graphs

set of vertices and edges

$G = (V, E)$

V: verticles — nodes

E: edges (pairs) — arcs

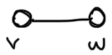
directed graph: if the edge pair is ordered (digraphs)

undirected graph: normal graph, not directed

adjacent: komşu
ajeysın

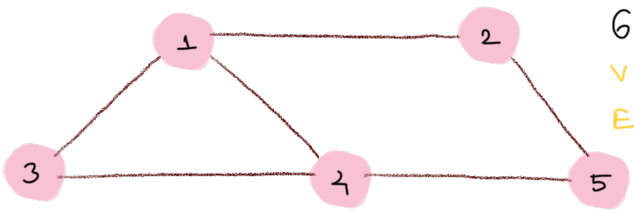
definitions

- adjacent:



- w is adjacent to v iff $(v,w) \in E$
- in undirected graph, if v is adjacent to w, then w is adjent to v too

- path: between two vertices, sequence of edges that begins at one vertex and ends at another vertex
  - ↳ simple path: passes through a vertex only once
  - ↳ cycle: is a path that begins and ends at the same vertex
  - ↳ simple cycle: is a cycle that does not pass through other vertices more than once.



$G = (V, E)$

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

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

adjacent: 1 and 2

path: 1,2,5 (simple) — 1,3,4,1,2,5 (not simple)

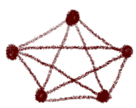
cycle: 1,3,4,1 (simple) — 1,3,4,1,4,1 (not simple)

- connected graph: has a path between each pair of distinct vertices


connected


disconnected

- complete graph: has an edge between each pair of distinct vertices


(it is also connected graph)

directed graph (digraphs)

if the edge pair is ordered.

- if there is a direct edge from v to w

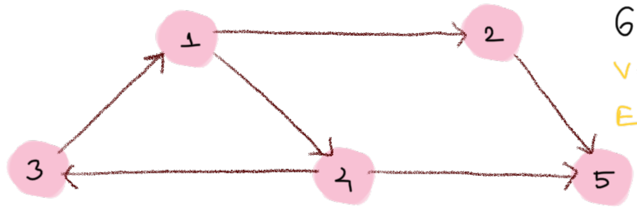
↳ w is successor of v

↳ v is predecessor of w

DAG = directed acyclic grap that has no cycle

strongly conneeted: if there is a path from every vertex to every other vertex (when the graph is undirected, it is called connected)

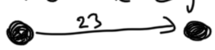
• if a directed graph is not strongly connected, but it is connected then it is called weakly connected. 



$G = (V, E)$
$V = \{1, 2, 3, 4, 5\}$
$E = \{(1,2), (1,4), (2,5), (3,1), (4,3), (4,5)\}$

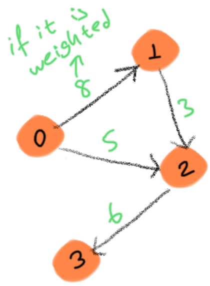
adjacent: 2 adjacent to 1, but 1 is not adjacent to 2.
path: 1, 2, 5 (a directed path)
cycle: 1, 4, 3, 1 (a directed cycle)

• weighted graph: if we label the edges of a graph with numerical values.



graph implementations

① adjacency matrix: two dimensional array



directed

	0	1	2	3
0		•8	•5	
1			•3	
2				•6
3				

undirected

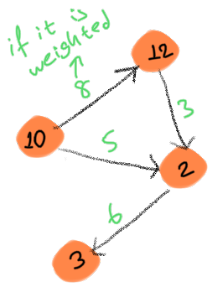
[i] \ [j]	0	1	2	3
0		•	•	
1	•		•	
2	•	•		•
3			•	

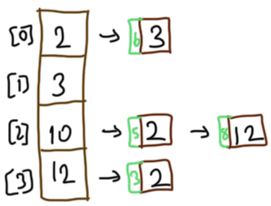
symmetrical

• n vertices, nxn matrix
• matrix[i,j] true if i→j
• matrix[i,j] = weight (if)
• space requirement $O(|V|^2)$→

good: determine whether there is an edge from vertex i to vertex j. $O(1)$
bad: find all vertices adjacent to a given vertex; $O(n)$
it is better if the graph is dense (sık, yoğun)

② adjacency list: for every vertex we keep a list of adjacent vertices





• consist of n linked list
• it is better if the graph is sparse (seyrek)
• space requirement $O(|E| + |V|) \to n$

bad: determine whether there is an edge from vertex i to vertex j. $O(n)$
good: find all vertices adjacent to a given vertex; $O(n)$

graph traversals

- starts from a vertex, visits all of the vertices that can be reachable from that vertex
- visits all if the graph is connected
- must mark each visited vertex to not to get into a infinite loop.

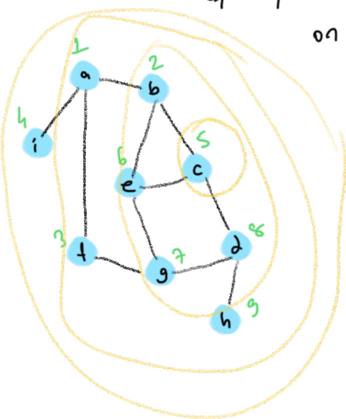
① breadth-first search (traversal) (en) (level order traversal in tree)

after visiting given vertex v, then visit every vertex adjacent to v

code

```
create_queue()
enqueue(v)
mark v as visited;
while( q is not empty)
        dequeue(w);
        for (each unvisited vertex
             u adjacent to w)
               mark u as visited;
               enque (u);
```

- it is useful for finding the shortest path on unweighted graphs



layer by layer

- O(V+E) —linear

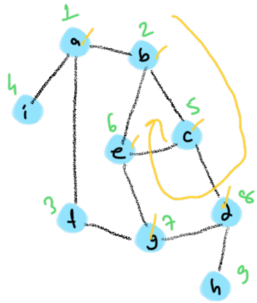
② depth-first search (traversal) (inorder traversal in tree)

a path from v as deeply into the graph as possible before backing up.

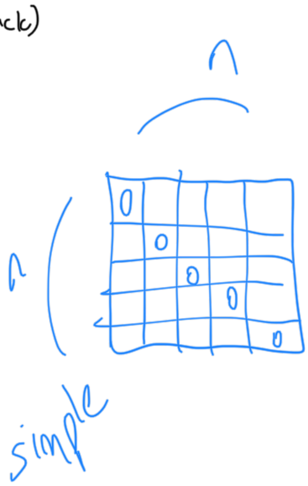
code

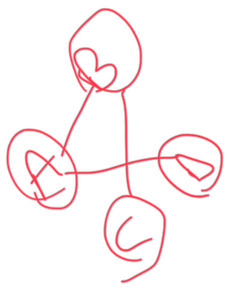
```
func(v)
  mark v as visited;
  for(each unvisited vertex
      u adjacent to v)
     fun(u)
```

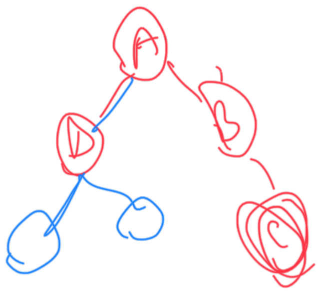
(using stack)

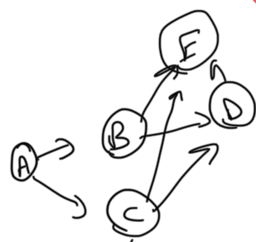


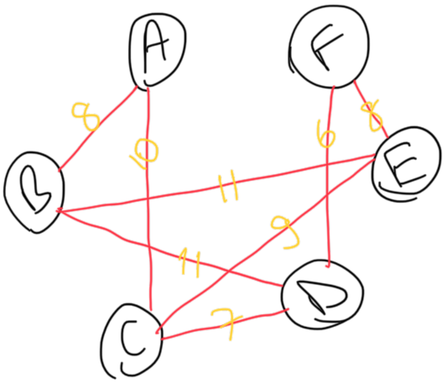
count connected nodes
determine connectivity
find bridges











							16
(16)	F		14	23		7	16
(22)	D	0	14	23	22	7	16
(23)	C	0	14	23	22	7	16