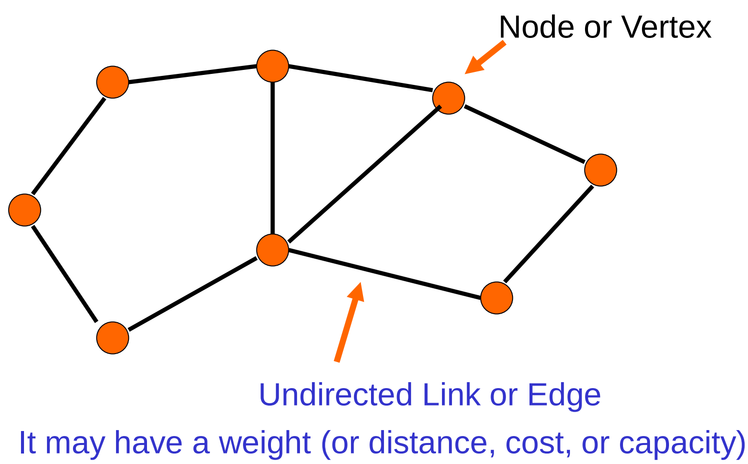
Graphs

- Definitions
- Special graphs
- Data structures
- Reading Assignment CLRS textbook
  - Appendix B.4 and B.5
  - Chapter 10
  - Chapter 3 (in case you forgot)

Definitions

- Undirected graph
  - Notation
- Directed graph
  - Notation

Undirected Graph



Why are we interested? Some examples:

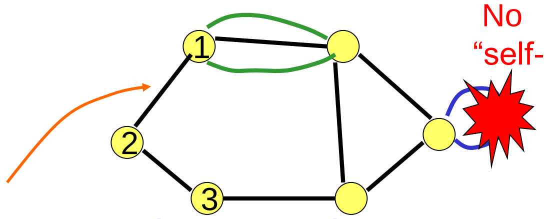
- Is the graph connected?
- Find a path between two nodes.
- Find a “shortest” (or least costly) path between two nodes.
- Find a subgraph that includes all the nodes.

Undirected Graph: Definitions

Simple graphs have at most one edge between a pair of nodes

Multigraphs may have more.

Node 2 is *adjacent* to 1 and 3, i.e., are *neighbors*



No "self-loops"

Edge (1,2) is *incident* to node 2

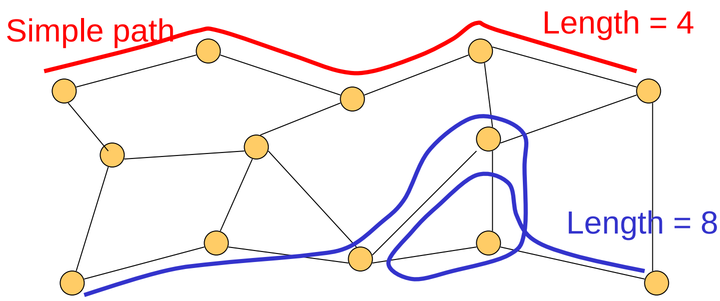
degree 2

degree 3

Undirected Graphs: Definitions

Path = a sequence of nodes, where consecutive nodes have edges between them. *Length = # hops*

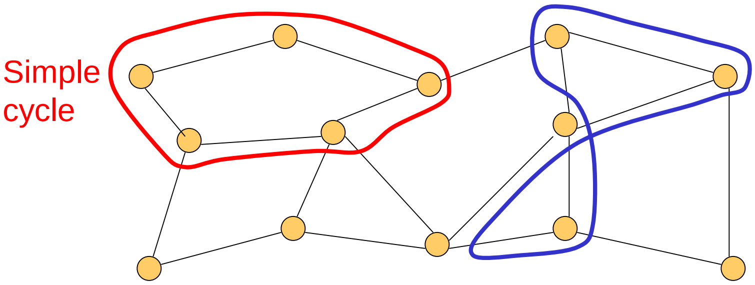
Simple path = path with distinct nodes.



Undirected Graphs: Definitions

Cycle = a path, but where the first and last nodes are the same.

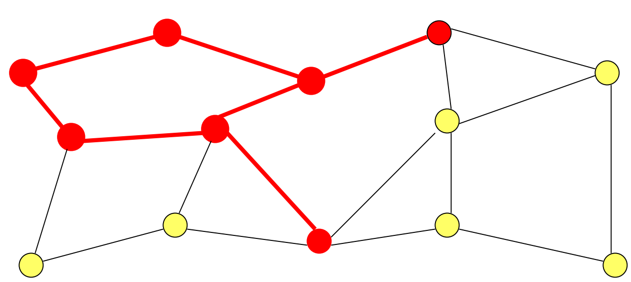
Simple cycle = a cycle, but where all intermediate nodes are distinct.



Undirected Graphs: Definitions

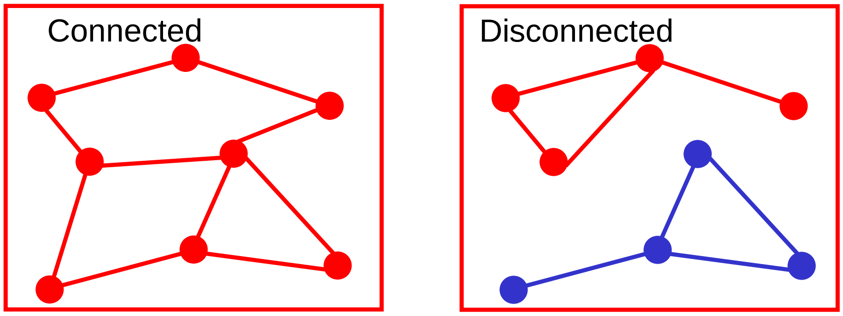
A subgraph is a graph that is a subset of a graph.

The red nodes induce the red subgraph



Undirected Graphs: Definitions

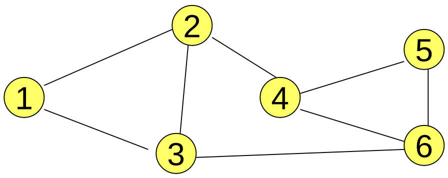
A graph is connected if all pairs of nodes have a path between them.



Undirected Graphs: Notation

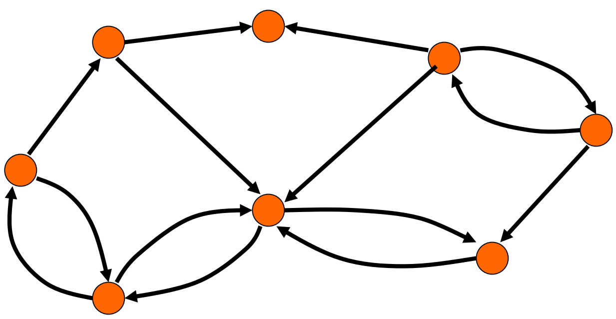
- **Graph G** = (*V*,*E*)
- **Nodes V**. Typically, {1, 2,...., |*V*|}.
  |*V*| = size or "cardinality" of set V.
- **Edges E**. An edge between nodes *u* and *v*: *(u,v)*. *Note (u,v) = (v,u).*
- **Path**: <v0, v1, ..., vk>, a sequence of nodes.

Notation



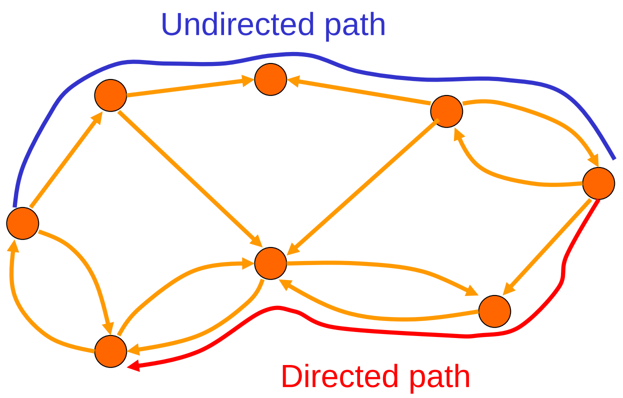
- V = {1, 2, 3, 4, 5, 6}
- E = {(1,2), (1,3), (2,3), (2,4),
  (3,6), (4,5), (4,6), (5,6)}

Directed Graphs



Directed edges or links

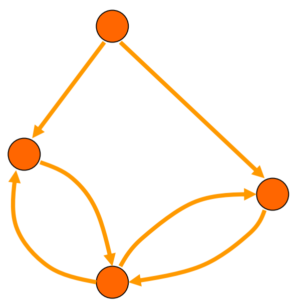
Directed Graphs



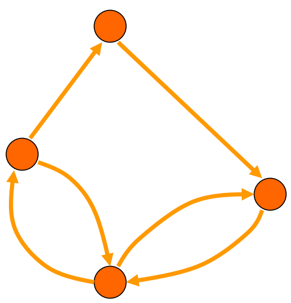
Directed paths follow the direction of edges

Directed Graphs

Strongly connected = directed paths between any (ordered) pair of nodes



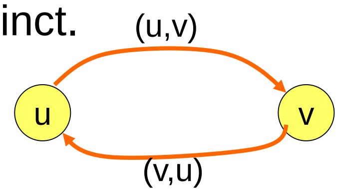
Connected, but not strongly



Strongly connected

Directed Graphs: Notation

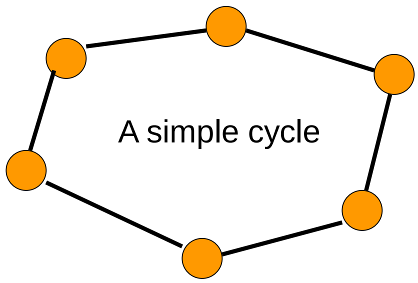
- Extension of notation for directed graphs
- (u,v) is not the same as (v,u) if u and v are distinct.



Special Graphs

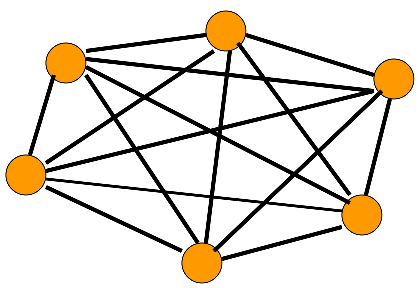
- Rings
- Complete Graphs
- Bipartite Graphs
- Trees

Ring



|V| = |E|

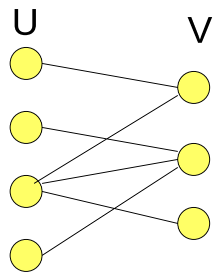
Complete Graph



Every pair of nodes has a link

Bipartite Graph

B = (U,V,E)

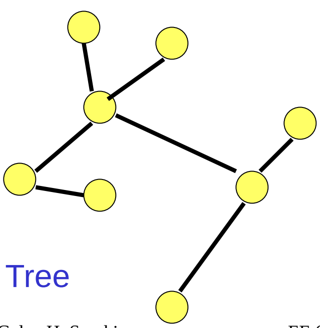


- U has no edges between them
- V has no edges between them
- B has only even cycles

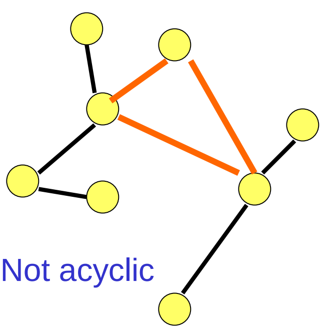
Example Application: Matchmaking
U = grooms, V = brides

Trees

(No cycles)
A tree is a connected, acyclic, undirected graph



Tree



Not acyclic

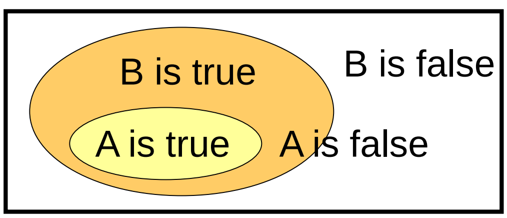
Trees

Theorem. Let G be an undirected graph. The following statements are equivalent

- G is a <u>tree:</u> a connected acyclic graph
- Any two vertices in G are connected to by a unique simple path
- G is connected, but if any edge is removed from E, the resulting graph is disconnected
- G is connected, and |E| = |V|-1
- G is acyclic, and |E| = |V|-1
- G is acyclic, but if any edge is added to E, the resulting graph contains a cycle.

Proof by Contradiction: We'll use this

Given A is true then B is true

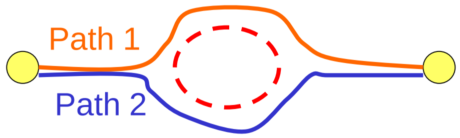


Given B is false then A is false

Properties of Trees

- G is a <u>tree</u> (connected and acyclic)
- Any two vertices in G are connected to by a unique simple path

1. Connectedness implies there is a simple path
2. There is unique path (*proof by contradiction*):
   Suppose there are multiple paths

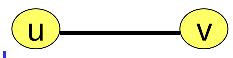


Not acyclic
Contradiction!

Properties of Fat Trees

- Any two vertices in G are connected to by a unique simple path
- G is connected, but if any edge is removed from E, the resulting graph is disconnected

1. G is connected is immediate
2. Arbitrary edge



- Unique path between u and v
- Delete it means u and v are disconnected

Properties of Fat Trees

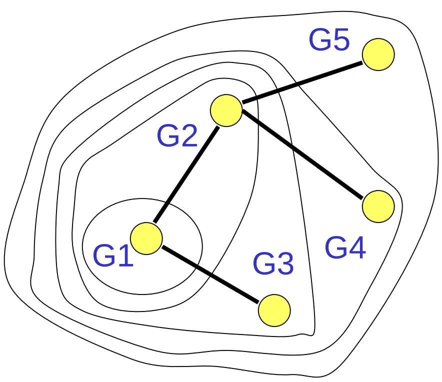
- G is connected, but if any edge is removed from E, the resulting graph is disconnected
- G is connected, and $|E| = |V|-1$

G is connected is immediate

We can grow a subgraph Gn (n = |V|-1)
of G that contains all the nodes and has |V|-1 edges

Properties of Trees

We can grow a subgraph Gn of G that contains all the nodes and has |V|-1 edges



Show an algorithm that can create Gn
- G1 = some node
- for k = 1, ..., |V|-2 do
  Find an edge (u,v) where u in Gk and v is not
  Gk+1 = Gk U {v}

We can do this since G is connected

Gn, n = |V|
- Connects all nodes
- Has |V|-1 edges

Properties of Trees

- *G* is connected, but if any edge is removed from *E*, the resulting graph is disconnected
- *G* is connected, and $|E| = |V|-1$

G_n, $n = |V|$
- connects all nodes
- has $|V|-1$ edges

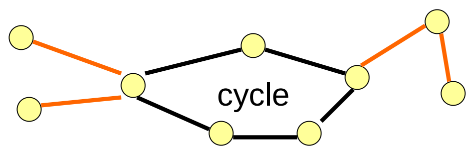
$G = G_n$, i,e, no extra edges. Why?
- Removing any extra edge will leave G still a superset of Gn and so G remains connected
- Thus, G has |V|-1 edges.

Properties of Trees

- G is connected, and |E| = |V|-1
- G is *acyclic*, and |E| = |V|-1

Proof by contradiction: we'll show G is acyclic

Suppose G has a cycle of length (size) k



Grow subgraph from cycle

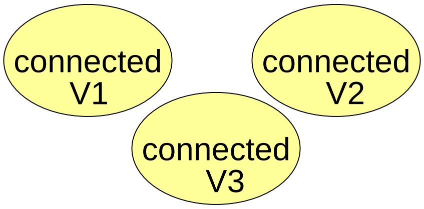
Cycle: k nodes, k edges

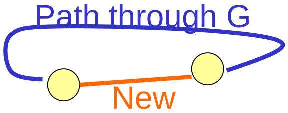
The rest: |V|-k nodes
|V|-k edges

Total edges ≥ |V|
Contradiction!

Properties of Trees

- G is acyclic, and |E| = |V|-1
- G is acyclic, *but if any edge is added to E, the resulting graph contains a cycle*.





Any new edge causes a cycle.

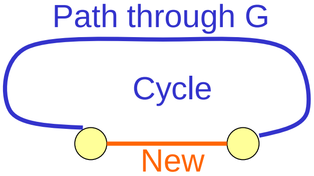
All connected and acyclic --> trees!

|E| = |V1| - 1
+ |V2| - 1
+ |V3| - 1
= |V| - 3

Since |E| = |V|-1 it implies one component
G is connected

Properties of Trees

- G is acyclic, but if any edge is added to E, the resulting graph contains a cycle.
- G is a fat tree (i.e., acyclic and *connected*)



- Suppose the graph is disconnected
- Then there are two nodes u and v without a path between them
- Adding the link (u,v) causes a cycle
- Thus, the nodes have two paths
  - {u,v}
  - Another path from u to v
- The other path already existed in G
- Contradiction

Trees

Theorem. Let G be an undirected graph. The following statements are equivalent

- G is a tree
- Any two vertices in G are connected to by a unique simple path
- G is connected, but if any edge is removed from E, the resulting graph is disconnected
- G is connected, and |E| = |V|-1
- G is acyclic, and |E| = |V|-1
- G is acyclic, but if any edge is added to E, the resulting graph contains a cycle.

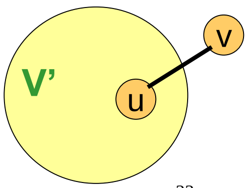
Facts About Trees and Rings

- A *spanning tree* T of a graph G is a subgraph that is a tree that contains all the nodes
- A tree is a connected graph, but one edge deletion makes it disconnected
- A tree has |E| = |V|-1
- Ring has |E| = |V|, and will survive any single node or edge deletion.
  Survive means remains connected

Spanning Trees

Grow-Tree
Input: G = (V,E)
Output: (V',E') a spanning tree in G if one exists

1. V' = v, some node in V. E' = empty set.
2. **while** V' not equal to V
3. **do** find an edge (u,v) such that u is in V' and v is not
4. add u to V'
5. add (u,v) to E'



Spanning Trees

Theorem. Grow-Tree finds a spanning tree in G if G is connected.

- Initially (V',E') is connected (trivial, single node)
- Since G is connected, we can always find a (u,v) leaving (V',E').
- Adding the edge (u,v) to (V',E') keeps it connected and (V',E') has an additional node.
- When V' = V, what are we left with?
  - (V',E') is connected and has |V|-1 edges.
  - Thus, it is a spanning tree.

Time Complexity

Grow-Tree

Input: G = (V,E)

Output: (V',E') a spanning tree in G if one exists

1. V' = {v}, some node in V. E' = empty set.
2. **while** V' not equal to V
3. **do** find an edge (u,v) such that u is in V' and v is not
4. add u to V'
5. add (u,v) to E'

O(f(V))

Time Complexity = O(|V|f(V))

Data Structures

- Arrays
- Link lists
- Adjacency matrix
- Adjacency list

Arrays

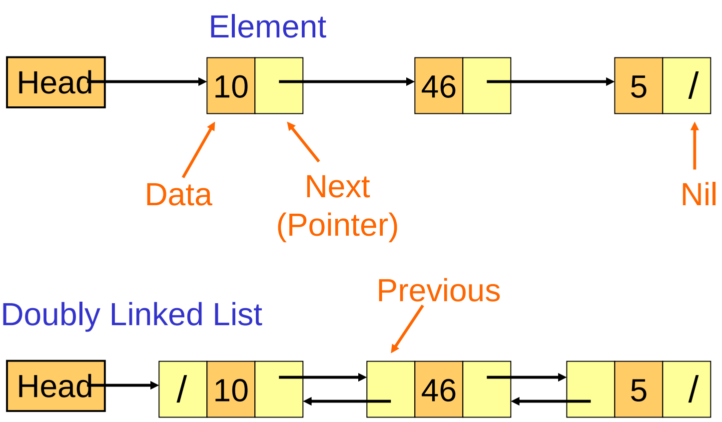
Array: A[0...n] = A[0], A[1],...., A[n].

2D array: A[i,j] for all i, j = 0,..., n.

C Language:

int A[100];
int A[10][10];
A[5] = 20;
A[0][3] = 85;

Linked Lists



Linked List

```
typedef struct listnode
{
int                 data
struct listnode     *nextptr
struct listnode     *prevptr
}  ListNode
```

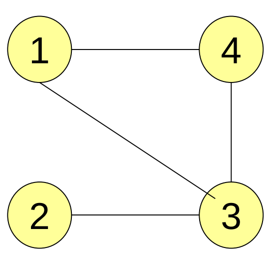

Undirected Graphs

<u>Adjacency-matrix</u>

a(i,j) = 1 if (i,j) is edge

0 otherwise

Notice: a(i,j) = a(j,i)



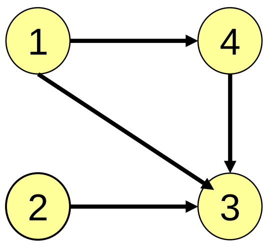
	1	2	3	4
1	0	0	1	1
2	0	0	1	0
3	1	1	0	1
4	1	0	1	0

Directed Graphs

<u>Adjacency-matrix</u>

a(i,j) = 1 if (i,j) is edge
0 otherwise

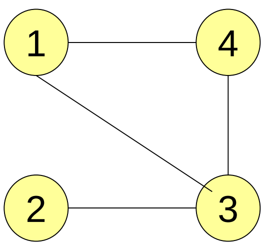
Notice: a(i,j) = a(j,i)

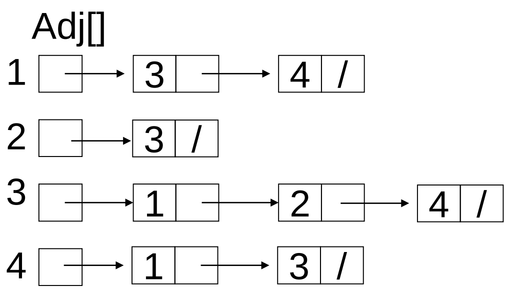


	1	2	3	4
1	0	0	1	1
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

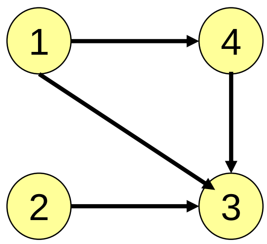
Undirected Graphs

<u>Adjacency-List</u>





Directed Graphs



Adjacency-List