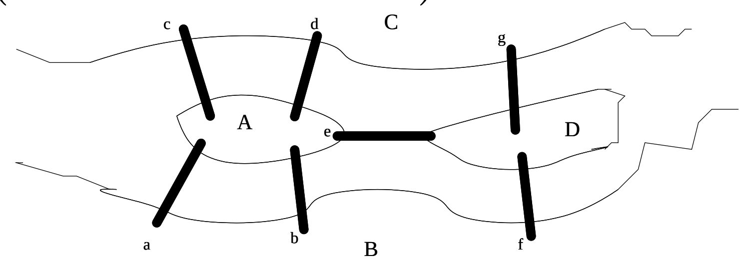
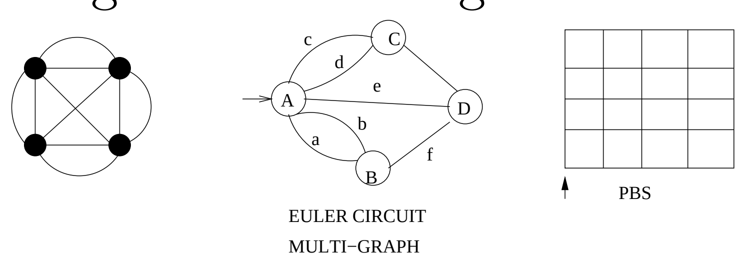
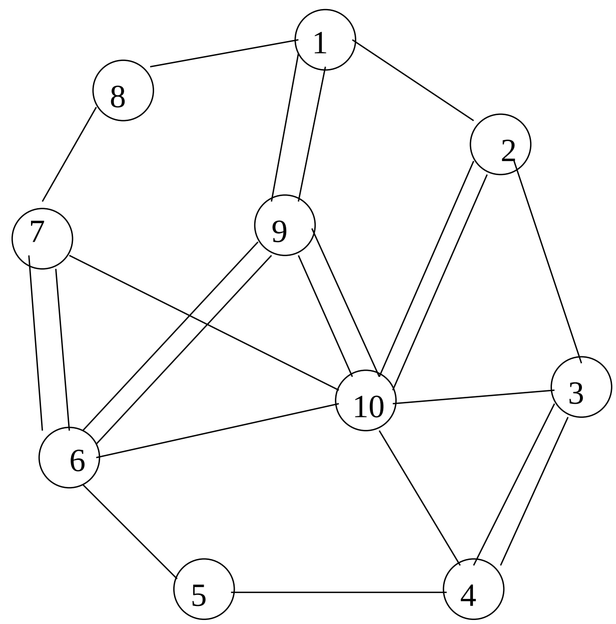
GRAPHS

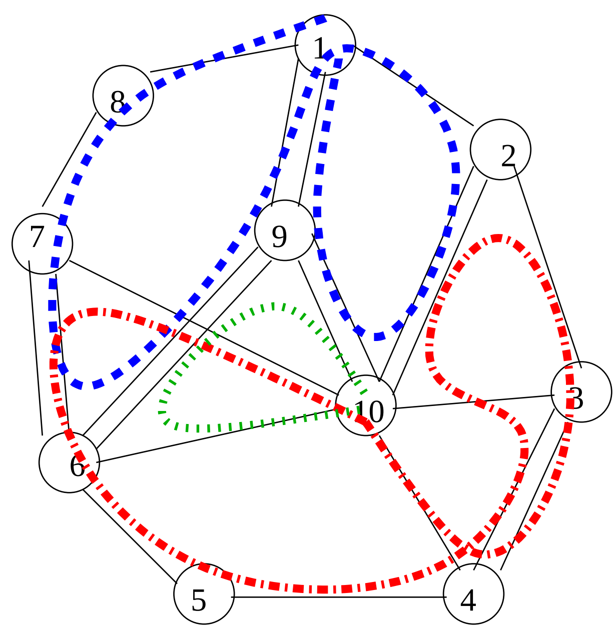
- Introduced by Euler 1736, Koeingsberg bridge problem (EAST PRUSSIA)



- Problem: Starting at some land area, is it possible to walk across all the bridges exactly once returning to the starting land area?




1
8
2
7
9
10
3
6
5
4


1
8
2
7
9
10
3
6
5
4

Theorem

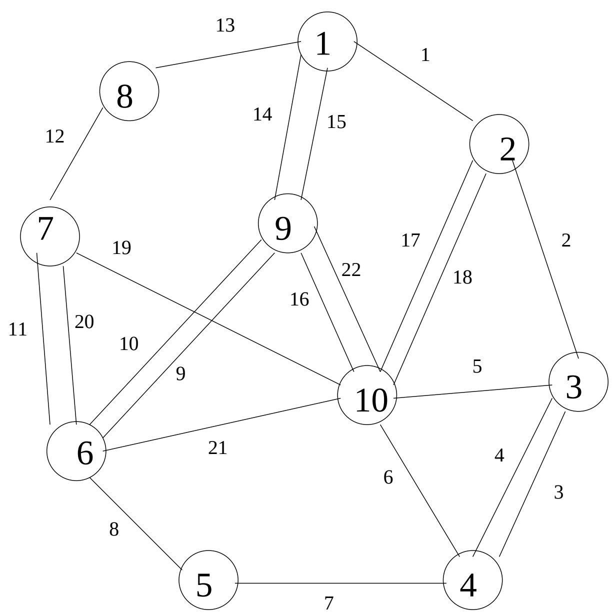
Multi-Graph G has an euler circuit if and only if G is connected and every node is of even degree.

Algorithm by Stephen Barnard. (Discuss Quickly)

```
function EULER(v:vertex) Returns Path
  { path:=NULL;
    for all vertices w adjacent to v
        and edge(v,w) not yet used do
      { mark (v,w) used;
        path:={(v,w)} || EULER(w) || path;
        // concatenate represented by ||
      }
    return path;
   }
```

C++ code appears elsewhere.

Possible Execution of Algorithm



The labels in the edges indicate the order in which they are used in the recursive calls. Next slide gives more details of the recursive calls.

Recursive Calls

E(1), E(2), E(3), E(4), E(3), E(10), E(4), E(5), E(6), E(9), E(6), E(7), E(8), E(1), E(9), E(1), E(10), E(2), E(10), E(7), E(6), E(10), E(9)

1st Iteration

edge label	(v,w)	Path
1	(1,2)	(1,2) (2,3) (3,4) (4,3) P_4
2	(2,3)	(2,3) (3,4) (4,3) P_4
3	(3,4)	(3,4) (4,3) P_4
4	(4,3)	(4,3) P_4
5	(3,10)	(3,10) (10,4) (4,5) (5,6) P_3
6	(10,4)	(10,4) (4,5) (5,6) P_3
7	(4,5)	(4,5) (5,6) P_3
8	(5,6)	(5,6) P_3
9	(6,9)	(6,9) (9,6) (6,7) (7,8) P_2
10	(9,6)	(9,6) (6,7) (7,8) P_2
11	(6,7)	(6,7) (7,8) P_2
12	(7,8)	(7,8) P_2
13	(8,1)	(8,1) (1,9) (9,10) P_1 (9,1)
14	(1,9)	(1,9) (9,10) P_1 (9,1)
15	(9,1)	
	∅	
17	(10,2)	(10,2) (2,10) (10,7) (7,6) (6,10) (10,9)
18	(2,10)	(2,10) (10,7) (7,6) (6,10) (10,9)
19	(10,7)	(10,7) (7,6) (6,10) (10,9)
20	(7,6)	(7,6) (6,10) (10,9)
21	(6,10)	(6,10) (10,9)
22	(10,9)	(10,9)
	∅	

2nd Iteration

edge Label	(v,w)	Path
⋮		
16	(9,10)	P_1 (9,1)

Algorithm Returns

These are used to reduce the amount of writing:

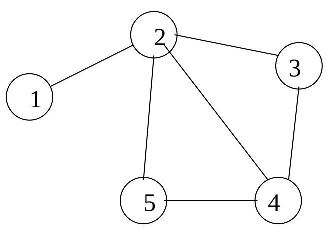
- Let P_1 = (10,2) (2,10) (10,7) (7,6) (6,10) (10,9)
- Let P_2 = (8,1) (1,9) (9,10) P_1 (9,1)
- Let P_3 = (6,9) (9,6) (6,7) (7,8) P_2
- Let P_4 = (3,10) (10,4) (4,5) (5,6) P_3

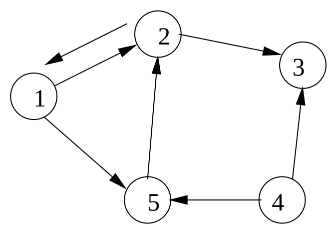
GRAPHS

Set of nodes (points or vertices)

Set of edges (lines or arcs)

UNDIRECTED ***DIRECTED***



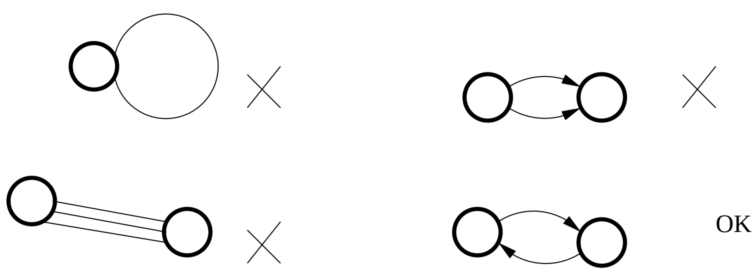


V={1,2,3,4,5}

E={{1,2},{2,3},{2,4},{2,5},{3,4},{4,5}}

V={1,2,3,4,5}

E={(1,2),(2,3),(4,3),(4,5),(5,2),(1,5),(2,1)}

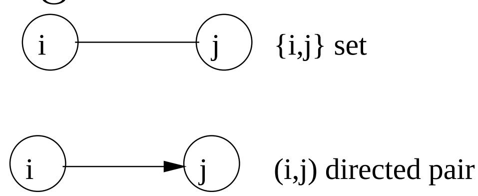


Definition

Graph $G = (V, E)$

V: Set of vertices

E: Set of edges



- $\{i, j\}$: Undirected
  - i and j are adjecent
  - $\{i, j\}$ is incident on vertices i and j
- (i, j): Directed
  - (i, j) is incident to vertex j and incident from i
  - i is adjecent to vertex j
  - j is adjecent from vertex i

No mutiple copies of edges

No self edge.

GRAPHS

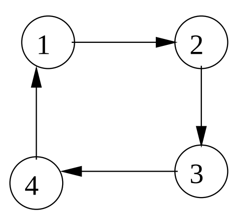
- A sequence of vertices $P = i_1, i_2, \ldots, i_k$ is an i_1 to i_k path if and only if $(i_j, i_{j+1}) \in E$ for every $1 \leq j < k$.
- Simple path: All vertices, except possibly for the 1^{st} and last, are different.
- Length of a path: # of edges in the path.

Representation

Adjacency Matrix

- (v_i, v_j) if and only if $A_{i,j} = 1$
- Space n^2 bits

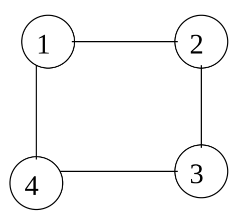
Directed Case



Bit Matrix

A	1 2 3 4
1	0 1 0 0
2	0 0 1 0
3	0 0 0 1
4	1 0 0 0

Undirected Case

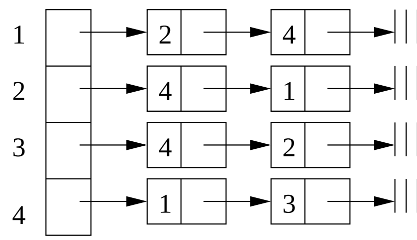


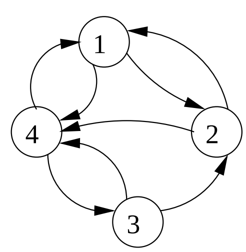
Bit Matrix

A	1 2 3 4
1	0 1 0 1
2	1 0 1 0
3	0 1 0 1
4	1 0 1 0

Adjacency Lists

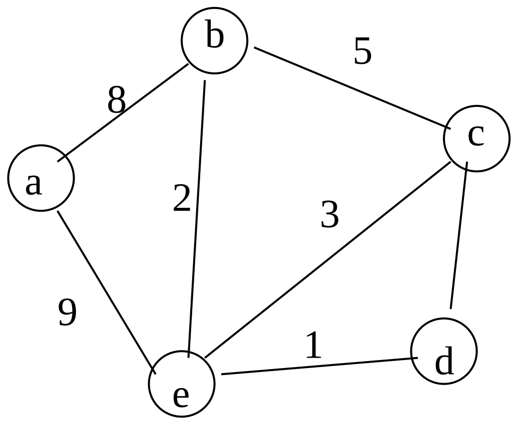
- Space $O(n + e)$

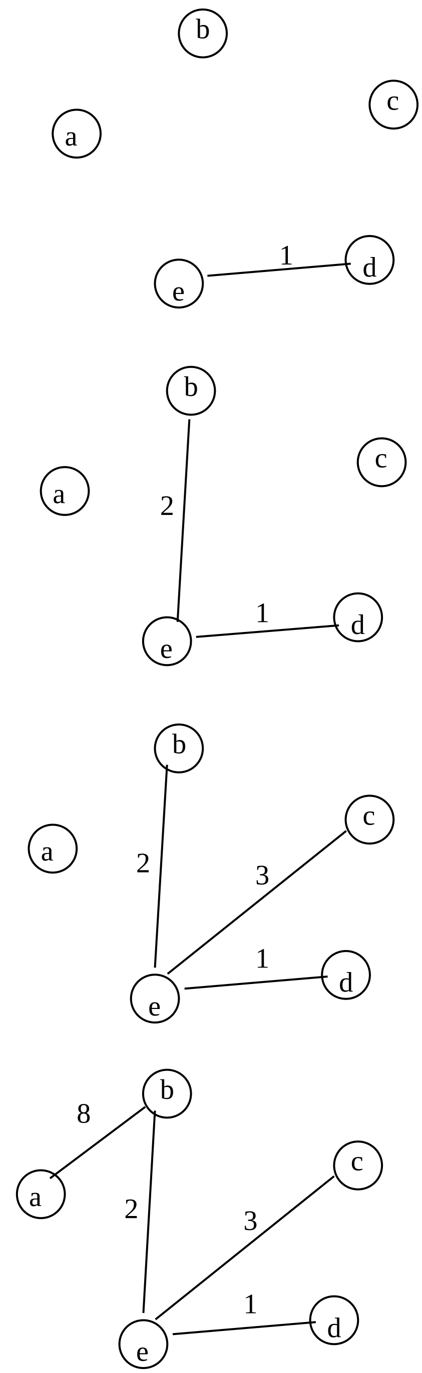




Minimum Cost Spanning Tree (MCST)

- Definition of Spanning Tree: Let $G = (V, E)$ be an undirected connected graph. A subgraph $T = (V, E')$ of G is a <u>spanning tree</u> if and only if T is a tree.
- Definition of Minimum Cost Spanning Tree: Let $G = (V, E)$ be an undirected connected graph and $w : E \rightarrow I^+$. A subgraph $T = (V, E')$ of G is a <u>minimum cost spanning tree</u> if $\sum_{(i,j) \in E} w(i, j)$ is minimum and T is a tree.
- Kruskal Algorithm: Greedy method. Add lowest cost edge that does not create a cycle.





{d,e} ✓

{b,e} ✓

{c,e} ✓

{b,c} ✗

{c,d} ✗

{a,b} ✓

Kruskal's Algorithm

```
n <- |V|; T <- NULL; E <- Set of edge in G;
while |T|<n-1 do
  e <- Deletemin(E);
  add e to T if it does not create a cycle
endwhile
---------
More Details
---------
Initialize Priority Queue(Q)
ADD all edges in G to priority queue
       Q(i,j,w(i,j));
T:=Empty;
while |T|<n-1 do
  {i,j} <- Deletemin(Q)
  if {T plus {i,j}} is not a cycle
             then add {i,j} to T
endwhile
```

Kruskal's Algorithm (refined)

```
/* G is connected*/
Initialize Union-Find(1..n):
           Priority Queue(Q)
Add all edges in G to priority queue Q
    as triplets (i,j,W(i,j))
    /* W(i,j) is the key for comparison */
T <- NULL;
while |T|<n-1 do
  {i,j} <- Deltemin(Q);
  I <- Find(i);
  J <- Find(j);
  if I!=J then Add {i,j} to T;
               Union(I,J);
  endif
endwhile
```

Time Complexity $O(e \log e) \rightarrow O(e \log n)$.