Chapter 5. Graph Theory

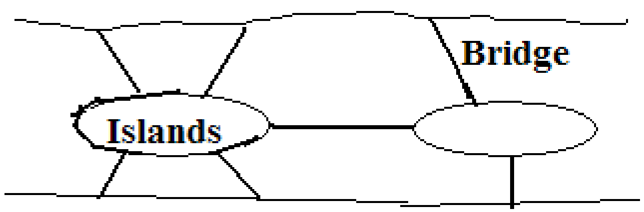
- 1. Definitions
  - 1.1 Undirected graph
  - 1.2 Terminology

- 2. Path and Cycle
  - 2.1 Euler Cycle   2.2 Euler Path
  - 2.3 Hamiltonian Circuit
  - 2.4 Shortest Path

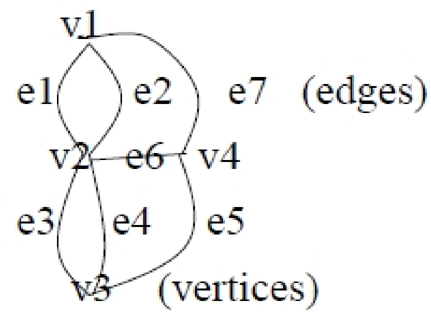
- 3. Graph Property
  - 3.1 Isomorphism   3.2 Planar graphs
  - 3.3 Vertex Coloring

1. Introduction

Konisberg Bridge problem → developed into Graph Theory (by Leonhard Euler)



[Graph model]



Question: Starting and ending at the same point, is it possible to cross all seven bridges just once and return to the same point.

Answer: No path exists Euler Path: Even edges leaving vertices

- Application area: Network, scheduling algorithm, state machine, circuit, data flow sorting and searching algorithm, social science, chemistry, EE, etc.

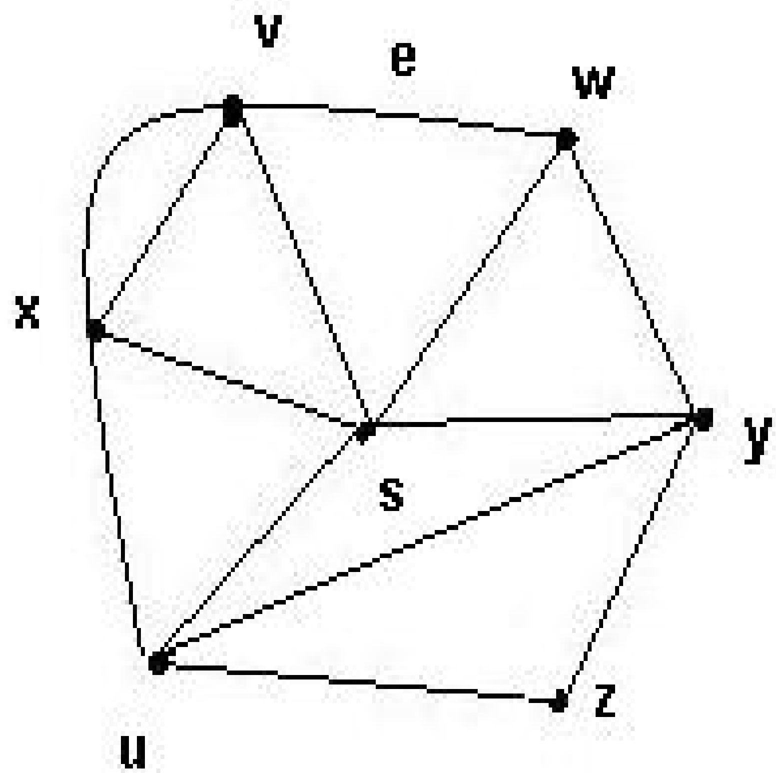
- Directed Graph - has arrow Undirected Graph - no arrow

1.1 Undirected Graph

- What is a graph G?
- It is a pair G = (V, E), where
  - V = V(G) = set of vertices
  - E = E(G) = set of edges

- **Example:**
  - V = {s, u, v, w, x, y, z}
  - E = {(x,s), $(x,v)_1$, $(x,v)_2$, (x,u), (v,w), (s,v), (s,u), (s,w), (s,y), (w,y), (u,y), (u,z),(y,z)}

  - E is a symmetric relation on V => (x,s) $\in$ E



Edges

- An edge may be labeled by a pair of vertices, for instance e = (u, w).

  e is between u and w and u

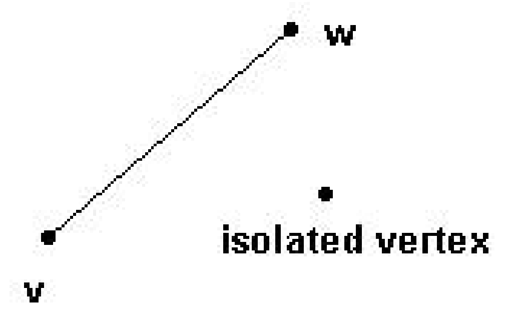
  u is **adjacent** to w

  Edge e **connects** u and w

  e is **unordered pair** , rather than (u,w) $\neq$(w,u)

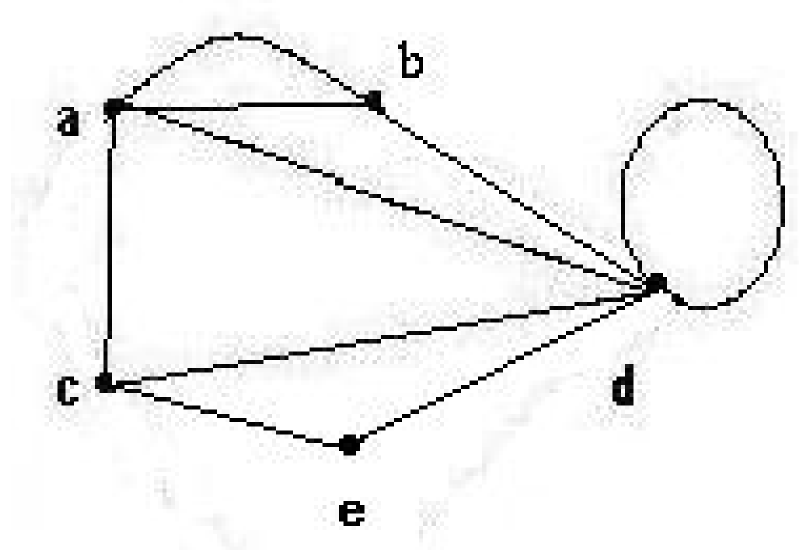
- e is said to be ***incident*** on v and w.
- Isolated vertex =

a vertex without incident edges.



Special edges

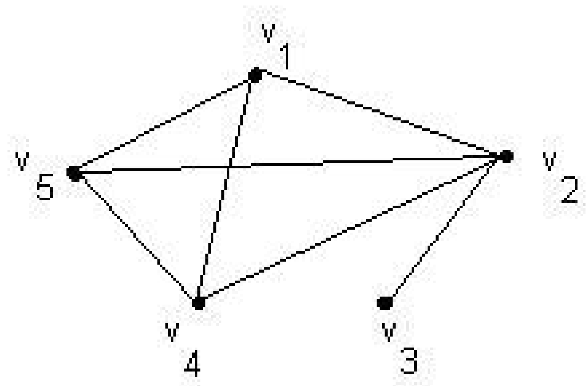
- Parallel edges
  - Two or more edges joining a pair of vertices
    - in the example, **a** and **b** are joined by two parallel edges



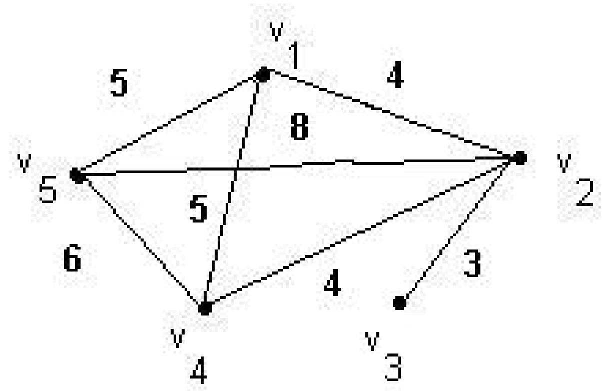
- Loops
  - An edge that starts and ends at the same vertex
    - In the example, vertex **d** has a loop
- Multi-graph
  - G에서 2개이상의 간선이 허용되는 그래프

Special graphs

- Simple graph
  - A graph without loops or parallel edges.



- Weighted graph
  - A graph where each edge is assigned a numerical label or “weight”.



Subgraph, Directed Graph

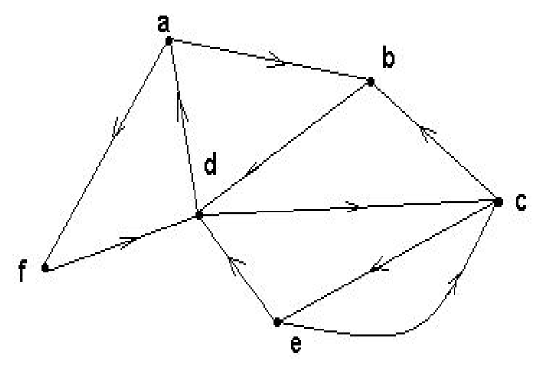
- Let G=(V,E) be a graph.

 Let G'=(V',E') be another graph, such that if

 Then G' is **subgraph** of G. $V' \subseteq V$ and $E' \subseteq E$

- G is a ***directed graph*** or **digraph**

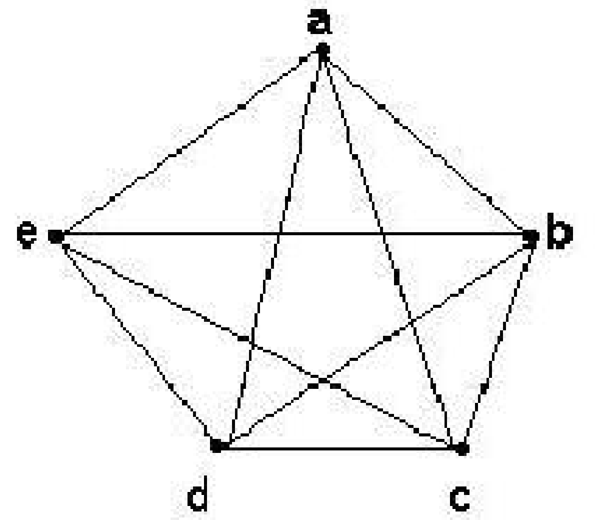
 if each edge has been
associated with an ordered
pair of vertices,
(i.e. each edge has a direction)



<$v1$, $v2$>, $v1$에서 $v2$로의 간선은
화살표(→)로 표시.

Complete graph K_n

- The ***complete graph*** K_n is the graph with **n vertices** and every pair of vertices is joined by an edge

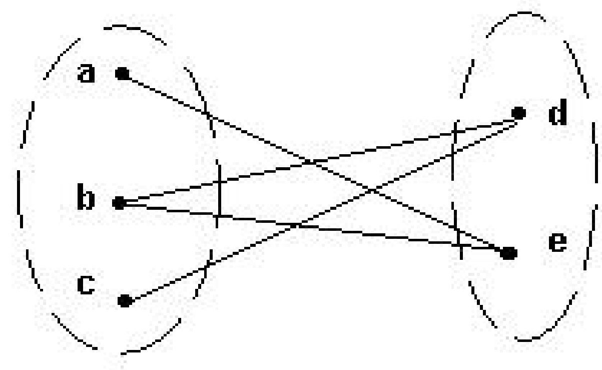


- Figure represents $\mathbf{K_5}$, (when $n \geq 3$)
- Edges for $K_n = n(n-1)/2$
- Every graph is a subgraph of K_n

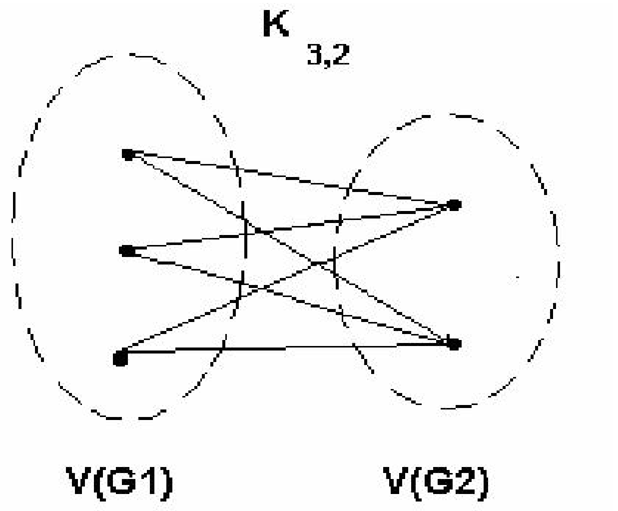
Bipartite graphs (이분그래프)

- A **<u>bipartite</u>** graph G is a graph such that
  - $V(G) = V(G_1) \cup V(G_2)$

    (if vertex set V can be partitioned into two subsets)
  - $|V(G_1)| = m$, $|V(G_2)| = n$
  - $V(G_1) \cap V(G_2) = \varnothing$
  - No edges exist between any two vertices in the same subset $V(G_k)$, $k = 1,2$



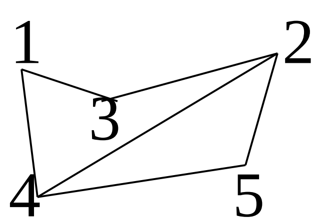
Complete bipartite graph $K_{m,n}$



- A bipartite graph is the *complete* bipartite graph $K_{m,n}$ if every vertex in $V(G_1)$ is joined to a vertex in $V(G_2)$ and conversely,
- $|V(G_1)| = m$
- $|V(G_2)| = n$

ex) Complete Bipartite graph ex) $K_{2,4}$ $K_{3,3}$?

ex)



Connected graphs

- A graph is *connected* if every pair of vertices can be connected by a path

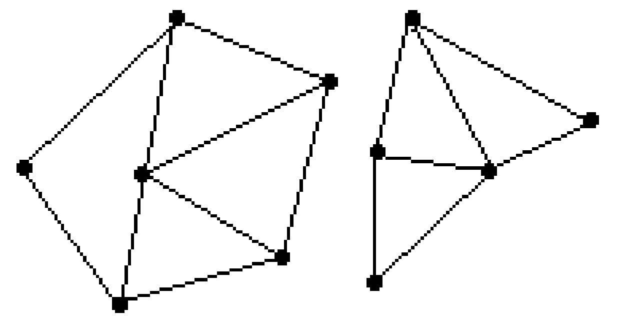
(strongly connected: $(a,b) \in E$, $(b,a) \in E$)

- Each connected subgraph of a non-connected graph G is called a ***component*** of G

- **Connectivity Number** = Number of connected components

Connectivity number of G = C(G)

- G is connected iff C(G) =1



2 connected components

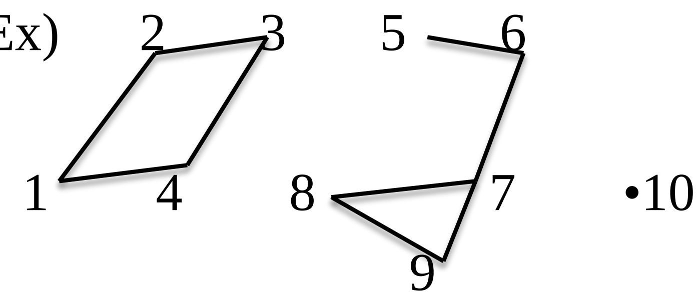
C(G) = 2

Algorithm for Connectivity

- {Let G=(V,E) be graph, C=C(G) }

begin
  V’ <- V
  C <- 0
  while V’ ≠ 0 do
  begin
    choose y ∈ V’
    Find all vertices connected to y
    remove these from V’
    C<- C+1
  end
end

Ex)



V’ = {1,2,3,...10}

Choose y = 5 (any number 1~10)

remove 5,6,7,8,9 from V’,
V’={1,2,3,410} C=1

Choose y=1,

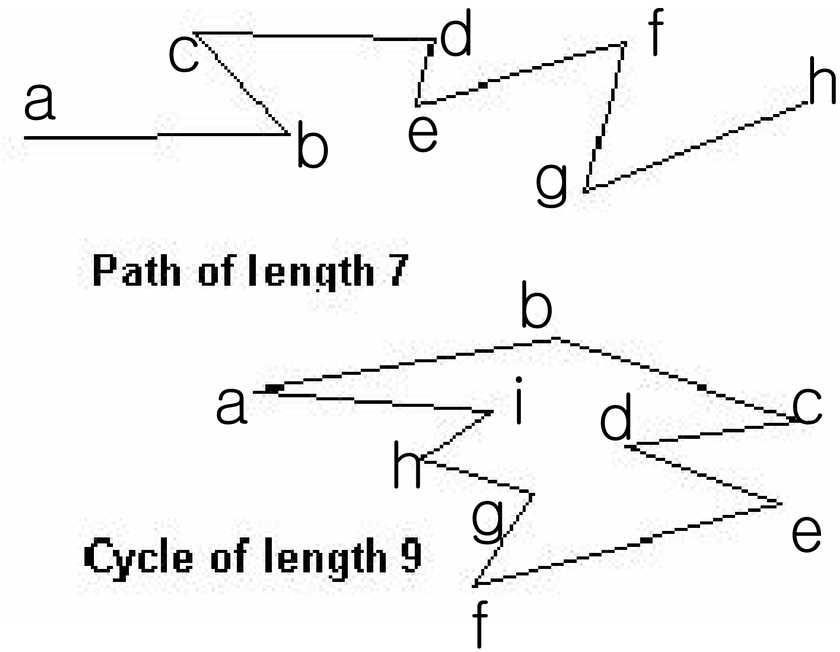
remove 1,2,3,4 from V’, V’={10} C=2

Choose y = 10

remove 10 from V’, V’={} C=3

Therefore C= 3

2. Path and cycle(Circuit)



- A ***path*** *of length n* is a sequence of n + 1 vertices and n consecutive edges

  (n+1 vertices and n edges)

- A ***cycle*** is a path that begins and ends at the same vertex

  ($v_0 = v_{k,}$), (no edges repeated)

- Must have at least 3 edges.

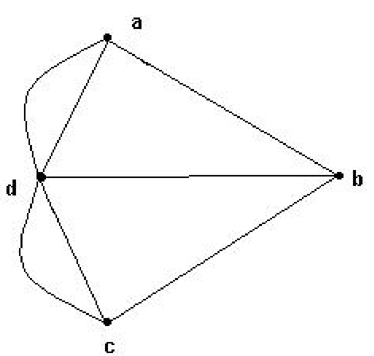
- Acyclic: does not have any cycle

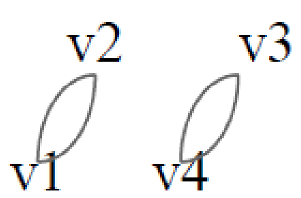
2.1 Euler cycle (Circuit)

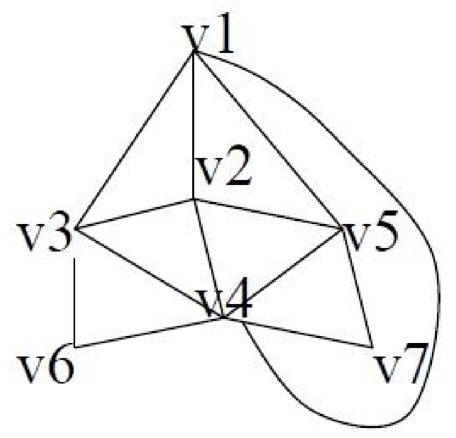
- Definition: Start and ends <u>at the same vertex</u>, uses every vertex <u>at least once</u>, and uses every edge <u>exactly once</u>.

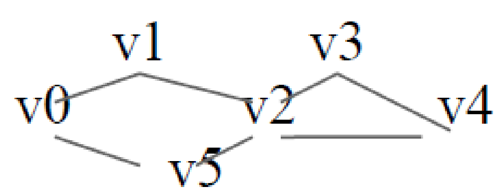
  (G has an Euler cycle if and only if G is connected and all its vertices have even degree)

- Ex) Find Euler Cycle









ex) Let G be a connected graph. Suppose that an edge **e** is in cycle.
Show that G with **e** removed is still connected

(pf) Suppose that e = (v,w) is in cycle. Then there is a path P from v to w not including e.

- Thm: If a graph G has Euler cycle, then G is connected and every vertex has even degree

(pf) Suppose G has an Euler cycle. From the previous argument for the graph,
every vertex in G has even degree.
If v and w are vertices in G, the portion of the Euler cycle that takes us from v to w serves as a path from v to w. Therefore, G is connected.

=> If some vertex of G has odd degree, then the G does not have an Euler cycle.

2.2 Eulerian Path

- Start≠End,
- v(start), w(end) => odd degree (2 vertices have odd degree)
- all other vertices of G have even degree

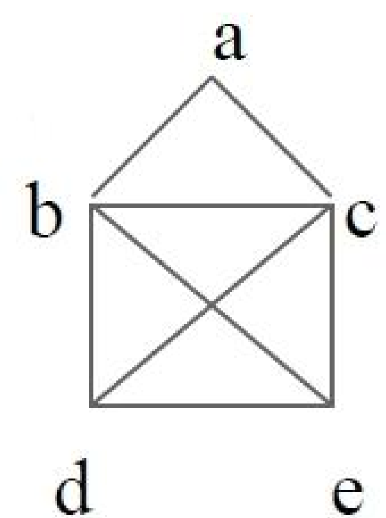
Ex)

d,e => degree 3 (odd degree)

all other => even degree

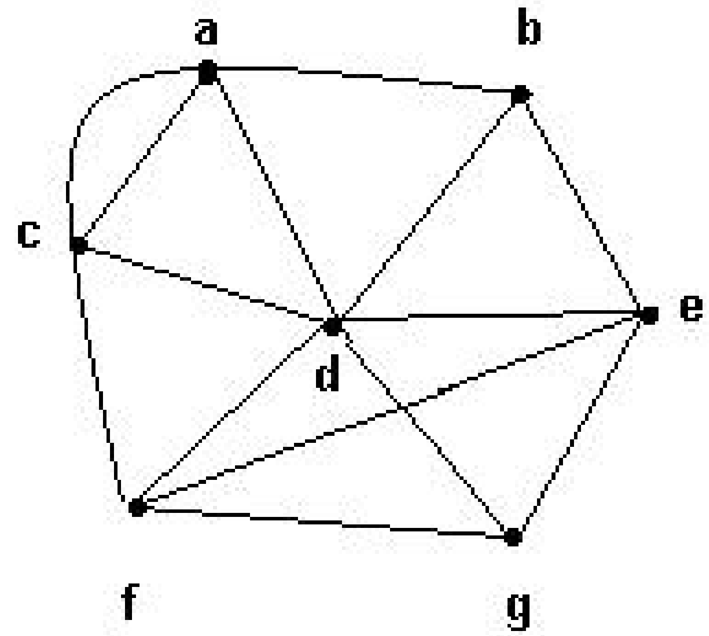
(must start from d or e)

d,e,c,a,b,c,d,b,e => has E. path



Degree of a vertex

- The *degree* of a vertex v, denoted by $\delta(v)$, is the number of edges incident on v



- Example:

$\delta(a) = 4$, $\delta(b) = 3$, $\delta(c) = 4$, $\delta(d) = 6$,

$\delta(e) = 4$, $\delta(f) = 4$, $\delta(g) = 3$.

- $$\sum_{i=1}^{n} \delta(v_i) = 2\,|E|$$

from example, number of E= 14,

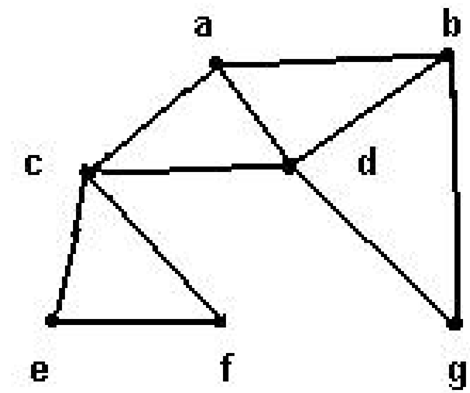
sum of $\delta(v)$ = 28

2.3 Hamiltonian cycles(circuit)

- Definition: Simple circuit that passes through **every vertex exactly once**

  (No rule to find H.C)

- Every known algorithm requires exponential or factorial time in worst case

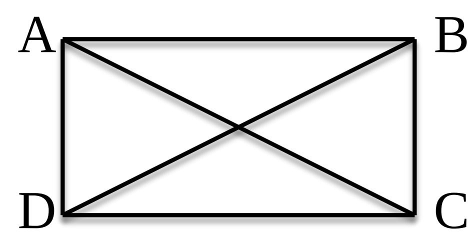


A non-Hamiltonian graph

- If edges are assigned positive weights, and object is to find H.C. of least total weight

  =>*Traveling salesperson problem(TSP)*

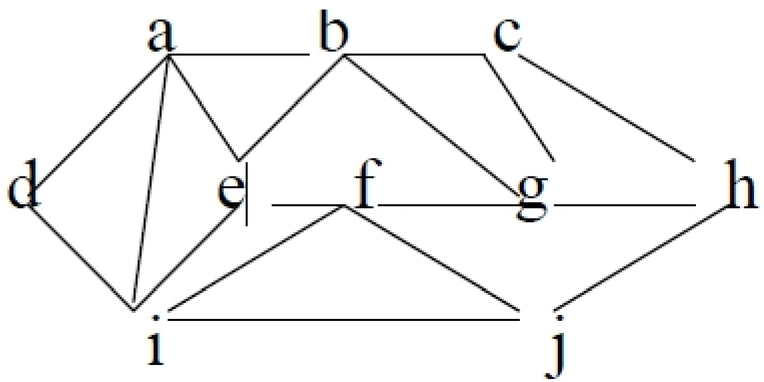
  - Given a weighted graph G, find a minimum- length Hamiltonian cycle in G.
  - No known algorithm to solve TSP in polynomial time.

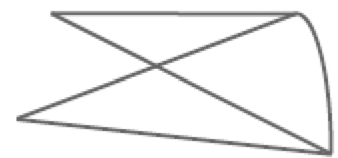


HC => (A, B, C, D, A)

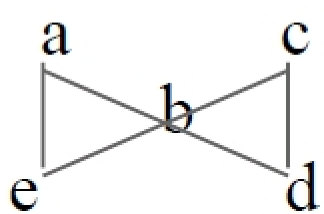
Hamiltonian Cycle (‘cont)

* Find Hamiltonian Cycle





Q: Give an Example of a graph that has an EULER cycle, but contains no H.C



Hamiltonian Cycle ('cont)

- **<u>Nearest Neighbor Method</u>** (sub-optimal solution)

1. Choose any v1 ∈ V
2. v' <- v1
3. w <- 0
4. add v' to list of vertices in path
5. while unmarked vertices remain do

    mark v'

    choose any unmarked vertex u, that is closest to v'

    add u to list of vertices in path

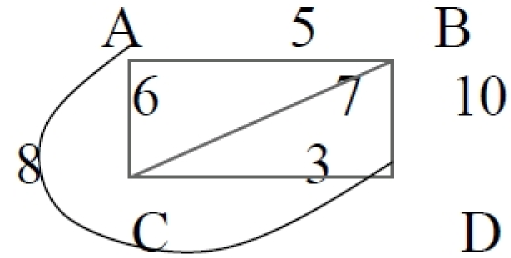
    w <- w + the weight of edge {v',u}

    v' <- u

6. add v' to list of path
7. w<- w+ weight of edge {v',v1}

apply Nearest Neighbor Algorithm

(choose ant vertex , start at B)

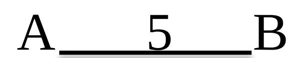


. w = 0 ;   path list = B

. unmarked vertices => A,C,D

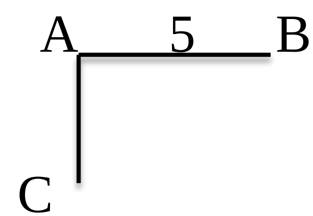
A is closest (since weight = 5)

add A to path (w=5)



mark A

unmarked vertices => C, D, (C is closest)



add C to path (w=5+6) ;   mark C

unmarked => D

add D to path (w=14)

no unmarked vertices

. Add B to path (path = B A C D B) A 5 B   and w = 14 + 10 = 24

Therefore, results & weight = 24.

➔ It may not be best solution!!   (Because there is shorter path, 23)

2.4 A shortest-path algorithm (Dijkstra)

- Dijkstra's algorithm finds the length of the shortest path from a single vertex to any other vertex in a connected weighted graph.

(shortest path => least total weight)

- Weight matrix representation=>

w(x,y) = 1) 0, if x=y

2) ∞, if (x,y) is not an edge

3) the weight of edge(x,y)

* Input: weighted graph (positive)

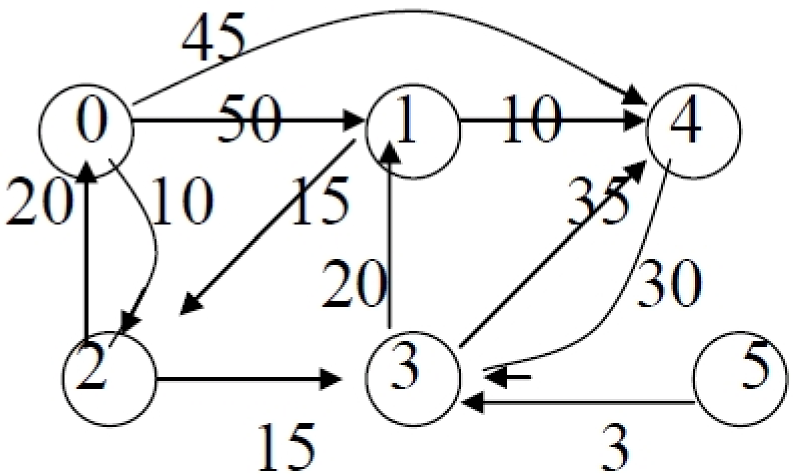
Output: L(z), the length of shortest path from a to z

```
Procedure dijkstra (w, a, z, L)
  L(a):= 0
  for all vertices x ≠ a do
      L(x) = ∞
  T:= set of all vertices

 While z ∈ T do
  begin
    choose v∈T with minimum L(v)
    T:= T - {v}
    for each x∈T adjacent to v do
        L(x) = min{L(x), L(v)+w(v,x)}
   end
end dijkstra
```

if (distance[v] + cost[v,x] <distance[x])
distance[x] = distance[v] + cost[v,x];

Shortest path example



<< 비용인접행렬 >>

	0	1	2	3	4	5
0		50	10		45	
1			15		10	
2	20			15		
3		20			35	
4				30		
5				3		

Vertex	0	1	2	3	4	5
Distance	0	50	10	999	45	999
S	1	0	0	0	0	0

1. S = {*v*0} : 초기는 공백
   distance(1) = 50
   distance(2) = 10 <= *min*
   distance(3) = 999
   distance(4) = 45
   distance(5) = 999

Vertex	0	1	2	3	4	5
distance	0	50	10	999	45	999
S	1	0	1	0	0	0

Shortest path example

2. $S = S \cup \{v2\} = \{v0, v2\}$

distance(1)<- min{distance(1), distance(2)+(*v*2,*v*1,999)} **50**
distance(3)<-min{distance(3), distance(2)+(*v*2,*v*3,15)} **25 <= *min***
distance(4)<- min{distance(4), distance(2)+(*v*2,*v*4,999)} **45**
distance(5)<- min{distance(5), distance(2)+(*v*2,*v*5,999)} **999**

vertex	0	1	2	3	4	5
distance	0	50	10	25	45	999
S	1	0	1	1	0	0

3. $S = S \cup \{v3\} = \{v0, v2, v3\}$

distance(1)<- min{distance(1), distance(3)+*v*3,*v*1,20)} **45 <= *min***
distance(4)<- min{distance(4), distance(3)+(*v*3,*v*4,35)} **45**
distance(5)<- min{distance(5), distance(3)+(*v*3,*v*5,999)} **999**

Vertex	0	1	2	3	4	5
Distance	0	45	10	25	45	999
S	1	1	1	1	0	0

Shortest path example

4. $S = S \cup \{v1\} = \{v0, v1, v2, v3\}$

distance(4) <-min{distance(4), distance(1)+(*v*1,*v*4,10)} **45 <=** ***min***
distance(5)<- min{distance(5), distance(1)+(*v*1,*v*5,999)} **999**

Vertex	0	1	2	3	4	5
dista	0	45	10	25	45	999
S	1	1	1	1	1	0

5. $S = S \cup \{v4\}$

distance(5) <-min{distance(5), distance(4)+(*v*4,*v*5,999)} **999 <=** ***in***

Vertex	0	1	2	3	4	5
Distance	0	45	10	25	45	999
S	1	1	1	1	1	1

6. $S = S \cup \{v5\}$

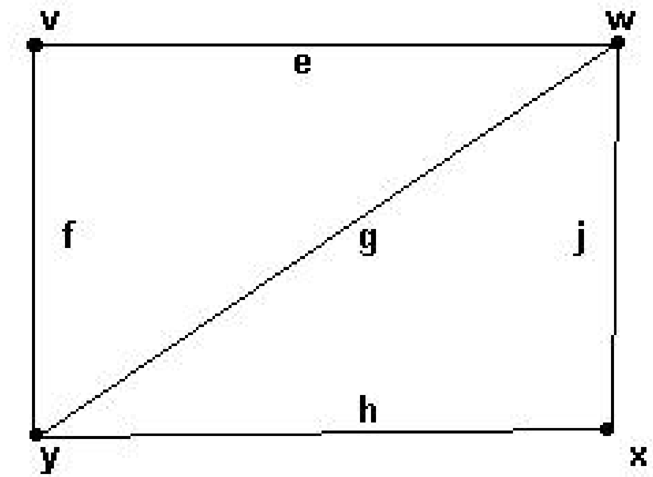
Representations of graphs

- **Adjacency matrix**

  Rows and columns are labeled with ordered vertices

  Write '1' if there is an edge

  '0' if no edge exists



	v	w	x	y
v	0	1	0	1
w	1	0	1	1
x	0	1	0	1
y	1	1	1	0

- **THM:** Let $G = (V,E)$, $n = |V|$

$E^* = E^1 \cup E^2 \cup E^3 \ldots \cup E^n$: composition of relation.

Given E has adjacency matrix M.

We compute M* by successive products of M itself.

$M^* = M^1 \cup M^2 \cup M^3 \ldots \cup M^n$

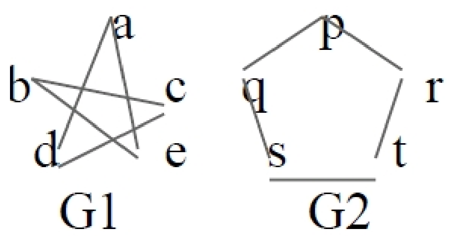
=> Too laborious, ..

3. Graph Properties 3.1 Isomorphic graphs

- DEF: In math terms, when two objects have essentially same structure, they are Isomorphic.
- Finding Isomorphic: 1) G_1 and G_2 are ***isomorphic,*** if there exist one-to-one onto (bijection) functions . Let G1=(V1, E1), G2=(V2, E2) be graphs.

  Let $f: V1 \to V2$, $\forall u,v \in V1$, $\{u,v\} \in E1$, iff $\{f(u), f(v)\} \in E2$

- Ex) Show two graphs are Isomorphic



v	a	b	c	d	e
f(v)	p	s	t	r	q

{u,v}	{f(u), f(v)}
a d	p r
a e	p q
b c	s t
b e	s q
c d	t r

arbitrarily, f(a) = p, a's neighbor -> d,e => f(d) = r
p's neighbor -> q,r, f(e) = q, f(b) = s, since f(e)=q, f(a) = p $\therefore$ f(c) = t

➔ Difficult if complex Graph

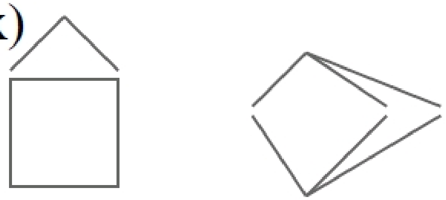
Isomorphic graphs

2) Alternative Method :   G1 and G2 are isomorphic iff their **Adjacency matrices are equal**

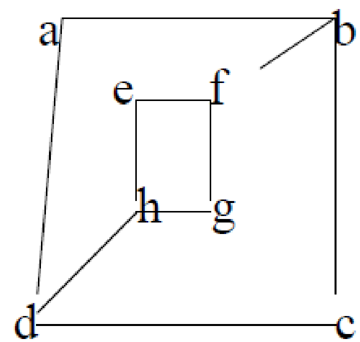
3) Easy way to find NOT isomorphic (Invariant property)

- compare same vertex, edges, if not => NOT isomorphic
- try to find cycles in the graph (try to find some characteristics)

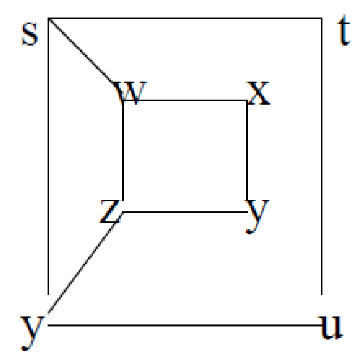
ex)



G1 G2



G



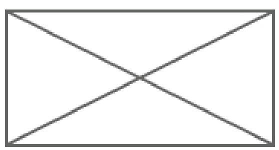
H

G1 has cycle of length3, G2?

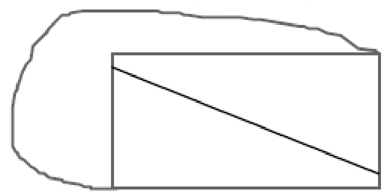
3.2 Planar graphs

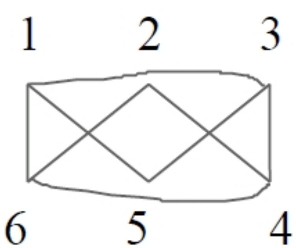
Definition: A graph is *planar* if it can be drawn in the plane without crossing edges

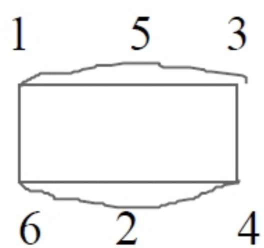
K4(complete graph)

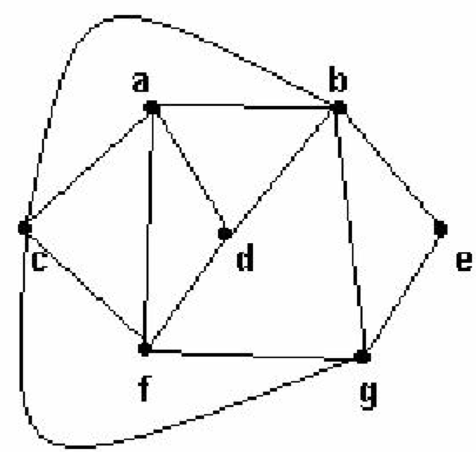


noncrossing edges

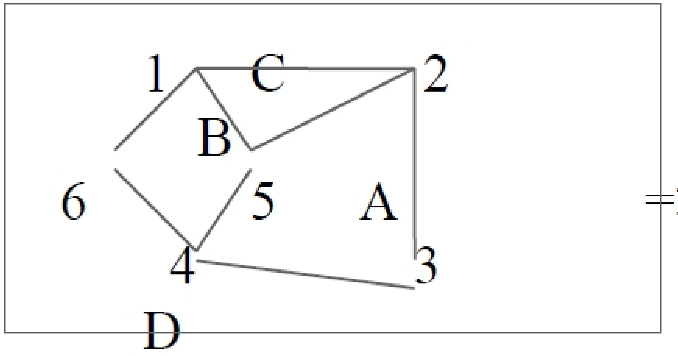








Finite Planar Graph (MAP)

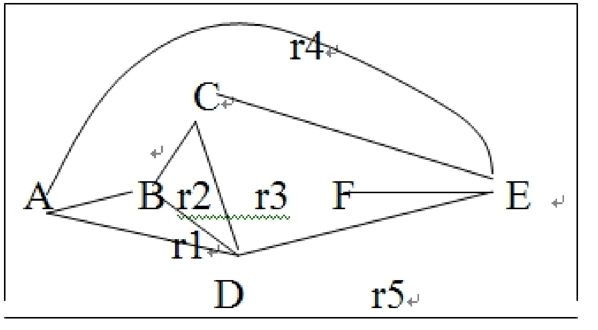


6 vertices , 8 edges
=> divides the plane into 4 regions
(A,B,C,D)

. MAP divides the plane into various regions
=> FACES

. we can assume map is contained in large rectangle
➔ border of each region consists of edges

. Degree of region, deg (r)= length of cycle borders r.



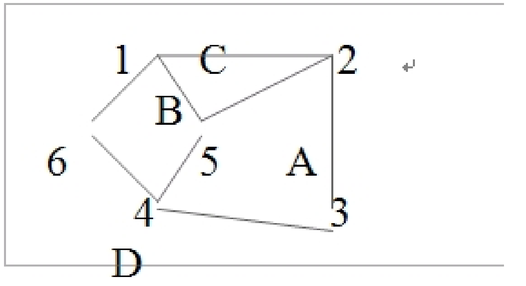
deg(r1) = 3, deg(r2) = 3,
deg(r3)=5, deg(r4)=4,
deg(r5)=3,
sum of deg = 18,
number of edges=9x2= 18

Thm:
The sum of degrees of the regions of map is equal to twice the number of edges ($\Sigma deg(r) = 2E$)

Finite Planar Graph (MAP)

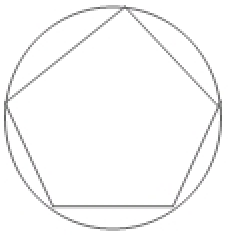
- **Euler's formula: (used to show certain Graph is not planar)**

  f = e - v + 2 (or V-E+F = 2)

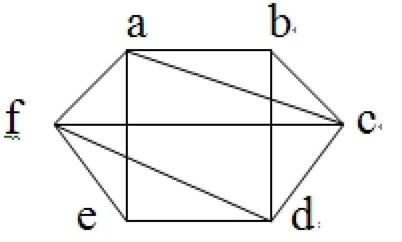


ex) V= 6, E=8,   so, f=8-6+2 => 4 faces(regions)

ex) f = 10-5+2=7



ex)



Ex) A connected planar graph has nine vertices having degrees, 2,2,2,3,3,3,4,4,5.

How many edges are there? How many faces are there?

(sol) 2e = 2+2+2+3+3+3+4+4+5 = 28   so, e=14   f = e-v+2 = 14-9+2 = 7

Ex) Why can there not exist a graph whose degree sequence is 5,4,4,3,2,1?

- (sol) because sum(d) is 19, which is not even,   not 2|E|

Non Planar Graph

ex) Show that K3,3 is not planar

(sol) $V=6$, $E= 9$ edges, by Euler's formula, F must be 5 ($F= 9 - 6 + 2 \Rightarrow 5$)

but no three vertices connected together, so degree of each regions 4 or more

and total degree is => (5 regions X 4 or more degree/region) = 20 or more degrees. but by the theorem " sum of degrees = 2E = 18", Graph has only 9 edges, which is contradiction

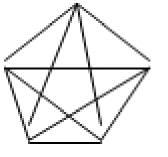
- **Thm: if G is connected & $V \geq 3$ then, $E \leq 3V-6$**

(pf) $V-E+F=2$, ($\Sigma deg(r) = 2E$) , $2E \geq 3F$ (each region's degree ≥degree 3)

$F \leq 2E/3$, $2 \leq V-E + 2E/3$ ($v-e+f=2$에서), $2 \leq V-E/3 \Rightarrow E \leq 3V-6$

ex) $V=5$, $E=10$, $3V-6=3x5-6=9$ $E=10$, $10 \leq 9$

The graph is not Planar

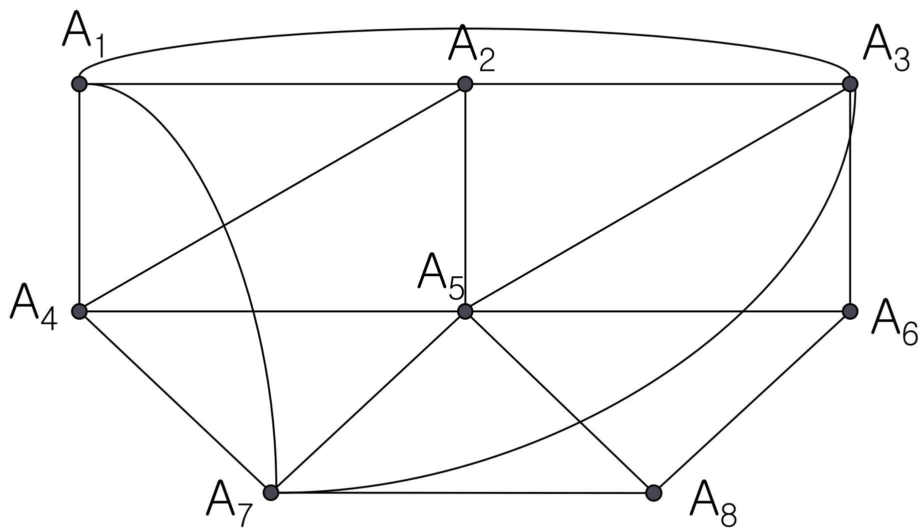


3.3 Vertex Coloring (정점의 착색)

- 두 개의 서로 다른 인접한 정점이 같은 색을 갖지 않으면 이러한 착색을 정점 착색(vertex coloring)이라 한다
- N가지 색으로 color가 가능하다면 N-colorable이라고 하고, 최소 색의 가지수를 착색수(chromatic number)라 하고, X(G)로 표기함.

- **Welch-Powell의 알고리즘**
  - (1) *G*의 정점의 차수가 내림차순이 되게 배열한다(이 배열은 차수가 같은 정점이 여러 개 있을 수 있으므로 몇 가지 다른 순서가 존재할 수 있다).
  - (2) 배열의 첫 번째 정점은 첫 번째 색으로 착색하고 계속해서 배열의 순서대로 이미 착색된 정점과 **인접하지 않은 정점을** 모두 같은 색으로 착색한다.
  - (3) 배열에서 착색되지 않은 다음 정점을 **두 번째 색으로** 착색하고 배열의 순서대로 이미 착색된 정점과 인접하지 않은 정점을 모두 착색한다.
  - (4) 계속해서 위의 과정을 그래프의 모든 정점이 착색될 때까지 반복한다.

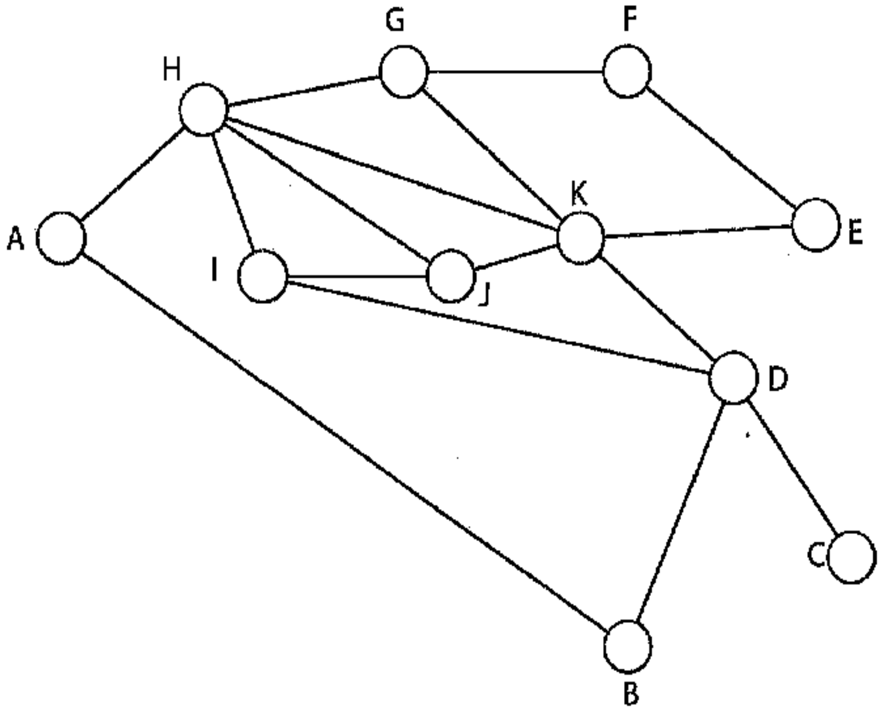
Ex) 다음 그래프를 Welch-Powell 알고리즘으로 착색하라.



[**sol**] 정점들을 내림차순으로 배열하면 $A5, A3, A7, A1, A2, A4, A6, A8$ 이다. $A5$와 $A1$이 먼저 첫 번째 색으로 착색되고, 다음 $A3, A4, A8$이 두 번째 색으로 착색되며, 마지막으로 $A7, A2, A6$이 세 번째 색으로 착색된다.

따라서 그래프는 3-색이 (3-colorable) 가능하다.

Coloring example



Vertex	Valence
A	2
B	2
C	1
D	4
E	2
F	2
G	3
H	5
I	3
J	3
K	5

H, K, D, G, I, J, A, B, E, F, C