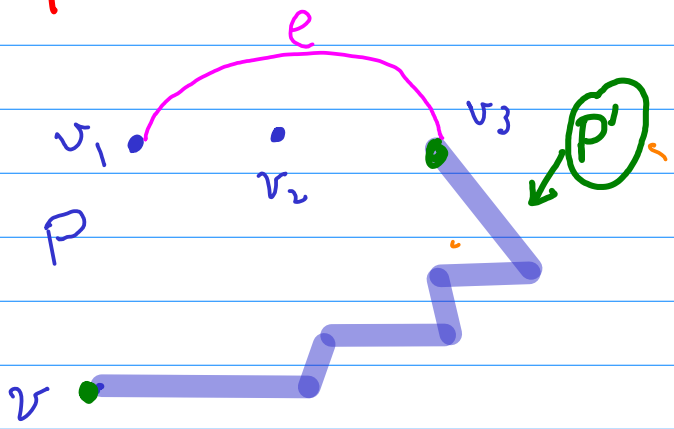
$v_2 \to v_1$ other additional path exists





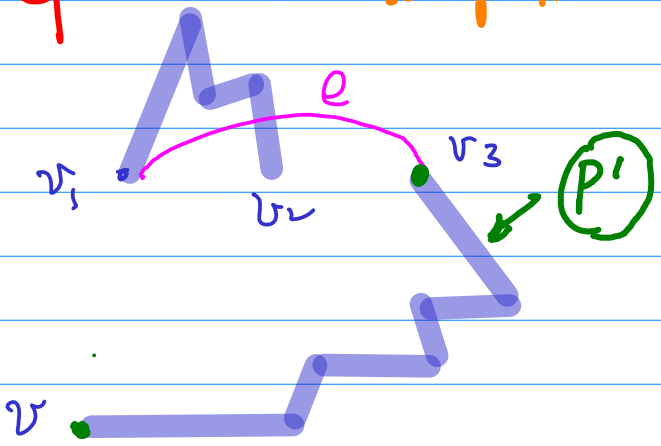


G' 2 components

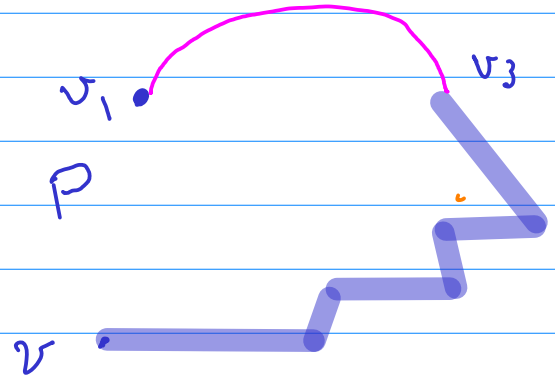


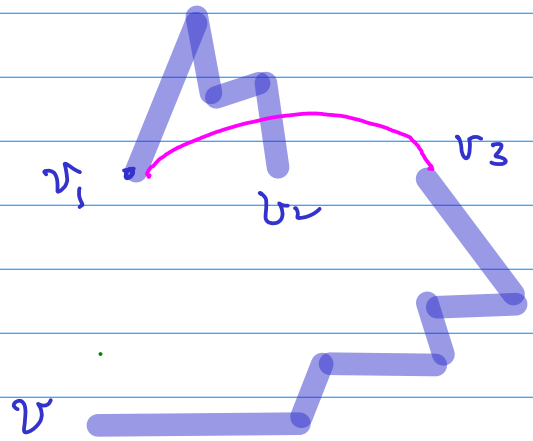
v는 v_1 & v_3 와 같은 component에 있다

G 1 component



v는 v_1, v_2, v_3 와 같은 component에 있다





v, v_1, v_3 만 갈 수 있음

v, v_1, v_2, v_3 모두 갈 수 있음

v_2가 연결 되어 있음

모두 연결 됨

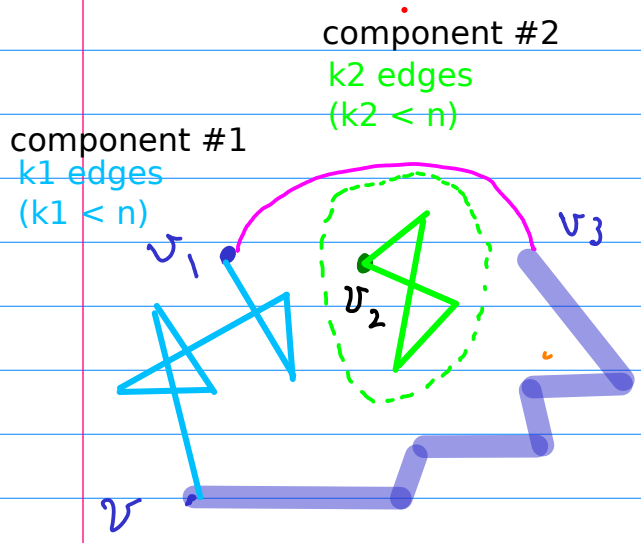
G'에 2개 component 존재

G'에 1개 component

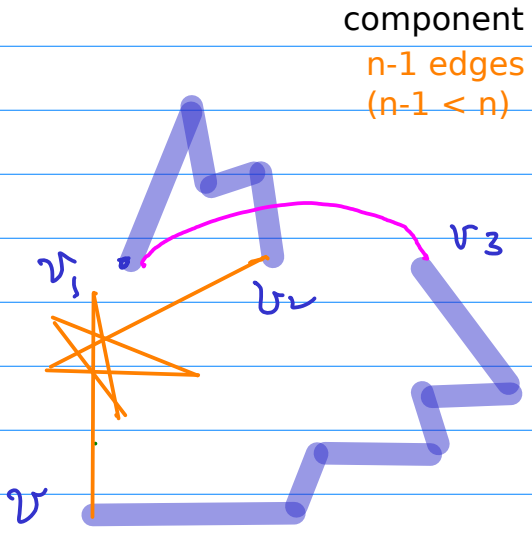
Connected G $\Longrightarrow$ G'

(remove e_1 e_2
add e)

G' 2개 component



G' single component



✳ G와 G'의 차이점은 e_1 e_2 가 빼지고 e가 추가된 것
⇒ 모든 component들은 connect되어있고 even degree이다

n보다 작은 모든 k에 대하여 $k < n$

k개의 edge를 가지고 모든 vertex가 even degree이고 connected graph ⇒ Euler cycle을 가진다

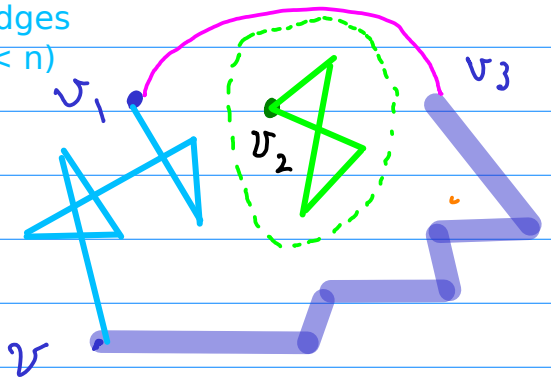
귀납적 가정

n개의 edge를 가지고 모든 vertex가 even degree이고 connected graph ⇒ Euler cycle을 가진다

G' 2개 component

component #1
k1 edges
(k1 < n)

component #2
k2 edges
(k2 < n)

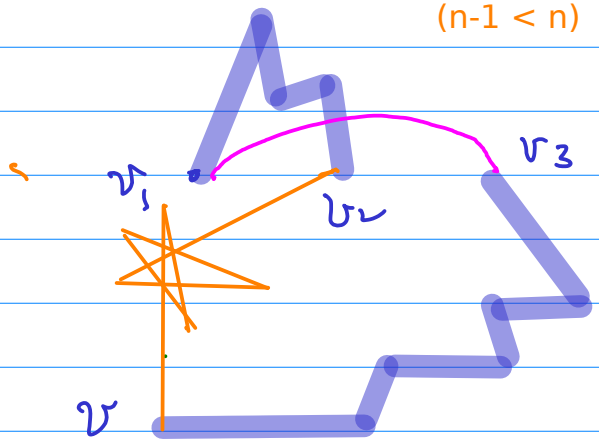


k_1, k_2 모두 edge 개수 even degree connected ⇒

G' 에서 Euler cycle 2개

G' single component

component #1
n-1 edges
(n-1 < n)

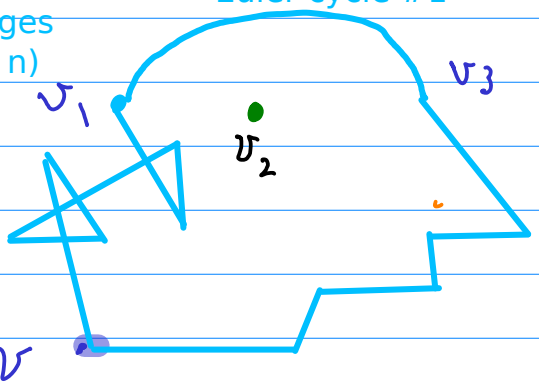


(n-1) 개 edge even degree connected ⇒

G' 에서 Euler cycle

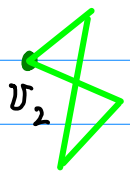
component #1
k1 edges
(k1 < n)

Euler cycle #1



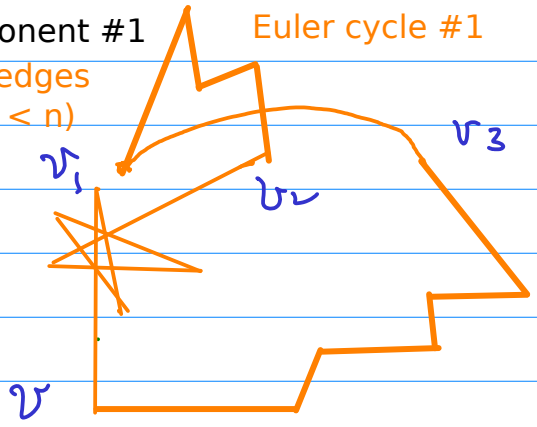
component #2
k2 edges
(k2 < n)

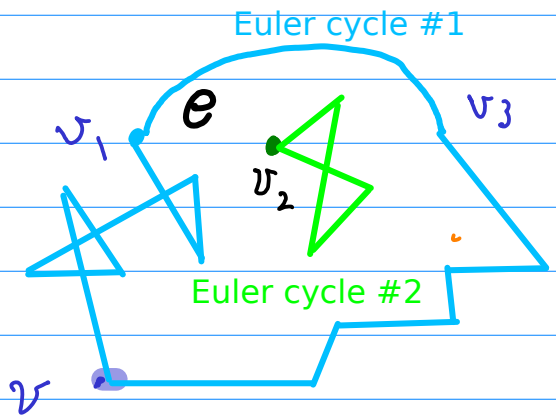
Euler cycle #2

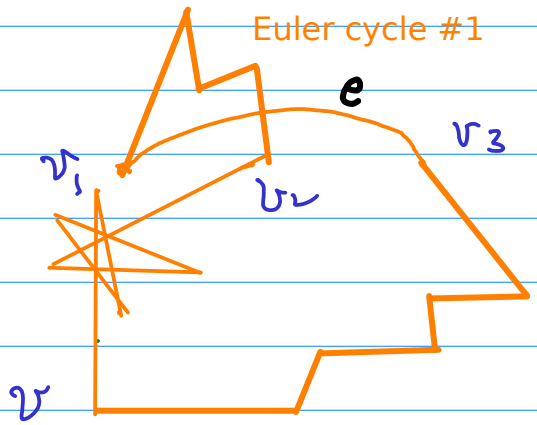


component #1
n-1 edges
(n-1 < n)

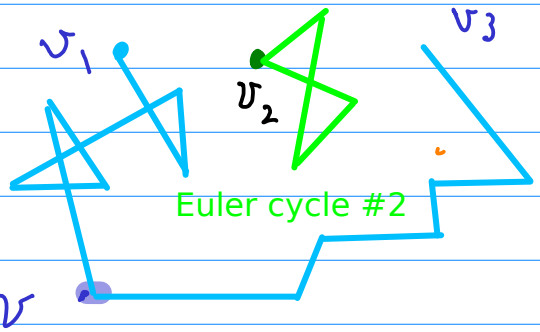
Euler cycle #1



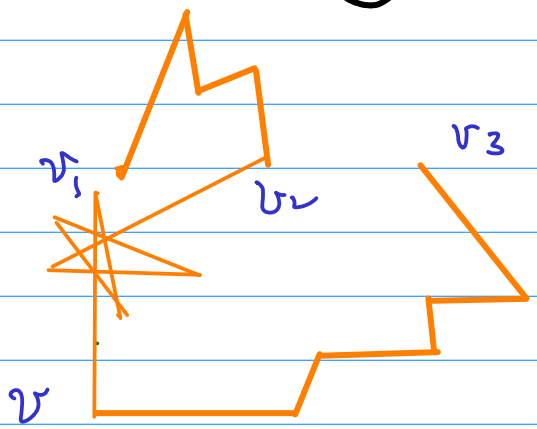




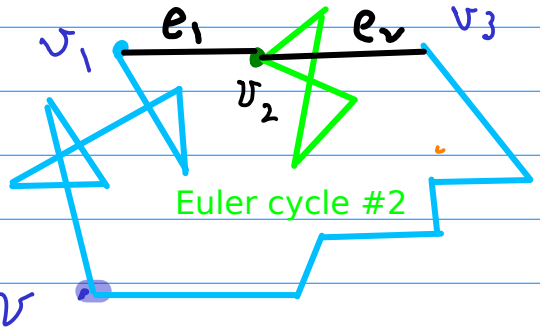
remove (e)



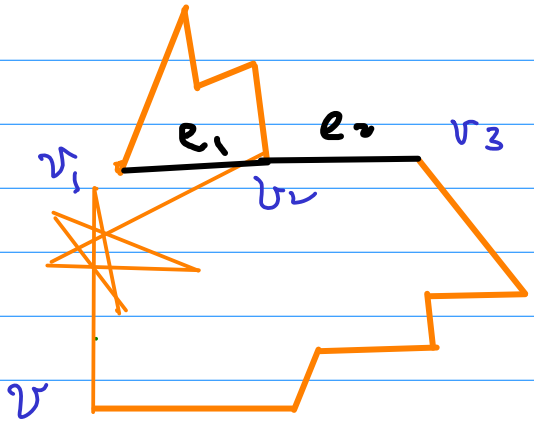
remove (e)



add (e_1) (e_2)



add (e_1) (e_2)



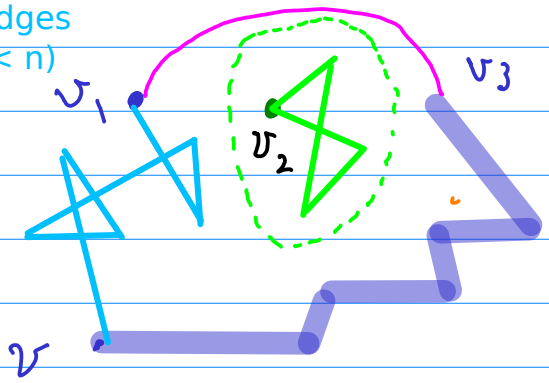
no an edge Graph Euler Cycle

no an edge Graph Euler Cycle

G' 2개 component

component #1
k1 edges
(k1 < n)

component #2
k2 edges
(k2 < n)

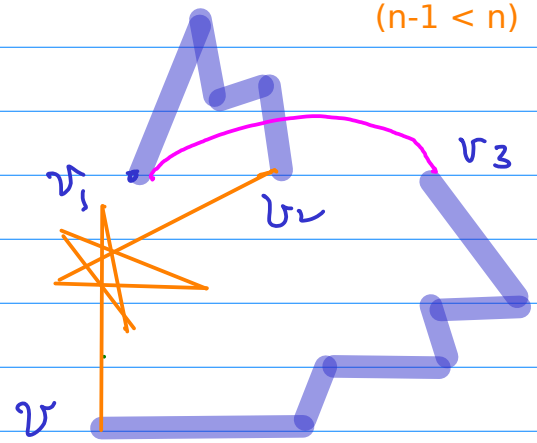


k_1, k_2 모두 짝수 edge
even degree
connected
⇒

G' 에서 Euler cycle 존재

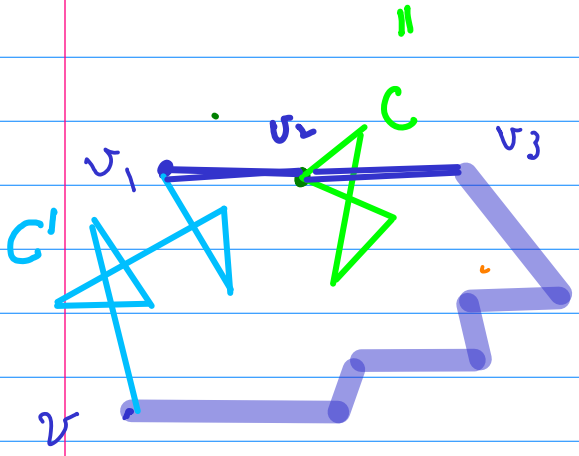
G' 단일 component

component #1
n-1 edges
(n-1 < n)



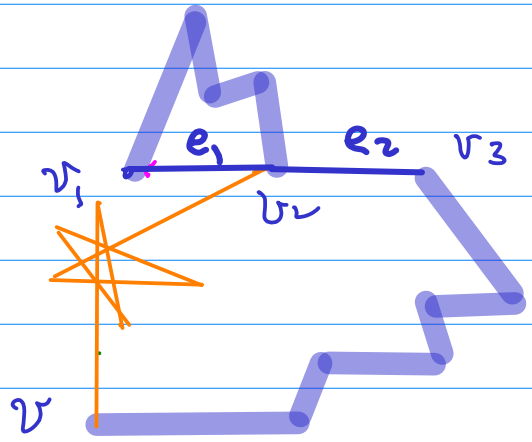
(n-1) 개 짝수 edge
even degree
connected
⇒

G' 에서 Euler cycle 존재



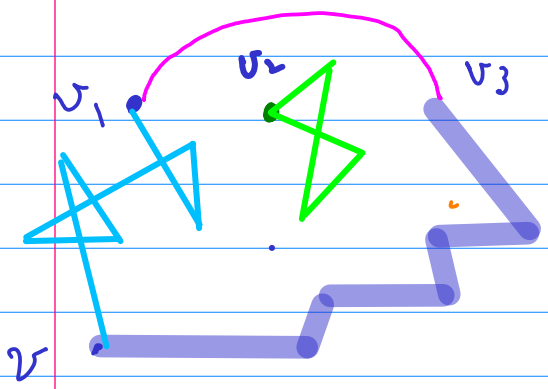
n 개 짝수 edge
even degree
connected
⇒

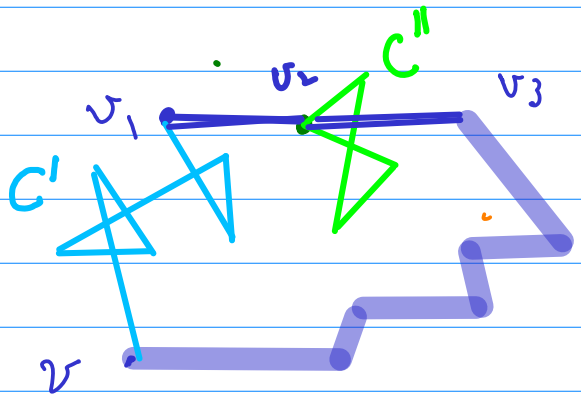
G 에서 Euler cycle 존재



n 개 짝수 edge
even degree
connected
⇒

G 에서 Euler cycle 존재





$v_1 \longrightarrow v_3$

$v_3 \longrightarrow v_1$

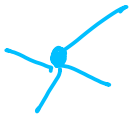
$c' \to v_1 \xrightarrow{e_1} v_2 \xrightarrow{c''} v_2 \xrightarrow{e_2} v_3 \to c'$

$c' \to v_3 \xrightarrow{e_2} v_2 \xrightarrow{c''} v_2 \xrightarrow{e_1} v_1 \to c'$

edge 개수 $n=0$
$n=1$
$n=2$
⋮

모든 graph

A connected graph → only one component → An Euler cycle
with even degree vertices



보려면 even number of edges

A proof by **induction** on the number of edges in G

A connected graph G
with even degree vertices only
and k edges ($k < n$) → An Euler cycle

A connected graph G
with even degree vertices only
and n edges → An Euler cycle

0 개 edge를 갖는 G. → Euler cycle
1 개 edge를 갖는 G. → Euler cycle

k개
($k<n$)

$n-2$ 개 edge를 갖는 G. → Euler cycle
$n-1$ 개 edge를 갖는 G. → Euler cycle

n 개 edge를 갖는 G. → Euler cycle

모든 connected고 even degree

Base case

trivial case

n = 0 edge

n = 1 edge

euler cycle을 갖는다 ...

n = 2 edge

"n 개 edge 갖는 G는
Euler cycle을 가진다"를
귀납법 한다

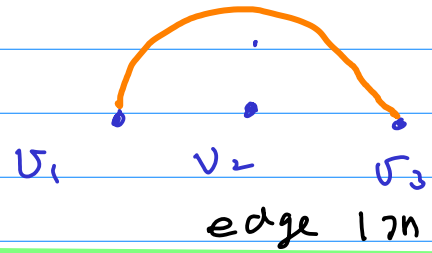
각각 component 또한
Euler cycle을 갖는다

→ Euler cycle 갖는다

귀납적 가정 : k개 edge를 갖는 G
- 1개 component
- 끊어짐마다 적은 수의 edge

이 component들은
edge의 개수 $< n$

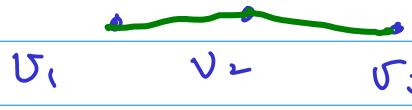
G : 1개 or 2개 component
끊어짐마다 적은 수의 edge
$(n-1)$개 edge



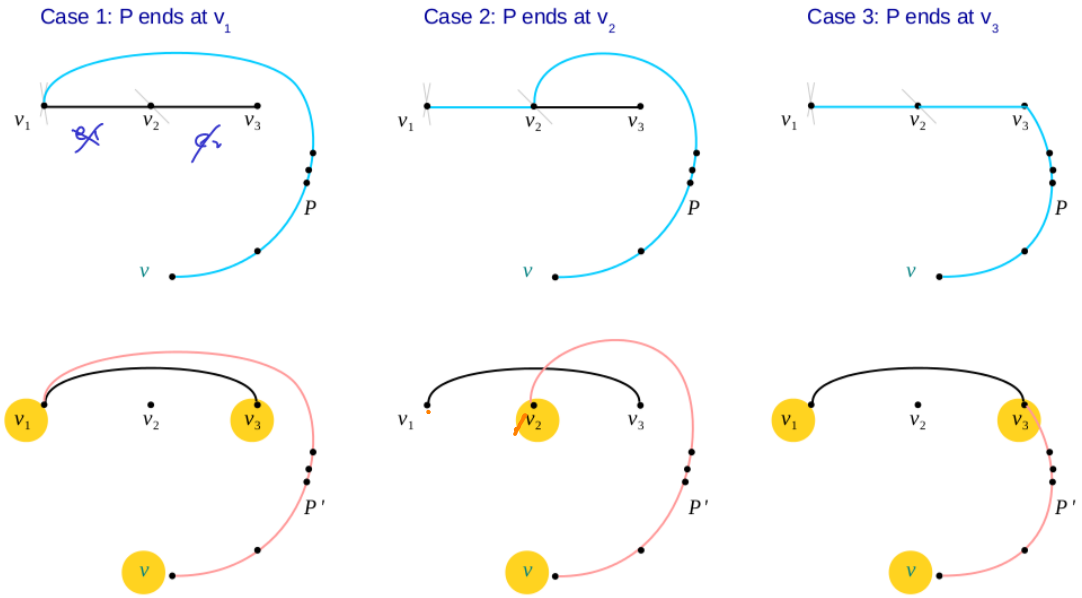
edge 1개

edge 2개

G : 1개 component
끊어짐마다 적은 수의 edge
n개 edge



1개 혹은 2개 component가 생긴다.



Johnsonbough, Discrete Mathematics

$\{v, v_1, v_3\} \sim \{v_2\}$

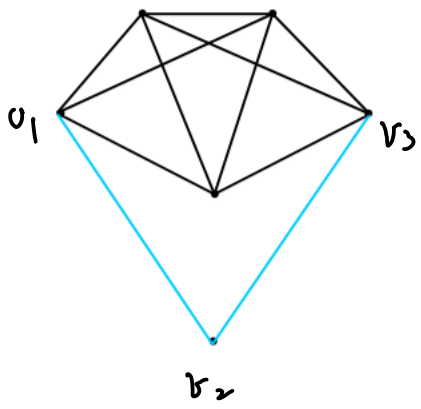
연결된 path가 없으면

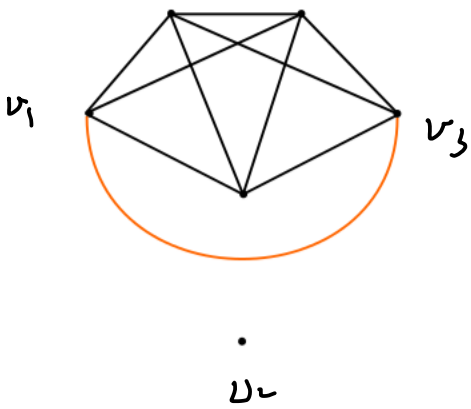
1개 component이고

연결되면

2개 component이다.

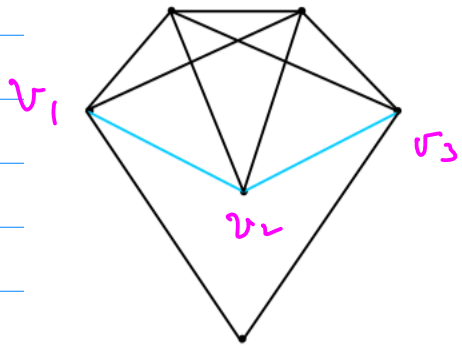
ex)

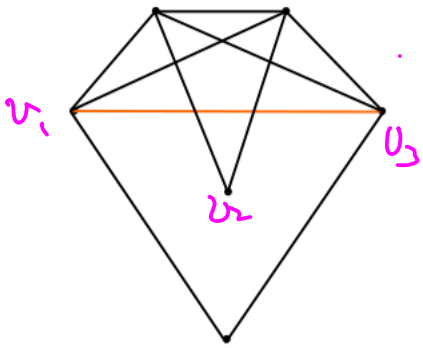




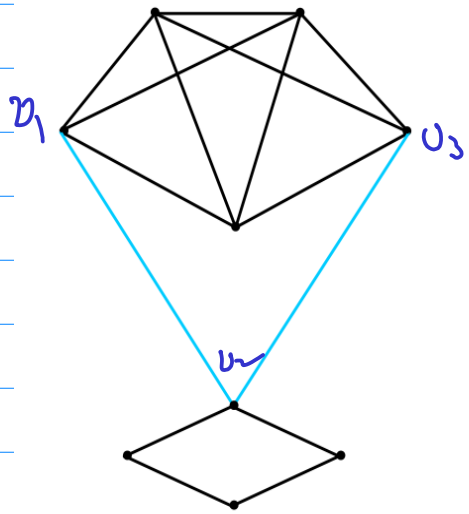
1 connected component

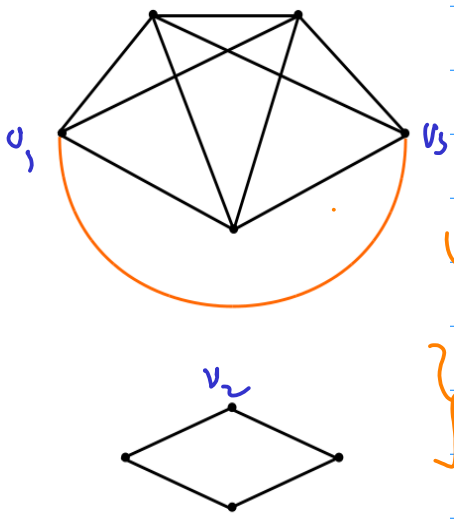
1 connected component

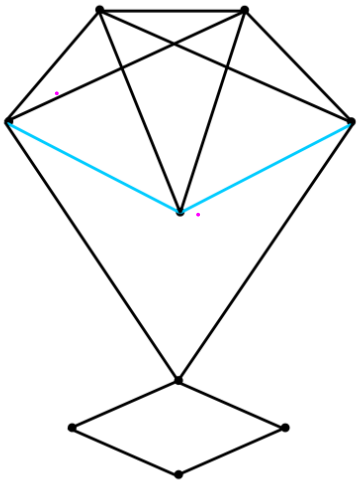


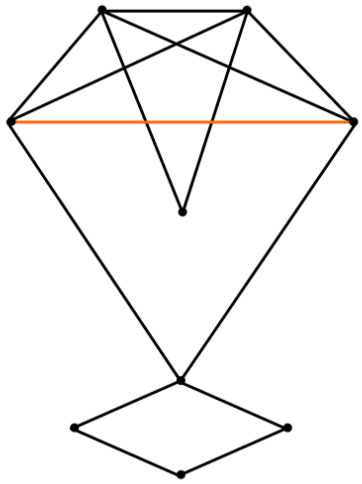


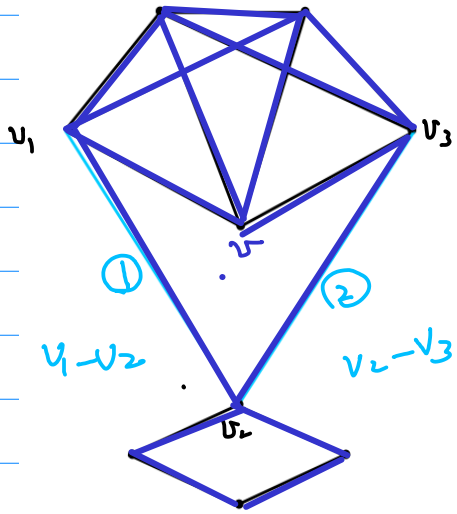
2 connected components

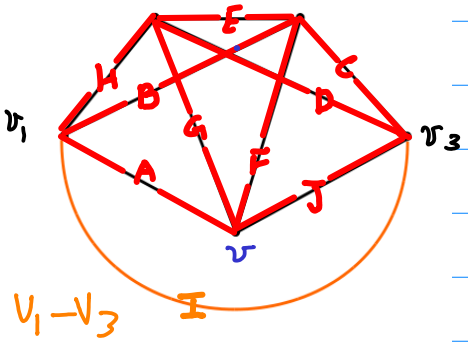

v_1
v_3
v_2

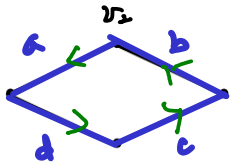

v_1
v_3
v_2
1 cc
component
1 cc
component




1 cc
component

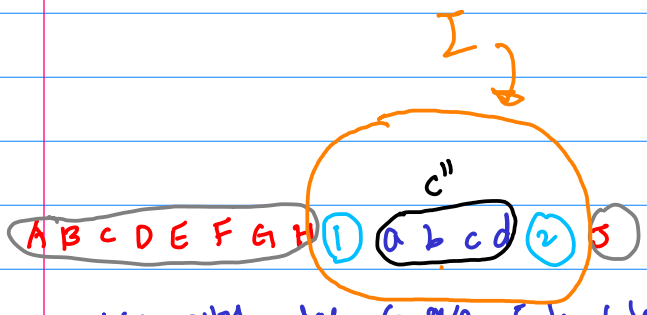




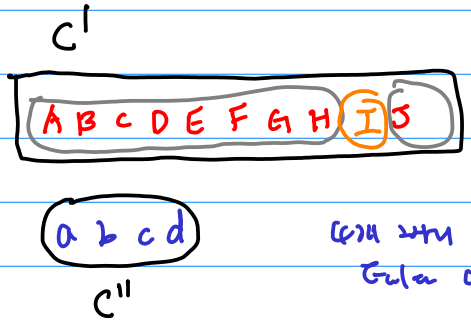


C'' 은 v_2에서 시작하는 폐경로

Euler cycle

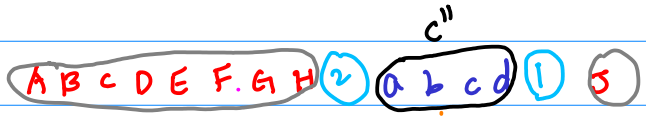


나중 처리 edge 가 있는 Euler cycle



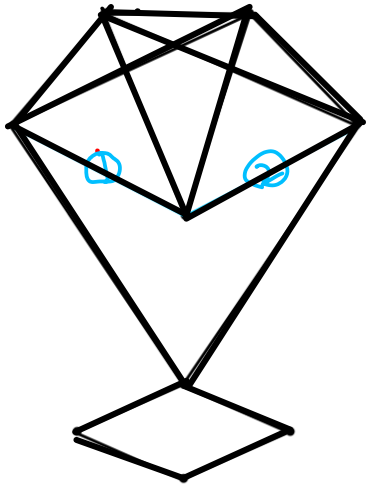
I 가 처리 edge 가 있는 Euler cycle

C'' 가 처리 edge 가 있는 Euler cycle

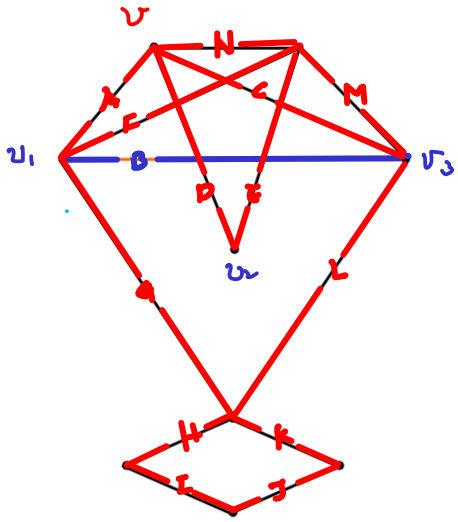


나중 처리 edge 가 있는 Euler cycle

A ① ② C D E F G H I J K L M N



A B C D E F G H I J K L M N



585 p

Vertex에 표시하는 문제

L(v) 가 표시 { temporary mark ··· 임시적인 표시
처음에 모든 v는 L(v) = ∞

final mark ··· 확정적인 표시

L(v)는 a ⟿ v 까지의 최단 경로.
(시작점)

T = {v_i} 임시적인 표시를 갖는 vertex들의 집합

확정적인 표시는 그림에서 ◯ 원을 사용한다.

v ∉ T 인 v에 대하여

L(v)는 a ⟿ v 까지의 최단 경로.
(시작점)

w : weight
a, z : vertex.
L(z) : the shortest path length from a to z

```
dijkstra (w, a, z, L) {
    L(a) = 0
    for every vertex x ≠ a { L(x) = ∞ }

    T = { all vertices that are not final }

    While ( v ∈ T ) {
        Choose v ∈ T with the smallest L(v)      (temporary mark)
        Eliminate v from T                       (final mark)   T = T − {v}
        for each neighbor x of v
            L(x) = min { L(x), L(v) + w(v,x) }
    }
}
```

Choose $v \in T$ with the smallest $L(v)$

Eliminate v from T $\quad T = T - \{v\}$

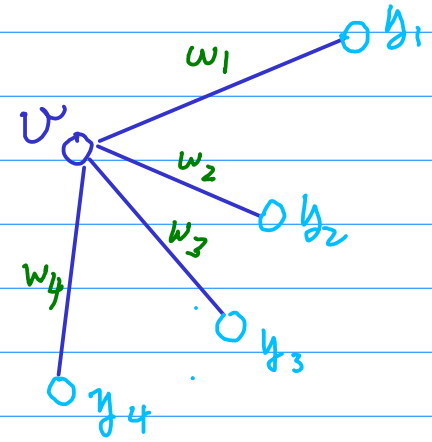
for each neighbor x of v

$$L(x) = \min\{L(x), L(v) + w(v,x)\}$$

$$T = \{\cdots, v, \cdots\} \qquad v \in T$$

$$= \{v, x_1, \cdots, x_n\} \qquad L(v) \le L(x_i)$$

$$T = \{x_1, \cdots, x_n\}$$

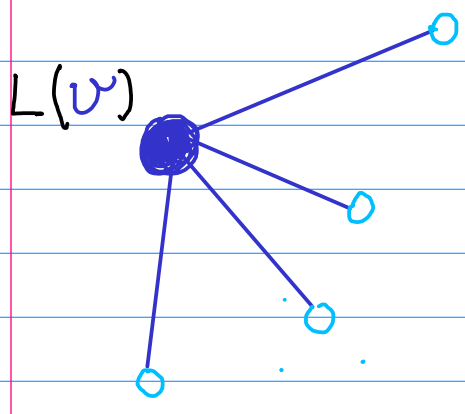


new (updated) old

$$L(y_1) = \min\{L(v) + w_1, L(y_1)\}$$
$$L(y_2) = \min\{L(v) + w_2, L(y_2)\}$$
$$L(y_3) = \min\{L(v) + w_3, L(y_3)\}$$
$$L(y_4) = \min\{L(v) + w_4, L(y_4)\}$$

10 12 (10)

12 (12) 14

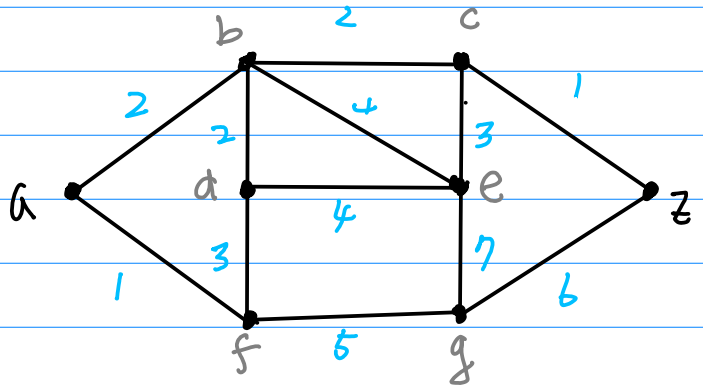


new (updated) fixed value

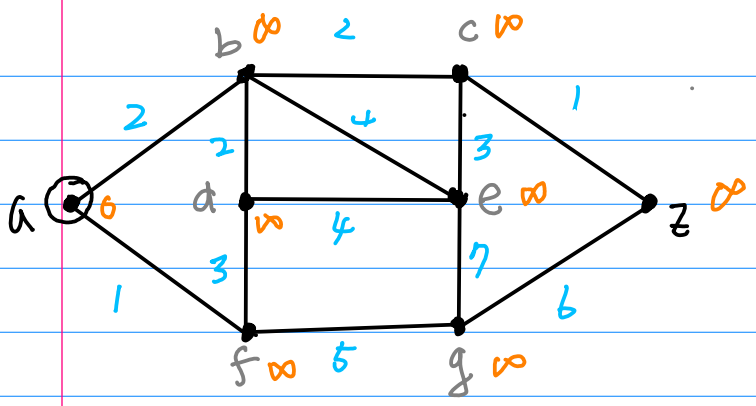
$$L(y_1) \le L(v) + w_1$$
$$L(y_2) \le L(v) + w_2$$
$$L(y_3) \le L(v) + w_3$$
$$L(y_4) \le L(v) + w_4$$

$10 \le 12$

$12 \le 12$


a
b
c
d
e
f
g
z
2
2
2
4
3
1
4
3
7
1
6
5

Algorithm ①

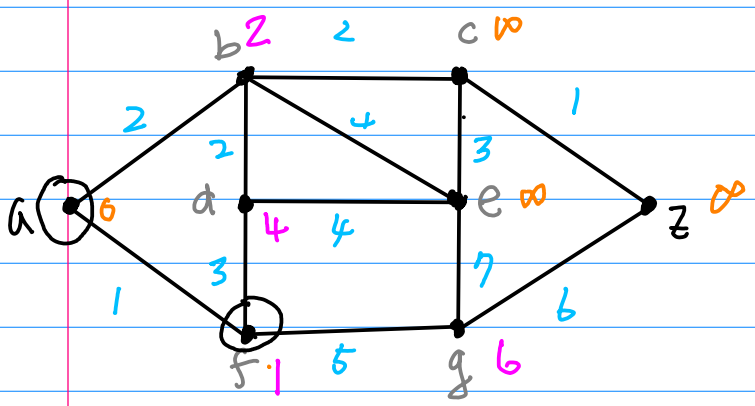


$T = \{a, b, c, d, e, f, g, z\}$

minimum $L(a) = 0$

$L(b) = \min\{\infty, 0+2\} = 2$

$L(f) = \min\{\infty, 0+1\} = 1$

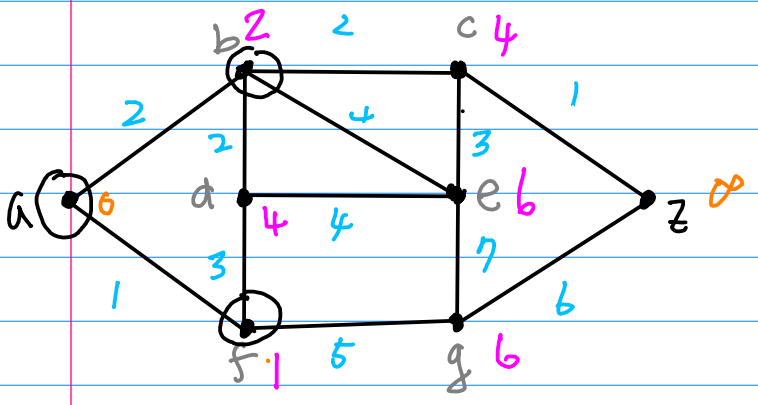


$T = \{b, c, d, e, f, g, z\}$

minimum $L(f) = 1$

$L(d) = \min\{\infty, 1+3\} = 4$

$L(g) = \min\{\infty, 1+5\} = 6$



$T = \{b, c, d, e, g, z\}$

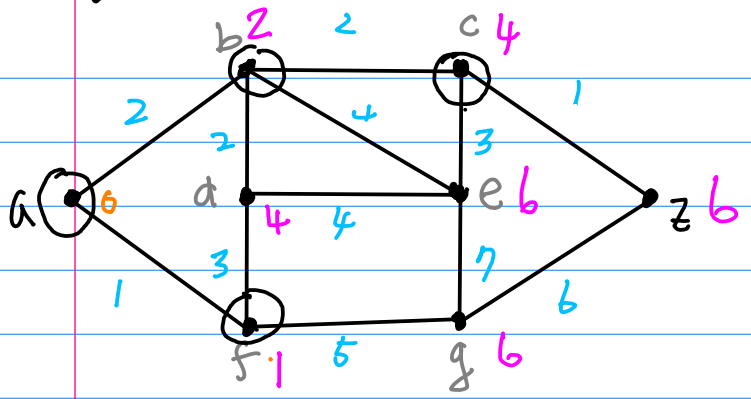
minimum $L(b) = 2$

$L(c) = \min\{\infty, 2+2\} = 4$

$L(d) = \min\{4, 2+2\} = 4$

$L(e) = \min\{\infty, 2+4\} = 6$

Algorithm (2)

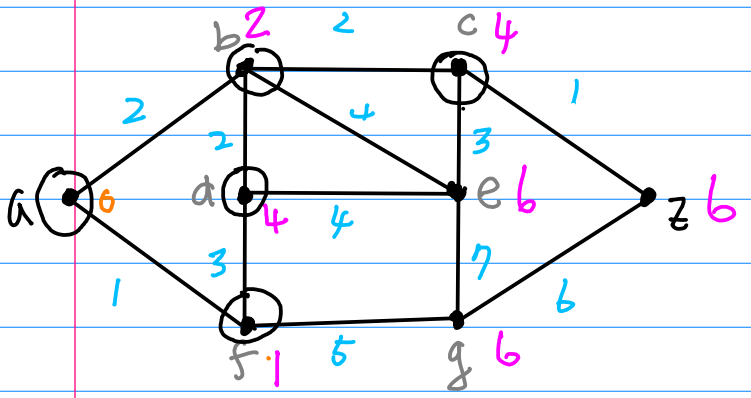


$T = \{c, d, e, g, z\}$

minimum $L(c) = 4$

$L(e) = \min\{6, 4+2\} = 6$

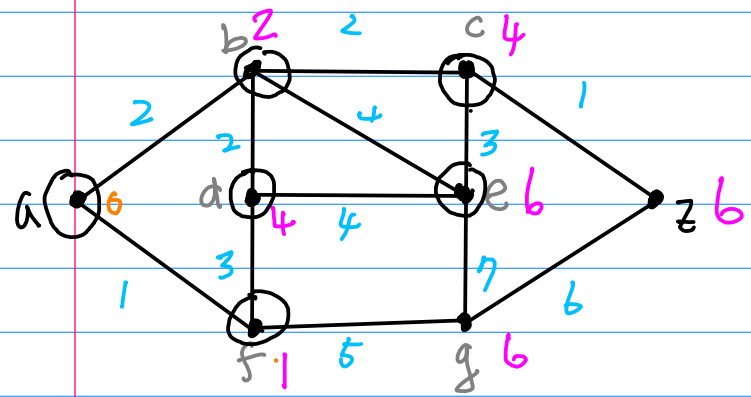
$L(z) = \min\{\infty, 4+2\} = 6$



$T = \{d, e, g, z\}$

minimum $L(d) = 4$

$L(e) = \min\{6, 4+4\} = 6$

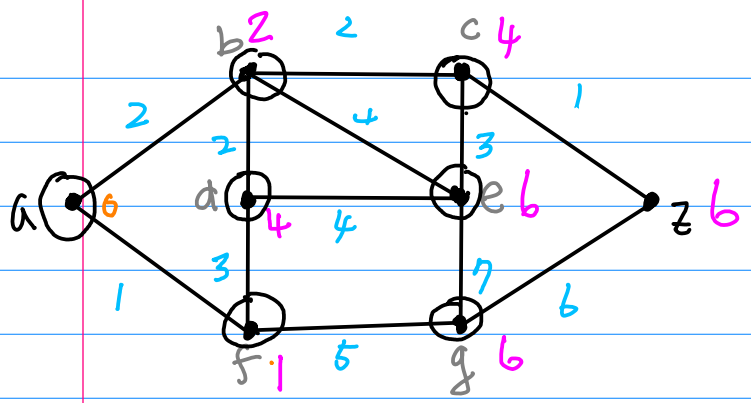


$T = \{e, g, z\}$

minimum $L(e) = 6$

$L(g) = \min\{6, 4+7\} = 6$

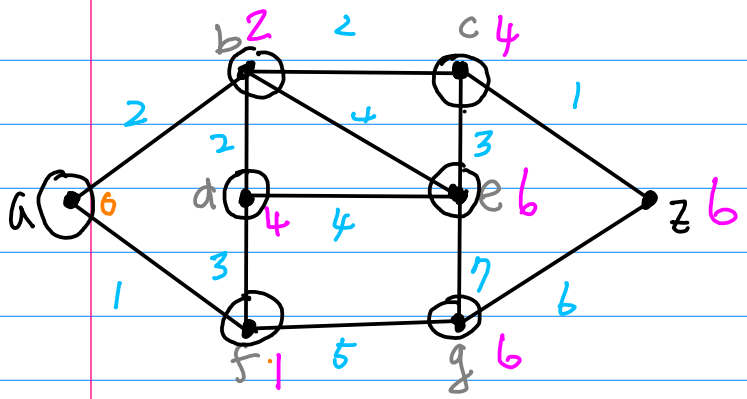
Algorithm (3)



$T = \{g, z\}$

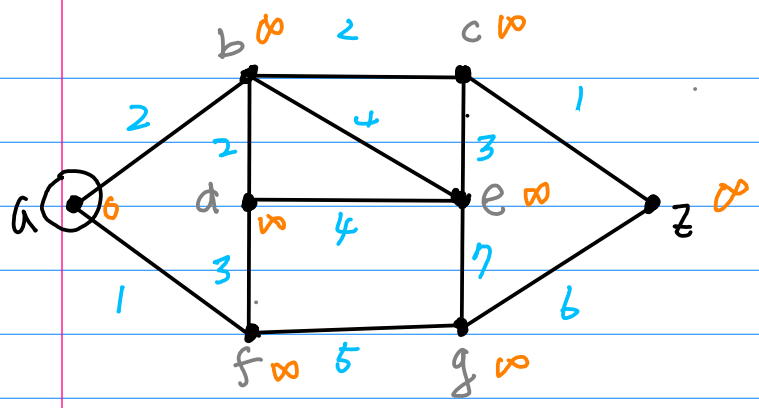
minimum $L(g) = 6$

$L(z) = \min\{6, 6+6\} = 6$



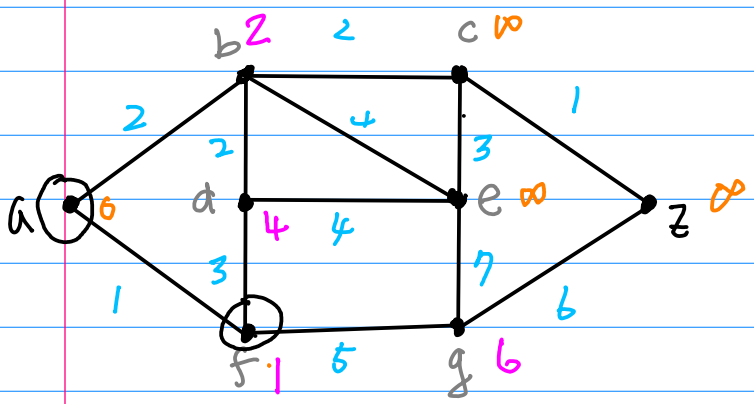
$T = \{z\}$

minimum $L(g) = 6$



$T = \{a, b, c, d, e, f, g, z\}$

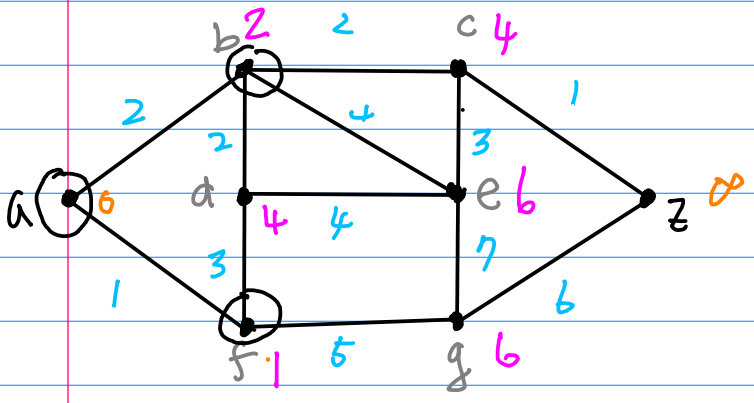
$L(a) = 0$ (1)



$T = \{b, c, d, e, f, g, z\}$

$L(a) = 0$ (1)

$L(f) = 1$ (2)

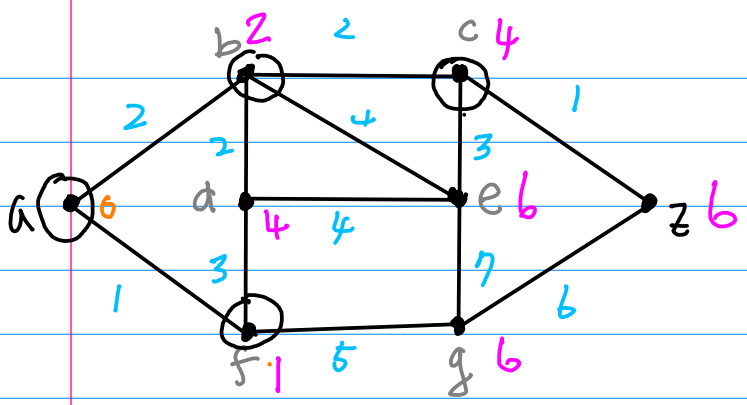


$T = \{b, c, d, e, g, z\}$

$L(a) = 0$ (1)

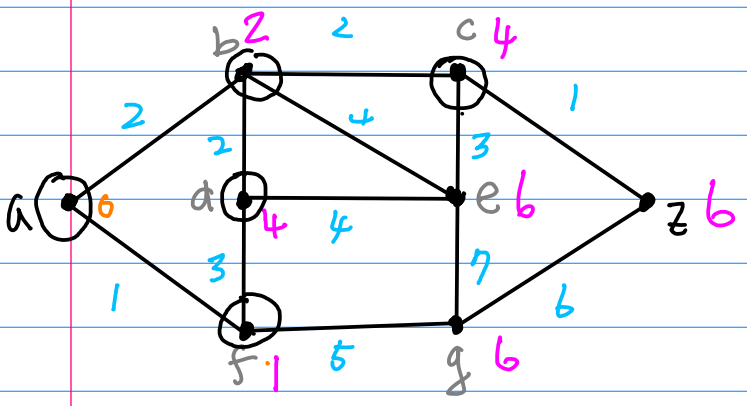
$L(f) = 1$ (2)

$L(b) = 2$ (3)



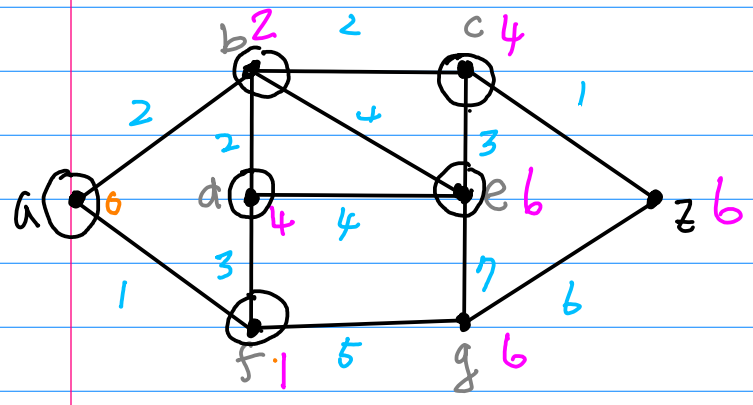
$T = \{c, d, e, g, z\}$

$L(a) = 0$ ①
$L(f) = 1$ ②
$L(b) = 2$ ③
$L(c) = 4$ ④



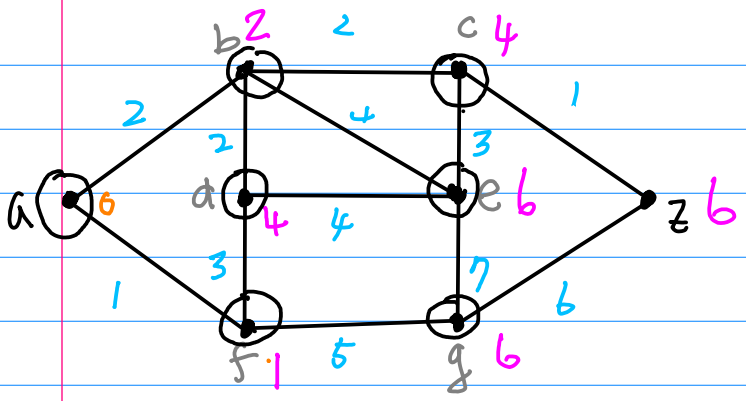
$T = \{d, e, g, z\}$

$L(a) = 0$ ①
$L(f) = 1$ ②
$L(b) = 2$ ③
$L(c) = 4$ ④
$L(d) = 4$ ⑤



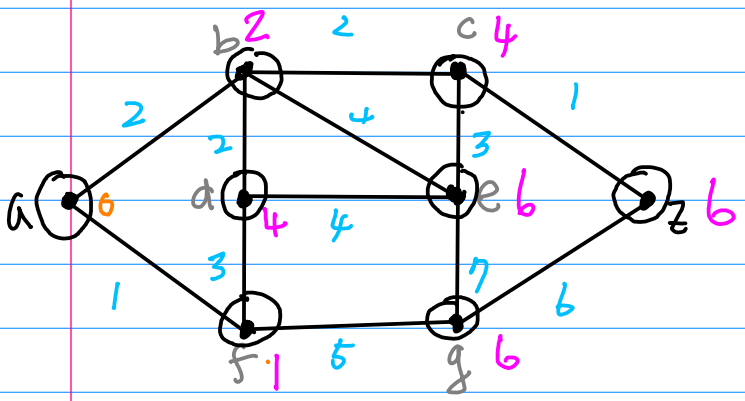
$T = \{e, g, z\}$

$L(a) = 0$ ①
$L(f) = 1$ ②
$L(b) = 2$ ③
$L(c) = 4$ ④
$L(d) = 4$ ⑤
$L(e) = 6$ ⑥



$T = \{g, z\}$

$L(a) = 0$ ①
$L(f) = 1$ ②
$L(b) = 2$ ③
$L(c) = 4$ ④
$L(d) = 4$ ⑤
$L(e) = 6$ ⑥
$L(g) = 6$ ⑦



$T = \{z\}$

$L(a) = 0$ ①
$L(f) = 1$ ②
$L(b) = 2$ ③
$L(c) = 4$ ④
$L(d) = 4$ ⑤
$L(e) = 6$ ⑥
$L(g) = 6$ ⑦
$L(z) = 6$ ⑧

Dijkstra's shortest-path algorithm
correctly finds the length of a shortest path
from a to z.

mathematical induction on i

proove this ↓

> the i-th time to choose v with the minimum L
>
> $L(v)$: the length of a shortest path (a, v)

If the above is true, then
when z is chosen,

$L(z)$: the length of a shortest path (a, z)

$\Rightarrow$ the algorithm works!

Basic step $i=1$

initialization : only $L(a)=0$,
$L(v)=\infty$ $(v \neq a)$

Choose $L(a)=0$ ... the length of the shortest path (a, a)

Inductive step i

$k < i$ assume this is true

the k-th time to choose v with the minimum L

$L(v)$: the length of a shortest path (a, v)

then, this is also true

the i-th time to choose v with the minimum L

$L(v)$: the length of a shortest path (a, v)

the i-th time to choose v with the minimum L

$L(v)$: the length of a shortest path (a, v)

Suppose

the i-th time to choose v with the minimum L

$v \in T$

$L(v)$ the smallest one

Let's show

if there is a path $P(a,w) < L(v)$
then $w \notin T$

Proof by contradiction

$\Rightarrow$ if there were a path from a to v
whose length $P(a,v) < L(v)$
then v would already have been selected
and should have been removed from T

$\Rightarrow$ every path $P(a,v) \geq L(v)$ at least

$\Rightarrow$ there is a path from a to v of length $L(v)$
and this is a shortest path from a to v.

Let's show

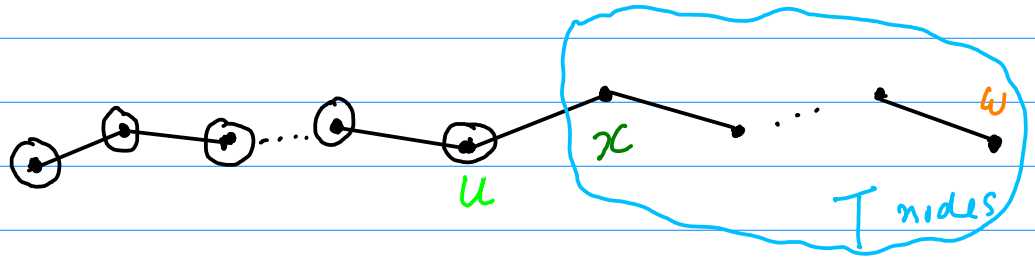
> if there is a path $P(a, w) < L(v)$
> then $w \notin T$

Proof by contradiction

assume this is true

> if there is a path $P(a, w) < L(v)$
> then $w \in T$

Let P : a shortest path from a to w
x : a vertex nearest a on P that is in T
u : a predecessor of x on P



$L(v)$ subject to change

$u \notin T$
therefore u was chosen
during the previous iteration

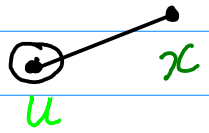
by inductive assumption
$L(u)$: the length of a shortest path (a, u)

$$L(u) \leq L(u) + w(u, x) \leq P(u, w) < L(v)$$

wrong assumption

if there is a path $P(a,w) < L(v)$
then $w \in T$

$$\underbrace{L(x) \leq L(u) + w(u,x)}_{\text{update equation}} \leq P(a,w) < L(v)$$



$v \in T$
$L(v)$: the smallest

$x \in T$
$L(x)$: the smallest

Contradiction

if there is a path from a to a vertex w
whose length is less than $L(v)$,
then w is not in T