Ch 11.2 Applications of Trees

- Problems can be studied using trees
- How should items in a list be stored so that an item can be easily located?
- What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?
- How should a set of characters be efficiently coded by bit strings?
- What sequence of moves does a player make in a game?

Applications of Trees

- Binary search trees
  – A simple data structure for sorted lists
- Decision trees
  – Minimum comparisons in sorting algorithms
- Prefix codes
  - Huffman coding
- Game trees
  - Determine moves a player makes

Applications of Trees

- Binary Search Trees

A binary search tree:

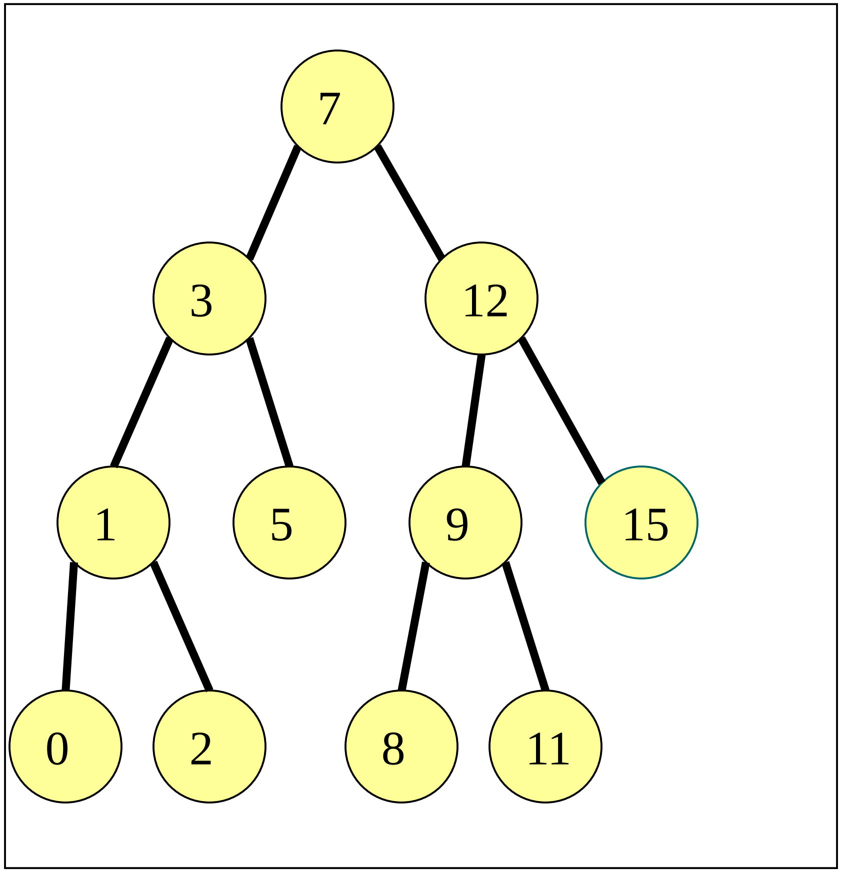
is a binary tree

if a node has value N, all values in its left sub-tree are less than or equal to N, and all values in its right sub-tree are greater than N.

Applications of Trees

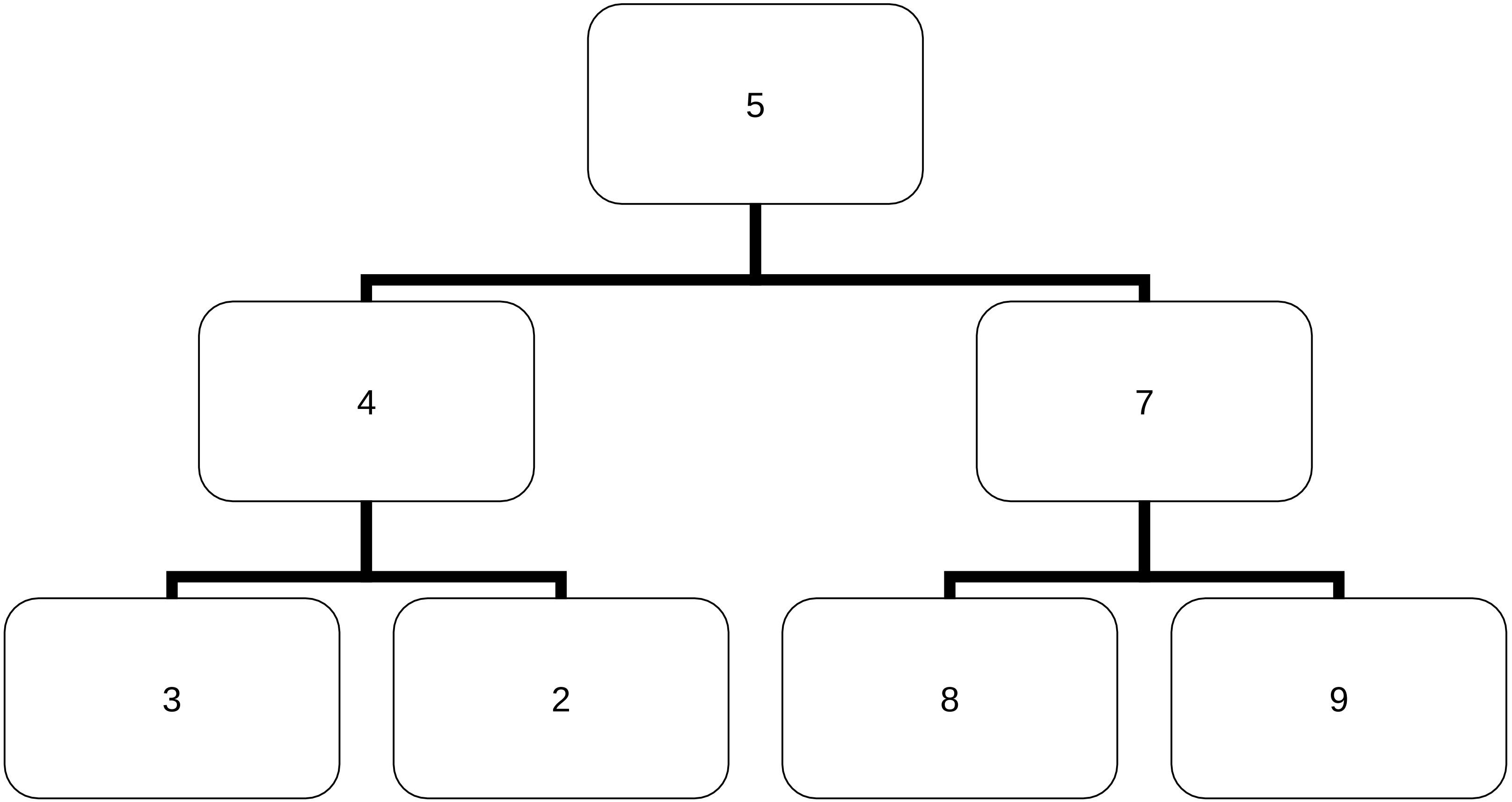
Binary Search Tree Format

- Items are stored at individual tree nodes.
- We arrange for the tree to always obey this invariant:
- For every item *x*,
    - Every node in *x*'s left subtree is less than *x*.
    - Every node in *x*'s right subtree is greater than *x*.



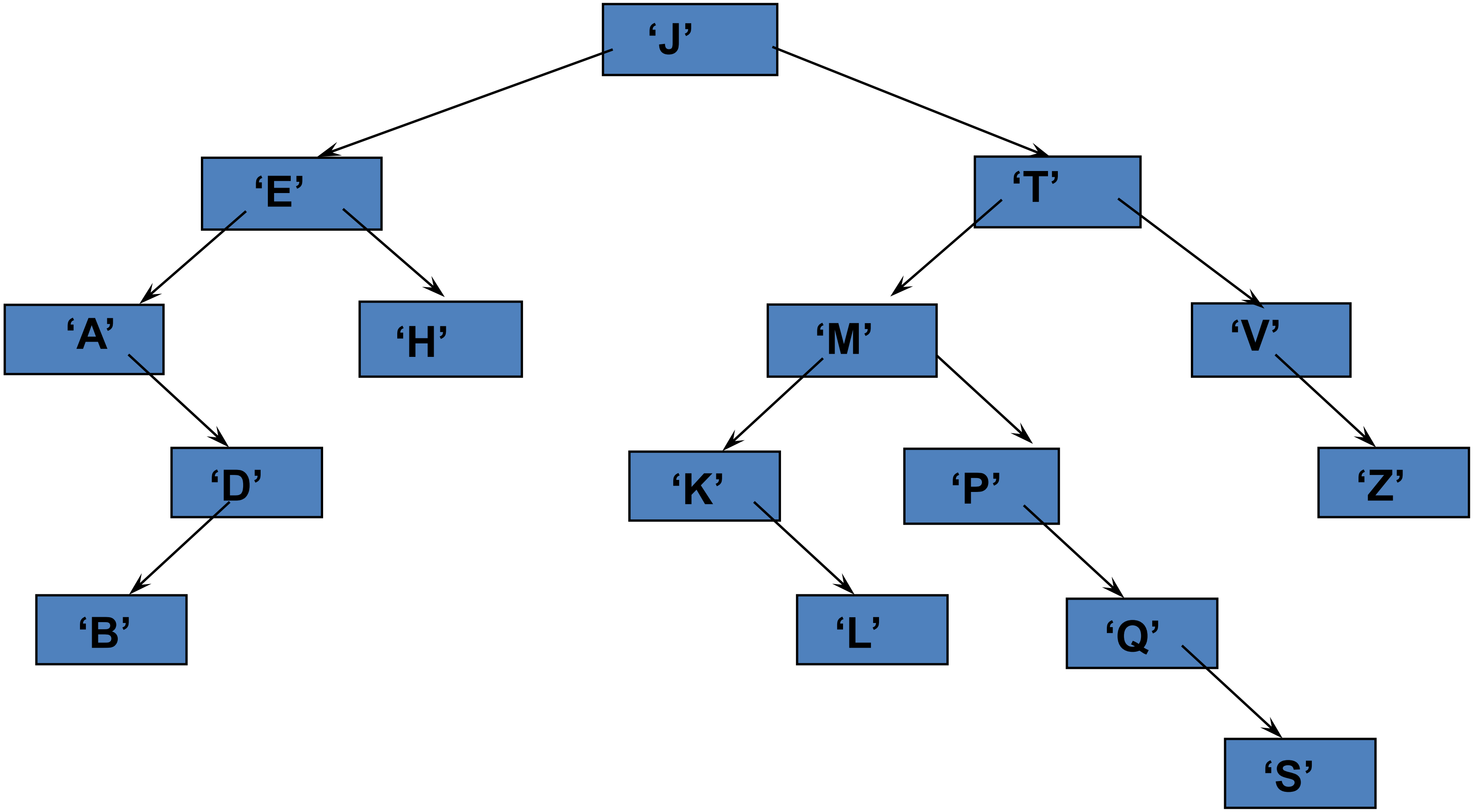
Applications of Trees

Is this a binary search tree?

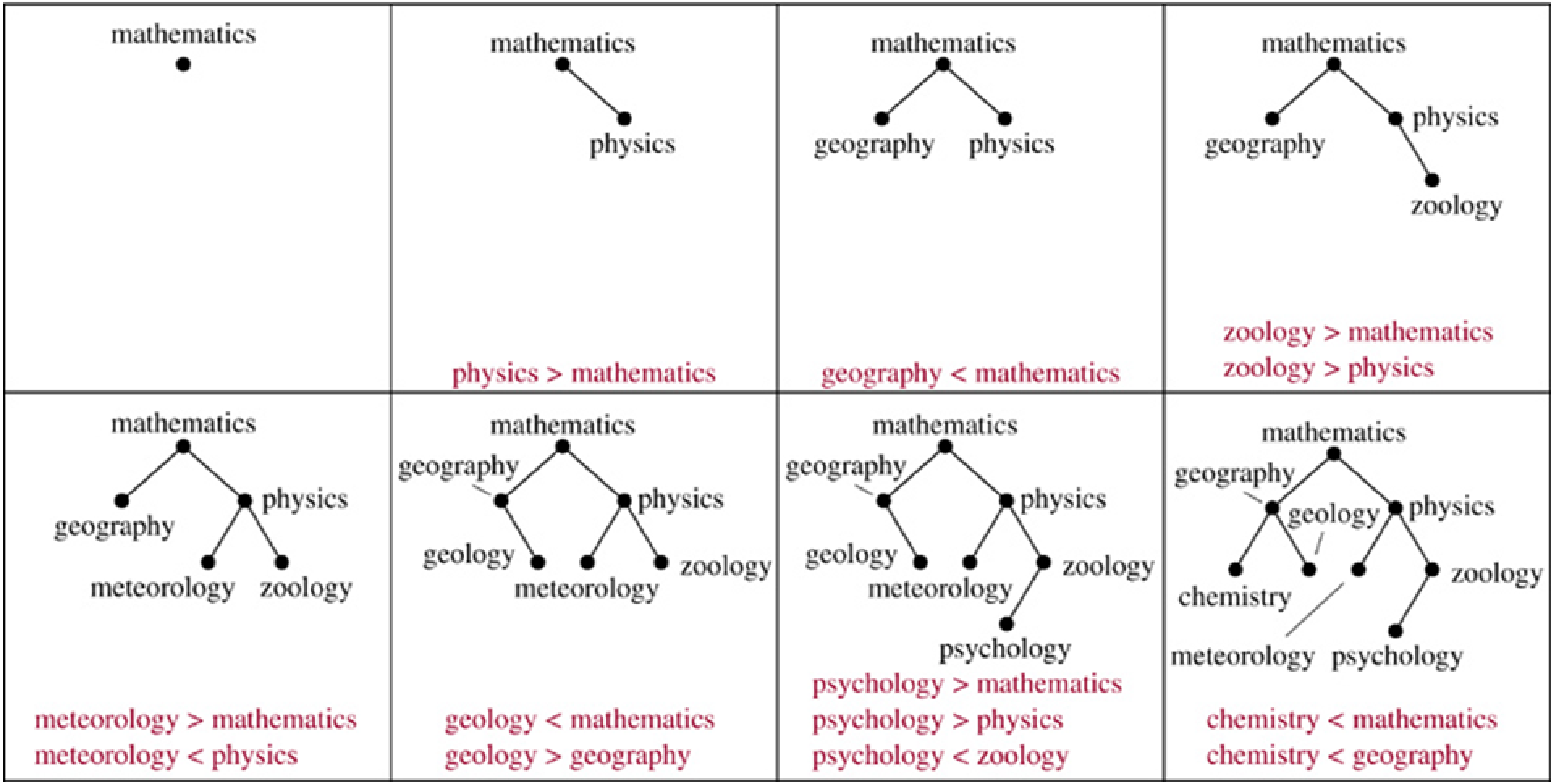


Applications of Trees

Is this a binary search tree?



Binary Search Trees



Applications of Trees

Searching a binary search tree

```
search(t, s) {
If(s == label(t))
   return t;
If(t is leaf) return null
If(s < label(t))
search(t's left tree, s)
else
search(t's right tree, s)}
```

Applications of Trees

Decision Trees

- A decision tree represents a *decision-making process*
- each internal vertex corresponds to a "decision point"
- a sub-tree at these vertices corresponds to each possible outcome of the decision
- In the extended decision trees used in *decision analysis -* also include nodes that represent random events and their outcomes

Applications of Trees

Coin-Weighing Problem

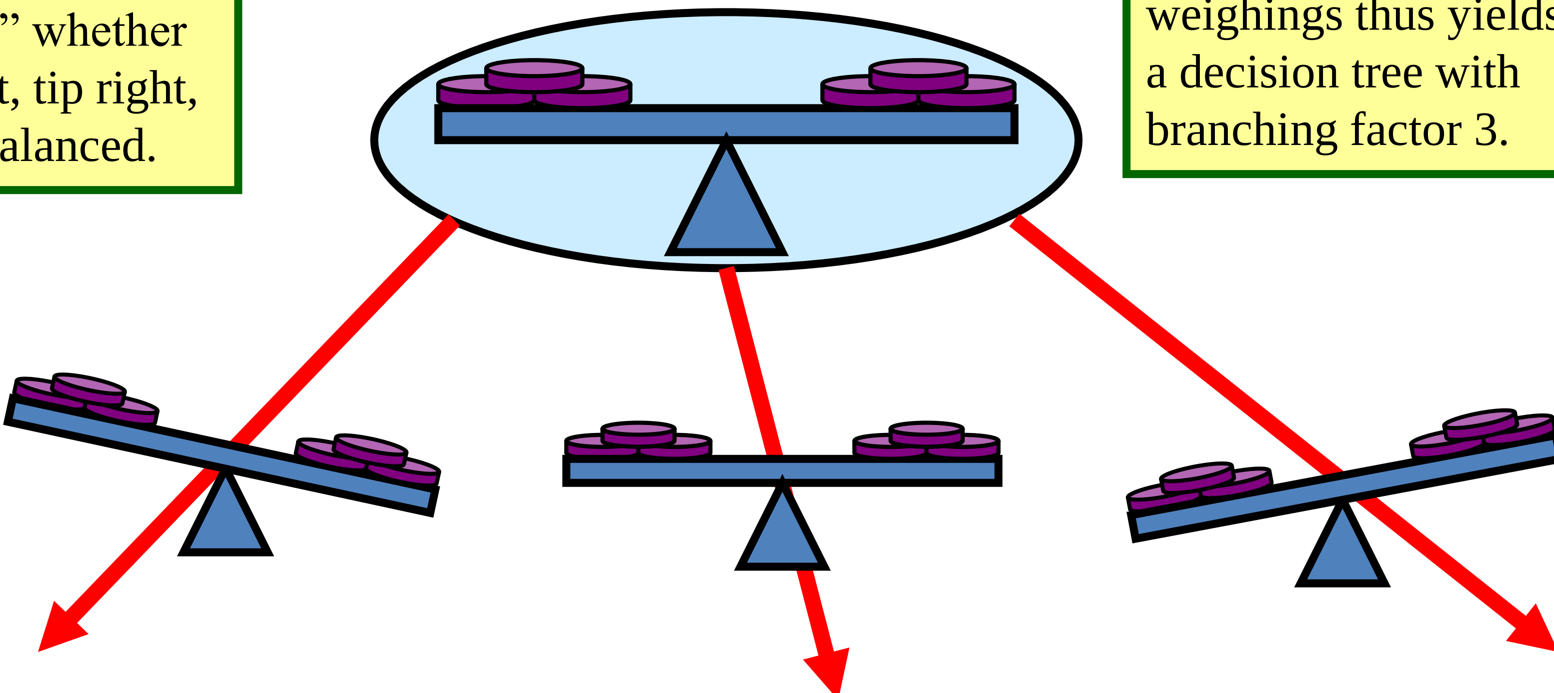
- Imagine you have 8 coins
- One of which is a lighter counterfeit
- A free-beam balance

- How many weighings are needed to guarantee that the counterfeit coin will be found?

Applications of Trees

- In each situation, we pick two disjoint and equal-size subsets of coins to put on the scale.

The balance then "decides" whether to tip left, tip right, or stay balanced.



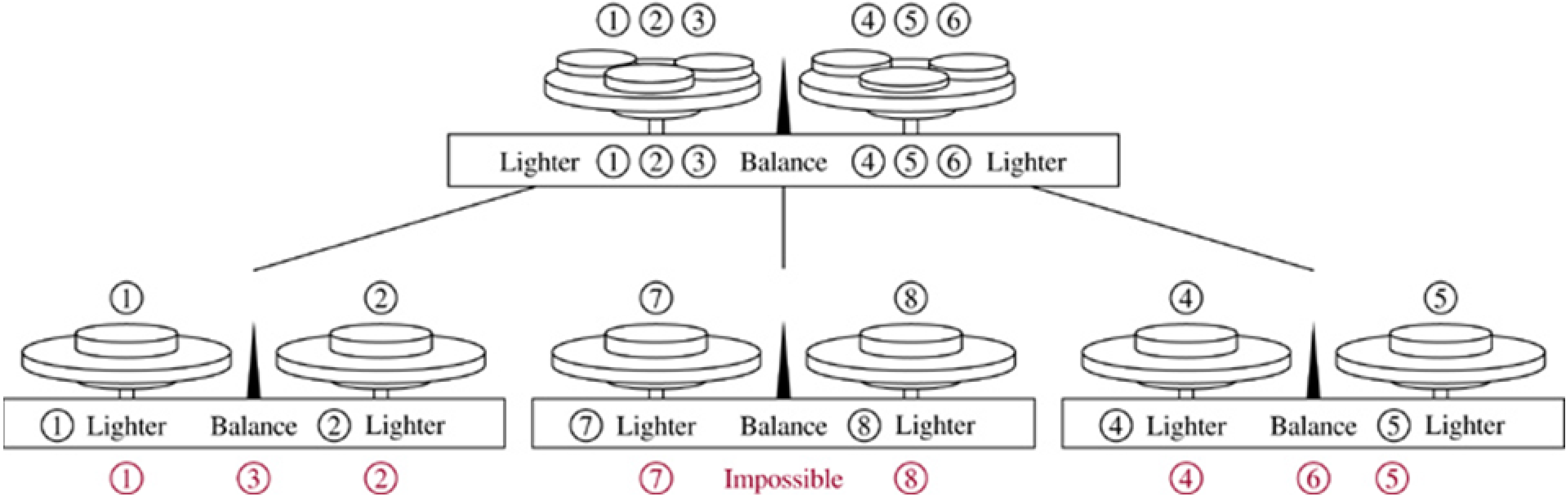
A given sequence of weighings thus yields a decision tree with branching factor 3.

Applications of Trees

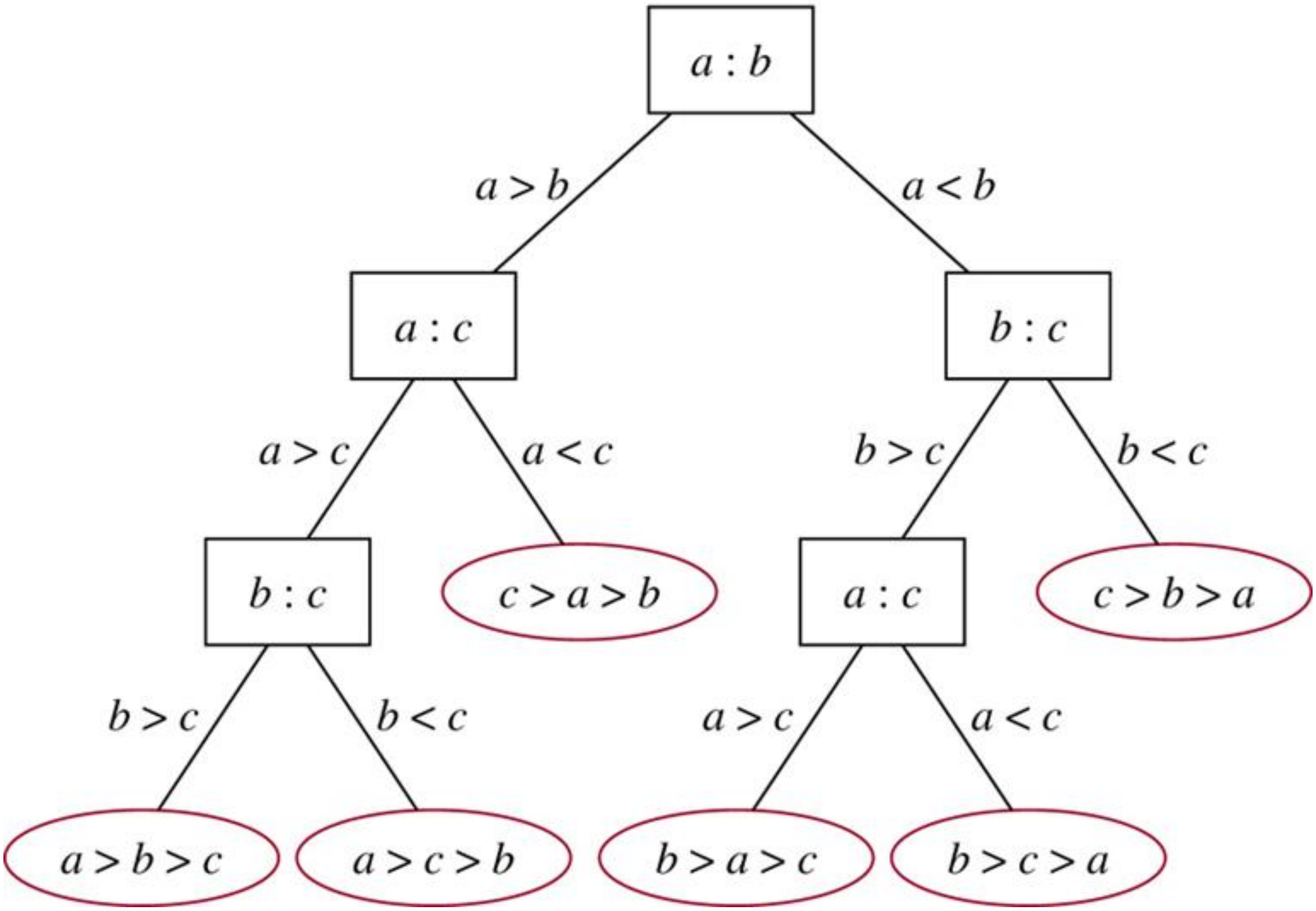
- Applying the Tree Height Theorem

- The decision tree must have at least 8 leaf nodes, since there are 8 possible outcomes.
  - In terms of which coin is the counterfeit one.
- Recall the tree-height theorem, $h \geq \lceil \log_m \ell \rceil$
  - Thus the decision tree must have height $h \geq \lceil \log_3 8 \rceil = \lceil 1.893\ldots \rceil = 2$
- Let's see if we solve the problem with *only* 2 weightings…

Decision Trees



Decision Trees



Applications of Trees

- Data Compression

- Suppose we have 3GB character data file that we wish to include in an email.
- Suppose file only contains 26 letters {a,…,z}.
- Suppose each letter α in {a,…,z} occurs with frequency f_α.
- Suppose we encode each letter by a binary code
- If we use a fixed length code, we need 5 bits for each character
- The resulting message length is $5\left(f_a + f_b + \cdots f_z\right)$
- **Can we do better?**

Applications of Trees

- Suppose the file only has 6 letters {a,b,c,d,e,f} with frequencies

a	b	c	d	e	f	
.45	.13	.12	.16	.09	.05	
000	001	010	011	100	101	Fixed length
0	101	100	111	1101	1100	Variable length

- Fixed length = 3 . (0.45 + 0.13 + 0.12 + 0.16 + 0.09 + 0.05) = 3G
- Variable length =

$$(.45\bullet 1+.13\bullet 3+.12\bullet 3+.16\bullet 3+.09\bullet 4+.05\bullet 4)=2.24G$$

Applications of Trees

- Is it possible to find a coding scheme of these letters such that, when data are coded, fewer bits are used?
- Use **Prefix Codes**
- Save memory
- Reduce transmittal time

Applications of Trees

Prefix Codes

- Bit string for a letter never occurs as the first part of the bit string for another letter
- Cannot encode *t* as 01 and *x* as 01101

  \- since 01 is a prefix of 01101

- Can encode e = 0, a = 10, t = 11

  \- word can be recovered, string 10110 - ate

Applications of Trees

- Binary tree representation for Prefix Codes

  characters are the labels of the leaves in the tree
- Label edges

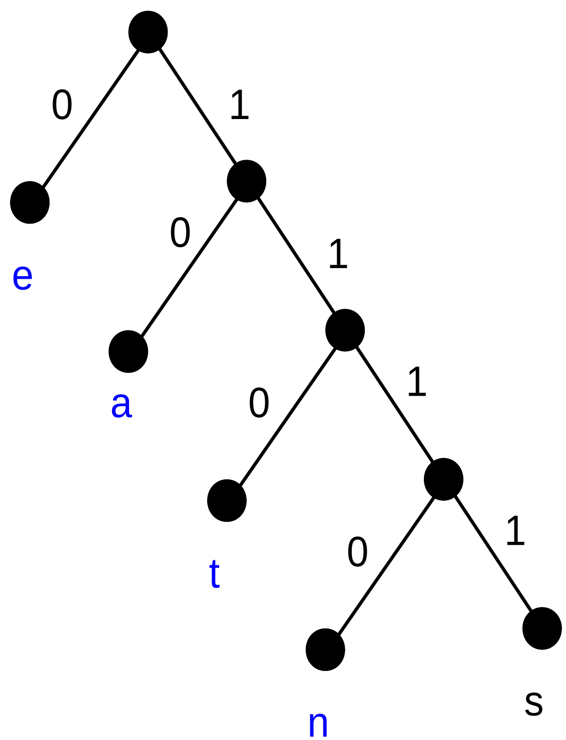
  edge leading to a left child is assigned a 0

  edge leading to a right child is assigned a 1
- Encoding a character

  sequence of labels of the edges in the unique path from the root to the leaf that has this character as its label

Applications of Trees

- Binary tree with a Prefix code
- e = 0, a = 10, t = 110, n = 1110, s = 1111



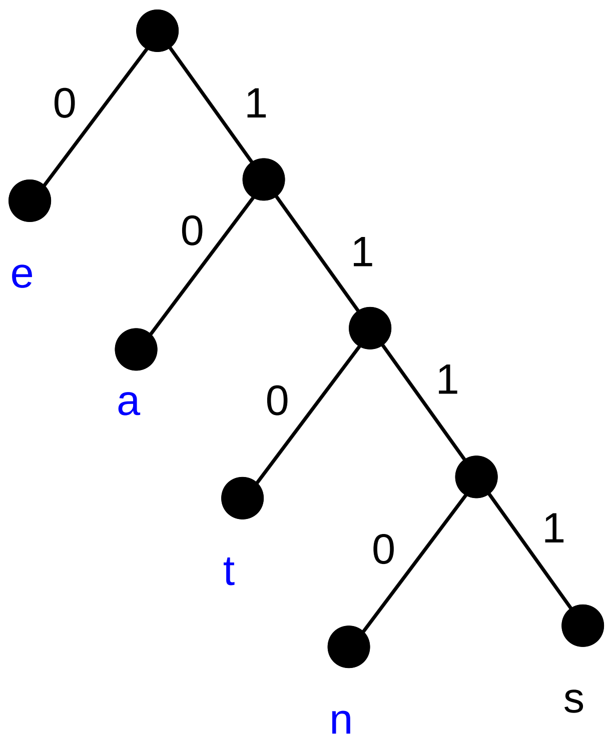
Applications of Trees

- Decoding prefix codes

- Tree representing a code can be used to decode a bit string
- Follow the tree until it reaches a leaf, and then repeat
- A message can be decoded uniquely

Applications of Trees

Prefix codes allow easy decoding



Decode:

11111011100

s 1011100

sa 11100

san 0

sane

Applications of Trees

Huffman Coding

- Fundamental algorithm in data compression
- Used extensively to compress bit string representing text
- Compress image and audio files
- Input - the frequencies of symbols in a string
- Output - A prefix code that encodes a string using the fewest possible bits, among all possible binary prefix codes for these symbols

Applications of Trees

David Huffman's idea

- Build the tree bottom-up in a greedy fashion
- Each tree has a weight in its root and symbols are labels of the leaves
- Start - forest of one vertex trees representing the input symbols
- Recursively merge two trees whose sum of weights is minimal until we have only one tree

Applications of Trees

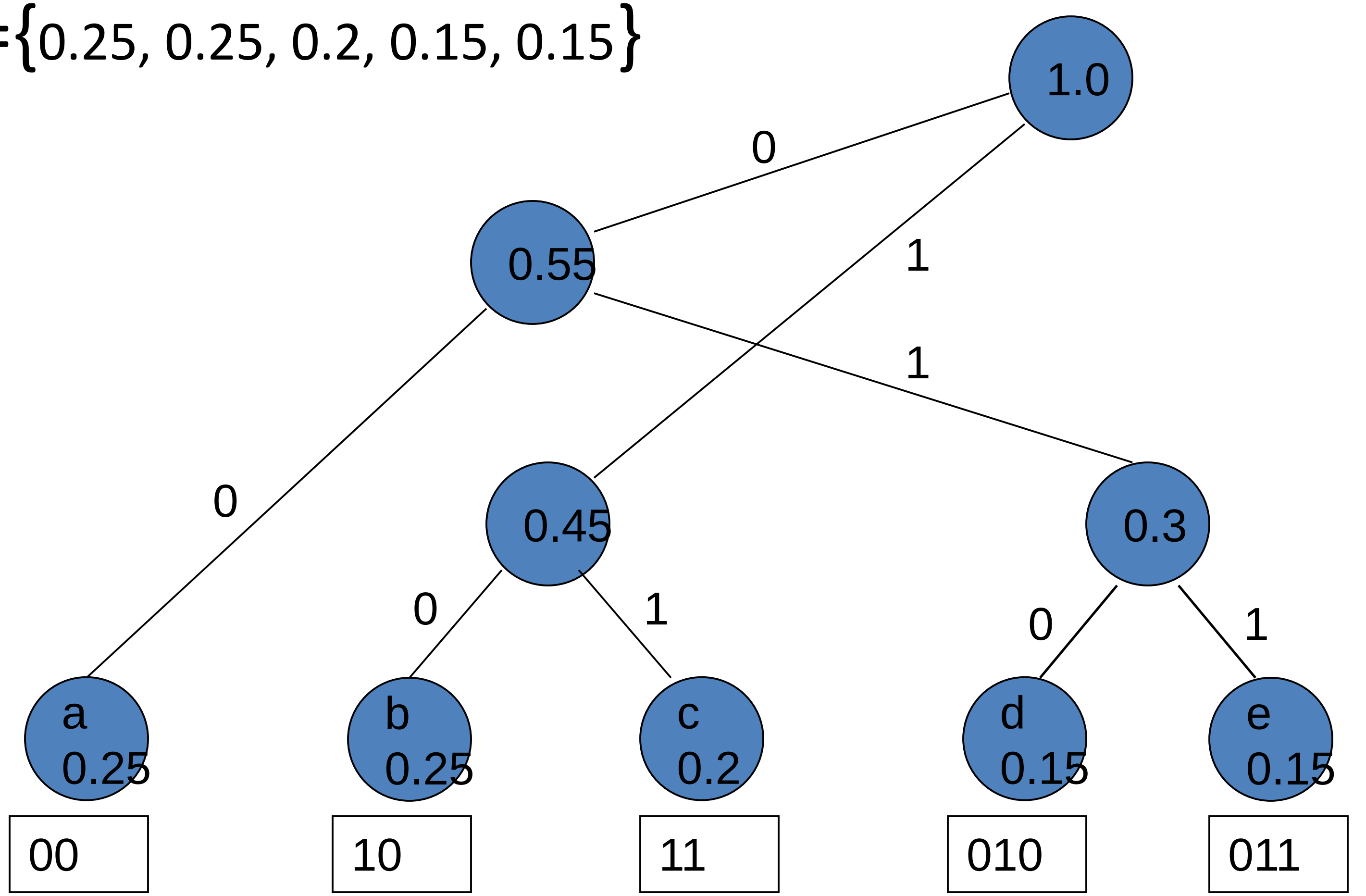
Huffman Coding Algorithm

1. Take the two least probable symbols in the alphabet

2. Combine these two symbols into a single symbol, and repeat

Applications of Trees

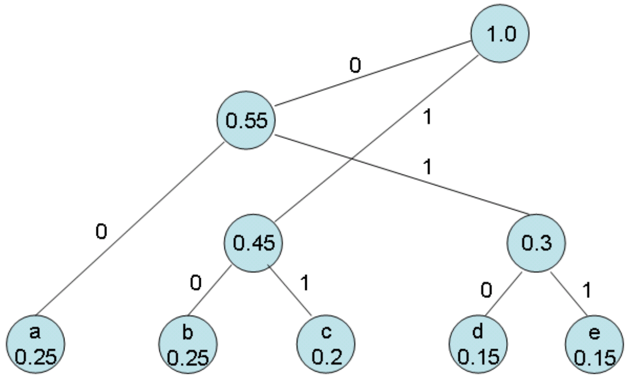
- $A_x = \{ a , b , c , d , e \}$
- $P_x = \{0.25, 0.25, 0.2, 0.15, 0.15\}$



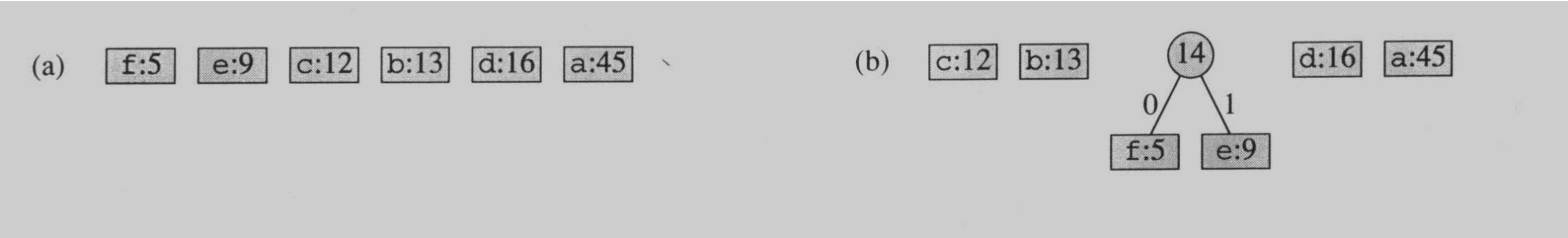
Applications of Trees

- A_x={ a , b , c , d , e }
- P_x={0.25, 0.25, 0.2, 0.15, 0.15}

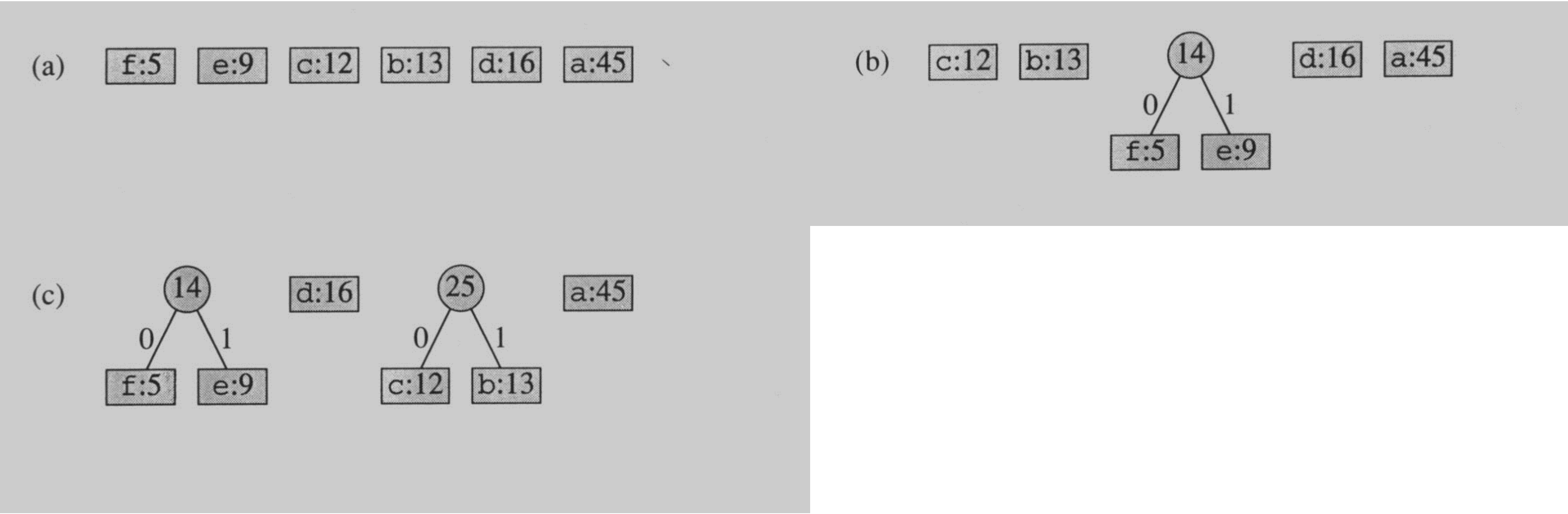
a_i	p_i	$h(p_i)$	l_i	$c(a_i)$
a	0.25	2.0	2	00
b	0.25	2.0	2	10
c	0.2	2.3	2	11
d	0.15	2.7	3	010
e	0.15	2.7	3	011



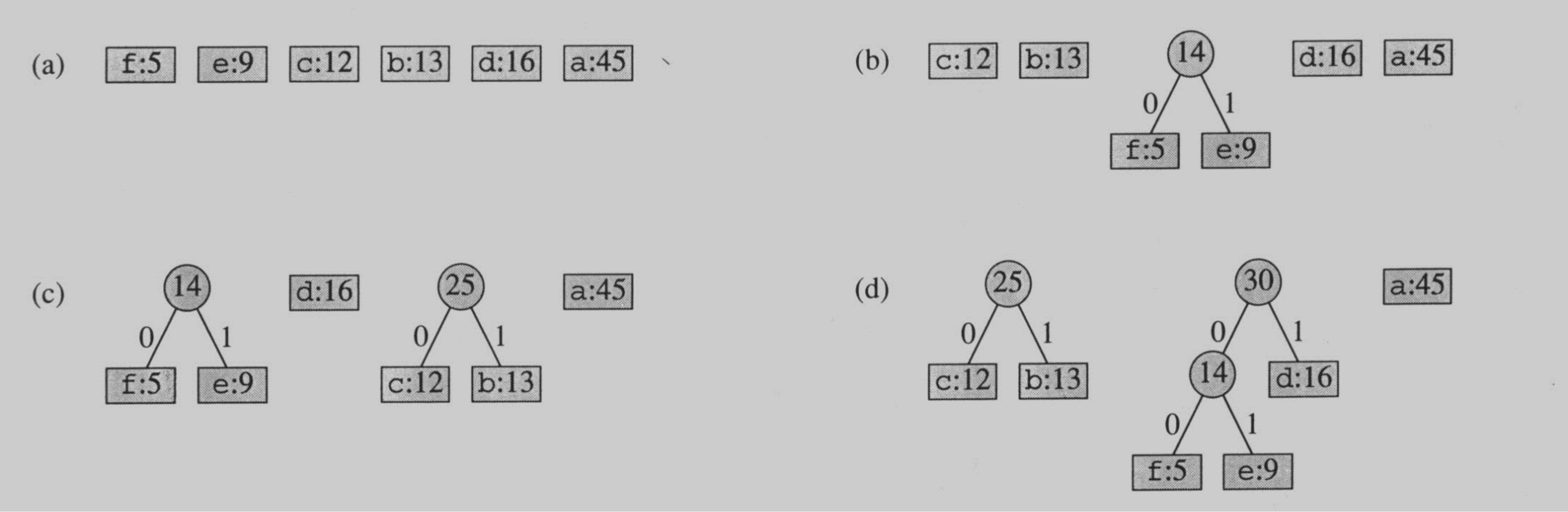
Building the Encoding Tree



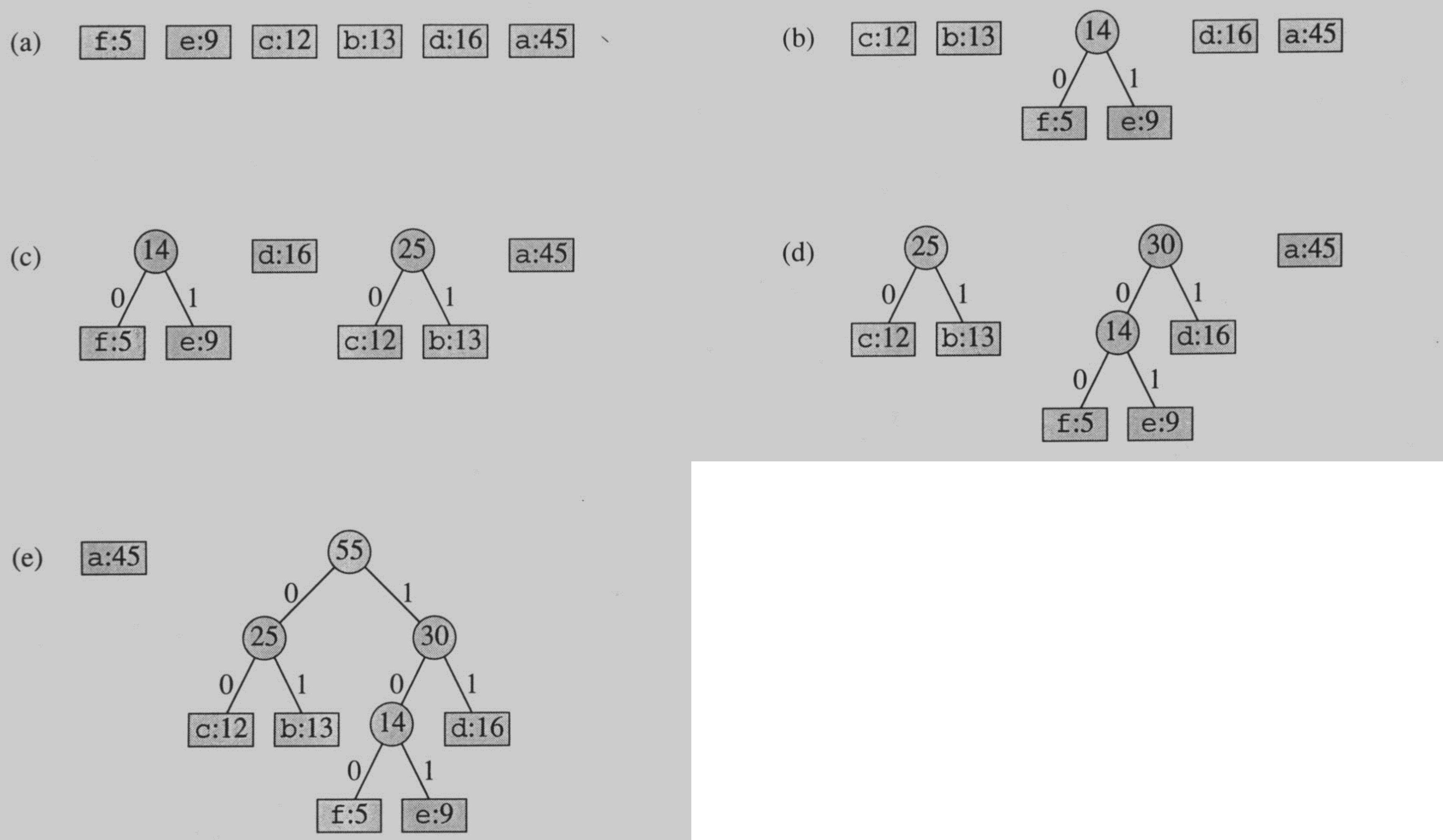
Building the Encoding Tree



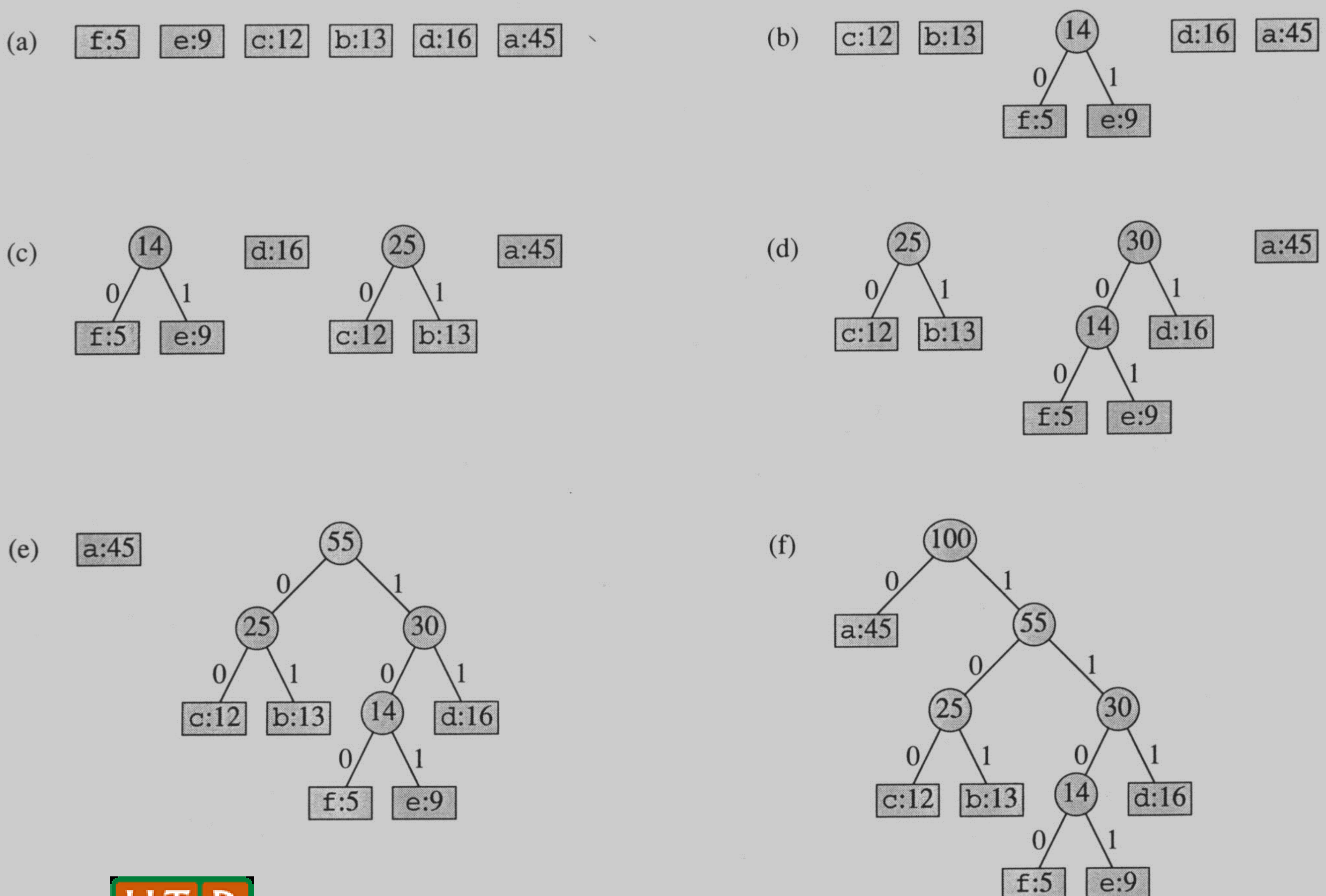
Building the Encoding Tree



Building the Encoding Tree



Building the Encoding Tree



Applications of Trees

- Game Trees – Analyze certain types of games
- Tic-tac-toe, checkers, nim, chess
- Vertices – positions that a game can be in as it progresses
- Edges – legal moves between these positions
- Root – Starting position

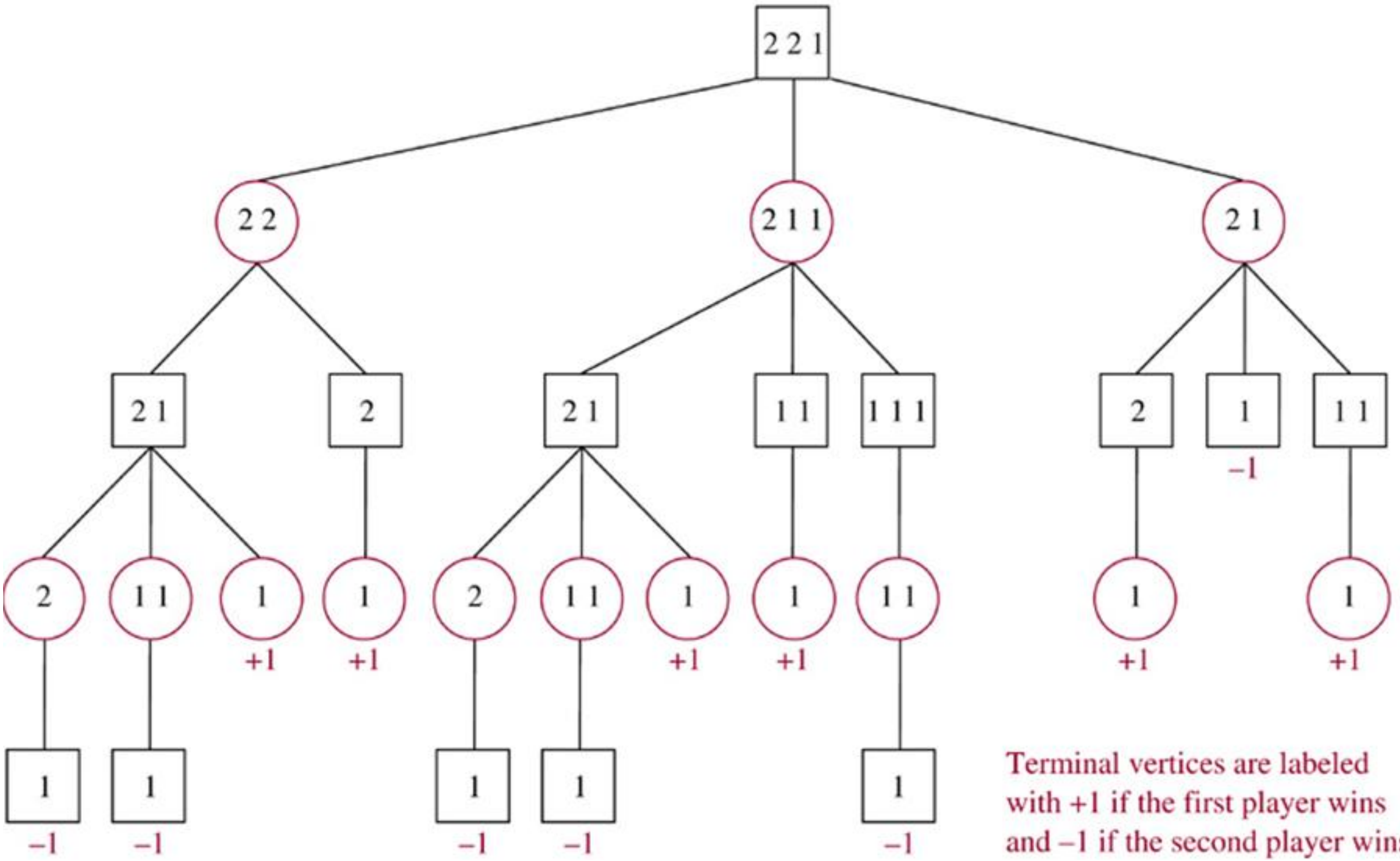
Applications of Trees

- Game of Nim
- Piles of Stones
- Legal move

  - removing one or more stones from one or more piles

  - without removing all the stones left
- A player without a legal move loses

Game Trees

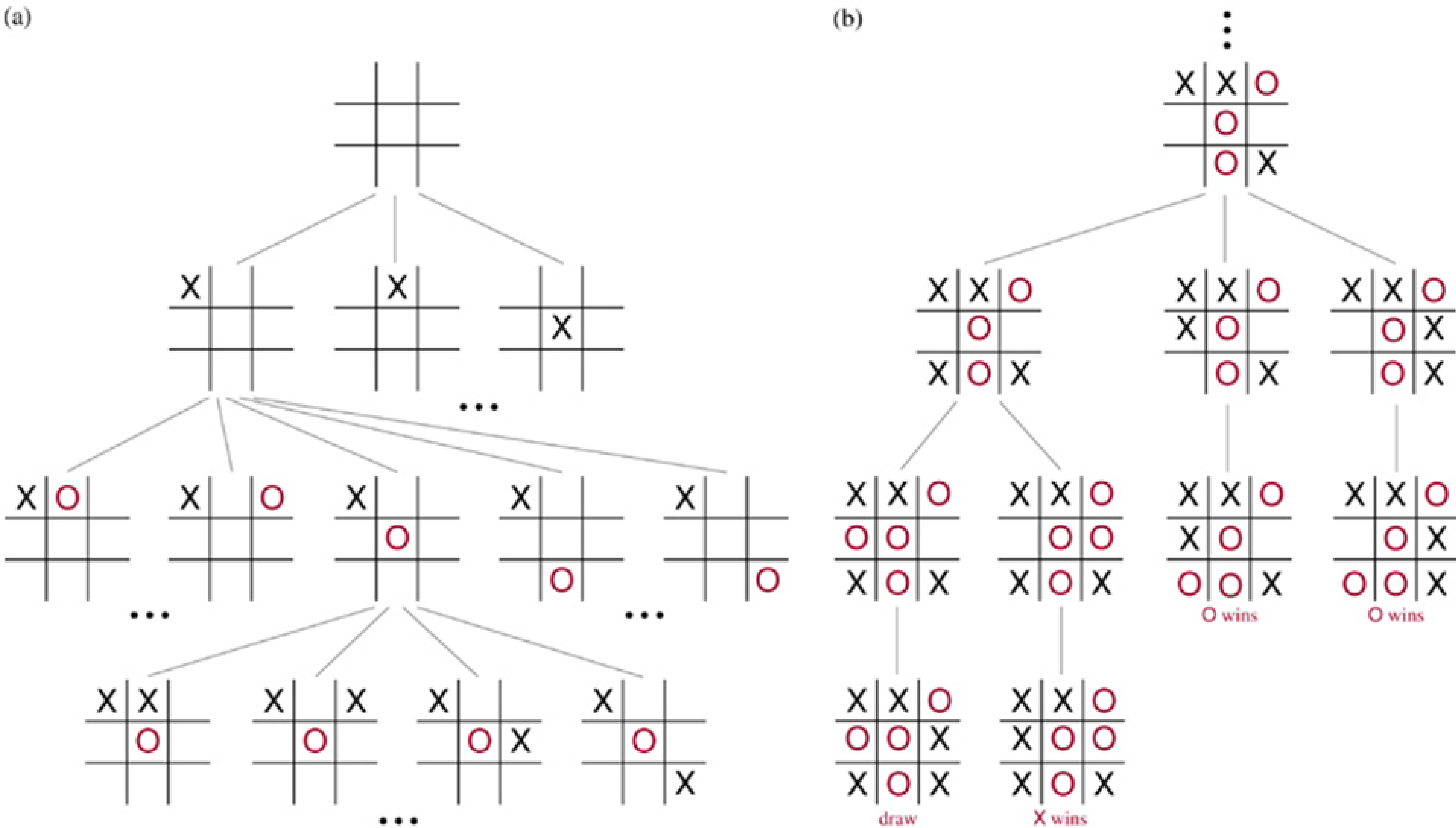


2 2 1
2 2
2 1 1
2 1
2 1
2
2 1
1 1
1 1 1
2
1
1 1
−1
2
1 1
1
1
2
1 1
1
1
1 1
1
1
+1
+1
+1
+1
+1
+1
1
1
1
1
1
−1
−1
−1
−1
−1
Terminal vertices are labeled
with +1 if the first player wins
and −1 if the second player wins

Applications of Trees

- Tic Tac Toe Game
- Three possible initial moves
- Subtree of the game
- Terminal positions
- Win, draw

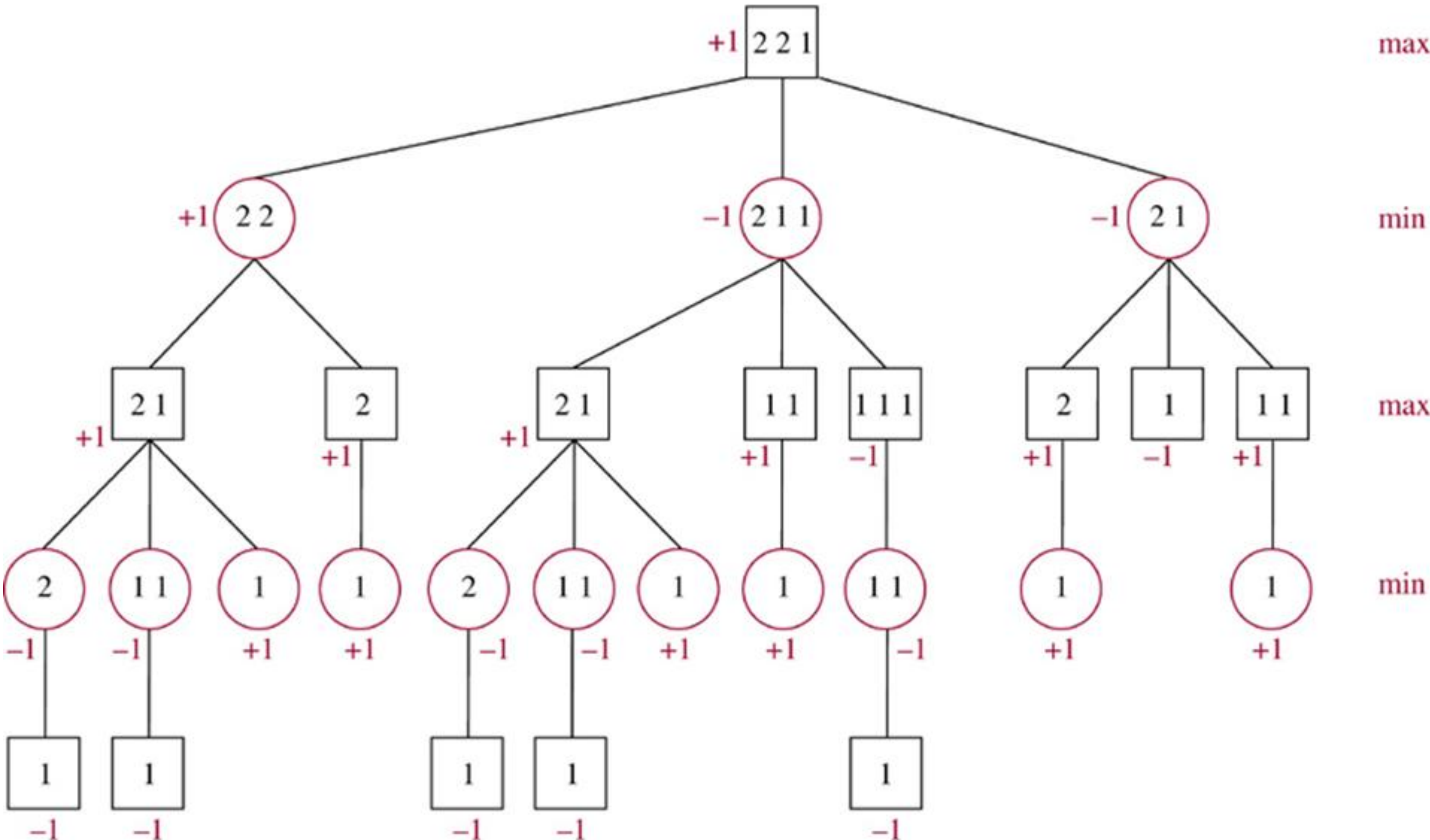
Game Trees



Applications of Trees

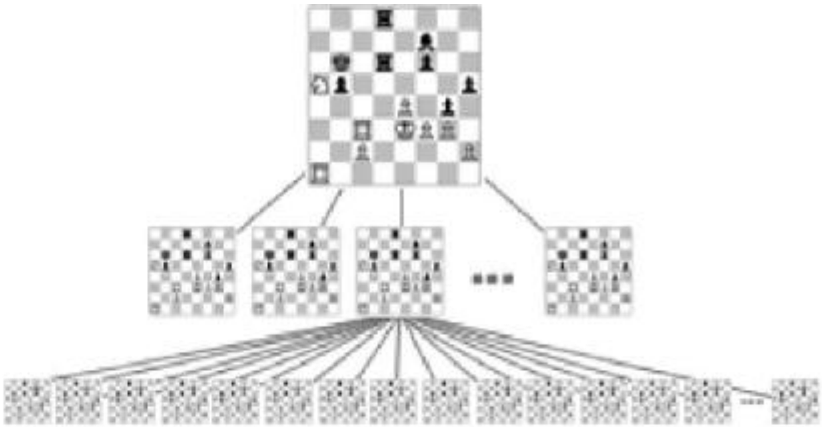
- Game of Nim
- Inductive Hypothesis
- Minmax strategy
- First player – position with largest value
- Second player – position with least value

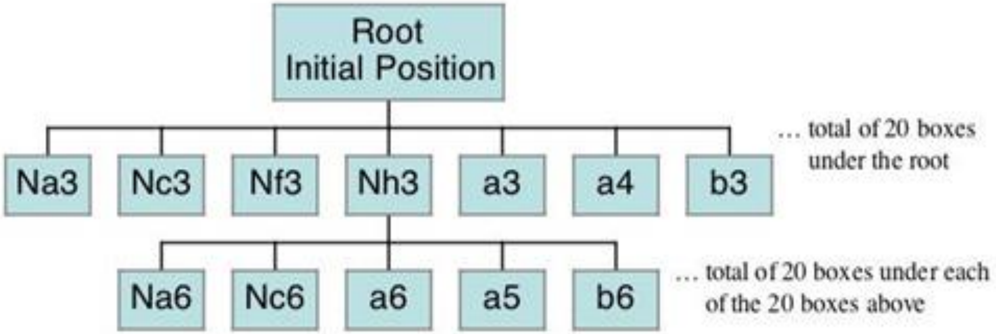
Game Trees



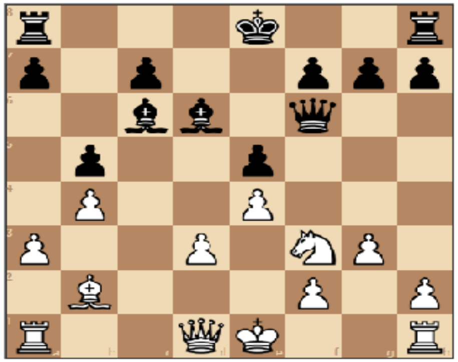
Game Trees

- Game of Chess

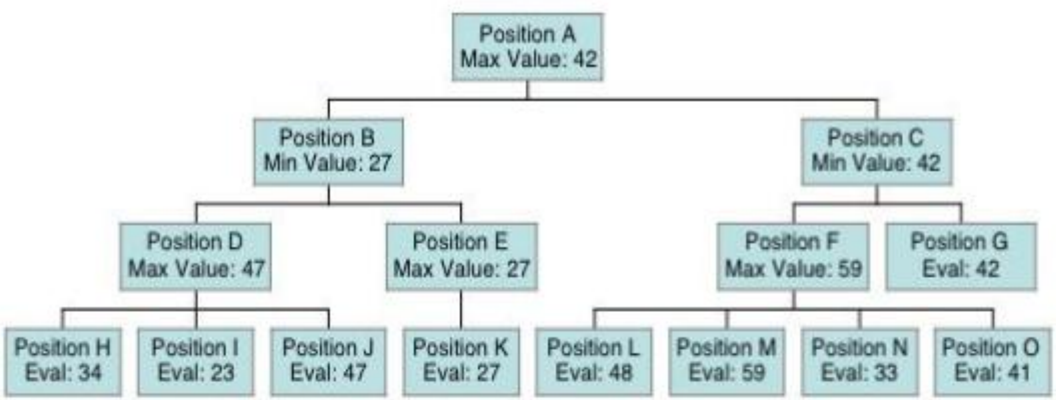




Game Trees

$f\Bigg($  $\Bigg) = 5$

- Game of Chess – Minmax Strategy



Opponent’s move

Your move

Opponent’s move

Your move