AI & Robotics

Trees & Graphs

PXL AI & ROBOTICS LAB

Goals

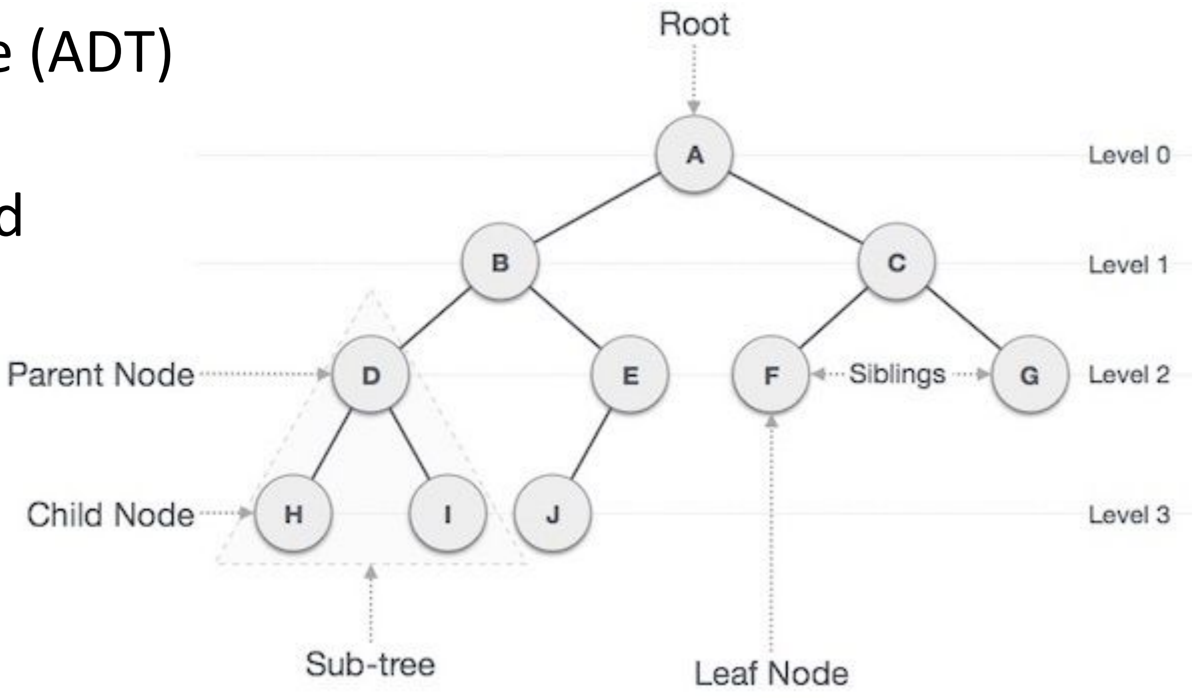
The **junior-colleague**

- can explain in own words what a tree is and its use cases
- can determine when a tree is a good data structure to use for a given problem
- can explain in own words what a binary tree is and its use cases
- can implement an arbitrary tree
- can implement pre-order, post-order and in-order tree traversal for arbitrary trees
- can explain in own words the state space of a problem
- can represent the state space of a problem
- can apply the state space procedure for a given problem
- can explain in own words what a graph is and its use cases
- can explain in own words the difference between undirected and directed in the context of graphs
- can explain in own words the difference between unweighted, weighted in context of graphs.
- can explain in own words the difference between acyclic and cyclic in context of graphs.
- can determine when a graph is a good data structure to use for a given problem
- can implement and differentiate the different graph representations

Trees

Trees

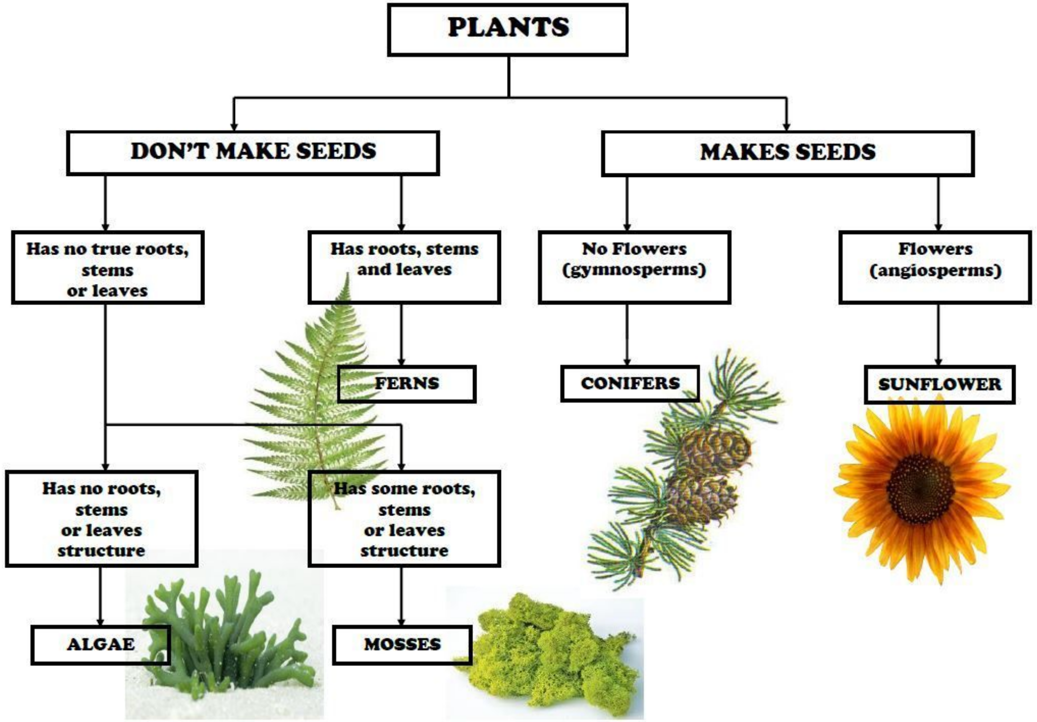
- Abstract Data Type (ADT)
- Hierarchical
- Recursively defined



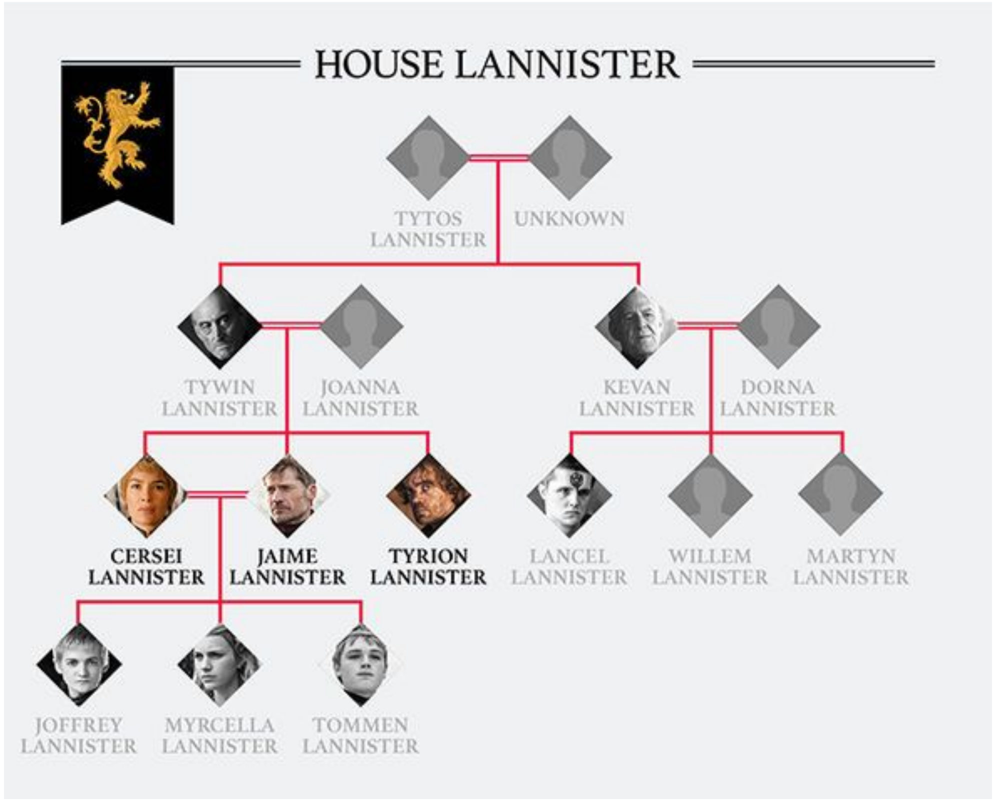
Why?

- Modeling hierarchical relationships
  - Biological taxonomies
  - Family trees
  - Organizational structure
  - etc

Why?

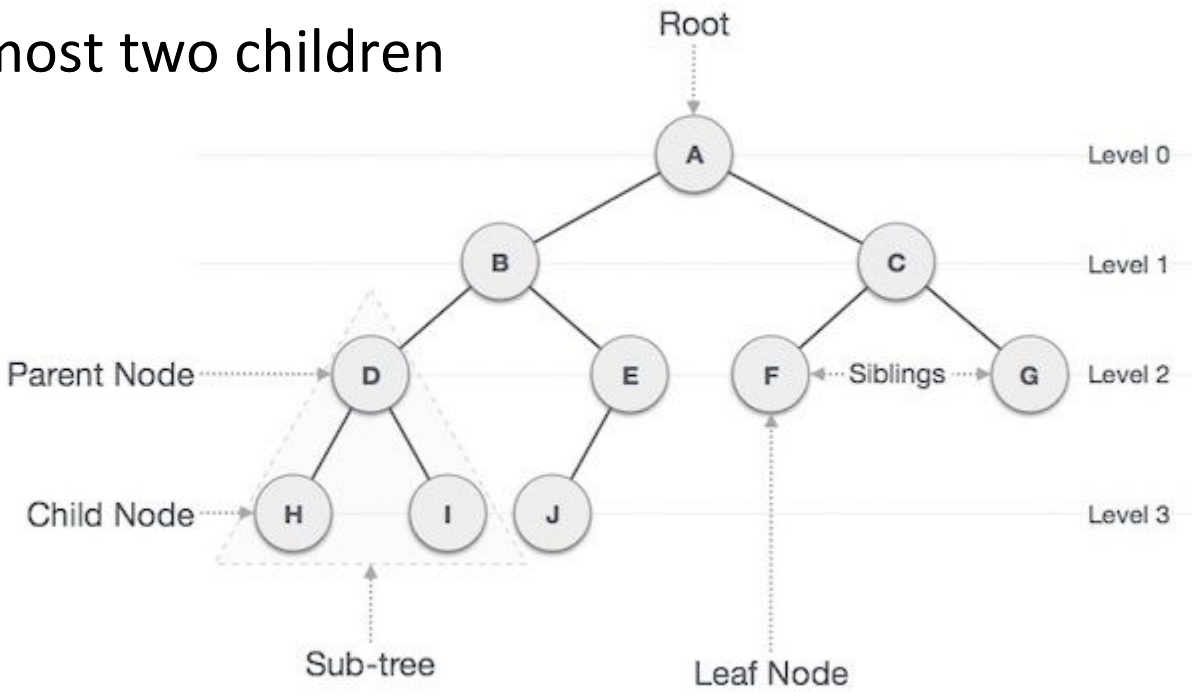

PLANTS
DON'T MAKE SEEDS
MAKES SEEDS
Has no true roots, stems or leaves
Has roots, stems and leaves
No Flowers (gymnosperms)
Flowers (angiosperms)
FERNS
CONIFERS
SUNFLOWER
Has no roots, stems or leaves structure
Has some roots, stems or leaves structure
ALGAE
MOSSES

Why?


HOUSE LANNISTER
TYTOS LANNISTER
UNKNOWN
TYWIN LANNISTER
JOANNA LANNISTER
KEVAN LANNISTER
DORNA LANNISTER
CERSEI LANNISTER
JAIME LANNISTER
TYRION LANNISTER
LANCEL LANNISTER
WILLEM LANNISTER
MARTYN LANNISTER
JOFFREY LANNISTER
MYRCELLA LANNISTER
TOMMEN LANNISTER

Binary Trees

- Each node has at most two children
  - – Left child
  - – Right child



Representation

```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data

    def print_tree(self):
        print(self.data)
```

Tree Traversal

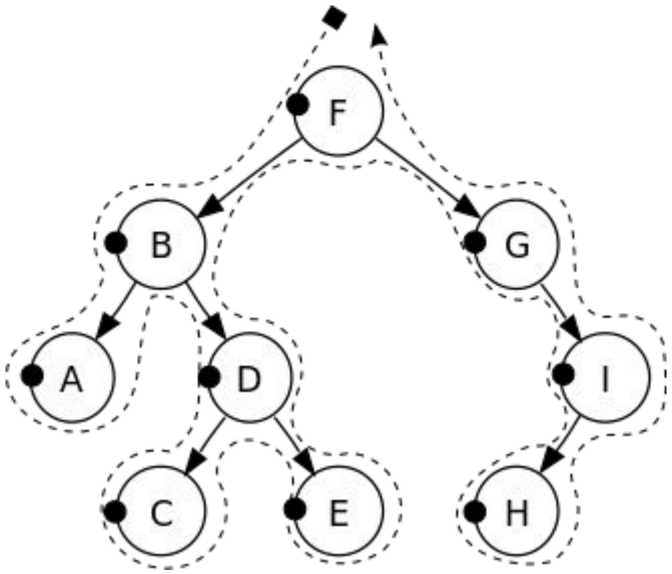
- Visiting: checking the value of a node when control is on the node
- Traversing: passing through nodes in a specific order
  - Pre-order
  - In-order
  - Post-order

Tree Traversal: Pre-order

```
def print_tree(self):
    print(self.data)

    if self.left:
        self.left.print_tree()

    if self.right:
        self.right.print_tree()
```

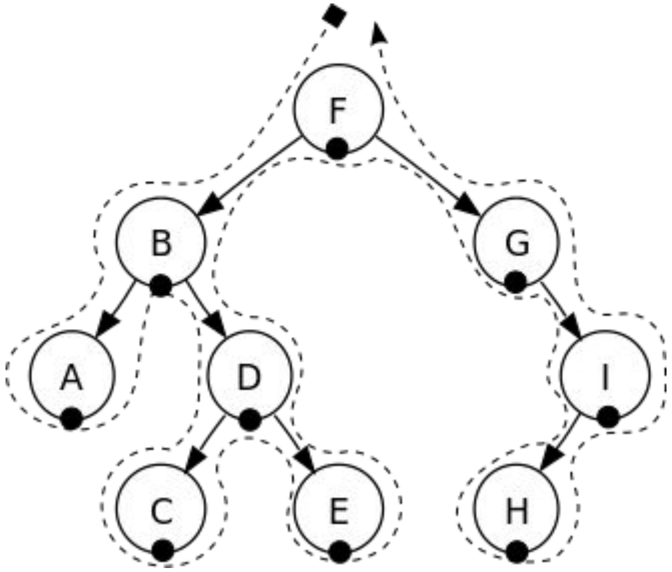


Tree Traversal: In-order

```
def print_tree(self):
    if self.left:
        self.left.print_tree()

    print(self.data)

    if self.right:
        self.right.print_tree()
```

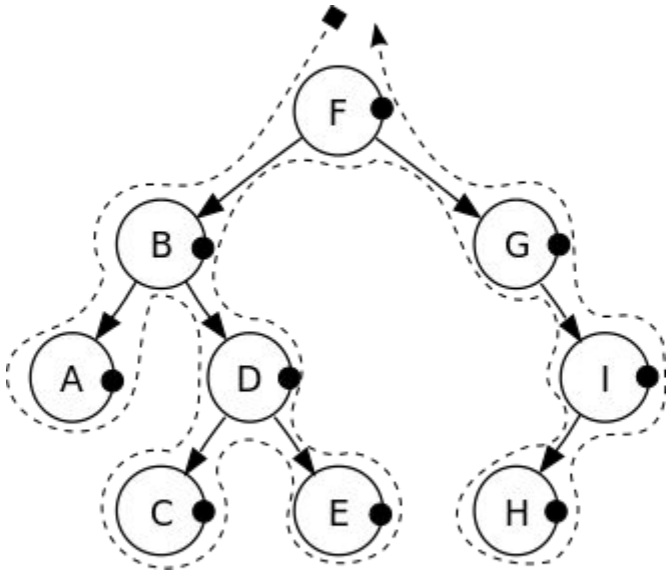


Tree Traversal: Post-order

```
def print_tree(self):
    if self.left:
        self.left.print_tree()

    if self.right:
        self.right.print_tree()

    print(self.data)
```



State space representation

State space representation

- World state includes every relevant detail of the environment
- Procedure:
  - Select some way to represent states in the problem in an unambiguous way.
  - Formulate all actions that can be performed in states including their preconditions and effects

    => Production Rules
  - Represent the initial state (s).
  - Formulate precisely when a state satisfies the goal of our problem.
  - Activate the production rules on the initial state and its descendants, until a goal state is reached.

State space representation: example

- The farmer, fox, goose and grain.

A farmer has to cross a river with his fox, goose and grain. His boat can only carry himself and one of his possessions, though. Thus he needs to make several crossings in order for all the animals to reach the other bank. However, bear in mind that an unguarded fox will eat the goose and an unguarded goose will eat the grain.

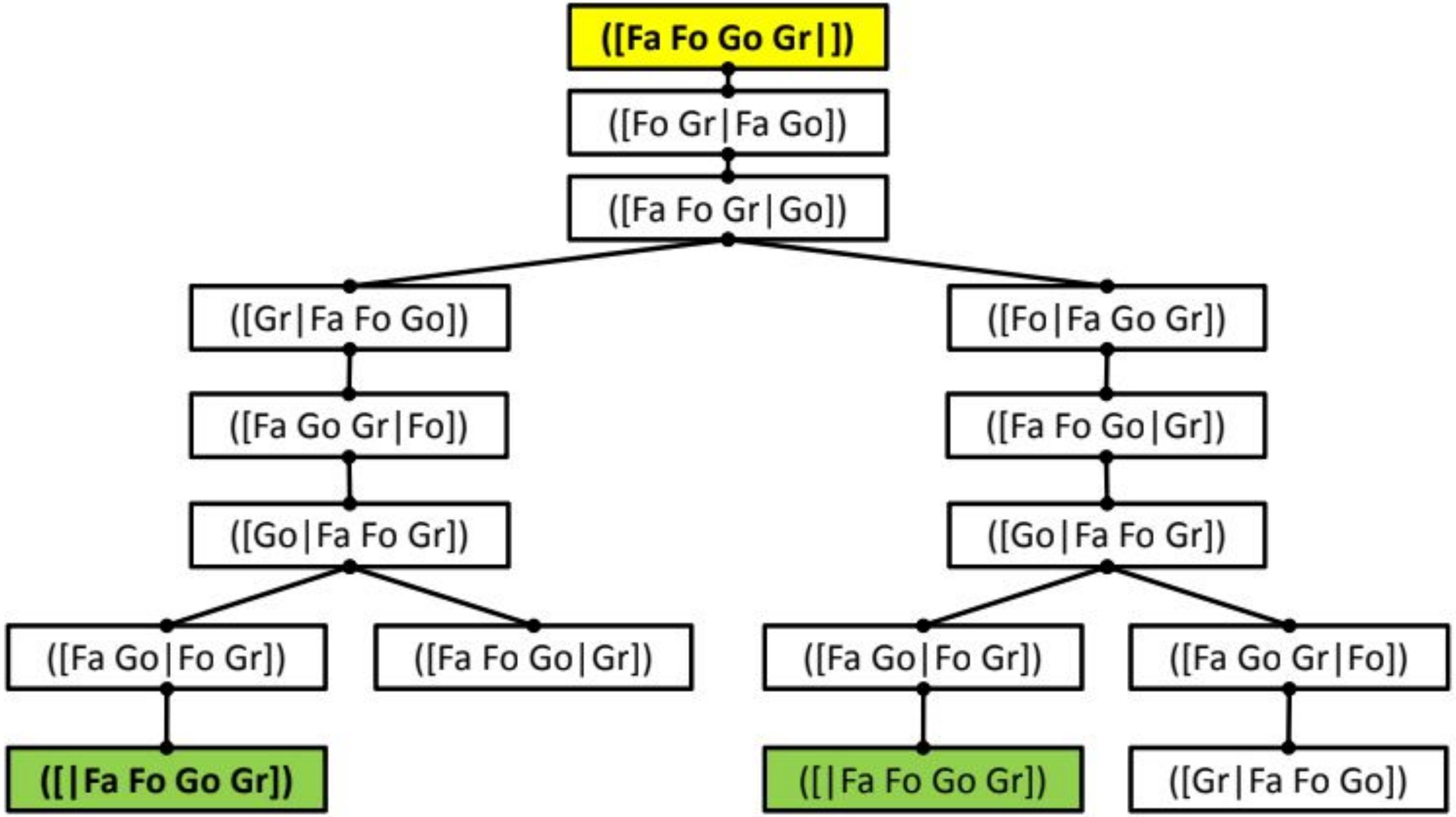
State space representation: example

- States of the form [L|R], where:
  - L: Items on left bank
  - R: Items on right bank
- L and R can contain:
  - Fa: Farmer
  - Fo: Fox
  - Go: Goose
  - Gr: Grain

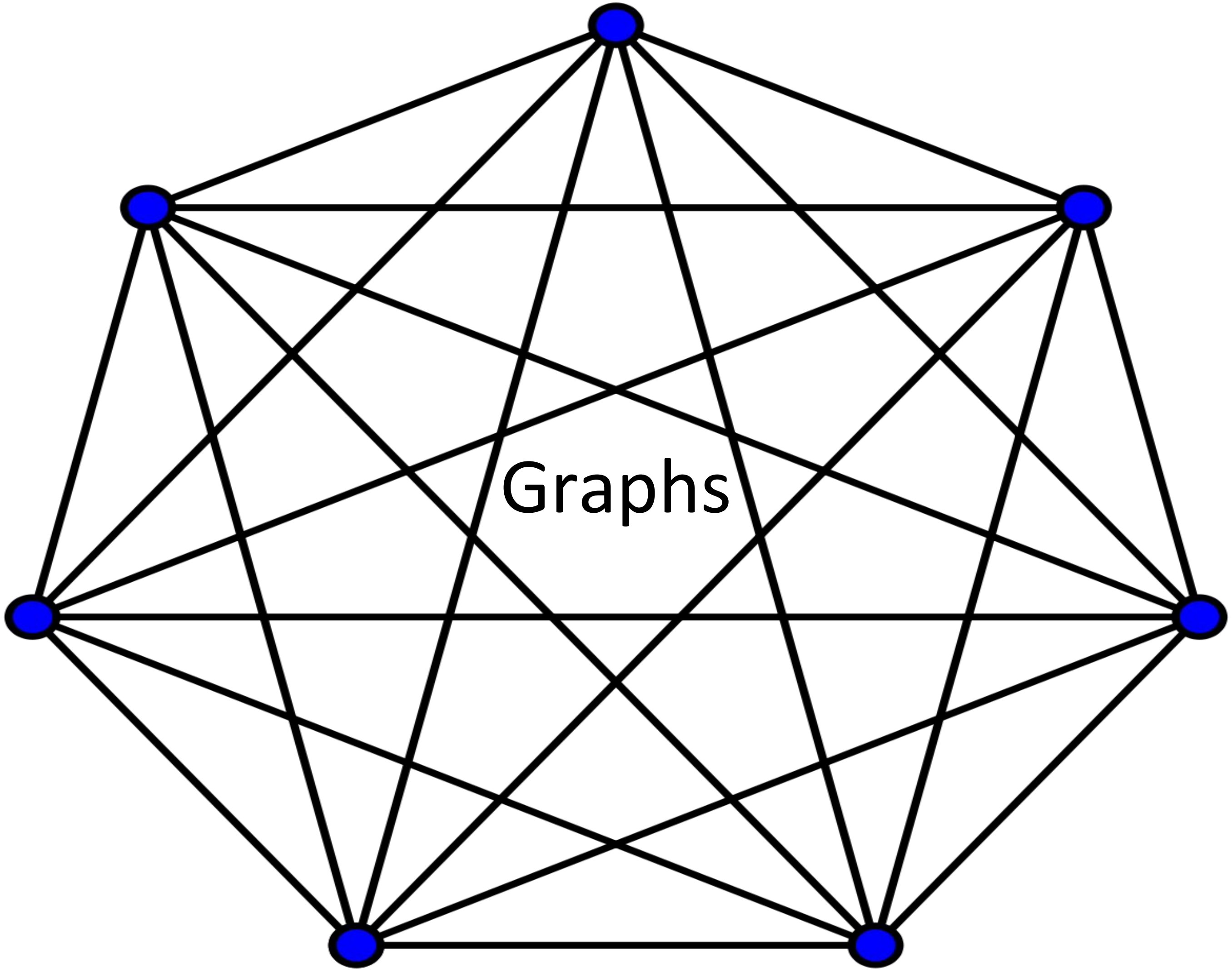
State space representation: example

- Start: [Fa Fo Go Gr|]
- Goal: [|Fa Fo Go Gr]
- Rules:
  - R1: [Fa `x`|`y`] => [`x`|Fa `y`]
  - R2: [`x`|Fa `y`] => [Fa `x`|`y`]
  - R3: [Fa `z` `x`|`y`] => [`x`|Fa `z` `y`]
  - R4: [`x`|Fa `z` `y`] => [Fa `z` `x`|`y`]
- No combination (Fo,Go) or (Go,Gr) on either bank, without the farmer.

State tree


([Fa Fo Go Gr|])
([Fo Gr|Fa Go])
([Fa Fo Gr|Go])
([Gr|Fa Fo Go])
([Fo|Fa Go Gr])
([Fa Go Gr|Fo])
([Fa Fo Go|Gr])
([Go|Fa Fo Gr])
([Go|Fa Fo Gr])
([Fa Go|Fo Gr])
([Fa Fo Go|Gr])
([Fa Go|Fo Gr])
([Fa Go Gr|Fo])
([|Fa Fo Go Gr])
([|Fa Fo Go Gr])
([Gr|Fa Fo Go])

Graphs



Graphs

- Object Set:

  G = (V, E)

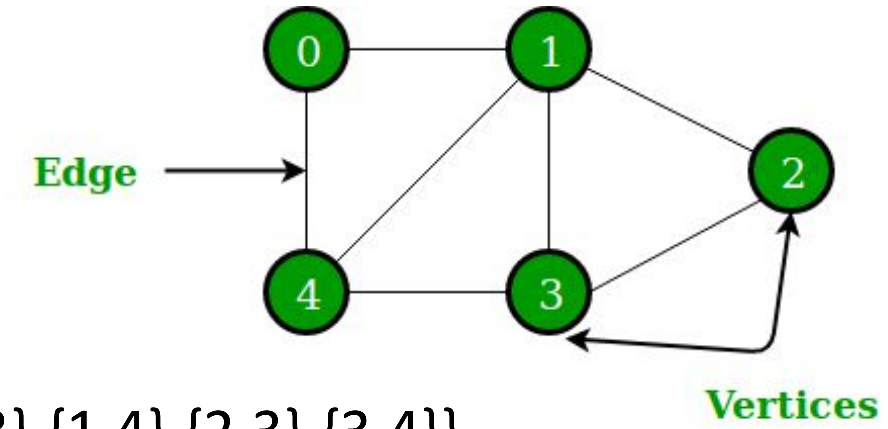
  – V: set of vertices

  {0,1,2,3,4}

  – E: set of edges

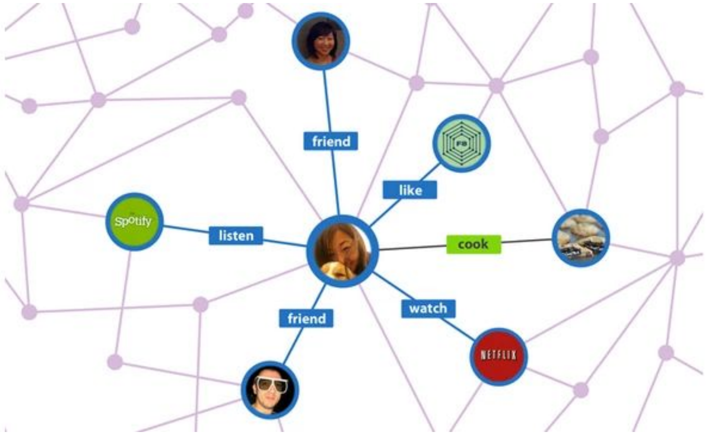
  {{0,1},{0,4},{1,2},{1,3},{1,4},{2,3},{3,4}}

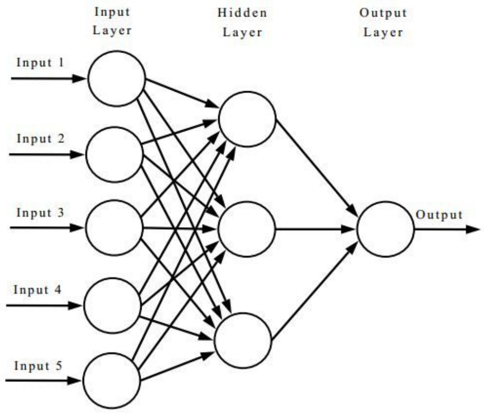
- Generalization of trees



Why?

- Modeling complex relationships
  - Social relationships:
    - Facebook
    - LinkedIn
  - Maps
  - Neural networks
  - etc.





Properties

- Undirected vs. Directed
- Unweighted vs. Weighted
- Acyclic vs. Cyclic

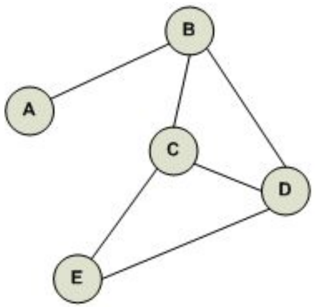


Fig 1. Undirected Graph

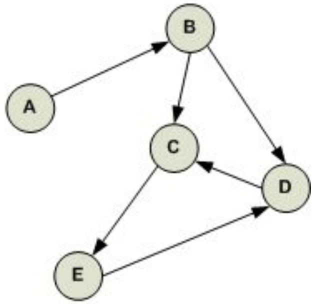
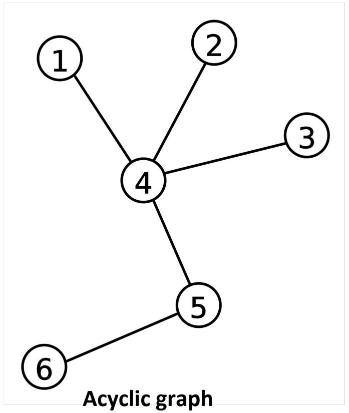
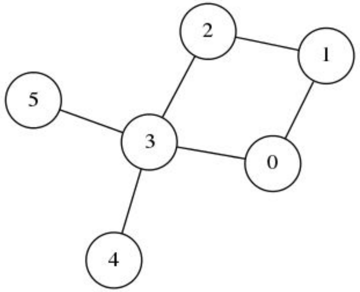


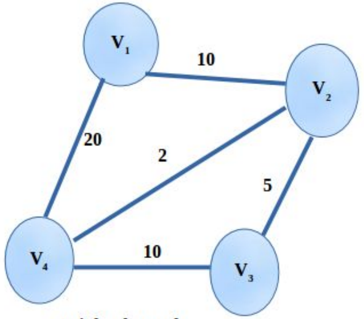
Fig 2. Directed Graph



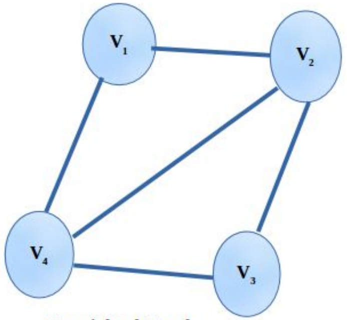
Acyclic graph



Cyclic graph



Weighted Graph



Unweighted Graph

https://towardsdatascience.com/graph-theory-basic-properties-955fe2f61914

Trees vs. Graphs

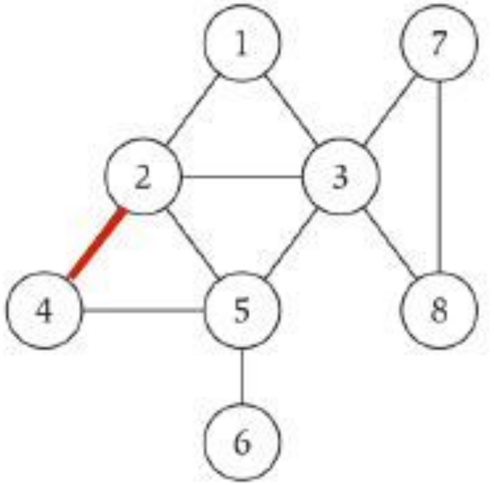
Trees	Graphs
only one path between any two vertices	more than one path between any two vertices
acyclic	cyclic or acyclic
undirected at first directed when a root node is chosen	directed or undirected
exactly one root node	no root node
n-1 edges	graph-dependent amount of edges
hierarchical model	network model

Representation

- Adjacency matrix
- Adjacency List

Adjacency matrix

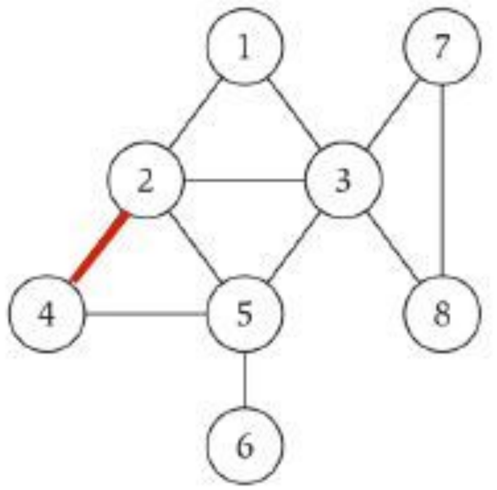
- Good for dense graphs (lots of edges)

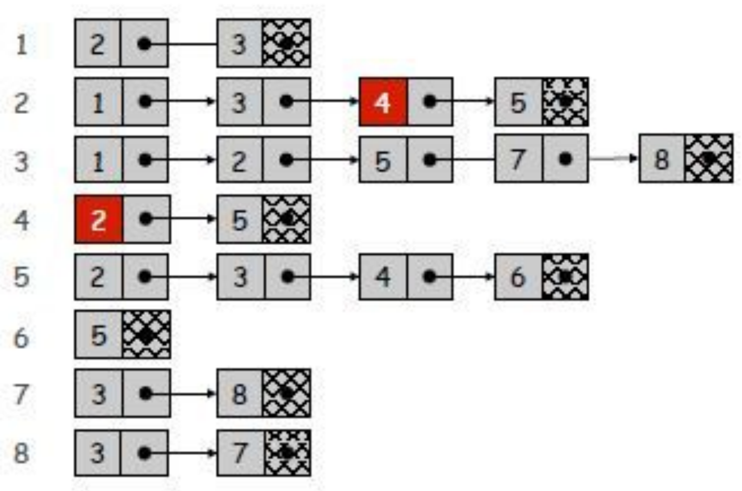


Nodes	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

Adjacency list

- Good for sparse graphs (few edges)




1 2 3
2 1 3 4 5
3 1 2 5 7 8
4 2 5
5 2 3 4 6
6 5
7 3 8
8 3 7