MATH-810 Mathematical Methods for Artificial Intelligence

DFS & BFS

Dr. Yasir Ali

DBS&H, CEME
National University of Sciences and Technology

Islamabad, Pakistan.

September 24, 2025

- $w : E \to \mathbf{R}$ is known as *weight function*
- w to an edge is called *weight of the edge*
- A graph $G = (V, E)$ in which each edge has a weight is called a *weighted graph.*
- Let $S = v_1, v_2, \ldots, v_k$ be a path in a weighted graph G then weight of S is given by

$$w(S) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

- A *shortest path* from a vertex v to a vertex v' is defined as any $v - v'$ path S with weight $w(S) = d(v, v')$ where $d(v, v')$ is given by

$$d(v, v') = \begin{cases} \min\{w(\widetilde{P}) \mid \widetilde{P} \text{ is a } v - v' \text{ path}\} & \text{if } v' \text{ is reachable from } v \\ +\infty & \text{otherewise.} \end{cases}$$

- A cycle C in a weighted graph with $w(C) < 0$ is called a *negative cycle.*

- The idea is to visit the nodes in order of their closeness to *source vertex* s.

The closest node to s, say x, must be adjacent to s and the next closest node, say y, must be either adjacent to s or x. The third closest node to s must be either adjacent to s or x or y, and so on.

Predecessor $\pi[v]$ & Shortest Path Estimate $d[v]$

We set the π attribute so that the chain of predecessors originating at a vertex v runs backwards along the shortest path form vertex s to v.

- Given a graph $G(V, E)$, we maintain for each vertex $v \in V$ a *predecessor* $\pi[v]$ that is either a vertex or `NIL`

For each vertex $v \in V$, we maintain an attribute $d[v]$, which is an upper bound on the weight of a shortest path from source s to vertex v.

- We call $d[v]$ a *Shortest Path Estimate*

We initialize the Shortest Path Estimate and Predecessor by the following procedure:

`INITIALIZE-SINGLE-SOURCE` (G, s)

1. **for** each vertex $v \in V$
2. **do** $d[v] \leftarrow \infty$
3. $\pi[v] \leftarrow$`NIL`
4. $d[s] = 0$

After initialization, $\pi[v] =$`NIL`, $d[v] = 0$ for $v = s$ and $d[v] = \infty$ for all $v \in V - s$

Relaxation

- The process of relaxing an edge (u, v) consist of testing whether we can improve the shortest path to v found so far by going through u and, if so, update $d[v]$ and $\pi[v]$.

A relaxation step may decrease the shortest path estimate $d[v]$ and updates's v's predecessor field $\pi[v]$.

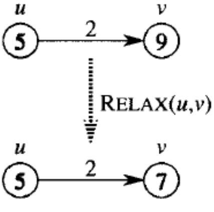
`RELAX` (u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$
2. **then** $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

`RELAX` (u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$ (if new shortest path found)
2. **then** $d[v] \leftarrow d[u] + w(u, v)$ (set new value of shortest path)
3. $\pi[v] \leftarrow u$ (add u to predecessor of v)

Relaxation of an edge (u, v).



(a)

(a) Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases.

RELAX (u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$ (if new shortest path found)
2. **then** $d[v] \leftarrow d[u] + w(u, v)$ (set new value of shortest path)
3. $\pi[v] \leftarrow u$ (add u to predecessor of v)

Relaxation of an edge (u, v).

(a) (b)

(a) Because $d[v] > d[u] + w(u, v)$ prior to relaxation, the value of $d[v]$ decreases.

(b) Here, $d[v] \leq d[u] + w(u, v)$ before the relaxation step, so $d[v]$ is unchanged by relaxation.

Dijkstra's algorithm maintains a set S of those vertices for which the shortest path weights from the source s have already been determined. The algorithm repeated selects the vertices $u \in V \setminus S$ with shortest path estimates, and insert u into S, and relaxes all the edges leaving u In the following implementation, we maintain a priority queue Q that contains all the vertices in $V \setminus S$ keyed by their d values.

DIJKSTRA (G, w, s)

INITIALIZE-SINGLE-SOURCE (G, s) (distance to source vertex is zero)
(set all other distances to infinity)

$S \leftarrow \emptyset$ (S, the set of visited vertices is initially empty)

$Q \leftarrow V[G]$ (Q, the queue initially contains all vertices)

while $Q \neq \emptyset$ (while the queue is not empty)

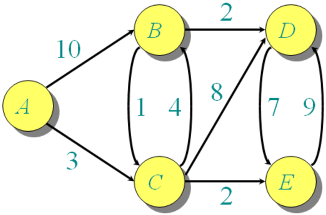
  do $u \leftarrow EXTRACT - MIN(Q)$ (select the element of Q with the min. distance)

   $S \leftarrow S \cup \{u\}$ (add u to list of visited vertices)

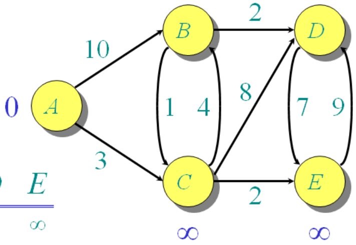
   for each vertex $v \in Adj[u]$

    do RELAX(u, v, w) (if new shortest path found)
(set new value of shortest path)

Graph with nonnegative edge weights:

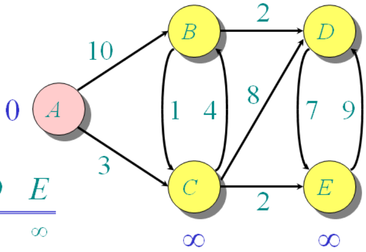


Initialize:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

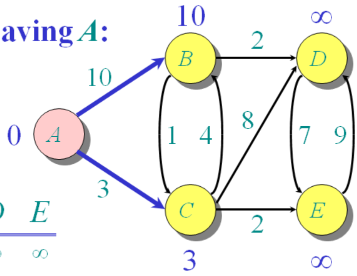
S: {}



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Relax all edges leaving A:

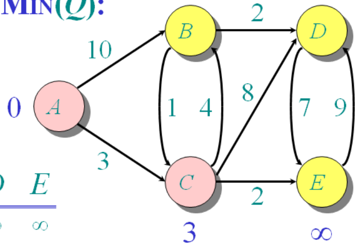


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: $\{ A \}$

Dijkstra's Algorithm Implementation

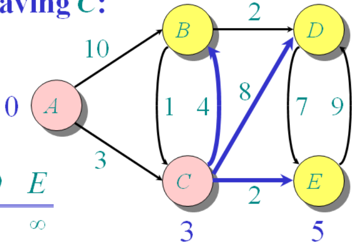
"C" $\leftarrow$ **EXTRACT-MIN(Q):**



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞

S: { A, C }

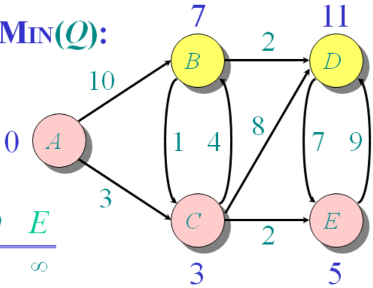
Relax all edges leaving C:



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	5

S: { A, C }

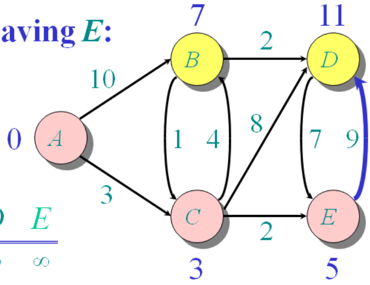
"E" $\leftarrow$ EXTRACT-MIN(Q):



Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$

S: { A, C, E }

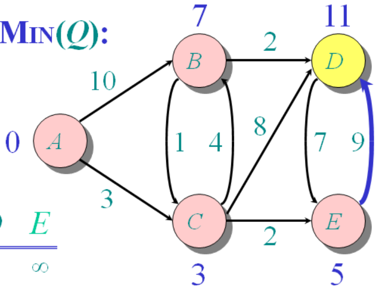
Relax all edges leaving *E*:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

"B" ← EXTRACT-MIN(Q):

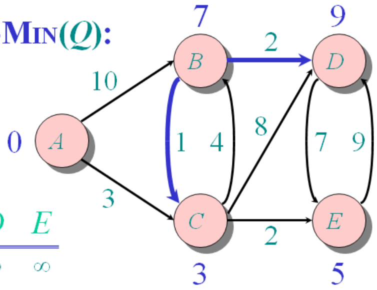


Q:

A	B	C	D	E
$\boxed{0}$	∞	∞	∞	∞
	10	$\boxed{3}$	∞	∞
	7		11	$\boxed{5}$
	$\boxed{7}$		11	

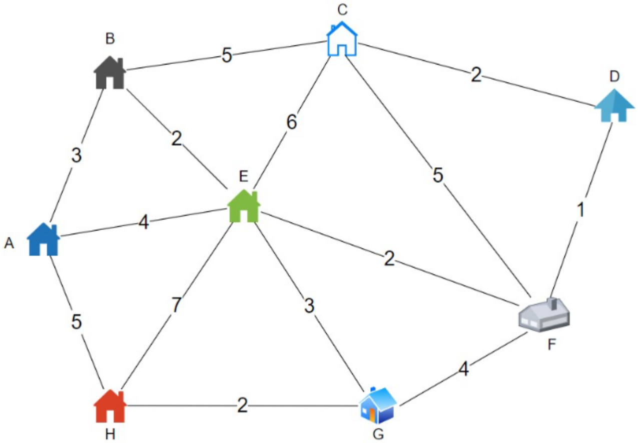
S: { A, C, E, B }

"D" $\leftarrow$ **EXTRACT-MIN(Q):**

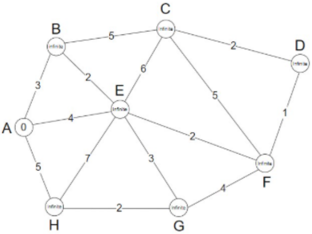


Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$
		$\boxed{7}$		11	
				$\boxed{9}$	

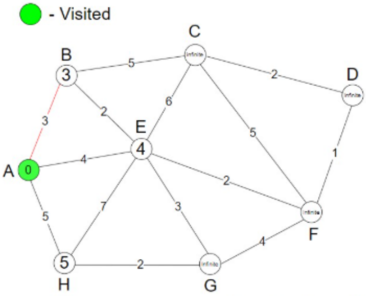
S: { A, C, E, B, D }


C
B
5
2
D
6
2
3
5
E
1
4
A
2
7
3
5
F
4
2
H
G

Step 1 – Start at the source node (let's say A) with distance 0 and all other nodes to infinity.



	B	C	D	E	F	G	H
A	Infinite	Infinite	Infinite	Infinite	Infinite	Infinite	Infinite



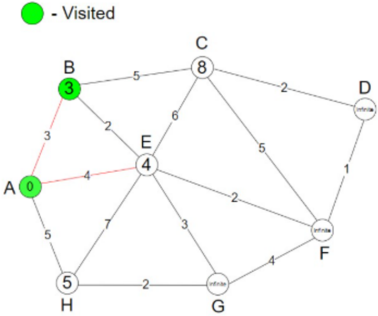
Step 2 – We mark the node A as visited.

A —> B, Cost of AB is 3, (Shortest Distance)

A —> E, Cost of AE is 4

A —> H, Cost of AH is 5

	B	C	D	E	F	G	H
A	3	Infinite	Infinite	4	Infinite	Infinite	5



Step 3 – We mark the node B as visited.

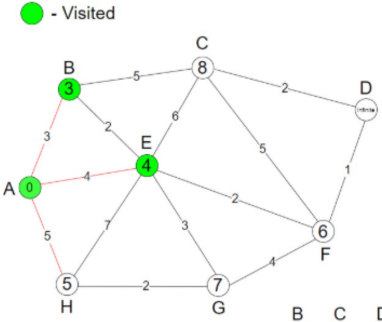
A —> B —> C, Cost of ABC is 8

A —> B —> E, Cost of ABE is 5

A —> E, Cost of AE is 4, (Shortest Distance)

A —> H, Cost of AH is 5

	B	C	D	E	F	G	H
A	3	8	Infinite	4	Infinite	Infinite	5



Step 4 – We mark the node E as visited.

A —> B —> C, Cost of AC is 8

A —> E —> F, Cost of AEF is 6

A —> E —> G, Cost of AEG is 7

A —> H, Cost of AH is 5, (Shortest Distance)

	B	C	D	E	F	G	H
A	3	8	Infinite	4	6	7	5

Step 5 – We mark the node H as visited.

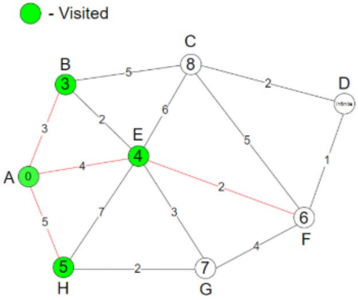
A —> B —> C, Cost of AC is 8

A —> E —> F, Cost of AEF is 6, (Shortest Distance)

A —> E —> G, Cost of AEG is 7

A —> H —> G, Cost of AHG is 7

We have to find the minimum cost. So, we update the shortest distance (AEF, 6).



	B	C	D	E	F	G	H
A	3	8	Infinite	4	6	7	5

Step 6 – We mark the node F as visited.

A —> B —> C, Cost of AC is 8

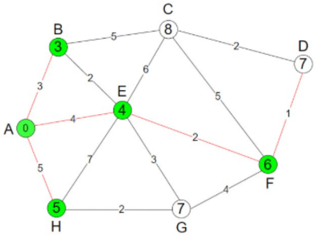
A —> E —> F —> D, Cost of AEFD is 7, (Shortest Distance)

A —> E —> G, Cost of AEG is 7

A —> H —> G, Cost of AHG is 7

We have to find the minimum cost. So, we update the shortest distance (AEFD, 7).

- Visited



	B	C	D	E	F	G	H
A	3	8	7	4	6	7	5

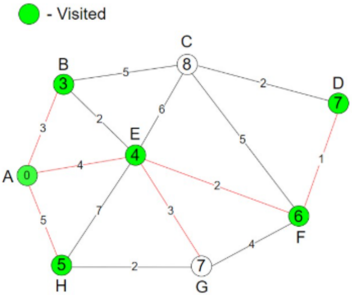
Step 7 – We mark the node D as visited.

A —> B —> C, Cost of AC is 8

A —> E —> G, Cost of AEG is 7, (Shortest Distance)

A —> H —> G, Cost of AHG is 7

We have to find the minimum cost. So, we update the shortest distance (AEG, 7).



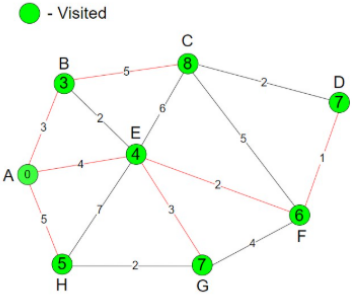
	B	C	D	E	F	G	H
A	3	8	7	4	6	7	5

Step 8 – We mark the node G as visited.

A —> B —> C, Cost of AC is 8, (Shortest Distance)

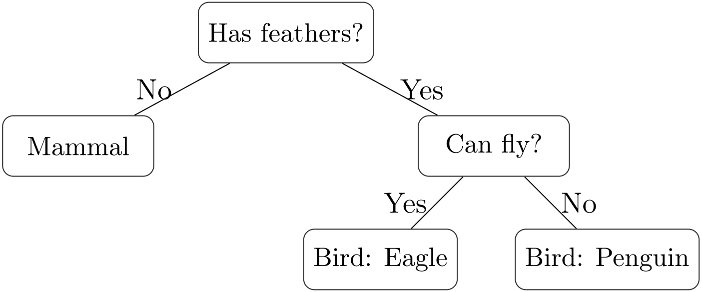
We have to find the minimum cost. So, we update the shortest distance (ABC, 8).

Also, mark the node C as visited.



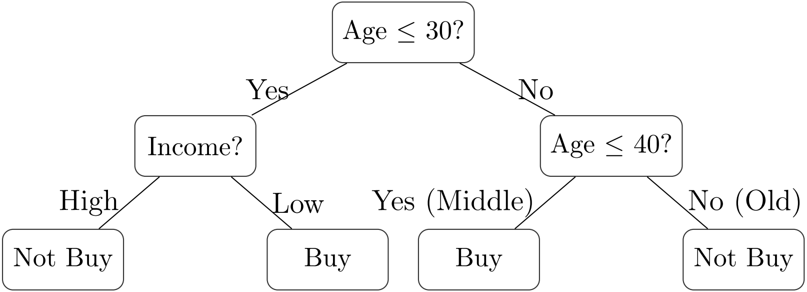
	B	C	D	E	F	G	H
A	3	8	7	4	6	7	5

Binary Tree in AI ? Decision Tree Example



- Each internal node asks a yes/no question.
- Leaves correspond to decisions or classifications.
- Binary trees like this form the basis of many AI algorithms

Binary Trees in AI ? Learning with Information Gain

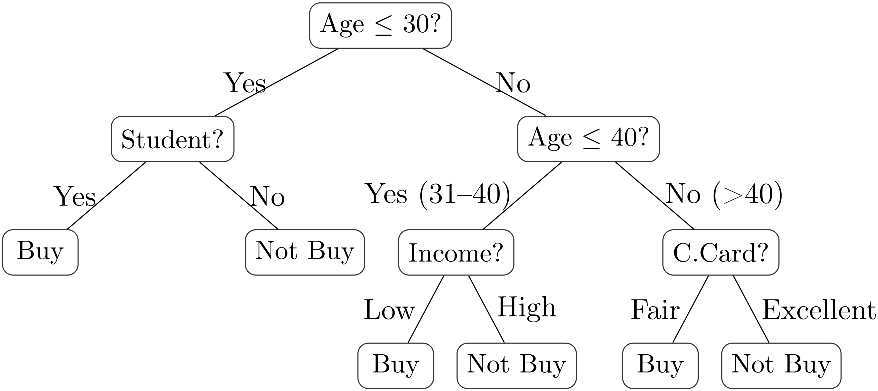


- Each node asks a binary question (Yes/No).
- Splits chosen by maximizing *information gain*:

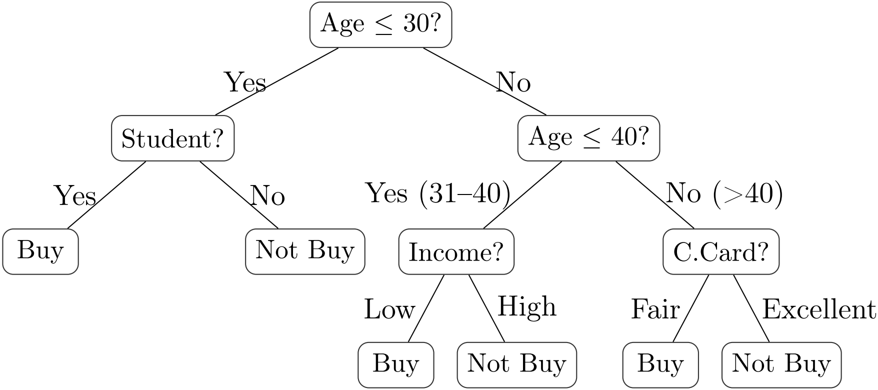
$$IG(S, A) = H(S) - \sum_{v \in \{Yes, No\}} \frac{|S_v|}{|S|} H(S_v).$$

$$H(S) = - \sum_{c \in \{Buy, NotBuy\}} p(c) \log_2 p(c)$$

Recursive Growth of Decision Trees



Recursive Growth of Decision Trees



- This dataset is an artificial pedagogical toy dataset where features were not chosen to model real-world economics perfectly ? just to demonstrate entropy/information gain.

Classic Dataset: "Buy Computer" (Mitchell, 1997)

Age	Income	Student	Credit Rating	Buys Computer
≤ 30	High	No	Fair	No
≤ 30	High	No	Excellent	No
≤ 30	Low	No	Fair	Yes
≤ 30	Low	Yes	Excellent	Yes
31–40	High	Yes	Fair	Yes
31–40	Low	Yes	Excellent	Yes
31–40	High	No	Fair	Yes
> 40	High	No	Excellent	No
> 40	Low	Yes	Fair	Yes
> 40	Low	No	Excellent	No

Summary: total examples $|S| = 10$, #Buy $= 6$, #NotBuy $= 4$.

Entropy (binary labels)

For binary label set S with class probabilities p_1, p_2,

$$H(S) = -\sum_i p_i \log_2 p_i.$$

Compute for dataset: $p(\text{Buy}) = \frac{6}{10} = 0.6, \quad p(\text{Not}) = \frac{4}{10} = 0.4.$

$$\begin{aligned} H(S) &= -\big(0.6 \log_2 0.6 + 0.4 \log_2 0.4\big) \\ &= -\big(0.6 \times (-0.736965594) \;+\; 0.4 \times (-1.321928094)\big) \\ &= 0.44217935649972373 \;+\; 0.5287712379549449 \\ &= 0.9709505944546686. \end{aligned}$$

Information Gain ? formula & Student split

Information Gain (IG)

For attribute A that partitions S into subsets S_v:

$$IG(S, A) = H(S) - \sum_v \frac{|S_v|}{|S|} H(S_v).$$

A = Student: values $= v \in \{Yes, No\}$.

- v = Yes: 4 examples, all 4 are **Buy** $\Rightarrow p(\text{Buy}) = 1, p(\text{Not}) = 0$ so $H(S_{\text{Yes}}) = 0$.
- v = No: 6 examples, $\#\text{Buy} = 2, \#\text{Not} = 4 \Rightarrow p(\text{Buy}) = \frac{2}{6} = 0.333333,\ p(\text{Not}) = \frac{4}{6} = 0.666666.$

$$H(S_{\text{No}}) = -\left(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6}\right) = -\big(0.33 \times (-1.58) + 0.67 \times (-0.58)\big)$$

$$= 0.5283208335737187 + 0.389974 \ = \ 0.9182958340544896.$$

Weighted entropy after splitting on Student:

$$\frac{4}{10} \cdot 0 \; + \; \frac{6}{10} \cdot 0.9182958340544896 \; = \; 0.5509775004326937.$$

Therefore

$$IG(S, \text{Student}) = H(S) - 0.5509775004326937 = 0.9709 - 0.5509 = 0.4199.$$

$\boxed{IG(\text{Student}) \approx 0.420}$ - The largest gain among attributes.

Split by Age values: ≤ 30 (4 examples), $31-40$ (3 examples), > 40 (3 examples).

Counts and entropies:

- Age ≤ 30: 4 examples, 2 Buy / 2 Not $\Rightarrow p = 0.5, 0.5$ so

$$H(S_1) = -\big(0.5 \log_2 0.5 + 0.5 \log_2 0.5\big) = 1.000000000.$$

- Age $31-40$: 3 examples, 3 Buy / 0 Not $\Rightarrow H(S_2) = 0$.
- Age > 40: 3 examples, 1 Buy / 2 Not

$$\Rightarrow H(S_3) = -\big(\tfrac{1}{3} \log_2 \tfrac{1}{3} + \tfrac{2}{3} \log_2 \tfrac{2}{3}\big) = 0.9182958340544896.$$

Weighted entropy:

$$\frac{4}{10} \cdot 1.0 + \frac{3}{10} \cdot 0 + \frac{3}{10} \cdot 0.92 = 0.4 + 0 + 0.28 = 0.68.$$

Information gain:

$$IG(S, \text{Age}) = 0.97 - 0.68 = 0.29.$$

$\boxed{IG(\text{Age}) \approx 0.2955}$.

Income (High / Low): each has 5 examples.

- High: 5 examples, 2 Buy / 3 Not
  $\Rightarrow H(\text{High}) = 0.9709505944546686.$
- Low: 5 examples, 4 Buy / 1 Not
  $\Rightarrow H(\text{Low}) = 0.7219280948873623.$
- Weighted entropy $=$
  $0.5 \cdot 0.97095 + 0.5 \cdot 0.721928 = 0.8464393446710154.$
- $IG(S, \text{Income}) = 0.9709505944546686 - 0.8464393446710154 = 0.12451124978365313.$

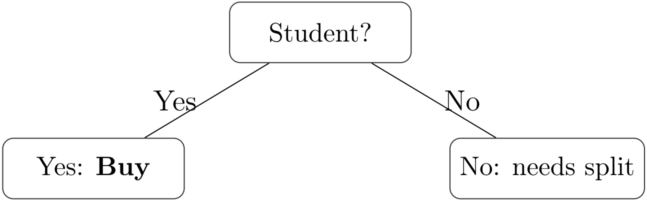
Information Gain ? Credit

Credit Rating (Fair / Excellent):

- Fair: 4 examples, 3 Buy / 1 Not
  $\Rightarrow H(\text{Fair}) = 0.8112781244591328.$
- Excellent: 6 examples, 3 Buy / 3 Not $\Rightarrow H(\text{Excellent}) = 1.0.$
- Weighted entropy
  $= 0.4 \cdot 0.8112781244591328 + 0.6 \cdot 1.0 = 0.9245112497836532.$
- $IG(S, \text{Credit}) = 0.9709505944546686 - 0.9245112497836532 = 0.0464393446710154.$

$\boxed{\text{Best root split: Student with } IG \approx 0.420}$

Tree after first split: Student



Explanation: Student=Yes subset is pure (4/4 Buy) so we make a leaf ‘Buy‘. For Student=No (6 examples, 2 Buy / 4 Not) we must split further ? see next slides.

Recursive split: analyze Student = No subset

Subset statistics

Student = No, this subset has 6 examples: #Buy = 2, #Not = 4.
Entropy:

$$H(S_{\text{No}}) = -\left(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6}\right) = 0.9182958340544896.$$

Candidate splits over: Age, Income, Credit. We compute IG inside this subset.

Age (within Student=No):

- Age ≤ 30: 3 examples (1 Buy, 2 Not) $\Rightarrow H = 0.9182958340544896$.
- Age $31-40$: 1 example (1 Buy) $\Rightarrow H = 0$.
- Age > 40: 2 examples (0 Buy, 2 Not) $\Rightarrow H = 0$.
- Weighted entropy $= \frac{3}{6} \cdot 0.9183 + \frac{1}{6} \cdot 0 + \frac{2}{6} \cdot 0 = 0.4591479170272448$.
- $IG_{\text{No}}(\text{Age}) = 0.9182958340544896 - 0.4591479170272448 = 0.4591479170272448$.

Recursive split: analyze Student = No subset

Income (within Student=No):

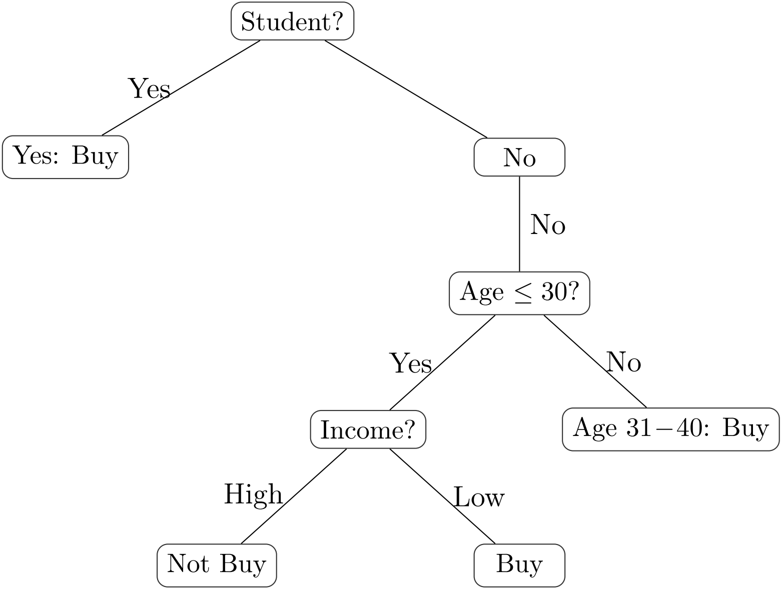
- High: 4 examples (1 Buy, 3 Not) $\Rightarrow H = 0.8112781244591328$.
- Low: 2 examples (1 Buy, 1 Not) $\Rightarrow H = 1.0$.
- Weighted entropy $= \frac{4}{6} \cdot 0.8112781244591328 + \frac{2}{6} \cdot 1.0 = 0.8741854163060885$.
- $IG_{\text{No}}(\text{Income}) = 0.9182958340544896 - 0.8741854163060885 = 0.04411041774840107 6$.

Credit (within Student=No):

- Fair: 3 examples (2 Buy, 1 Not) $\Rightarrow H = 0.9182958340544896$.
- Excellent: 3 examples (0 Buy, 3 Not) $\Rightarrow H = 0$.
- Weighted entropy $= \frac{3}{6} \cdot 0.9182958340544896 + \frac{3}{6} \cdot 0 = 0.4591479170272448$.
- $IG_{\text{No}}(\text{Credit}) = 0.9182958340544896 - 0.4591479170272448 = 0.4591479170272448$.

Best splits on Student=No: Age and Credit tie with $IG \approx 0.45915$.

Continue recursion: Student=No, choose Age

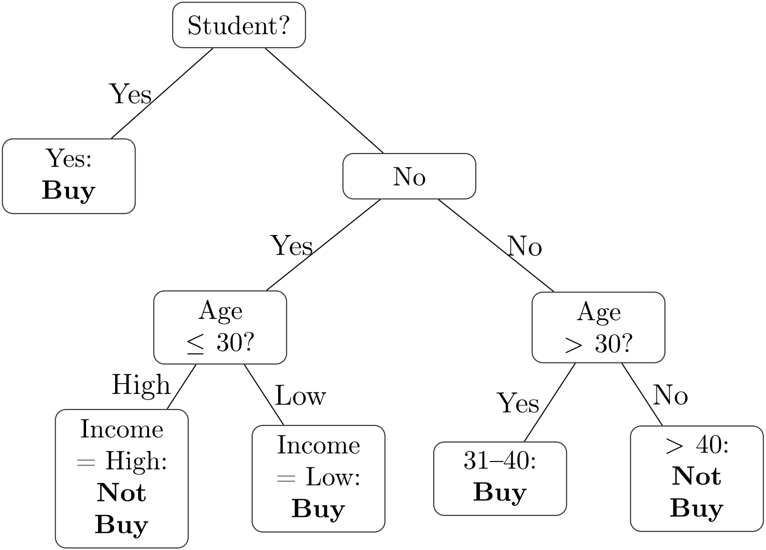


Continue recursion: Student=No, choose Age

Notes on final small splits:

- For Student=No and Age ≤ 30 (3 examples): splitting by **Income** yields
- (i) Income $=$ High $\Rightarrow$ 2 examples (both NotBuy) $\Rightarrow$ leaf **Not Buy**.
- (ii) Income $=$ Low $\Rightarrow$ 1 example (Buy) $\Rightarrow$ leaf **Buy**.
- Age $31-40$ was pure (Buy) in this subset; Age > 40 gave NotBuy.

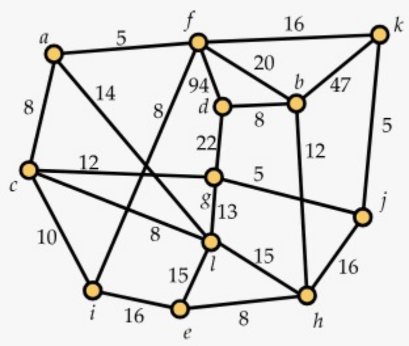
Final decision tree (summary)

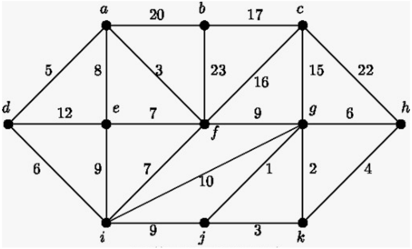


Final decision tree (summary)

Summary:

- Numeric entropy & IG calculations explain & justify every split.
- Student had the highest IG at the root (0.420 bits) and became a pure leaf for Student=Yes.
- Recursion and local IG checks (Age / Credit) led to the rest of the tree; local splits produced pure or near-pure leaves.


a
5
f
16
k
8
14
8
94
20
b
47
d
8
5
22
12
12
c
5
g
j
13
10
8
15
l
16
15
i
16
e
8
h


a
20
b
17
c
5
8
3
23
16
15
22
d
12
e
7
f
9
g
6
h
6
9
7
10
1
2
4
i
9
j
3
k