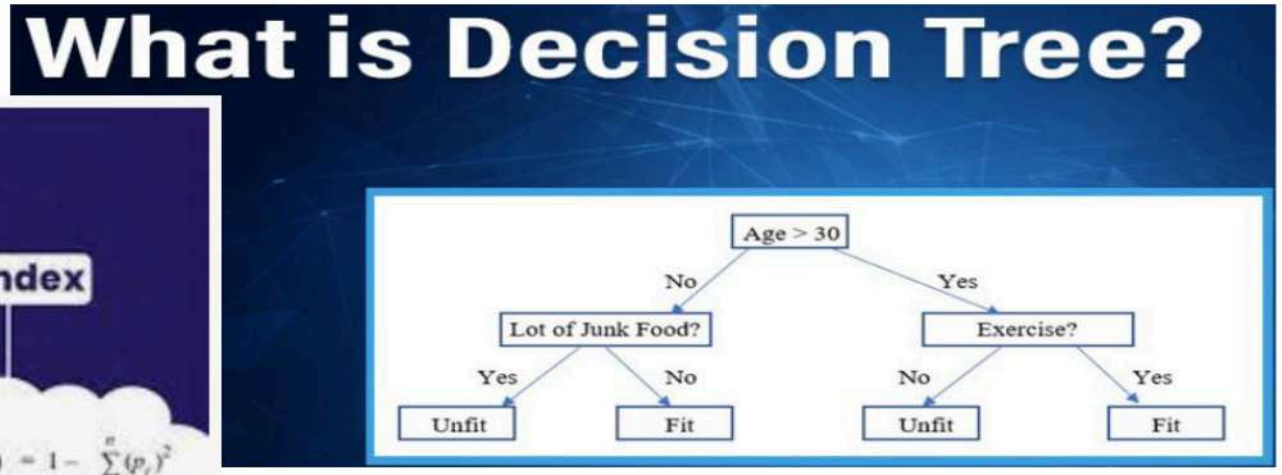
PACE UNIVERSITY

SEIDENBERG SCHOOL OF CSIS



Seidenberg School Of Computer Sciences | Pace University

The Mathematical Foundation for Analytics

Graphs, Paths, Circuits, Trees / Euler Path & Circuit

Spanning Tree – Minimum-cost Spanning Tree

Decision Trees -- Impurity Measurements: Entropy, Gini Index, Information Gain

Prof. Tassos H. Sarbanes

Gain Ratio

Gain Ratio: When a partner retires, the remaining partners gain in terms of the profit sharing ratio. This gain is called the Gain Ratio.

Formulas used in gaining ratio

Gaining Ratio = New Ratio – Old Ratio
New Ratio = Old Ratio + Gain
Gaining Ratio = Retiring partner's share x Acquisition Ratio
New Ratio = Old Ratio + Gaining Ratio

SHANNON ENTROPY

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

CS660 --- Spring 2025

Calculus– III

<< **Decision Trees – DTs** >>

- Graphs, Paths, Circuits, Trees / Euler Path & Circuit
- Spanning Tree – Minimum-cost Spanning Tree
- Kruskal's Algorithm
- Decision Trees
- Impurity Measurements: Entropy, Gini Index, Information Gain
- Decision Trees Split Rule
- Random Forests

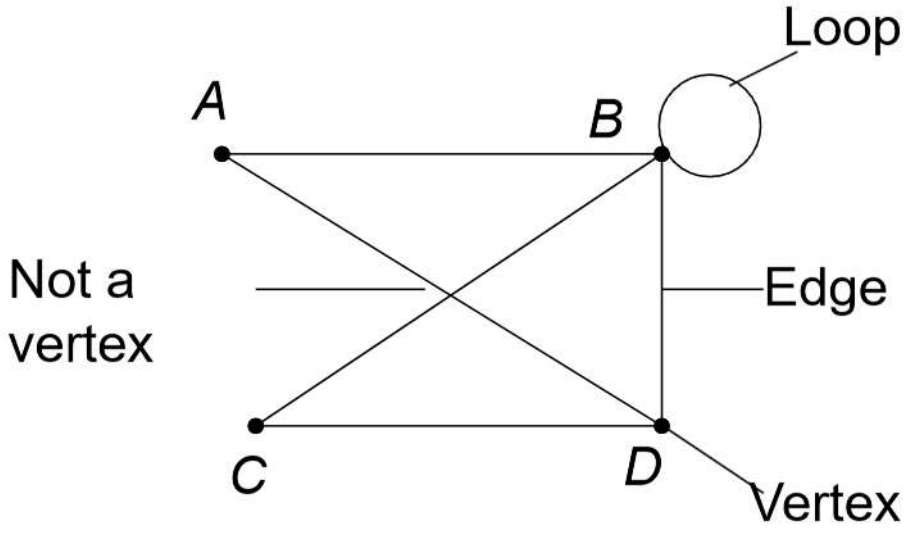
<< **k-Nearest Neighbors – kNNs** >>

- The k-Nearest Neighbor (k-NN) Classifier
- The Minkowski Distance: A Distance Metric

Graph Theory – Definitions

- A ***graph*** is a finite set of points called ***vertices*** (singular form is ***vertex***) connected by line segments (not necessarily straight) called ***edges*** or ***arcs***.

- A ***loop*** is an edge that connects a vertex to itself.

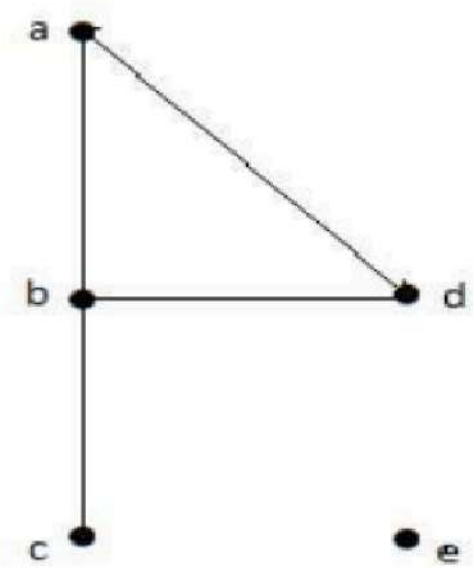


Degree of a Vertex

- The ***degree*** of a vertex is the number of edges that connect to that vertex.
- A vertex with an even number of edges connected to it is an ***even vertex***, a vertex with an odd number of edges connected to it is an ***odd vertex***.

In a simple graph with n number of vertices, the degree of any vertex is –

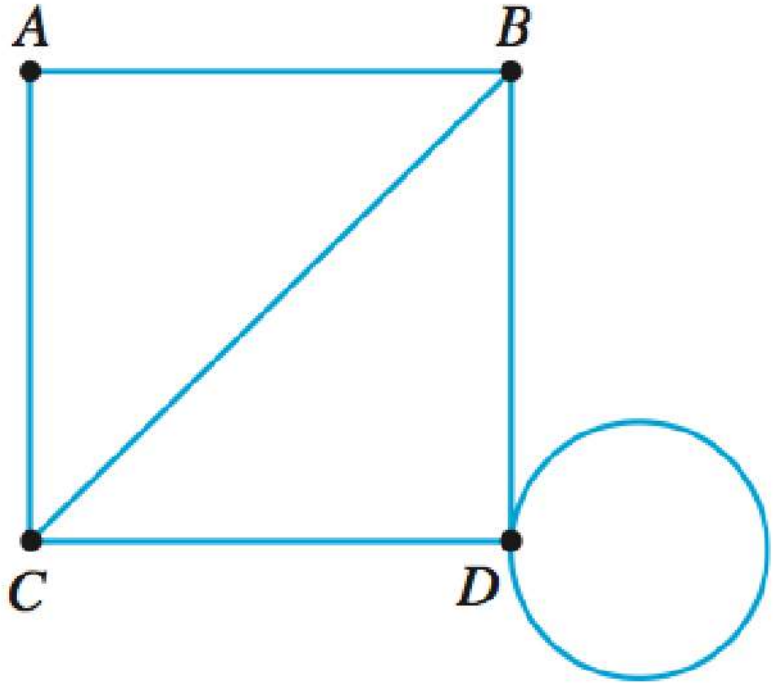
$\deg(v) = n - 1 \; \forall \; v \in G$



- deg(a) = 2, as there are 2 edges meeting at vertex 'a'.
- deg(b) = 3, as there are 3 edges meeting at vertex 'b'.
- deg(c) = 1, as there is 1 edge formed at vertex 'c'
- deg(d) = 2, as there are 2 edges meeting at vertex 'd'.
- deg(e) = 0, as there are 0 edges formed at vertex 'e'.

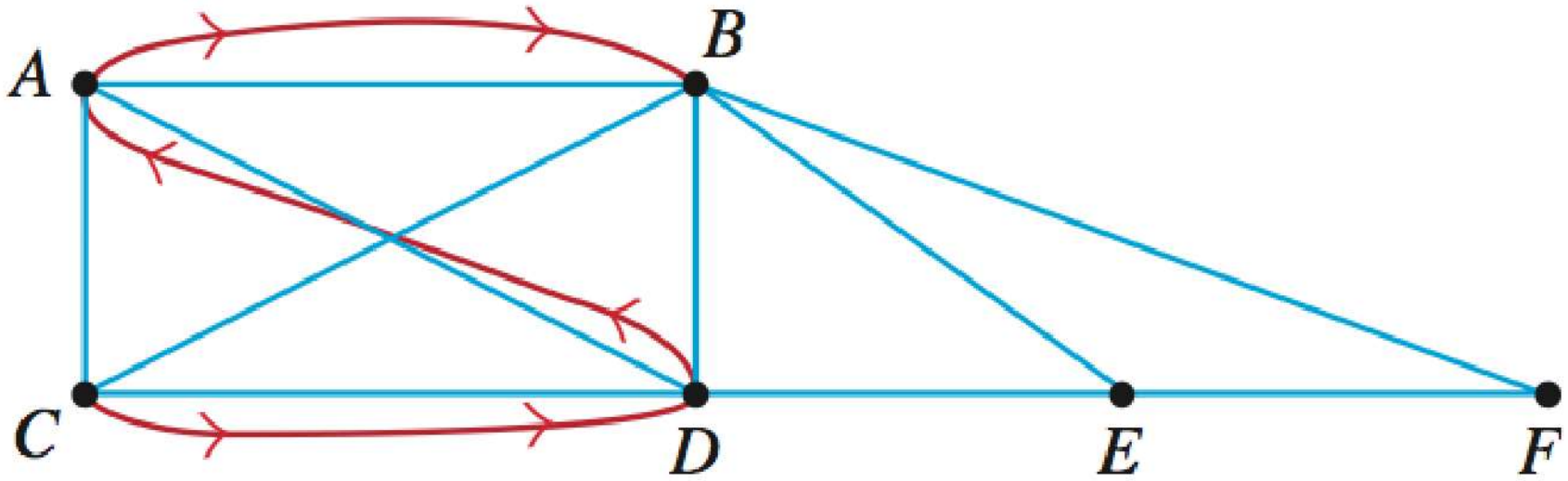
Even / Odd Vertices

In the figure, vertices *A and D are <u>even</u>, and vertices B and C are <u>odd</u>.*



Paths

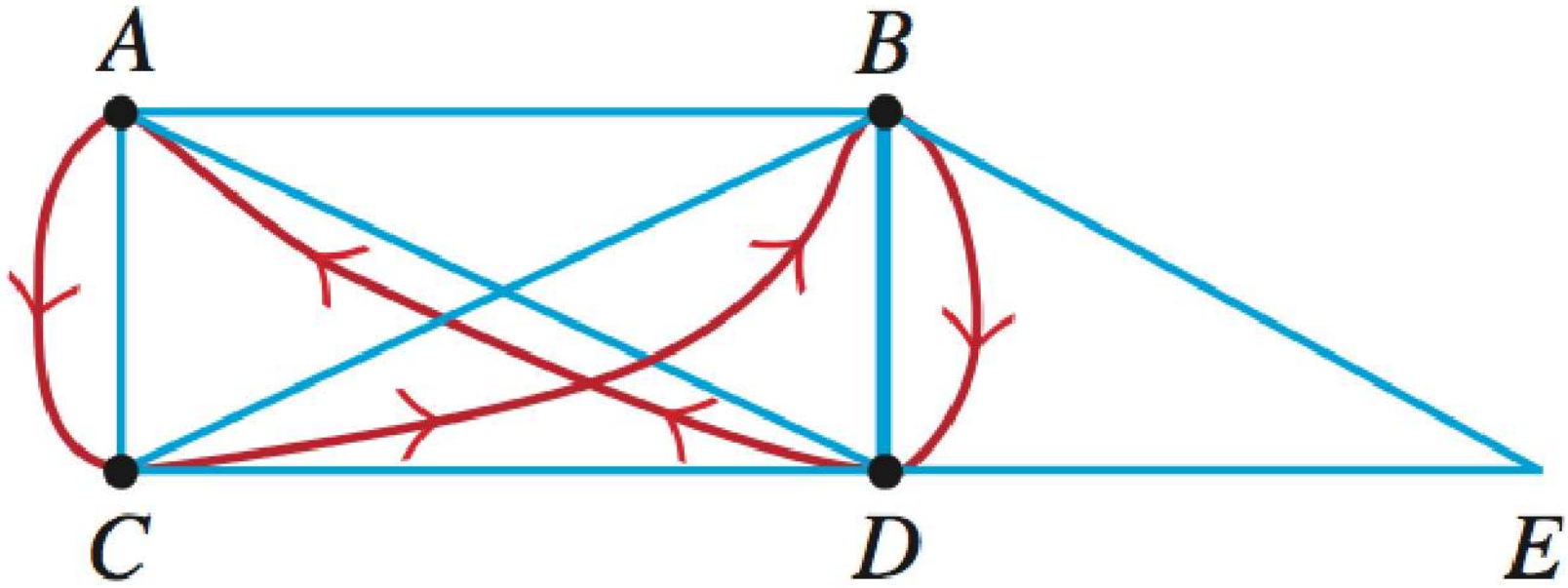
A **path** is a sequence of adjacent vertices and edges connecting them.



C, D, A, B is an example of a path

Circuit

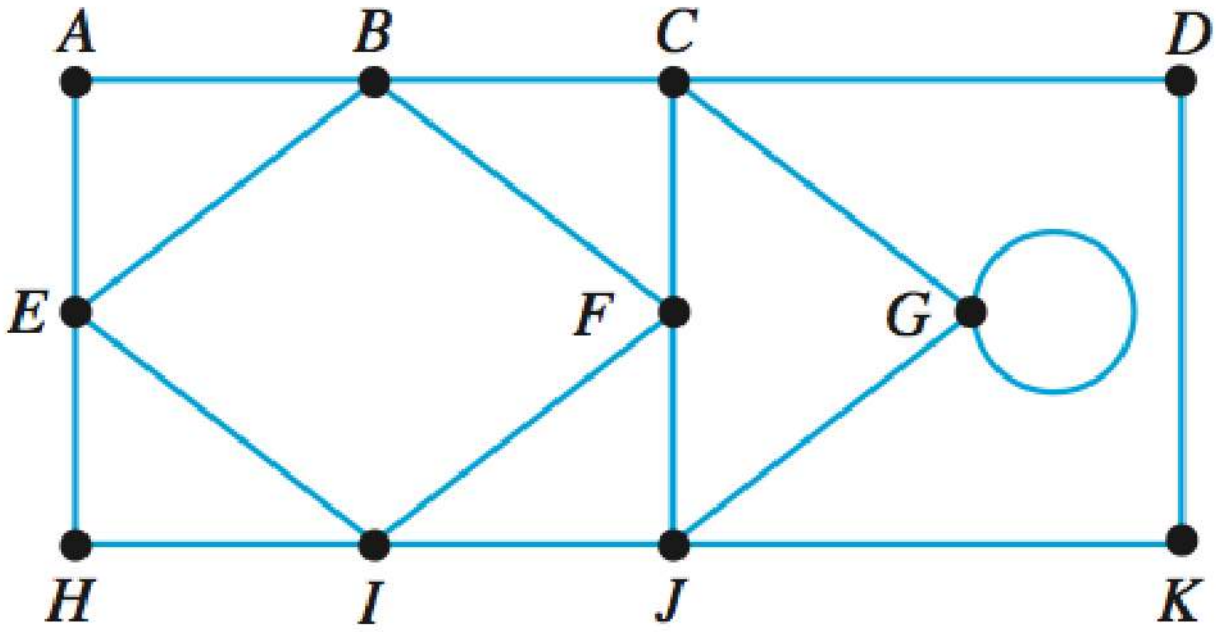
A **circuit** is a path that begins and ends at the same vertex.



Path *A, C, B, D, A* forms a circuit.

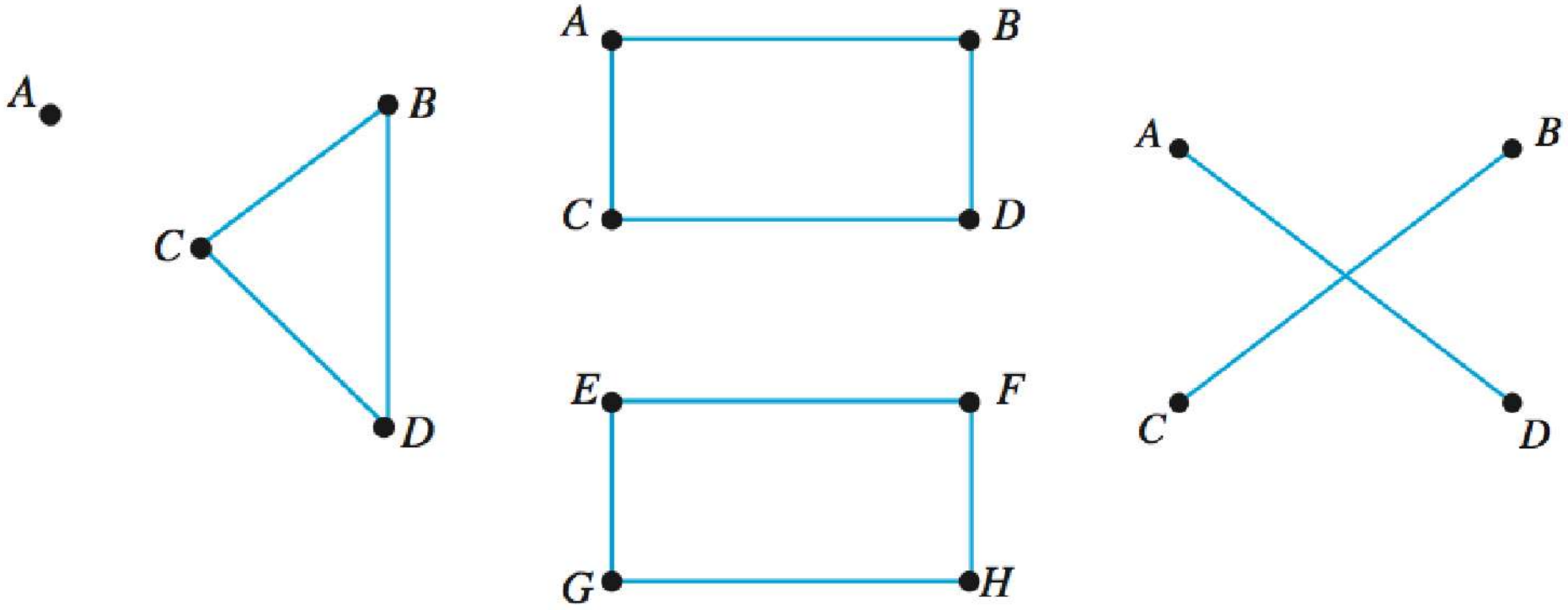
Connected Graph

A graph is ***connected*** if, for any two vertices in the graph, there is a path that connects them.



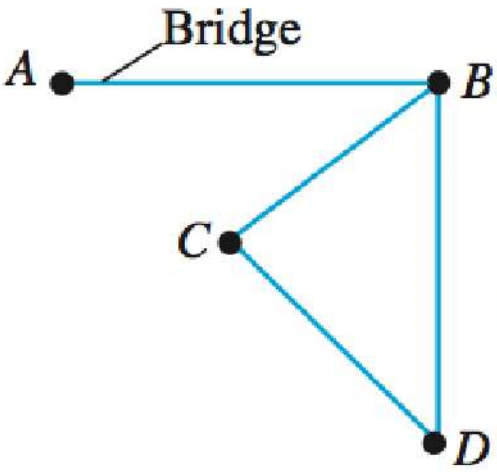
Disconnected Graph

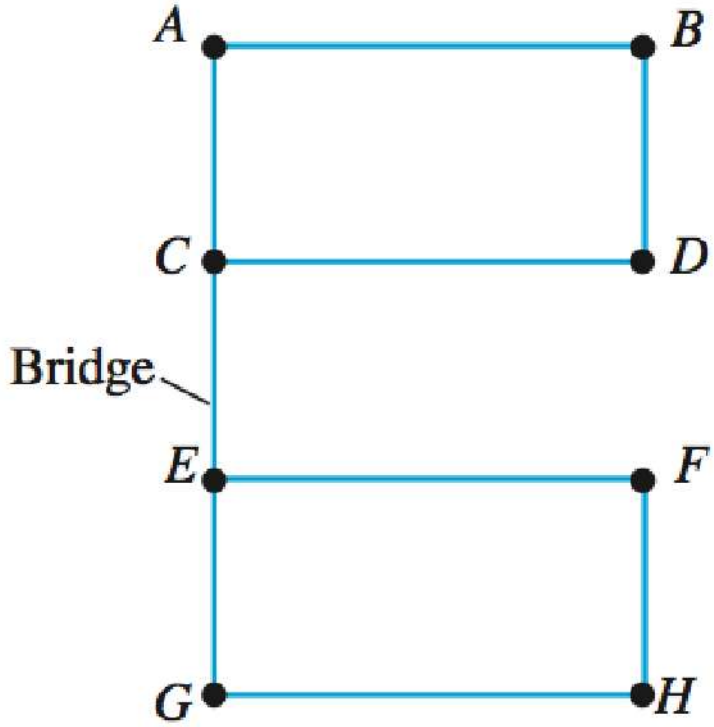
If a graph is not connected, it is ***disconnected***.

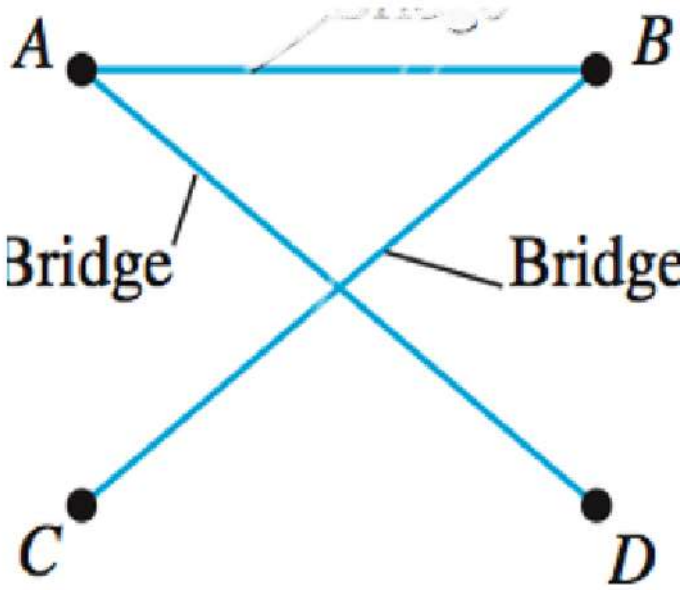


Bridge

A **bridge** is an edge that, if removed from a connected graph, would create a disconnected graph.



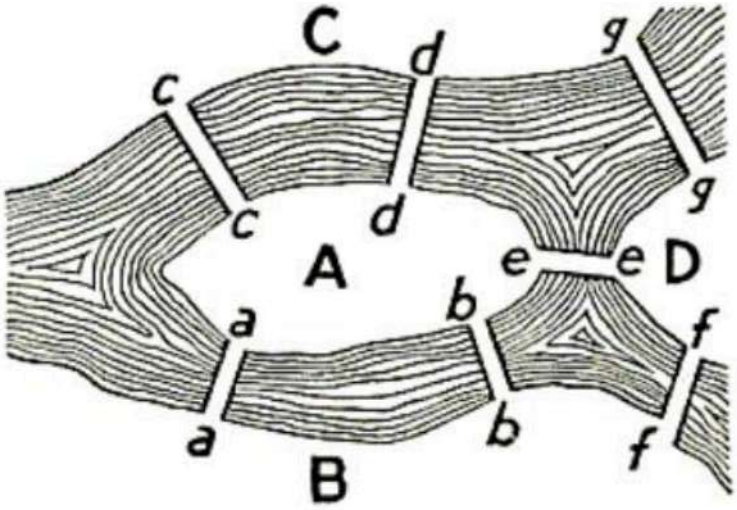


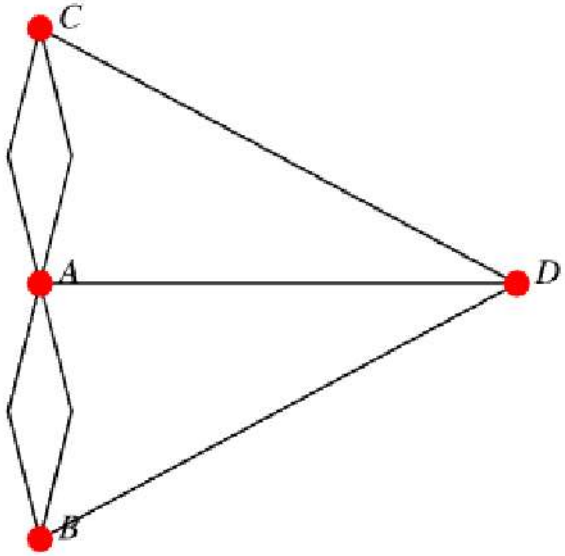


The Königsberg Bridge Problem

The **Königsberg bridge problem** asks if the seven bridges of the city of Königsberg (left figure; Kraitchik 1942), formerly in Germany but now known as Kaliningrad and part of Russia, over the river Preger can all be traversed in a single trip without doubling back, with the additional requirement that the trip ends in the same place it began.

This is equivalent to asking if the multigraph on four nodes and seven edges (right figure) has a **Eulerian cycle**. This problem was answered in the negative by **Euler** (1736) and represented the **beginning of graph theory**.

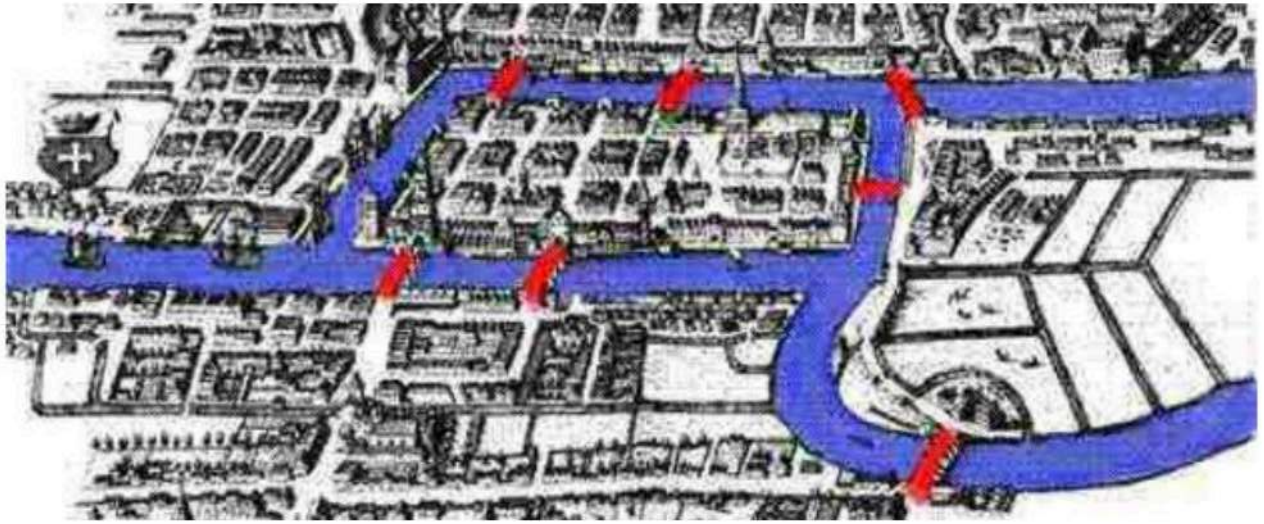




The Königsberg Bridge Problem – No Solution!

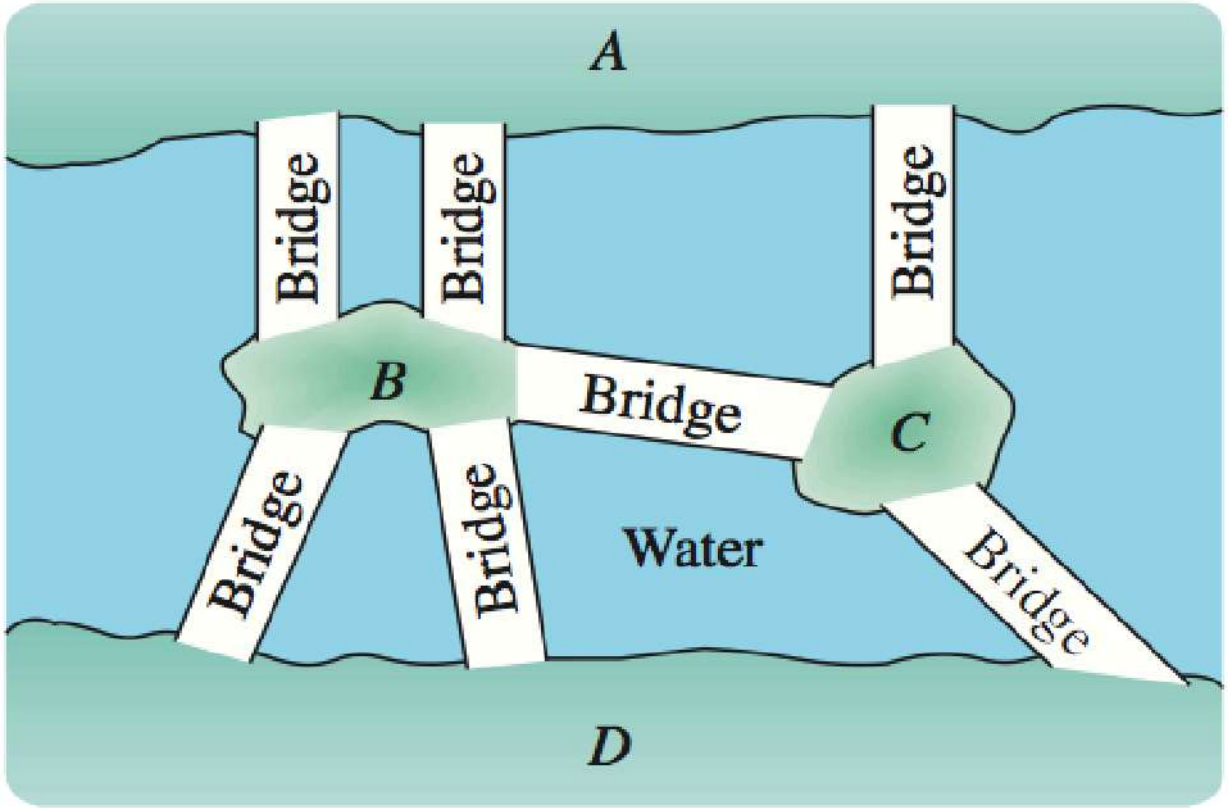
It is impossible to travel the bridges in the city of Königsberg once and only once.

We'll prove it (see next slides) using Euler's Path definition!



Representing the Königsberg Bridge Problem

- Using the definitions of vertex and edge, represent the Königsberg bridge problem with a **graph**.
- Königsberg was situated on both banks and two islands of the Prigel River. From the figure, we see that the sections of town were connected with a series of seven bridges.

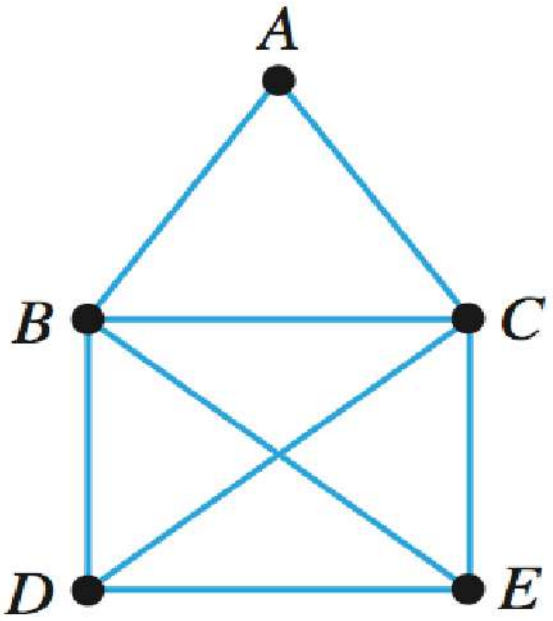


Euler Path – Euler Circuit

- A **Euler path** is a path that passes through **each edge** of a graph exactly one time.
- If a graph has a **Euler path**, we say the graph is ***traversable***
- A **Euler circuit** is a circuit that passes through **each edge** of a graph exactly one time.

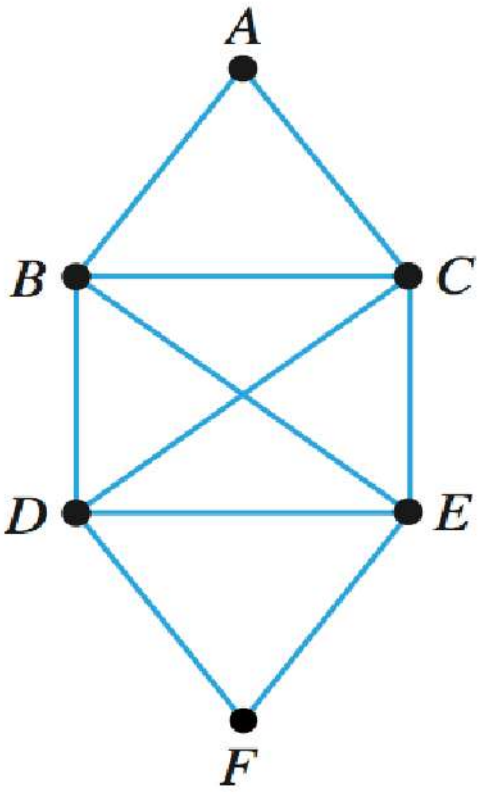
Euler Path vs Euler Circuit

The difference between a Euler path and a Euler circuit is that a Euler circuit must start and end at the same vertex.



Euler Path

D, E, B, C, A, B, D, C, E



Euler Circuit

D, E, B, C, A, B, D, C, E, F, D

Euler's* Theorem

For a connected graph, the following statements are true:

1. A graph with ***no odd vertices*** (all even vertices) has at **least one Euler path**, which is also a **Euler circuit**. A Euler circuit can be started at any vertex, and it will end at the same vertex.

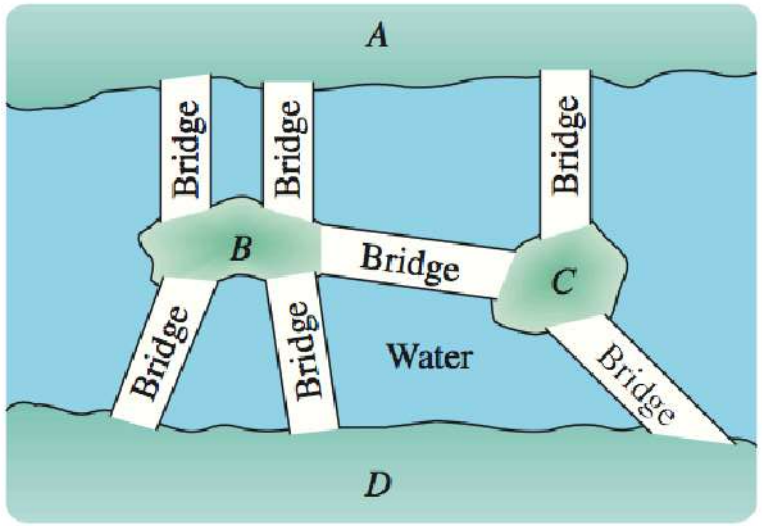
2. A graph with ***exactly two odd vertices*** has at **least one Euler path** but **no Euler circuits**. Each Euler path must begin at one of the two odd vertices, and it will end at the other odd vertex.

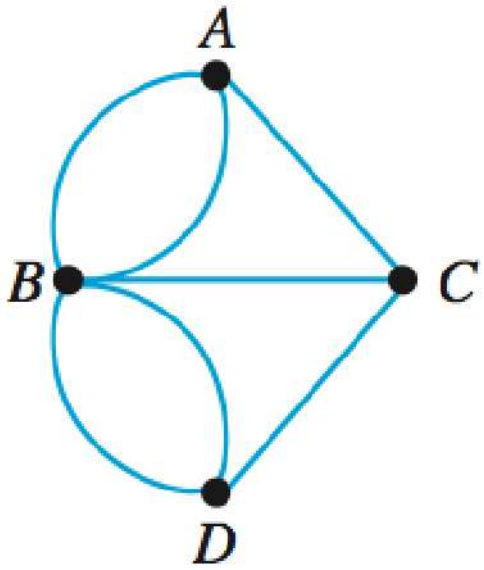
3. A graph with *more* ***than two odd vertices*** has **neither** a **Euler path nor** a **Euler circuit**.

(*) Leonhard **Euler** was a Swiss mathematician, physicist, astronomer, geographer, logician and engineer who founded the studies of **graph theory** and **topology** and made pioneering and influential discoveries in many other branches of mathematics.

Solving the Königsberg Bridge Problem

- Does a Euler path exist?
- **Four odd vertices**: *A, B, C, D*
- So, according to item 3 of **Euler's Theorem**, **no Euler path exists**.

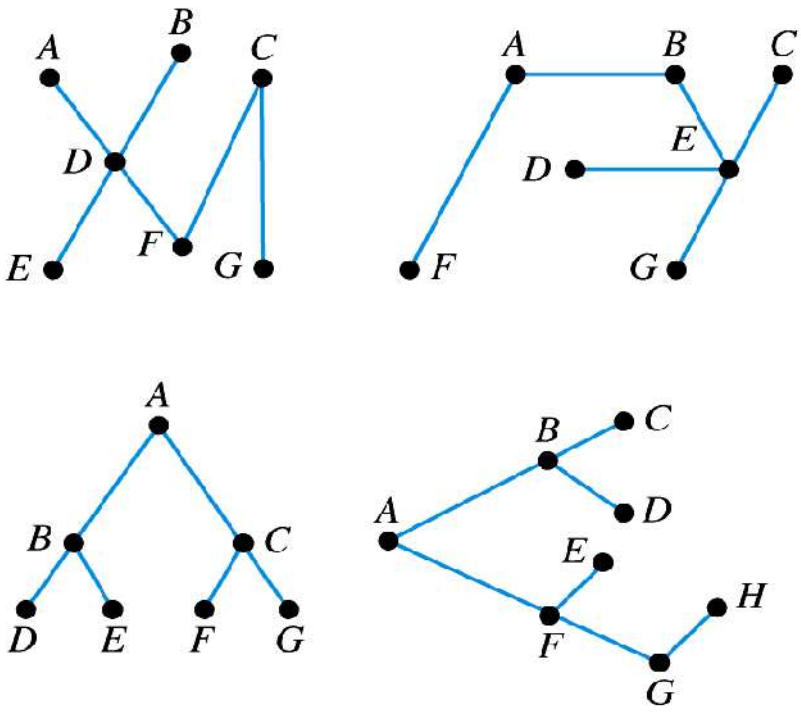




Tree -- Definition

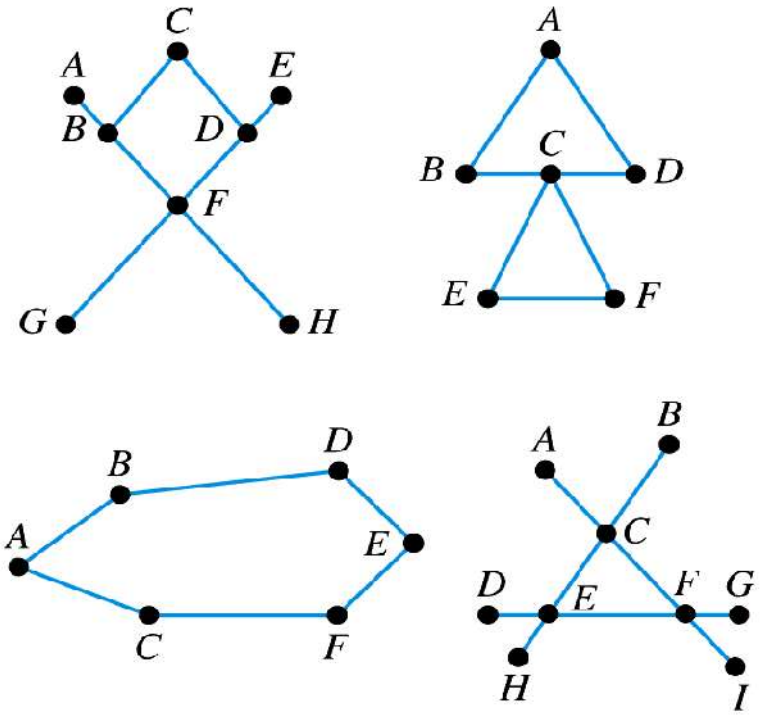
A **tree** is a connected graph in which **each edge** is a **bridge** (slide 10)

Trees



(a) *Graphs that are trees*

No Trees

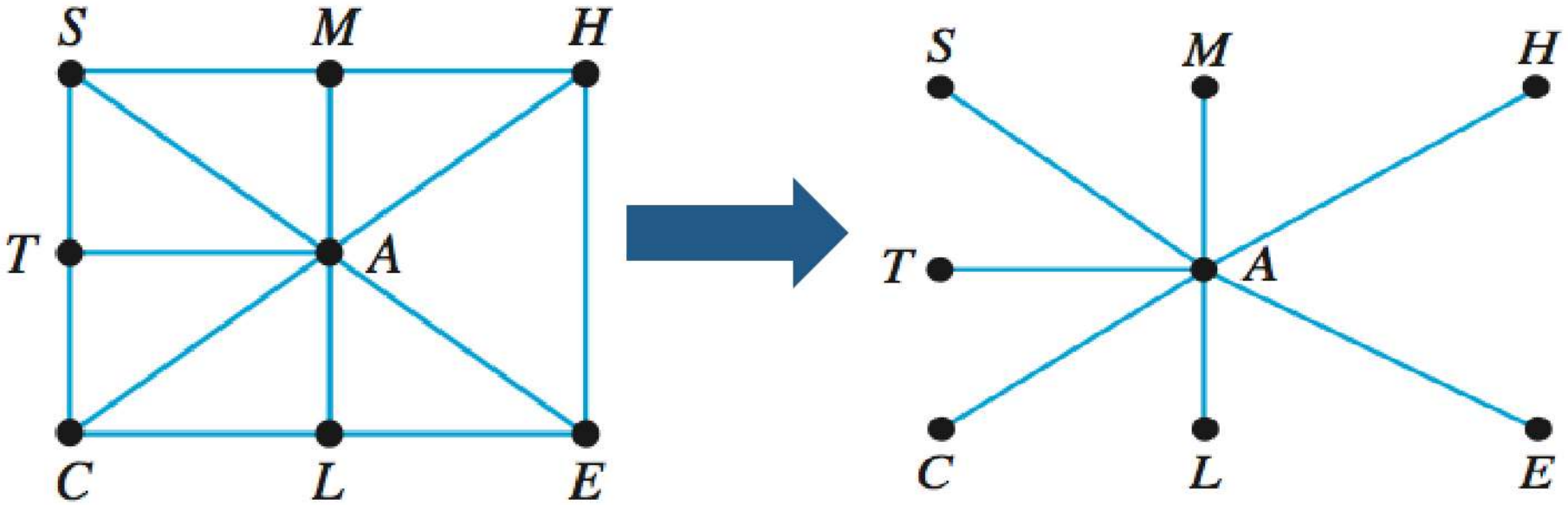


(b) *Graphs that are not trees*

Spanning Tree

A **spanning tree** is a tree that is created from another graph by removing edges while still maintaining a path to each vertex.

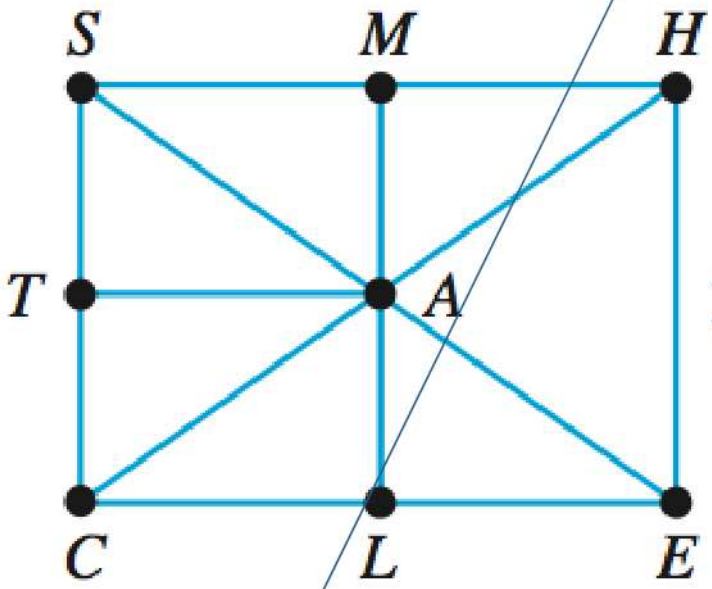
Example : To create a spanning tree, we **remove non-bridge edges** until a tree is created.



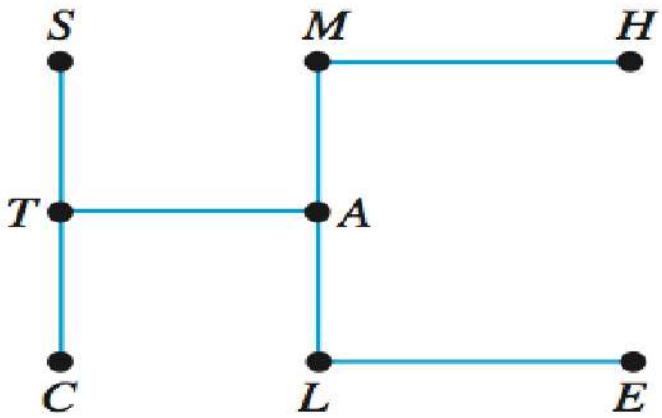
Spanning Tree – Example

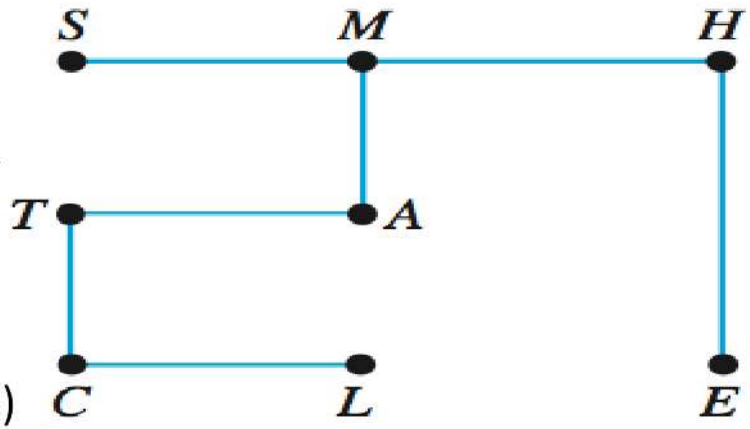
If the graph is a **complete graph** K_n with n vertices, the number

$$T(K_n) = n^{n-2}$$



Here two (2) more spanning trees from the original graph (previous slide)





Formulas and methods to **calculate the number of spanning trees** that can be formed from a graph:

1. **Cayley's Formula** (For Complete Graphs)
2. **Kirchhoff's Matrix-Tree Theorem** (For General Graphs)

a_ Construct the Laplacian Matrix L

b_ Calculate the Determinant of a Cofactor:

> Remove any one row and the corresponding column from L to form a cofactor matrix L'.

> T=det(L ')

Tree vs Spanning Tree

Tree:

- A **tree** is a type of **graph**.
- It is a **connected** graph with **no cycles**.
- It has **N nodes and N-1 edges**.
- It stands on its own — not necessarily derived from another graph.

Think of it as a standalone structure like a family tree or an organizational chart.

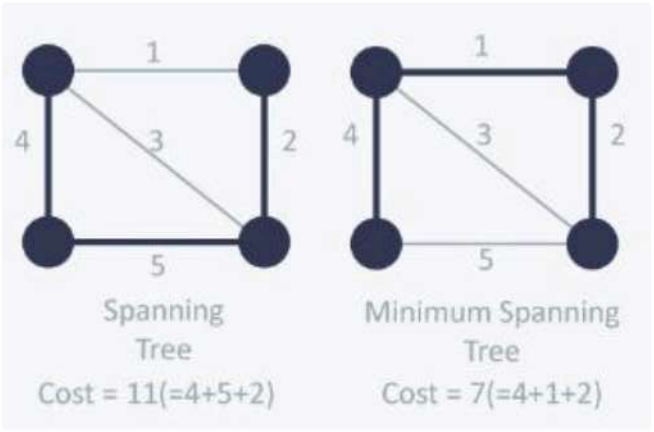
Spanning Tree:

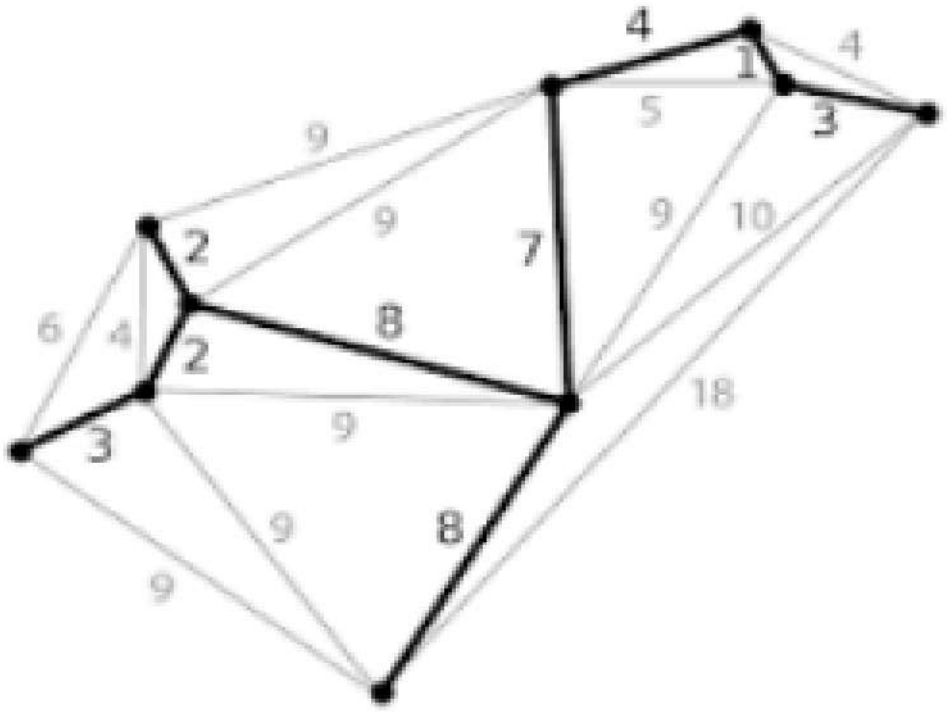
- A **spanning tree** is a **subgraph** of a connected graph.
- It includes **all the nodes** of the original graph.
- It is also a **tree** (connected and no cycles).
- It has exactly **N-1 edges** if there are N nodes.
- There can be **many different spanning trees** for a single graph.

Think of it as a "minimal" way to connect all nodes of a bigger graph without any loops.

Minimum-Cost Spanning Tree

A **minimum spanning tree** (MST) or minimum cost (weight) spanning tree is a subset of the edges of a connected, **edge-weighted undirected graph** that **connects all the vertices** together, without any cycles and with the minimum possible total edge weight.





Kruskal's* Algorithm

To **construct the minimum-cost spanning tree** from a weighted graph:

1. Select the lowest-cost edge on the graph.
2. Select the next lowest-cost edge that does not form a circuit with the first edge.
3. Select the next lowest-cost edge that does not form a circuit with the previously selected edges.
4. Continue selecting the lowest-cost edges that do not form circuits with the previously selected edges.
5. When a spanning tree is complete, you have the minimum-cost spanning tree.

(*) Joseph Bernard **Kruskal** (passed away September 19, 2010) was an American mathematician, statistician, computer scientist.

Let's jump into the Machine Learning or Data Science world!

Let's study Decision Trees!

A decision tree is a very different concept from the graph theory "tree" or "spanning tree."

Decision Tree (DT) – Definition

A **decision tree** is an efficient algorithm for describing a way to traverse a dataset while also defining a tree-like path to the expected outcomes.

This branching in a tree is based on control statements or values, and the data points lie on either side of the **splitting node**, depending on the value of a specific feature.

DTs in Machine Learning

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, decision tree algorithm can be used for solving **regression & classification problems** too (**CART**: **C**lassification **A**nd **R**egression **T**ree).

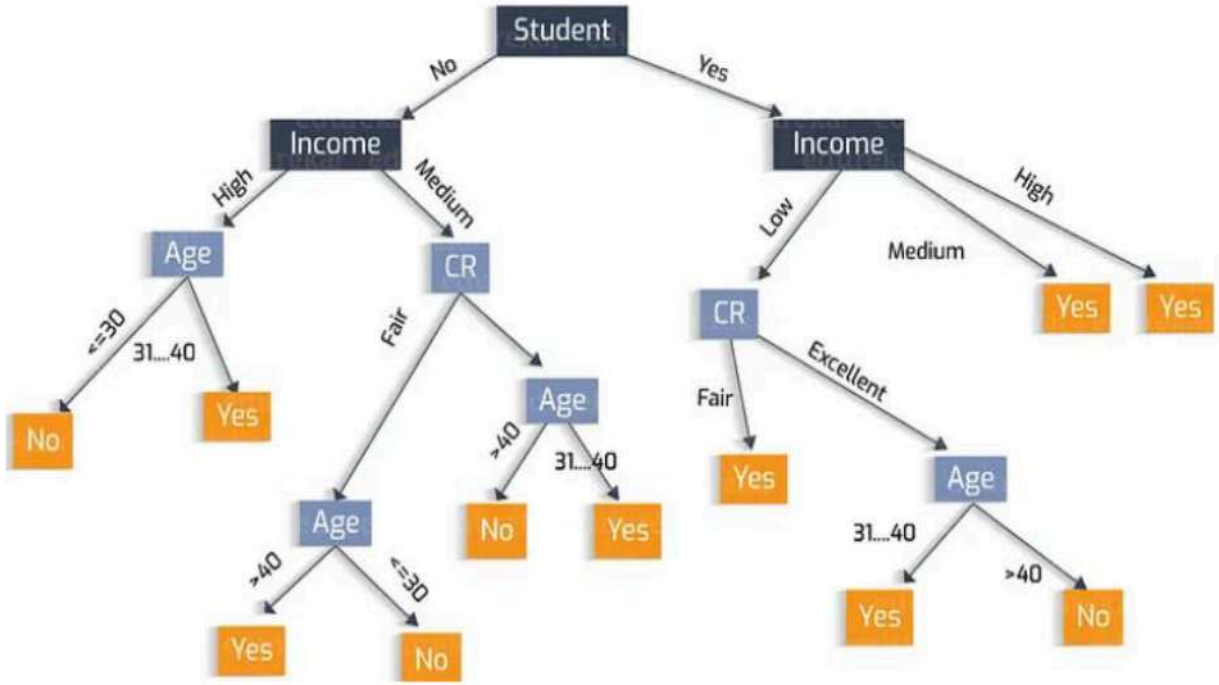
Goal of using a Decision Tree is to create a training model that can used to **predict the class or value** of the target variable by **learning simple decision rules** inferred from prior data (training data).

Decision Tree Structure

The **structure of a decision tree** can be defined by a root node, which is the **most important splitting feature**. The internal nodes are **tests on an attribute**.

E.g., if an internal node has a control statement (age<40), then the data points satisfying this condition are on one side, and the rest on the other.

The **leaf nodes** belong to the available **classes** that the dataset represents.



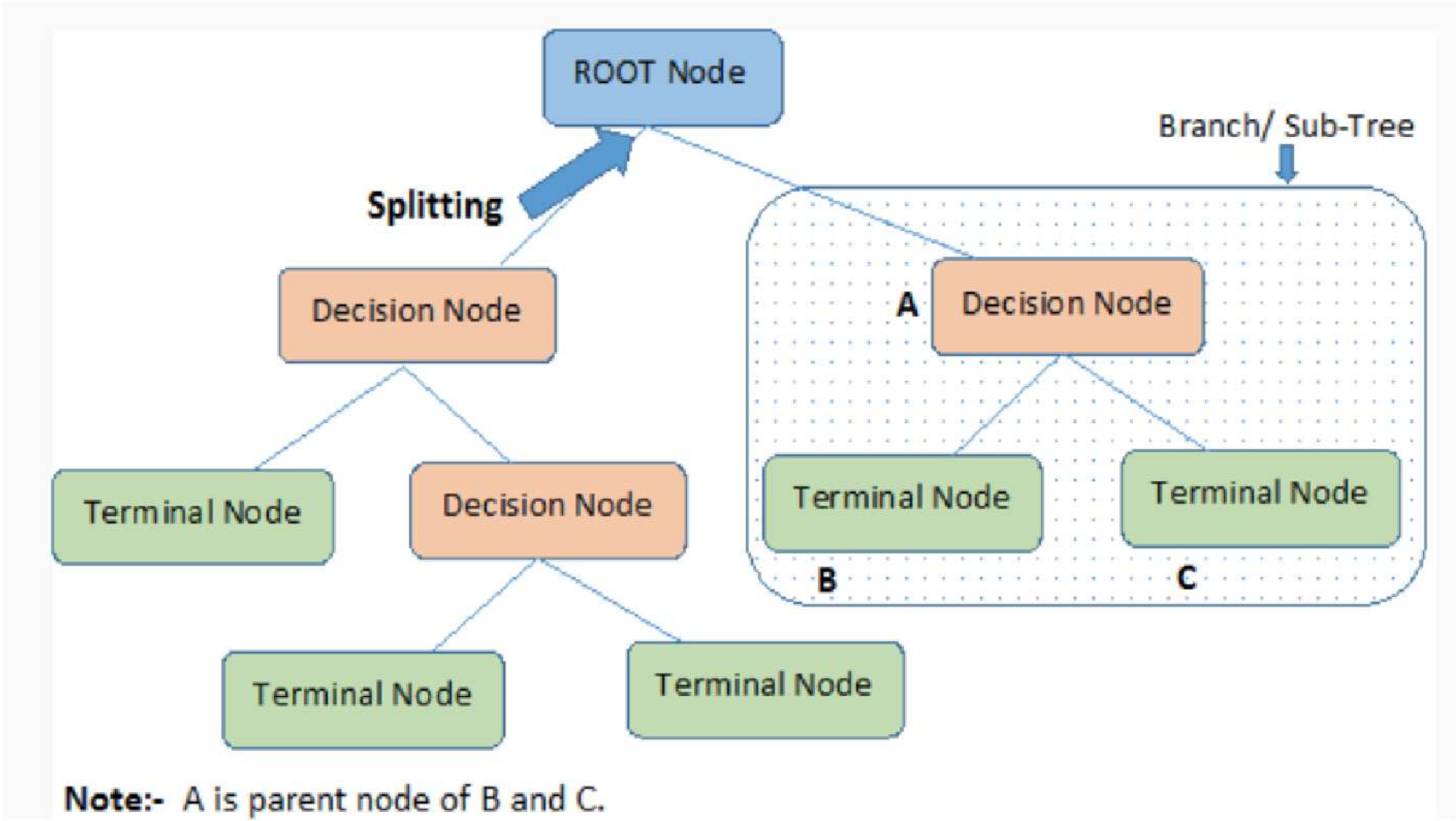
Terminology related to Decision Trees

Terms used within a Decision Tree data structure:

- **Root Node:** It represents the entire population or sample, and this further gets divided into two or more homogeneous sets.
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
- **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
- **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
- **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.
- **Parent and Child Node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

See pictorial representation, next slide...

Terms of Decision Trees – Visualize It



Generate Decision Tree – Algorithm

Generate a decision tree from dataset (usually a training dataset in ML)

>> Input:

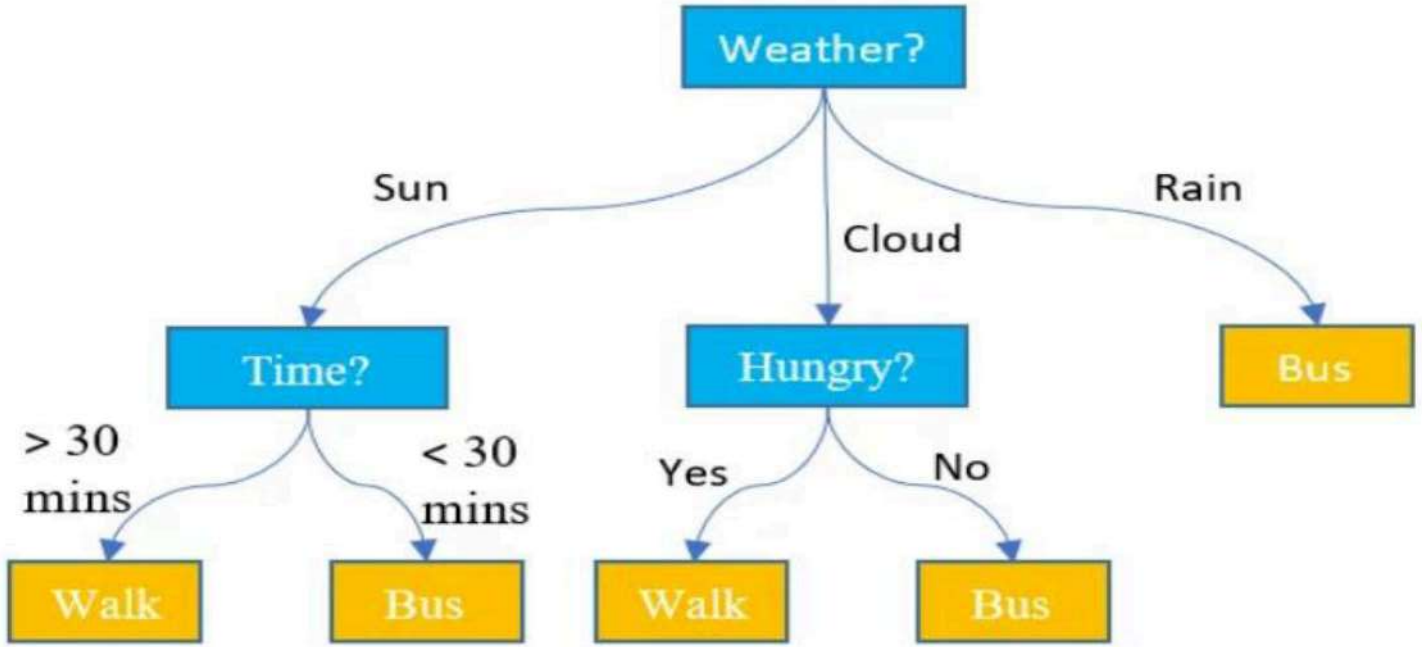
- **Dataset** D, a set of training instances and their associated labels
- **Attribute_List**: The set of candidate attributes
- **Attribute_Selection_Method**: A procedure to determine the **splitting criterion** that "best" partitions the data instances into individual classes. This criterion consists of a splitting_attribute and, possibly, either a split point or splitting subset

>> Output: A Decision Tree

Decision Tree Induction -- Example #1

Applying the algorithm (from previous slide) of deciding whether to walk or take a bus, we can develop a decision tree by selecting a root node, internal nodes, leaf nodes, and then by defining step-by-step the **splitting criteria** for the class.

We select the **most important node as root node**, which is weather. Depending on the weather and other conditions like time and hunger, we can build our tree as follows:



Decision Tree Induction -- Example #2

UCSD Medical Center (1970s):

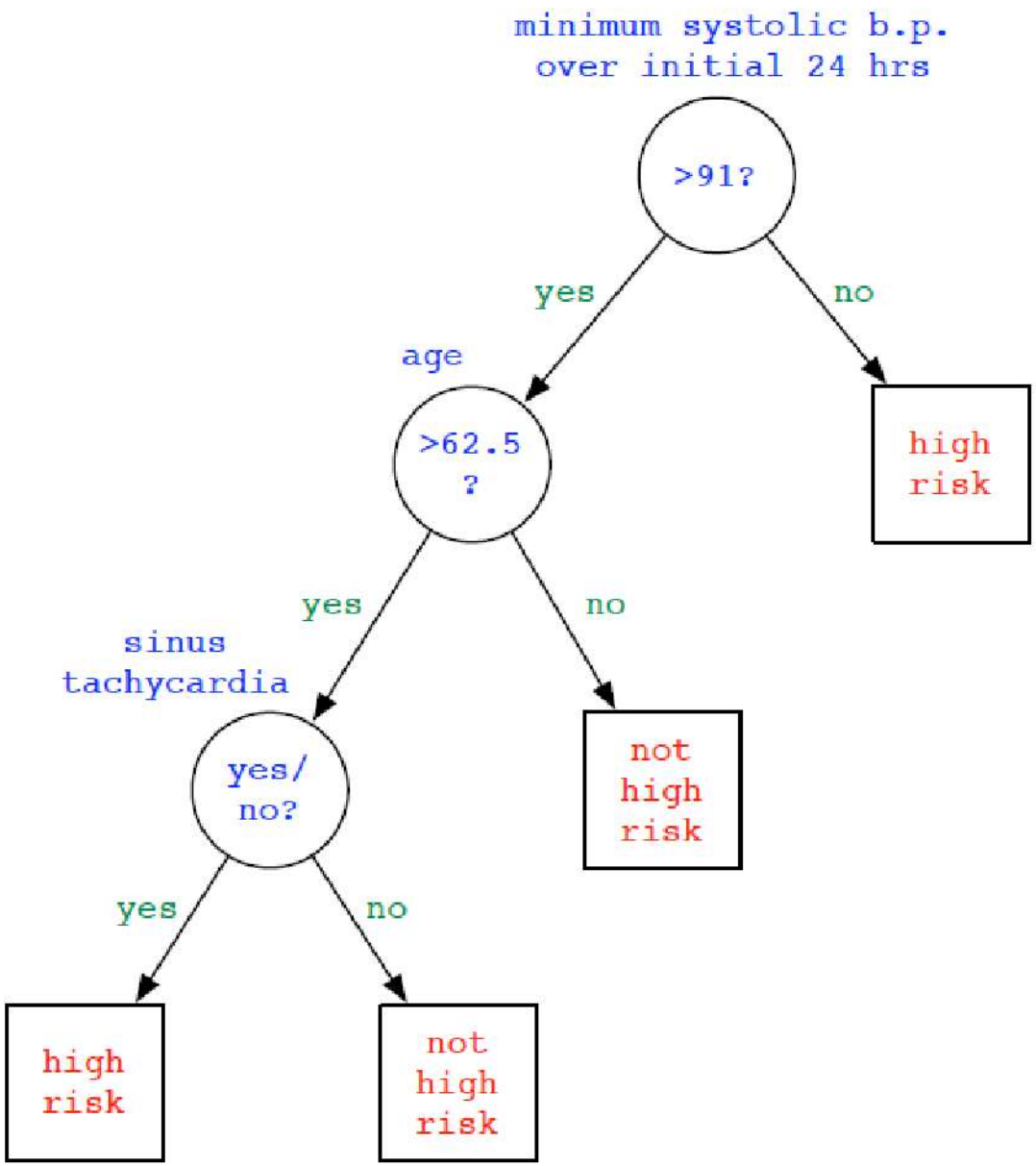
identify patients at risk of dying within 30 days after heart attack.

Data set:

215 patients.

37 (=20%) died.

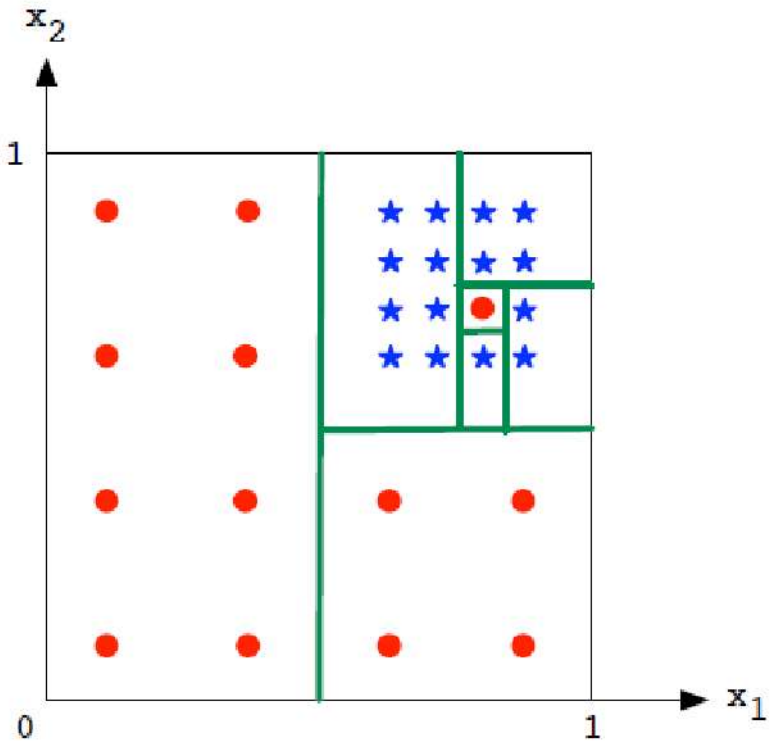
19 features.

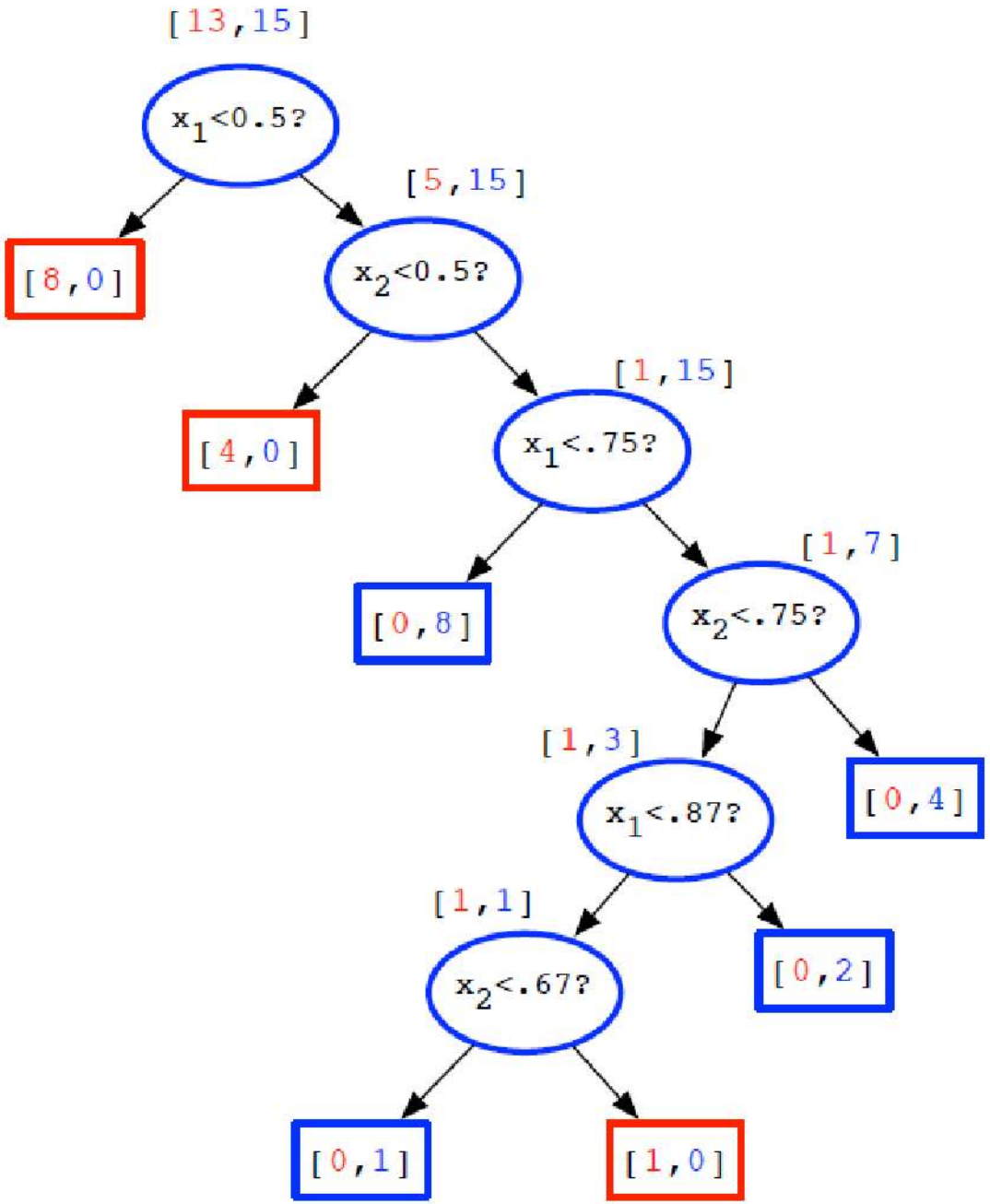


Decision Tree Induction* – Leaf Nodes Example

Continuing from previous slide.

See how the DT '**isolates**' / reaches a leaf node.





What is Decision Tree Induction?

Decision Tree Induction is the process of **automatically constructing** a decision tree from a dataset. It's a core part of Machine Learning, where the algorithm learns patterns from data and builds a tree that can make predictions.

How It Works:

1. **Start with all data** at the root.
2. **Choose the best attribute** to split the data (based on criteria like **Information Gain**, **Gini index**, etc.).
3. **Split the dataset** based on that attribute's values.
4. **Repeat** the process recursively for each subset (creating branches).
5. **Stop** when:
    1. All samples belong to the same class
    2. No more attributes left
    3. Or a stopping condition is met (e.g., max depth, min samples)

Building a Decision Tree

Greedy algorithm: build tree, top-down.

- Start with a single node (root) containing all data points
- Repeat:
  - Look at all current leaves and all possible splits
  - Choose the **split that most reduces uncertainty in prediction**.

    Several ways to quantify: **Gini Index**, **Entropy**, **Information Gain**

When to stop?

- ✓ When each leaf is **pure**?
- ✓ When the tree is already **pretty big**?
- ✓ When each leaf has **uncertainty** below some threshold?

Types of Decision Trees

Types of decision trees are based on the **type of target variable** we have. It can be of two types:

- **Categorical Variable Decision Tree:** Decision Tree which has a categorical target variable then it called a **Categorical variable decision tree.**
- **Continuous Variable Decision Tree:** Decision Tree has a continuous target variable then it is called **Continuous Variable Decision Tree.**

Decision Tree Algorithms

- Decision trees use **multiple algorithms to decide to split** a node into two or more sub-nodes. The creation of sub-nodes increases the **homogeneity** of resultant sub-nodes. In other words, we can say that the **purity of the node increases with respect to the target variable**. The decision tree splits the nodes on all available variables and then **selects the split which results in most homogeneous sub-nodes**.
- The algorithm selection is also based on the type of target variables. Here are some algorithms used in Decision Trees:
- **ID3** → (Iterative Dichotomiser 3, extension of D3, Ross Quinlan in 1980s)
  C4.5 → (successor of ID3, Ross Quinlan in 1993)
  CART → (Classification And Regression Tree, Leo Breiman, 1984)
  CHAID → (Chi-Square Automatic Interaction Detection, performs multi-level splits when computing classification trees, Gordon V. Kass in 1980)
  MARS → (Multivariate Adaptive Regression Splines, Jerome Friedman 1991

The ID3* (Iterative Dichotomiser) Algorithm

ID3 algorithm builds decision trees using a **top-down** greedy search approach through the space of possible branches with no backtracking. A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.

Steps in ID3 algorithm:

- It begins with the **original set** S as the root node.
- On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates **Entropy(H)** and **Information Gain(IG)** of this attribute.
- It then selects the **attribute** which has the **smallest Entropy** or **largest Information Gain**.
- Set S is **split by the selected attribute** to produce a subset of the data
- The algorithm **continues to recur** on each subset, considering only **attributes never selected** before.

(*) **ID3**: An algorithm invented by **Ross Quinlan**

Quinlan, J. R. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (Mar. 1986),

ID3 Algorithm – Properties

ID3 does not guarantee an optimal solution. It can converge upon local optima. It uses a greedy strategy by selecting the locally best attribute to split the dataset on each iteration. The algorithm's optimality can be improved by using backtracking during the search for the optimal decision tree at the cost of possibly taking longer.

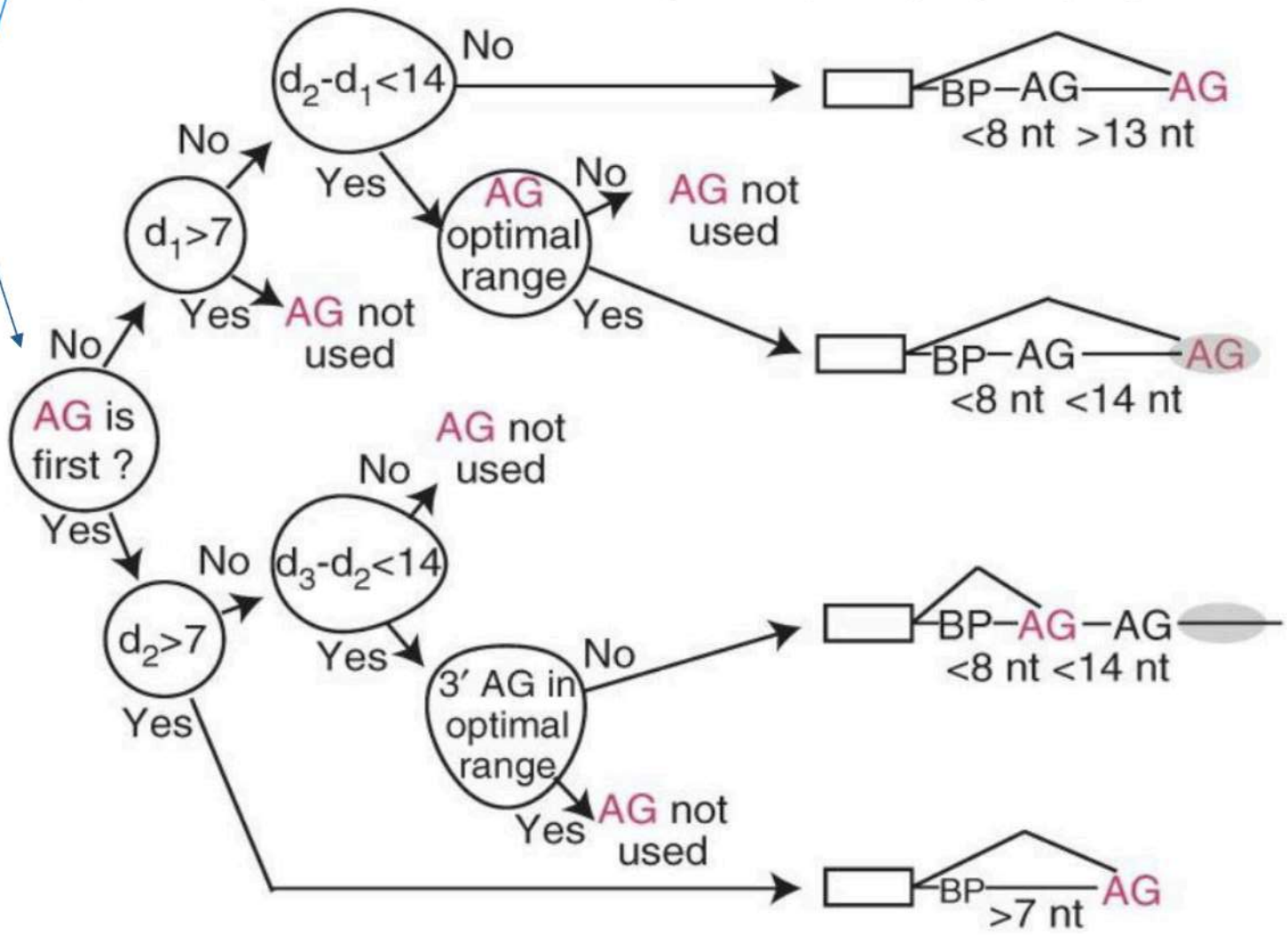
ID3 can **overfit** the training data. To avoid overfitting, smaller decision trees should be preferred over larger ones. This algorithm usually produces small trees, but it does not always produce the smallest possible decision tree.

ID3 is harder to use on continuous data than on factored data (factored data has a discrete number of possible values, thus reducing the possible branch points). If the values of any given attribute are continuous, then there are many more places to split the data on this attribute, and searching for the best value to split by can be time-consuming.

PROTEIN SYNTHESIS

Splicing is the removal of **introns** (non-coding regions) from a **pre-mRNA transcript** and the joining of the remaining **exons** (coding regions) together.

Presence of the AG dinucleotide at root node is a critical feature for identifying 3' splice sites in pre-mRNA sequences. The decision tree uses this feature to **differentiate between splice site and non-splice site regions**, helping in the accurate identification of boundaries between **exons** and **introns** during the **splicing process**.



__ID3-generated decision tree__ used to determine whether a particular nucleotide pair within a pre-mRNA sequence corresponds to an mRNA splice site. This tree has been shown to have a __95% correct prediction rate__.

Decision Trees in Biology – Explain (I)

Context: What are splice sites?

In **DNA/RNA**, your genes have two parts:

- **Exons**: the coding regions (like important "words")
- **Introns**: the non-coding regions (like "junk" that gets removed)

Before a gene can be used to make a protein, the introns need to be **spliced out**, and the exons stitched together. This process happens in a copy of DNA called **pre-mRNA**.

What is a 3' splice site?

- A **3' splice site** is the point **at the end of an intron**, where the cutting happens to remove the intron.
- The body's molecular scissors look for **specific patterns** to know where to cut.

Decision Trees in Biology – Explain (II)

What's special about AG* at the root node?

- **AG** is a **common and critical sequence** at the **3' end** of an intron — it's like a flag that says, "Cut here!"
- The **decision tree** in your image uses **whether this AG pattern is present** as the **first (root) decision**. That means it's the **most important feature** for telling:
  - ✓ "This **is** a splice site" vs.
  - ✓ "This **is not** a splice site"

Think of the **decision tree as a smart yes/no** quiz:

- First question: "Does this location have AG?"
  - Yes → likely a splice site
  - No → probably not

(*) **AG** is a base with 2 **nucleotides**: **Adenine** (**A**) & **Guanine** (**G**)

DNA (The Blueprint of Life), RNA (Messenger & Worker), Nucleotide

A **nucleotide** is the basic building block of DNA and RNA.
Each nucleotide contains:

- A **sugar**
- A **phosphate**
- A **nitrogenous base**

There are **4 main bases** in RNA:

- **A** = Adenine
- **U** = Uracil (replaces T from DNA)
- **G** = Guanine
- **C** = Cytosine

In **DNA**, it's A, T, G, and C — but in **RNA**, T is replaced by U.

Biology & YOU (Life)

So, why are DNA & RNA important for life?

Because **proteins do almost everything in your body** — and DNA & RNA are how those proteins get made.

- No DNA → no instructions.
- No RNA → no way to read or follow the instructions.
- No proteins → no cells, no organs, no YOU.

Decision Trees in Biology – Explain (III)

So, what is the decision tree doing?

- It's trained on examples of actual splice sites and non-splice sites.
- It **learns the patterns** that best distinguish them.
- The **tree starts with AG** at the top (most powerful clue).
- It then checks other smaller details lower down (like surrounding letters, context, etc.).

Why does this matter?

- This kind of model helps **bioinformatics tools** accurately identify where to cut pre-mRNA.
- **Accurate splicing = proper protein creation.**
- Misidentifying splice sites can lead to **genetic diseases** or malfunctioning proteins.

Decision Tree Node Splitting Approach

From slide #29…

An *attribute selection measure* is a **heuristic** for selecting the **splitting criterion** that's most capable of deciding how to partition data in such a way that would **result in individual classes**.

Attribute selection measures are also known as **splitting rules** because they define how the data points are to be split on a certain level in a decision tree.

The **attribute** that has the **best score** for the measure is chosen as the **splitting attribute** for the given data points.

Decision Tree – Attribute Selection Measures

Most well-known **attribute selection methods**:

- **Entropy**
- **Information Gain** (IG)
- Gain Ratio (GR)
- **Gini Index** (GI)
- Pruning
- Reduction in Variance
- **Chi-Square**

Criteria will calculate values for every attribute. Values are sorted, and attributes are placed in the tree by following the order i.e., the attribute with a high value (in case of information gain) is placed at the root.

Using **Information Gain** (**IG**) as criterion, attributes should be **categorical**, and for the **Gini Index** (**GI**), attributes are assumed to be **continuous**.

The Concept of Entropy

Entropy is a scientific concept, as well as a **measurable physical property**, that is most commonly associated with a state of disorder, randomness, or uncertainty.

The term and the concept are used in diverse fields:

- Classical Thermodynamics, where it was first recognized
- To the microscopic description of nature in Statistical Physics
- Principles of Information Theory (Shannon* Entropy).

Entropy

A measure of disorder, uncertainty, and randomness.

It is **zero for a perfectly ordered system**.

(*) Claude Shannon: The father of **Information Theory.**
One of the founders of Artificial Intelligence (Summer 1955 at Dartmouth)

Entropy – 2nd Law of Thermodynamics

Concept of **entropy** is a fundamental idea that plays a crucial role in various fields, including **thermodynamics**, **statistical physics**, & **information theory**.

Let's understand what entropy means in each context.

1. Entropy in Classical Thermodynamics

- **Definition**: In classical thermodynamics, **entropy** (denoted as S) is a measure of the **disorder** or **randomness** of a system. It was first introduced by the physicist **Rudolf Clausius** in the 19th century.
- **Key Concept**: Entropy is linked to the **Second Law of Thermodynamics**, which states that the **total entropy of an isolated system can never decrease over time**. In simpler terms, natural processes tend to move toward a state of higher entropy, meaning **systems evolve toward greater** disorder or equilibrium.

Example:

- When **ice melts into water**, the molecules move from a highly ordered state (solid) to a more disordered state (liquid), resulting in an increase in entropy.
- In a heat engine, such as a car engine, **some energy is always lost as heat**, increasing the entropy of the surroundings and making it impossible to convert all the energy into useful work.

Entropy – Statistical Physics

2. Entropy in Statistical Physics

- **Definition**: In the realm of statistical physics, entropy is given a more **microscopic interpretation**. Here, entropy measures the number of possible microstates (i.e., arrangements of particles) that correspond to a system's macroscopic state.
- **Key Concept**: This version of entropy formalized by **Ludwig Boltzmann**
- **Interpretation**:
  - ✓ If a system can exist in many different configurations (**microstates**) while maintaining the same overall appearance (**macrostate**), it has high entropy.
  - ✓ Systems naturally evolve toward configurations with the highest number of microstates, thus maximizing entropy.

Example:

- ❖ A **gas in a container has higher entropy than a solid** because gas molecules can move freely, resulting in many possible arrangements, while molecules in a solid are tightly packed in a fixed structure.

Entropy – Information Theory

3. Entropy in Information Theory (Shannon Entropy)

- **Definition**: In information theory, **entropy** is a measure of **uncertainty** or **information content** in a message or data set (based in probability distribution.) This concept was introduced by **Claude Shannon** in 1948.
- **Key Concept**: Shannon entropy quantifies the average amount of information produced by a stochastic source of data. Calculated using this formula:

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i$$

- H is the entropy.
- p_i is the probability of occurrence of the i^{th} possible event.
- The sum is taken over all possible events in the system.

Interpretation:

- Shannon entropy is used to **measure the efficiency of communication systems**, where a **higher entropy indicates more uncertainty or variability** in the information source.
- System with low entropy (e.g., a repetitive, predictable message) carries less information, while a system with high entropy (e.g., a random string of bits) carries more information.

Example:

- In digital communication, Shannon entropy can help determine the optimal way to encode data to minimize the number of bits required, thereby **increasing the efficiency of data transmission**.

The Shannon Entropy formula* $H = -\sum_i p_i \log p_i$

Interpretation:

1. **Each logp_i** measures how much information (or surprise) we get when observing event i.
    1. Rare events carry more "surprise" (higher information content).
    2. Common events carry less.
2. **Multiplying by p_i** weights that surprise by how often it occurs.
3. **Summing** over all events gives the **average information content** — that's the **entropy**.

Intuition:

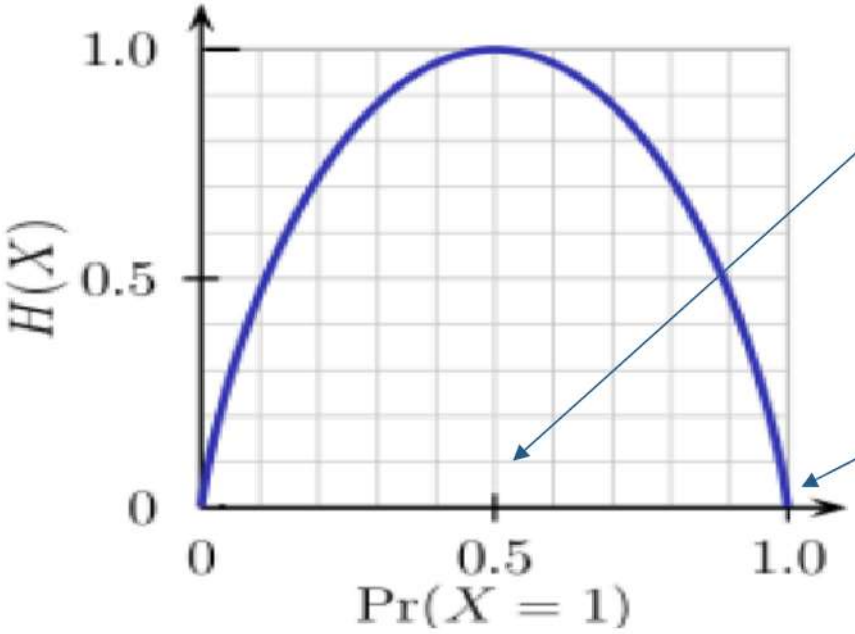
Entropy tells you **how "surprising" or "uncertain"** a random variable is.

- If the distribution is **uniform** (all outcomes equally likely), entropy is **high** — maximum uncertainty.
- If one outcome is **much more likely** than others, entropy is **lower** — less uncertainty.
- If one outcome is **certain** (i.e., probability = 1), entropy is **zero** — no uncertainty.

(*) The negative sign (–): Because logp_i is negative for $0<p_i<1$, and entropy is a **non-negative** measure.

Entropy – Example

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



>>> **Entropy is maximum** when the probability is 0.5 because it projects perfect randomness in the data and there is no chance in perfectly determining the outcome.

>>> **Entropy H(X) is zero** when the probability is either 0 or 1 (validate it via the formula.)

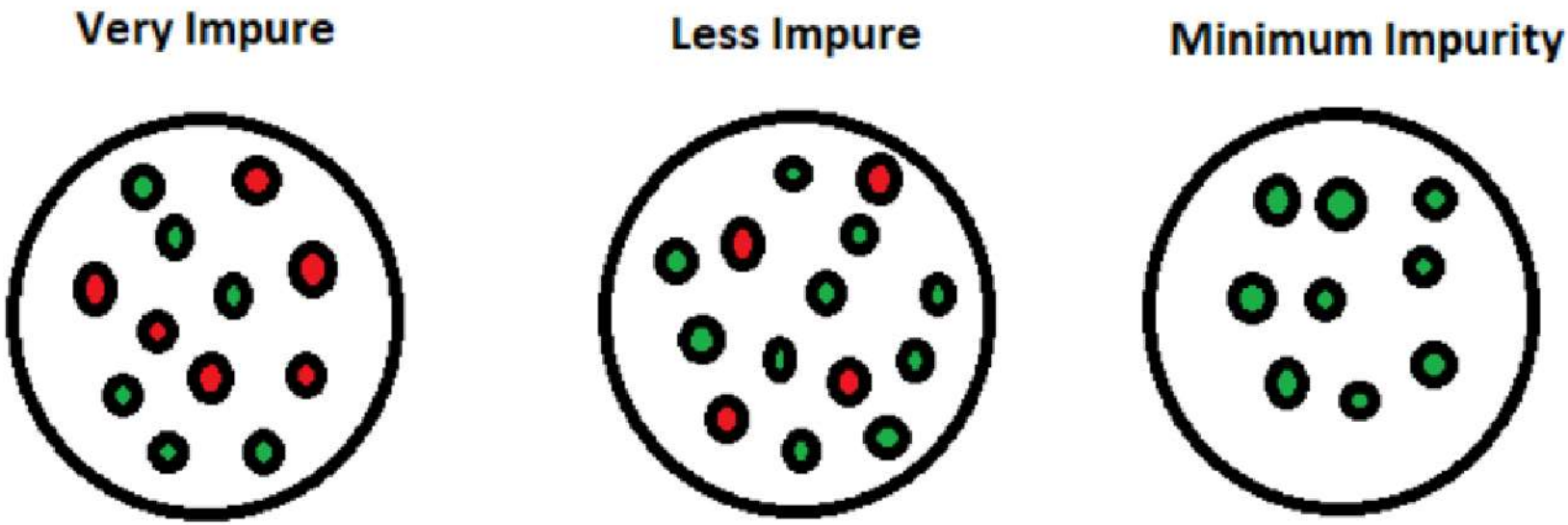
ID3 algorithm follows this rule:

A (tree) branch with an entropy of zero is a leaf node and a branch with entropy more than zero needs further splitting.

Attribute Selection Measures – Entropy (in DT)

Entropy is an information theory metric that measures the **impurity** or **uncertainty** in a group of observations. It **determines how a decision tree chooses to split data**.

The image below gives a better **description of the purity** of a set.



Entropy – Mathematical Formula (I)

Mathematically Entropy for one (1) attribute is calculated by this formula:

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

5/14 = **0.36**

9/14 = **0.64**

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

= Entropy (0.36, 0.64)

= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)

= 0.94

Where **S → Current state, and Pi → Probability of an event *i* of state S or Percentage of class *i* in a node of state S.**

Entropy – Mathematical Formula (II)

Mathematically Entropy for multiple attributes is represented as:

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

where **T**→ **Current state and**
X → **Selected attribute**

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

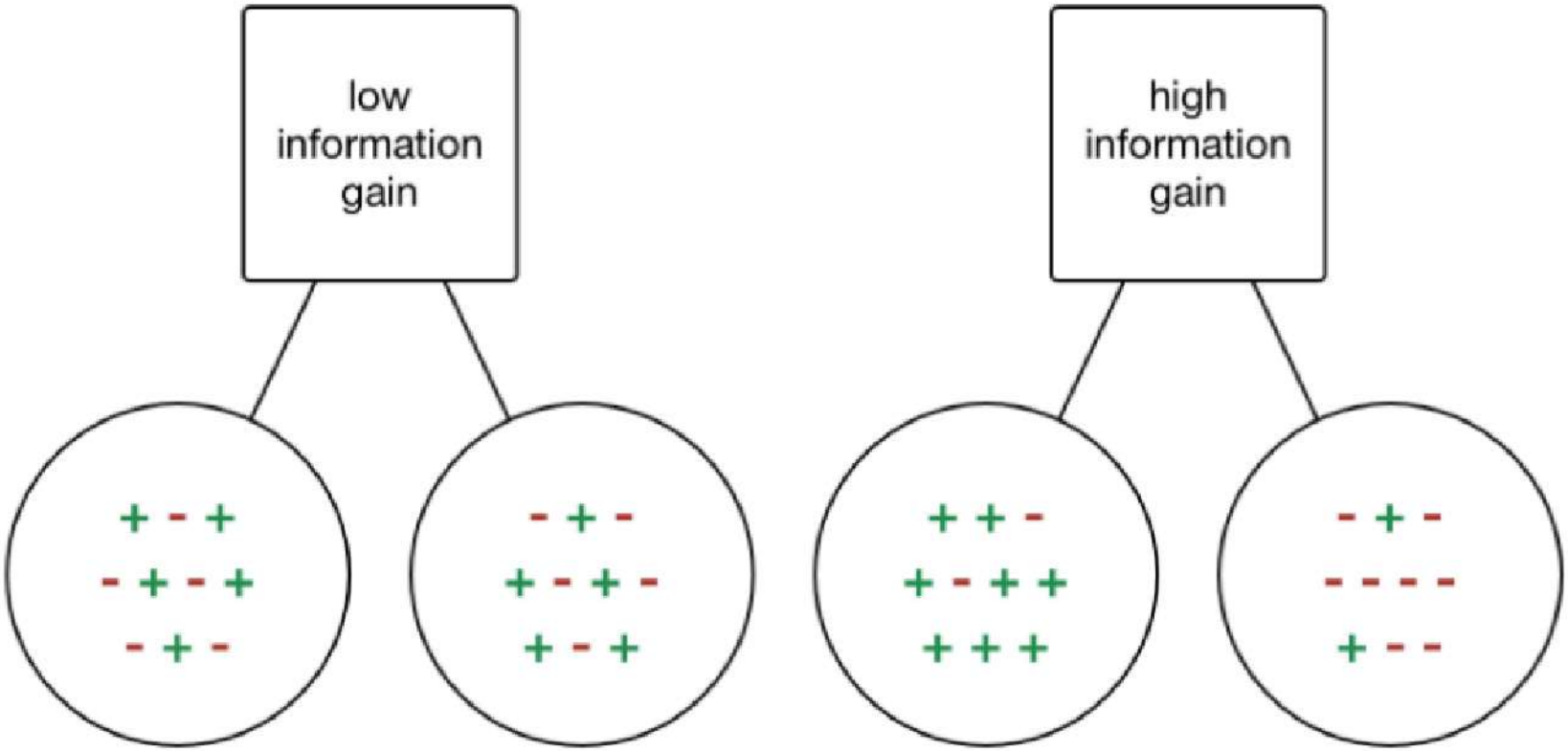
E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Information Gain (IG)

Information Gain : A statistical property that **measures** **how well a given attribute separates** **the training examples** according to their target classification



Information Gain is a <u>decrease</u> in Entropy. It computes the difference between entropy <u>before</u> split and average entropy <u>after</u> split of the dataset based on given attribute values. ID3 decision tree algorithm uses Information Gain metric.

Constructing a Decision Tree is all about finding an attribute that returns:

>>> Highest Information Gain and

>>> Smallest Entropy.

Attribute Selection Measures – Information Gain

Information Gain (IG) is a decrease in entropy.

IG = **entropy before split** minus **average entropy after split** of the dataset based on given attribute values.

$$\text{Information Gain}(T,X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

$$\begin{aligned} \text{IG(PlayGolf, Outlook)} &= \text{E(PlayGolf)} - \text{E(PlayGolf, Outlook)} \\ &= 0.940 - 0.693 \\ &= 0.247 \end{aligned}$$

In a much simpler way, we can conclude that:

$$Information\ Gain = Entropy(before) - \sum_{j=1}^{K} Entropy(j, after)$$

Where "before" is the dataset **before the split**, K is the number of subsets generated by the split, and (**j**, after) is subset j after the split.

The Gini Index (GI)

Gini Index is a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of each class from one. It **favors larger partitions** and easy to implement whereas information gain favors smaller partitions with distinct values.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

>>> **Gini Index works with the categorical target variable** "Success" or "Failure". It performs only Binary splits.

>>> **Higher value of Gini index** implies higher inequality, **higher heterogeneity**.

Gini Index Calculation

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Steps to Calculate Gini index for a split

- Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p^2+q^2).
- Calculate the Gini index for split using the weighted Gini score of each node of that split.
- **CART*** (Classification and Regression Tree) uses the **Gini Index** method (by default) to create split points.

(*)CART (**C**lassification **A**nd **R**egression **T**ree) is a variation of the decision tree algorithm. It can handle both classification and regression tasks.
Scikit-Learn uses the Classification And Regression Tree (CART) algorithm to train Decision Trees
CART was first produced by **Leo Breiman**, **Jerome Friedman**, Richard Olshen, and Charles Stone in 1984.

Gini Index & Information Gain – Summary

Algo / Split Criterion	Description	Tree Type
Gini Split / Gini Index	Favors larger partitions. Very simple to implement.	CART
Information Gain / Entropy	Favors partitions that have small counts but many distinct values.	ID3 / C4.5

Gini Index

- Favors larger partitions.
- Uses squared proportion of classes.
- Perfectly classified, Gini Index would be zero.
- Evenly distributed would be 1 – (1/# Classes).
- You want a variable split that has a low Gini Index.
- The algorithm works as 1 – (P(class1)^2 + P(class2)^2 + … + P(classN)^2)

Information Gain / Entropy

- Favors splits with small counts but many unique values.
- Weights probability of class by log(base=2) of the class probability
- A smaller value of Entropy is better. That makes the difference between the parent node's entropy larger.
- Information Gain is the Entropy of the parent node minus the entropy of the child nodes.
- Entropy is calculated [P(class1)*log(P(class1),2) + P(class2)*log(P(class2),2) + … + P(classN)*log(P(classN),2)]

CART Decision Tree in Scikit-Learn

sklearn.tree.DecisionTreeClassifier

sklearn.tree.DecisionTreeClassifier

class **sklearn.tree.DecisionTreeClassifier**(*, *criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0*) [sourc

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy", "log_loss"}, default="gini"*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "log_los and "entropy" both for the Shannon information gain, see Mathematical formulation.

splitter : *{"best", "random"}, default="best"*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

Continued from slide #46 – Attribute Selection Methods...

The Gain Ratio (GR)

Gain Ratio – an enhancement of Information Gain!

Information Gain is inherently biased toward attributes with many distinct values, often leading to overfitting by favoring highly specific splits as root nodes.

>>> **C4.5**, an improvement of ID3, uses **Gain Ratio** which is a modification of Information gain that reduces its bias and is usually the best option.

>>> Gain Ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split.

>>> Gain Ratio corrects information gain by taking the Intrinsic Information (II) of a split into account (SplitInfo). (see next slide…)

The Gain Ratio Formula

Gain Ratio attempts to lessen the bias of Information Gain on highly branched predictors by introducing a normalizing term called the **Intrinsic Information**.

$$GainRatio = \frac{\text{Information Gain}}{\text{Intrinsic Information}}$$

The **Intrinsic Information (II)** is defined as the **entropy of sub-dataset proportions**. In other words, it is **how hard for us to guess in which branch a randomly selected sample is put into**.

$$II = -(\sum \frac{|D_j|}{|D|} * \log_2 \frac{|D_j|}{|D|})$$

A better, complete formula for GR is:

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^{K} Entropy(j, after)}{\sum_{j=1}^{K} w_j\ log_2\ w_j}$$

Where "before" is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Attribute Selection Measures -- Reduction in Variance

Reduction in Variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the **standard formula of variance** to choose the best split. The **split with lower variance** is selected as the criteria to split the population:

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for **each split as the weighted average of each node variance**.

Attribute Selection Measures -- Chi-Square

CHAID (*Chi*-squared Automatic Interaction Detector) is one of the oldest tree classification methods.

- It finds out the **statistical significance** between the differences between sub-nodes and parent node.
- We measure it by the **sum of squares of standardized differences** between observed and expected frequencies of the target variable.
- It works with the **categorical target variable** "Success" or "Failure". It can perform two or more splits.
- **Higher** the value of **Chi-Square higher** the **statistical significance** of differences between sub-node and Parent node.
- It generates a **tree** called **CHAID** (Chi-square Automatic Interaction Detector).

Chi-Squared Formula

Mathematically, **Chi-squared** is represented as:

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

Where:

χ^2 = Chi Square obtained
$\sum$ = the sum of
O = observed score
E = expected score

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

CHAID (*Chi*-squared Automatic Interaction Detector) vs Others

Comparison of CHAID with Other Decision Tree Algorithms:

Feature	ID3	C4.5	CART	CHAID
Splitting Criterion	Information Gain	Gain Ratio	Gini Impurity / MSE	Chi-square Test
Split Type	Multi-way	Multi-way	Binary Only	Multi-way
Handles Continuous Data	No	Yes	Yes	Yes (with Interval Binning)
Pruning	No	Post-pruning	Cost-complexity Pruning	Implicit via Significance Testing
Task Type	Classification Only	Classification Only	Classification & Regression	Classification Only
Handling Missing Values	No	Yes	No (basic version)	Yes (by treating as separate category)

The Biggest Issue dealing with Decision Trees Algorithms is overfitting* (aka variance**)

(*) Overfitting: Good performance on training data,
Poor generalization to other (validation, test) data.

(**) Bias vs Variance Dilemma / Tradeoff: The conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set.

Decision Tree Overfitting Problem

How to avoid/counter Overfitting in Decision Trees?

- The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus, this affects the accuracy when predicting samples that are not part of the training set (**validation**, **test** datasets).

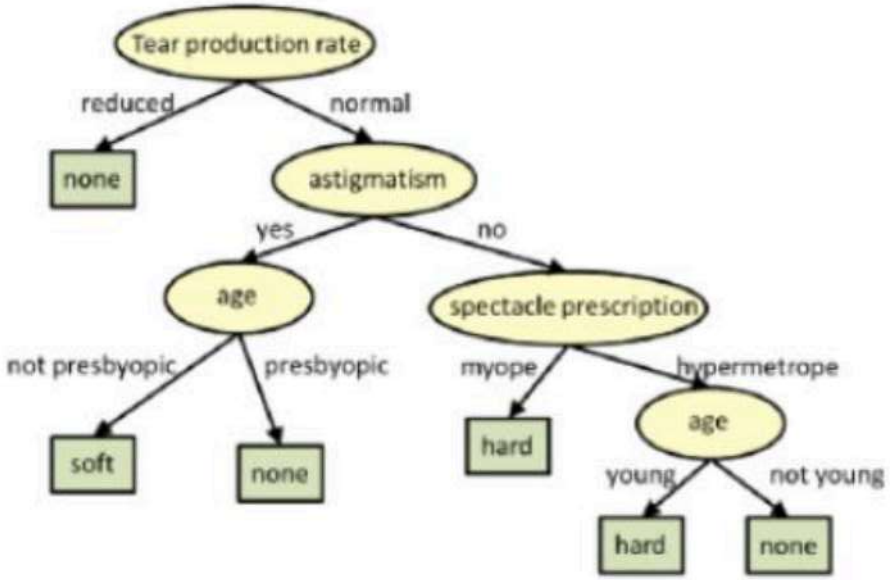
Here are two ways to deal with (fight back!) overfitting:

- **Pruning** Decision Trees.
- **Random Forest** (RF) Algorithm

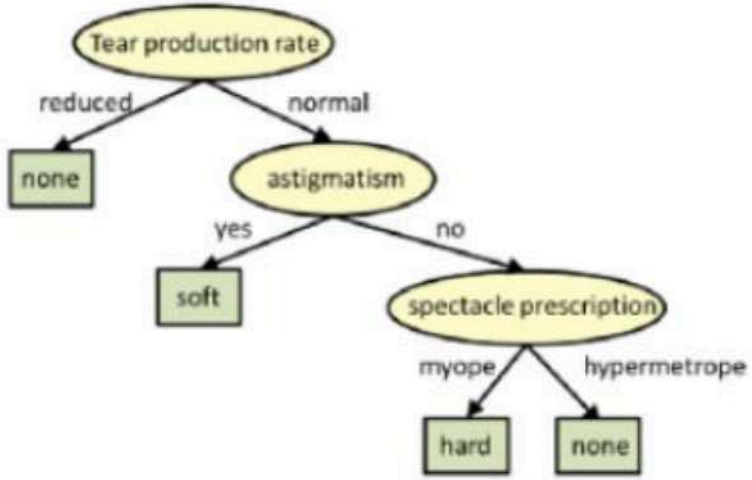
Pruning Decision Trees

Splitting process results in fully grown trees until the stopping criteria are reached. Fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data.

In **pruning**, you trim off the branches of the tree, i.e., **remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed**.



Original Tree



Pruned Tree

In above, the 'Age' attribute in the left-hand side of the tree has been pruned as it has less importance on the right-hand side of the tree, hence **removing overfitting**.

Random Forest (RF) – Ensemble Method

Random Forest is an example of **ensemble learning**, in which we combine multiple machine learning algorithms to obtain better predictive performance (weak learner → strong learner)

Why the name "Random"?

Two key concepts that give it the name random:

1. A random sampling of training data set when building trees.
2. Random subsets of features considered when splitting nodes.

A technique known as **bagging (Boosting Aggregation)** is used to create an ensemble of trees where multiple training sets are generated **with replacement**.

In the bagging technique, a data set is divided into **N** samples using randomized sampling. Then, using a single learning algorithm a model is built on all samples. Later, the resultant predictions are combined using **voting or averaging** in parallel.

Decision Trees in Scikit-Learn

Implementing **decision trees in scikit-learn**

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.tree.export import export_text
>>> iris = load_iris()
>>> decision_tree = DecisionTreeClassifier(random_state=0,
max_depth=2)
>>> decision_tree = decision_tree.fit(iris.data, iris.target)
>>> r = export_text(decision_tree,
feature_names=iris['feature_names'])
>>> print(r)
|--- petal width (cm) <= 0.80
|    |--- class: 0
|--- petal width (cm) >  0.80
|    |--- petal width (cm) <= 1.75
|    |    |--- class: 1
|    |--- petal width (cm) >  1.75
|    |    |--- class: 2
```

Parameter tuning in scikit-learn:

1.max_depth: *int, default=None. defines t*he maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

2.min_samples_split: *int or float, default=2.* The minimum number of samples required to split a node—a minimum of 2 samples are required for splitting the node. Can be tuned as per the features and their values. (Helps control tree growth and prevents overfitting)

3.min_samples_leaf: *int or float, default=1.* Provides information about *t*he minimum number of samples required to be at a leaf node. This helps in smoothing the model.

Advantages of Decision Trees

- Interpretable and Simple: Decision trees are able to generate understandable rules. The trees are simple to understand and to interpret and can be visualized (MLI: ML Interpretability).
- Handle all kinds of data well: Decision trees can handle both numerical and categorical data, making them widely-useable.
- Non-Parametric: Decision trees are considered to be non-parametric. This means that decision trees have **no assumptions about the data points'** space or the classifier's structure, nor is there a need for assuming any seed values.
- Robust: Decision trees require less effort from users for pre-processing data. They aren't influenced by outliers and missing values either.
- Fast: The cost of using the tree (i.e. making predictions) is logarithmic in the number of data points (why?) used to train the tree.

Disadvantages of Decision Trees

- Overfitting: Overly complex trees can be developed due to overfitting. Pruning, setting the minimum number of samples required at a leaf node, or setting the max depth of tree are necessary steps to avoid this problem.
- Instability: Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. Ensemble techniques like bagging and boosting can help avoid such instabilities in outcomes.
- Bias: Decision tree learners create biased trees if some classes are more likely to be predicted or have a higher number of samples to support them. Balancing the dataset before decision tree induction is a good practice to provide every class with a fair and equal chance.
- Optimality: The problem of learning an **optimal decision tree** is known to be NP-complete, since the number of samples or a slight variation in the splitting attribute can change results drastically.

The NP-complete issue is a fundamental concept in computer science related to the field of computational complexity theory.

It deals with the classification of problems based on their inherent difficulty and the resources needed to solve them.

P vs NP vs NP-Complete

Key Concepts:

1. P vs. NP:

1. **P** (Polynomial time): The set of problems that can be solved efficiently (in polynomial time) by a deterministic algorithm. These are problems for which an algorithm exists that can produce a solution relatively quickly as the size of the input grows.
2. **NP** (Nondeterministic Polynomial time): The set of problems for which, given a solution, it can be verified quickly (in polynomial time) whether that solution is correct. However, finding the solution may not necessarily be quick.

2. NP-Complete:

1. An **NP-complete** problem is a special type of problem that is both in **NP** and **NP-hard**.
2. **NP-hard** problems are at least as hard as the hardest problems in NP, meaning no polynomial-time algorithm is known to solve these problems efficiently (unless **P = NP**).
3. If any NP-complete problem can be solved in polynomial time, then all problems in NP can be solved in polynomial time (this would imply **P = NP**).
4. Conversely, if no NP-complete problem can be solved in polynomial time, then **P ≠ NP**.

Characteristics of NP-Complete Problems:

- They are **decision problems**, meaning the answer is either "yes" or "no".
- They are the **most challenging problems** in NP because they can be reduced to each other in polynomial time (using a process called **polynomial-time reduction**).
- If you can **solve one NP-complete problem efficiently, you can solve all NP problems efficientl**y.

Examples of NP-Complete Problems:

1. **Traveling Salesman Problem (Decision Version)**: Given a list of cities and the distances between them, is there a route that visits each city exactly once and returns to the origin with a total distance less than or equal to a given number?
2. **Knapsack Problem**: Given a set of items, each with a weight and a value, can you fit a selection of these items into a knapsack with a limited capacity to achieve at least a given value?
3. **Boolean Satisfiability Problem (SAT)**: Given a Boolean expression, can the variables be assigned in such a way that the entire expression evaluates to true?
4. **Graph Coloring**: Can the vertices of a graph be colored with a given number of colors so that no two adjacent vertices have the same color?

NP-Complete Summary

Why It Matters:

- The question of whether **P = NP** is one of the **most important open questions** in computer science. It has far-reaching implications for fields like cryptography, optimization, algorithm design, artificial intelligence, and more.
- **Clay Mathematics Institute's Millennium Prize**: A $1 million prize is offered for a proof or disproof of whether **P = NP**.

In summary, the **NP-complete issue** centers around the difficulty of solving problems that, while their solutions can be quickly verified, do not have a known efficient way to find the solution.

The **P vs. NP problem** remains one of the most profound unsolved problems in computer science.

Questions?