Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 4/10: Search and Reasoning

DR VARUN OJHA

Department of Computer Science

University of Reading

Learning Objectives

On completion of this week, you will be able to

- Understand One-Player and Two-Player Game and Their solutions using Search Techniques.
- Learning two different categories of search techniques of AI
  - Systematic Search
  - Non Systematic Search
- Learn techniques to improve search speed
  - Alpha-beta pruning
  - A* Search
- Apply methods to solve search problems
- Learning methods of Reasoning

Content of this Lecture

Introduction

- Part – I : Search Problem Formulation
- Part – II : Systematic Search
- Part – III : Non-Systematic Search
- Part – IV : Reasoning
- Part – V : Practical Exercise

Quiz

Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 4/10: Search and Reasoning

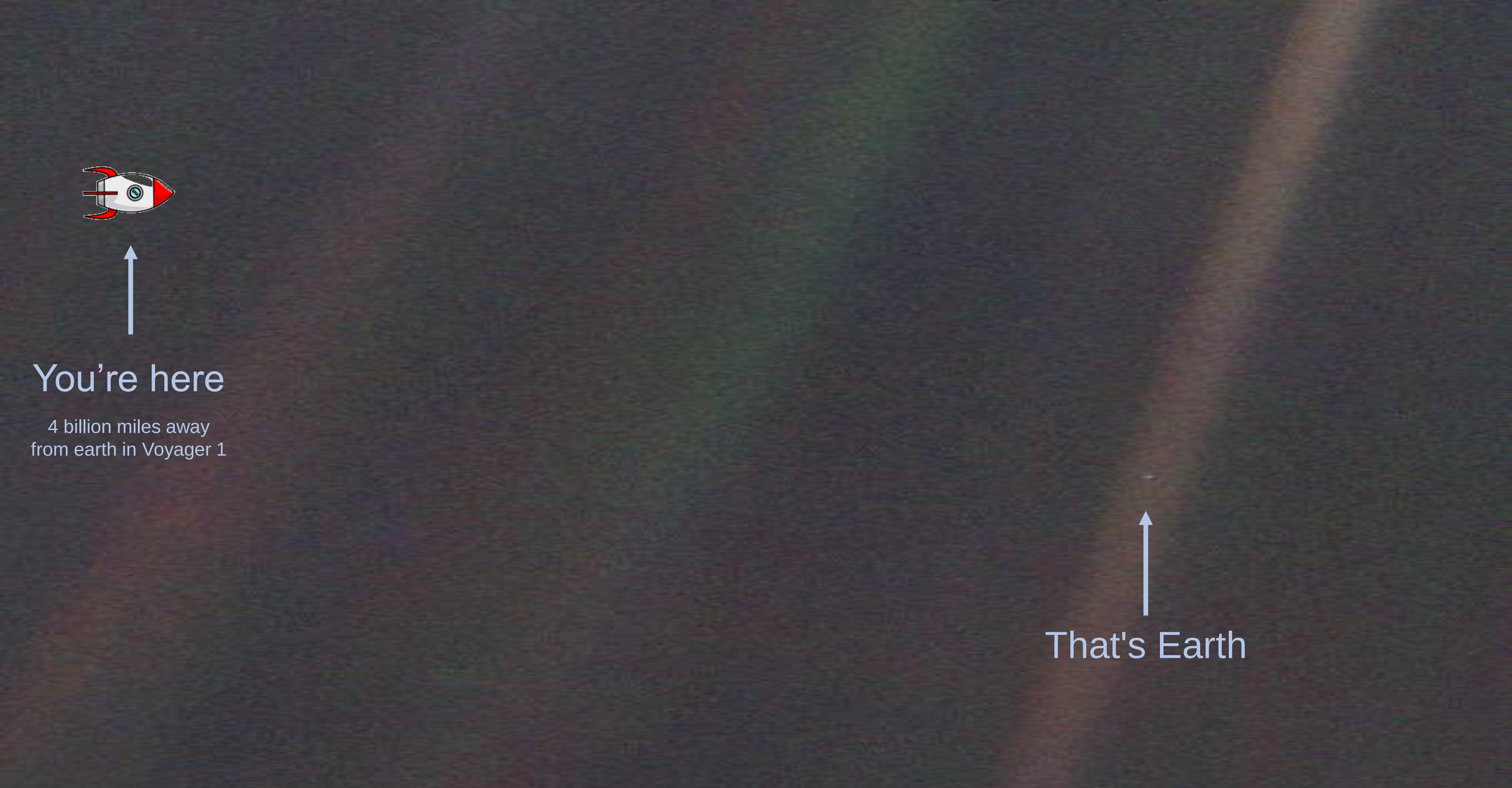
Part 1
Search

DR VARUN OJHA

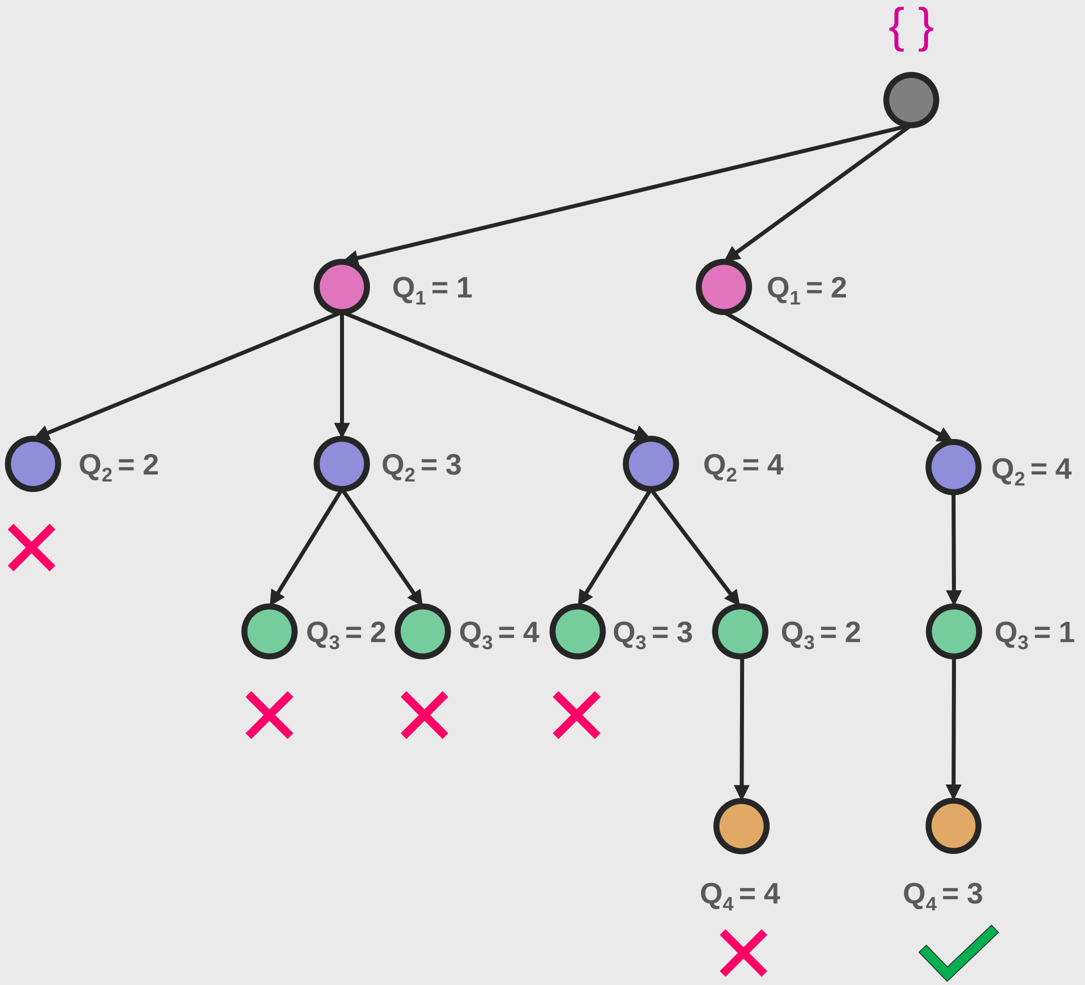
Department of Computer Science

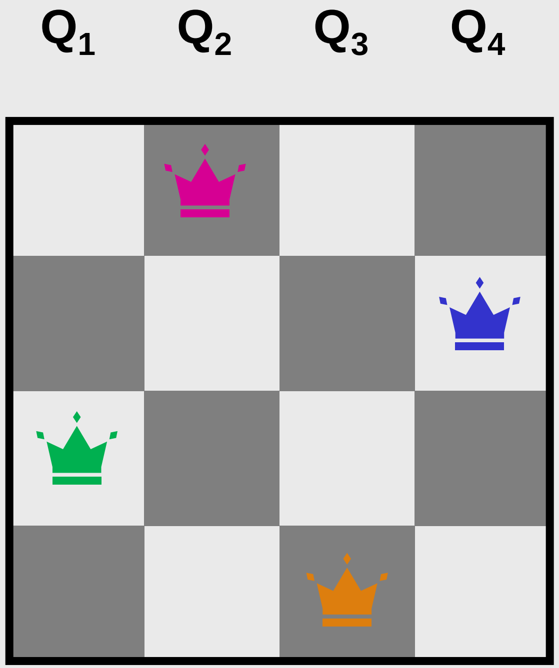
University of Reading

Image Credit: NASA/JPL.

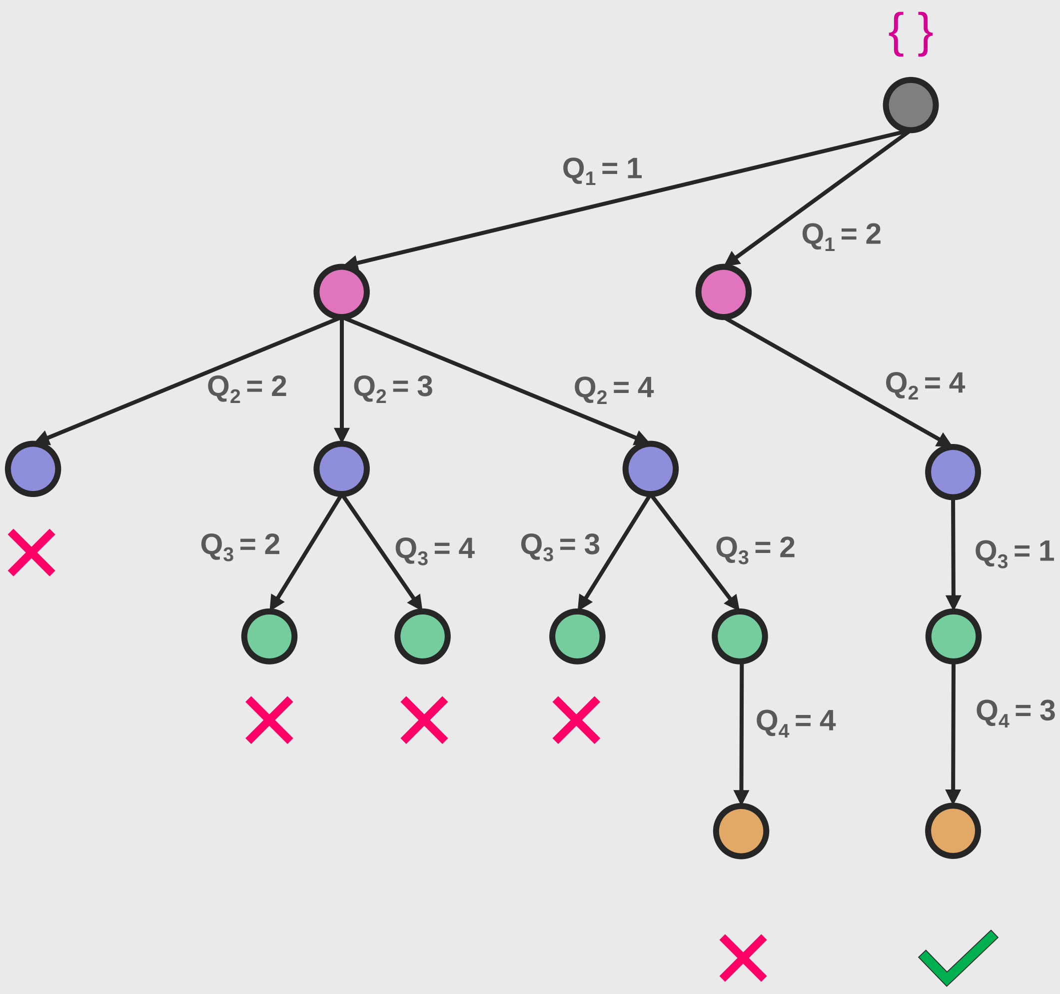

You’re here
4 billion miles away
from earth in Voyager 1
That's Earth

Search for Solution(s) in a Tree





Game Trees (Definition)

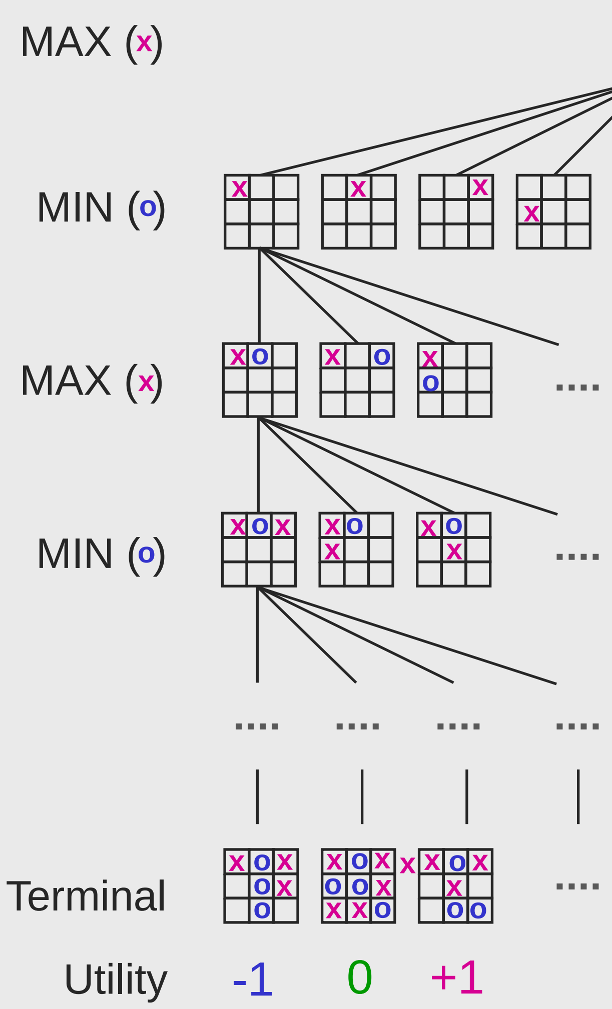


- INITIAL STATE ({ })
- ACTIONS function (Q_1 move to col 2)
- RESULT function (✗ ✓)

- the **nodes** are game states
- the **edges** are moves.

Game Trees (Tic-Tac-Toe)

6:43 PM



INITIAL STATE

MAX(x) has 9 moves

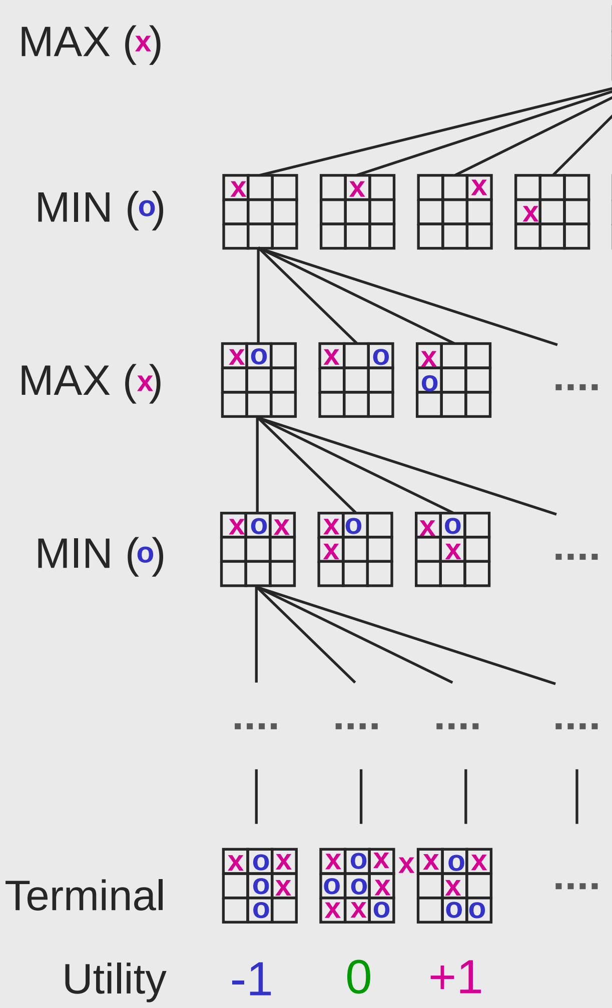
ACTIONS function

Alternatively MAX places **x** and MIN places **o** *until reach* leaf (terminal) node

RESULT function

utility value of the terminal state from the point of view of MAX; high values are assumed to be good for MAX and bad for MIN

Game Trees (Tic-Tac-Toe)

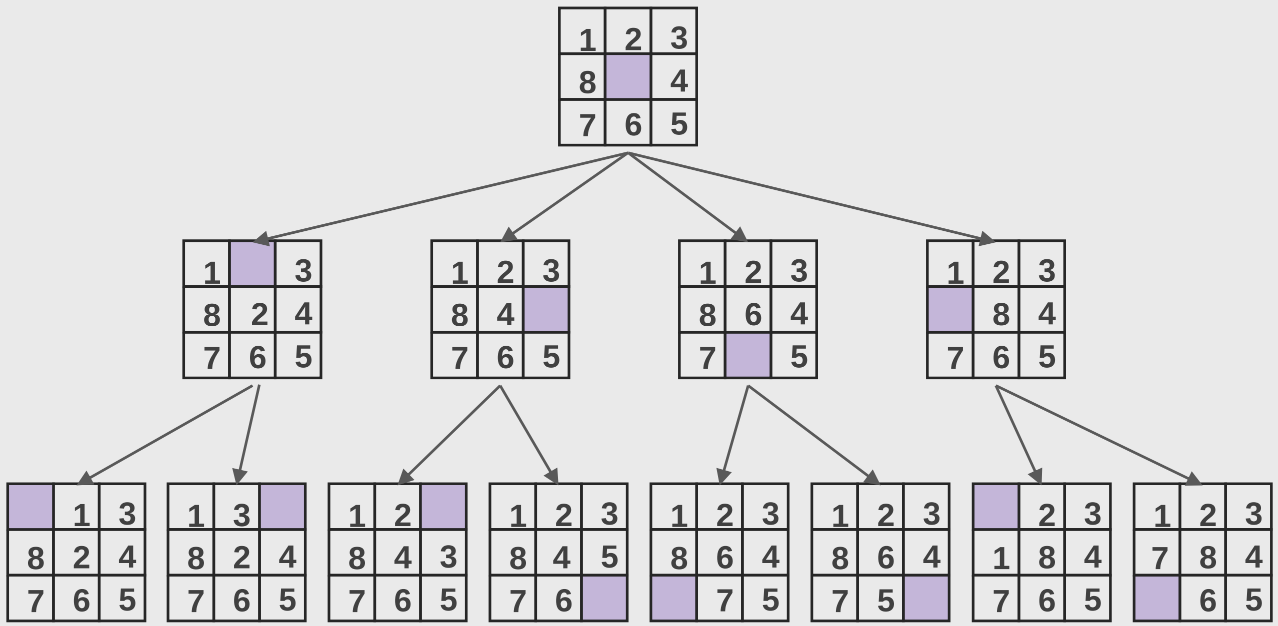


- Terminal node for tic-tac-toe game tree is: fewer than

  9! = 362,880 nodes.

- For **chess** there are over $\mathbf{10^{40}}$ **nodes**.

Game Tree Types



Example: Sliding puzzle

Single-player path finding problems.

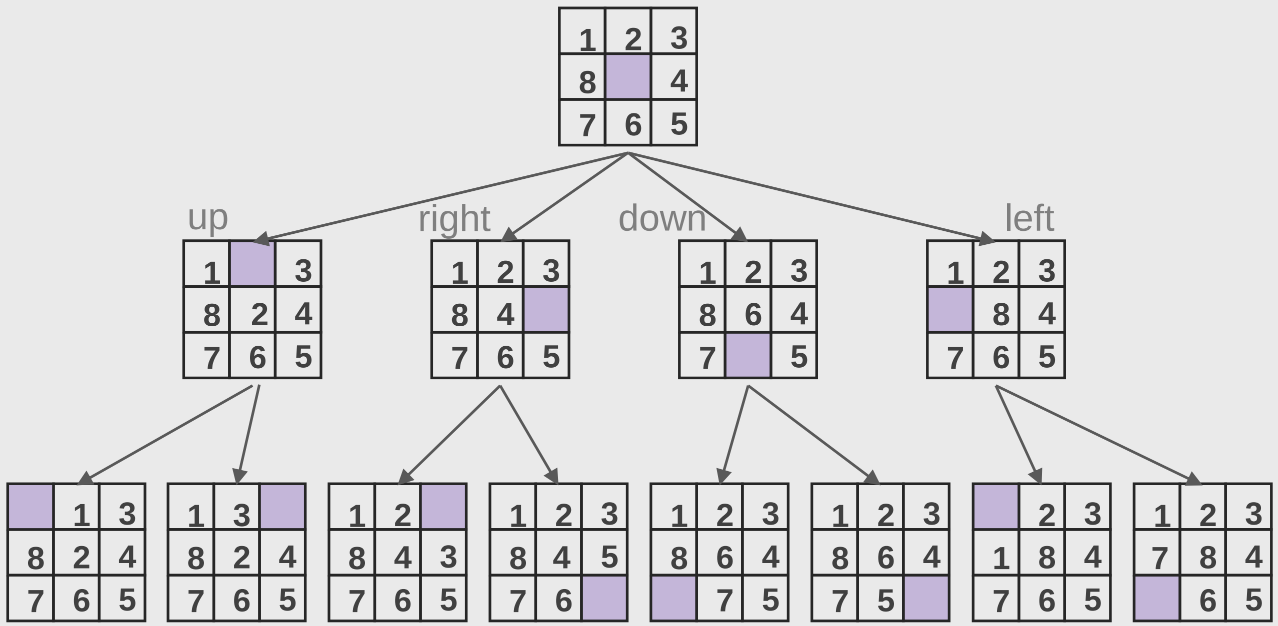
- Rubik’s Cube
- Sliding puzzle.
- Travelling Salesman Problem.

Two-player games.

- Tic-Tac-Toe
- Chess
- Checkers
- Othello

- **Constraint satisfaction problems**.
  - Eight Queens (N-Queen)
  - Sudoku

Game Tree – Problem Space



Example: Sliding puzzle

Each game consists of

- a problem space,
- an initial state, and
- a single (or a set of) goal states.

A **problem space** is a mathematical abstraction in the form of a tree:

- the root represents current state
- nodes represent states of the game
- edges represent moves
- leaves represent final states (win, loss or draw)

Example: **8-Puzzle game**

- nodes: the different permutations of the tiles.
- edges: moving the blank tile up, down, right or left.

Game Tree – Problem Space

Choice of a problem space

- not so obvious for some problems.
- One general rule is that a **smaller representation, in the sense of fewer states to search, is often better then a larger one**. A problem space is characterized by two major factors.

The branching factor - the average number of children of the nodes in the space.

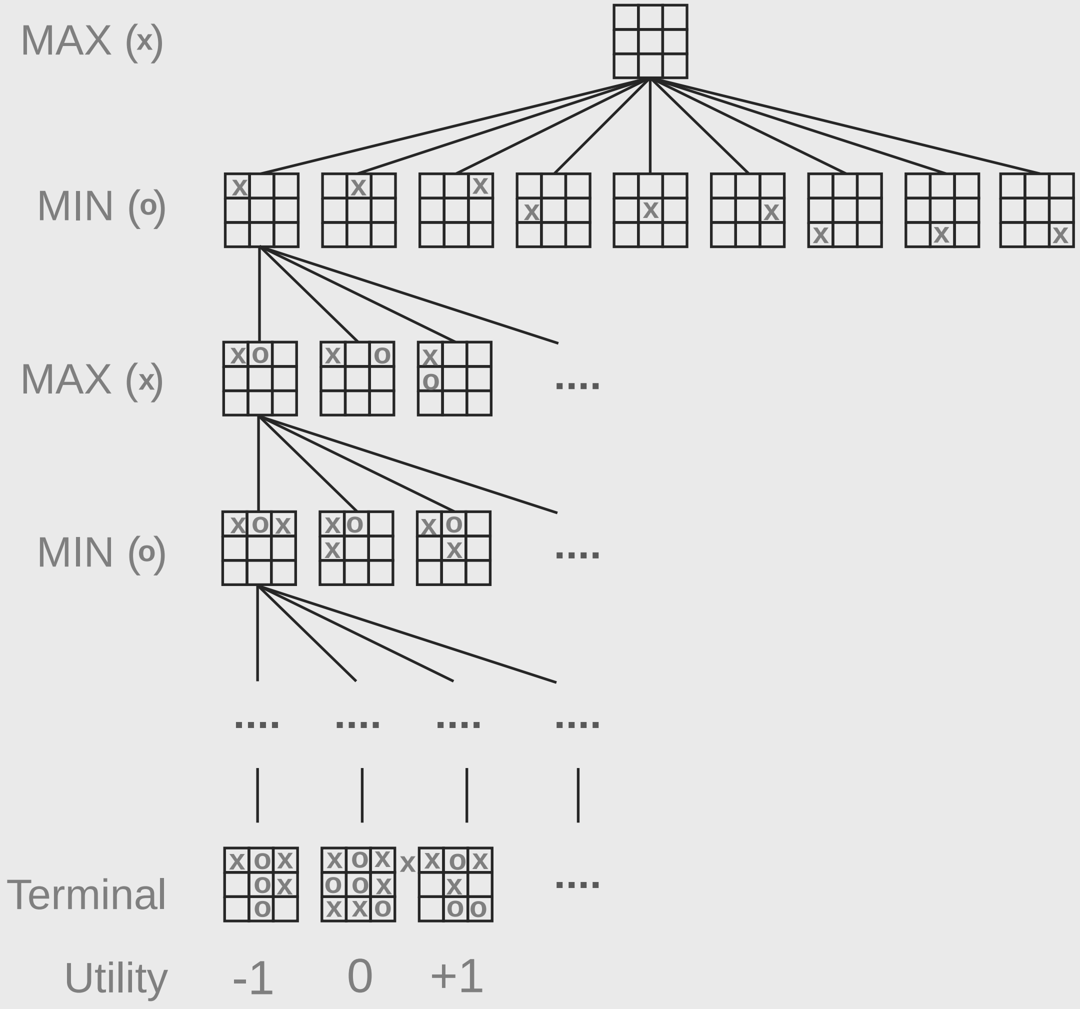
- The eight puzzle has a branching factor of **2.13**
- Rubik's cube has a branching factor of **13.34**
- Chess has a branching factor of about **35**

The solution depth

- The length of the shortest path from the initial node to a goal node.
- **The size of a solution space**:
  - Tic-Tac-Toe is 9! = 362,880
  - 8-puzzle - 9!/2
  - Checkers - $\mathbf{10^{40}}$
  - Chess - $\mathbf{10^{120}}$ (40 moves, 35 branch factor - $\mathbf{35^{(2*40)}}$)

Game Trees - Search for a Move

- Brute-Force Search
- Minimax
- Heuristic Search
  - Dijkstra Algorithm
  - Best-First Search
  - A* algorithm



Search

- **Uninformed / Systematic**
  - Brute force search
  - Minimax search
- **Informed / Non-Systematic**
  - Best-First search
  - A* Algorithm

Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 4/10: Search and Reasoning

Part 2
Systematic Search

DR VARUN OJHA

Department of Computer Science

University of Reading

Systematic Search

Brute-Force Search And Minimax

Brute-Force Search

Breadth-First Search

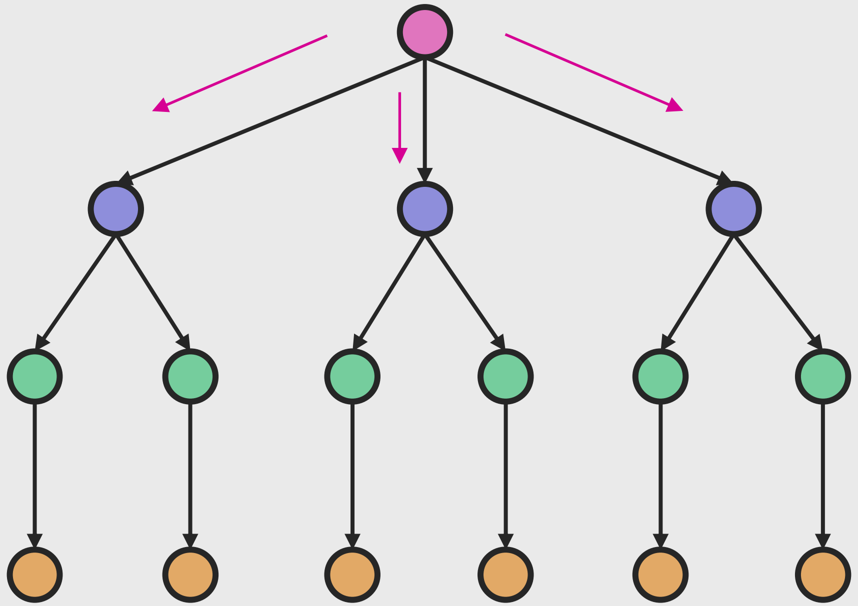
Depth-First Search

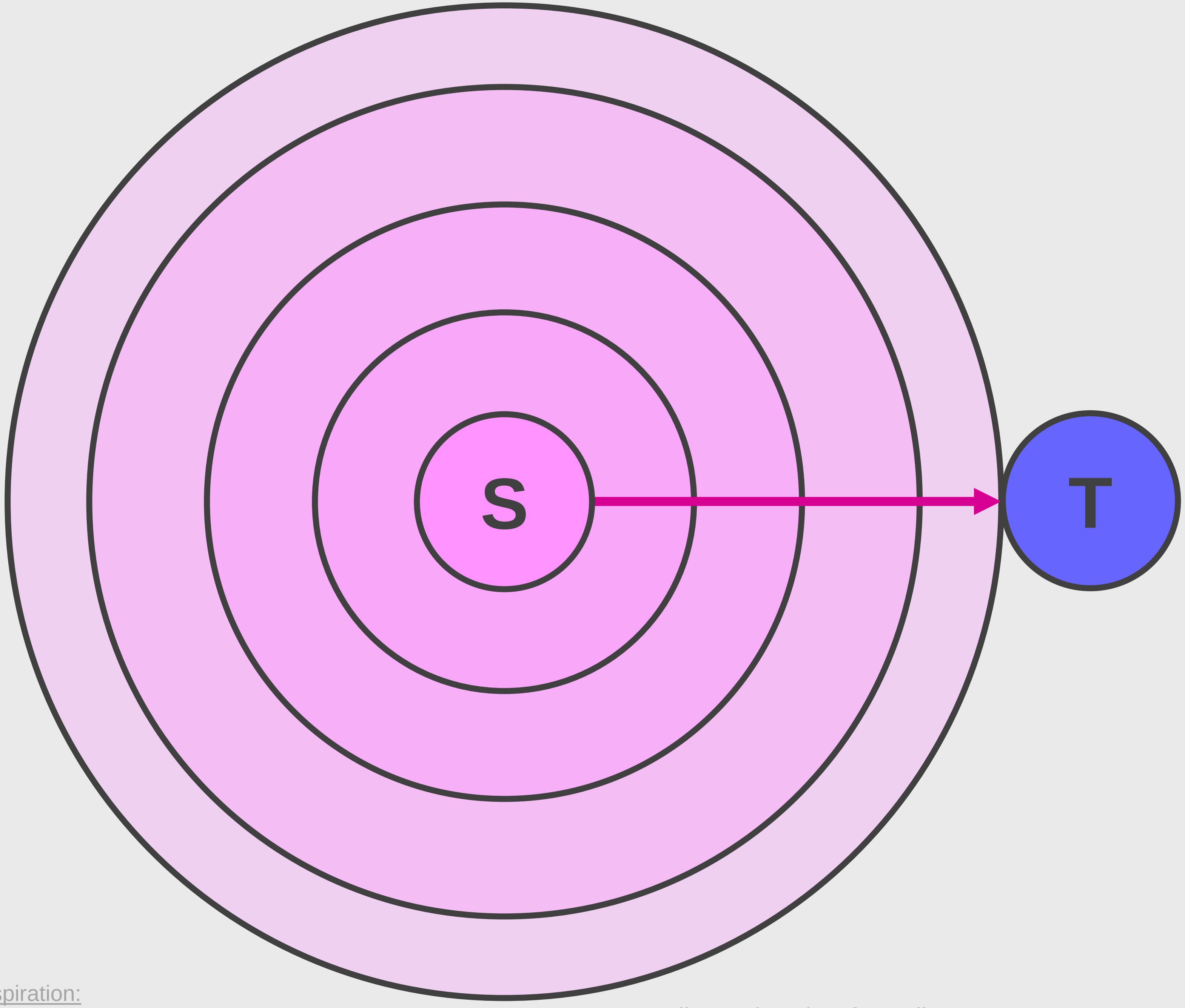
Breadth-First Search

- Breadth-First search (BFS) expands nodes in order of their depth from the root.
- Implemented by first-in first-out (FIFO) queue.
- BFS will find a shortest path to a goal.
- Time/Space Complexity - **branching factor b** and the solution depth **d**.
- Generate all the nodes up to level d.
- Total number of nodes in BFS

$$1 + b + b^2 + \ldots + b^d = O(b^d)$$

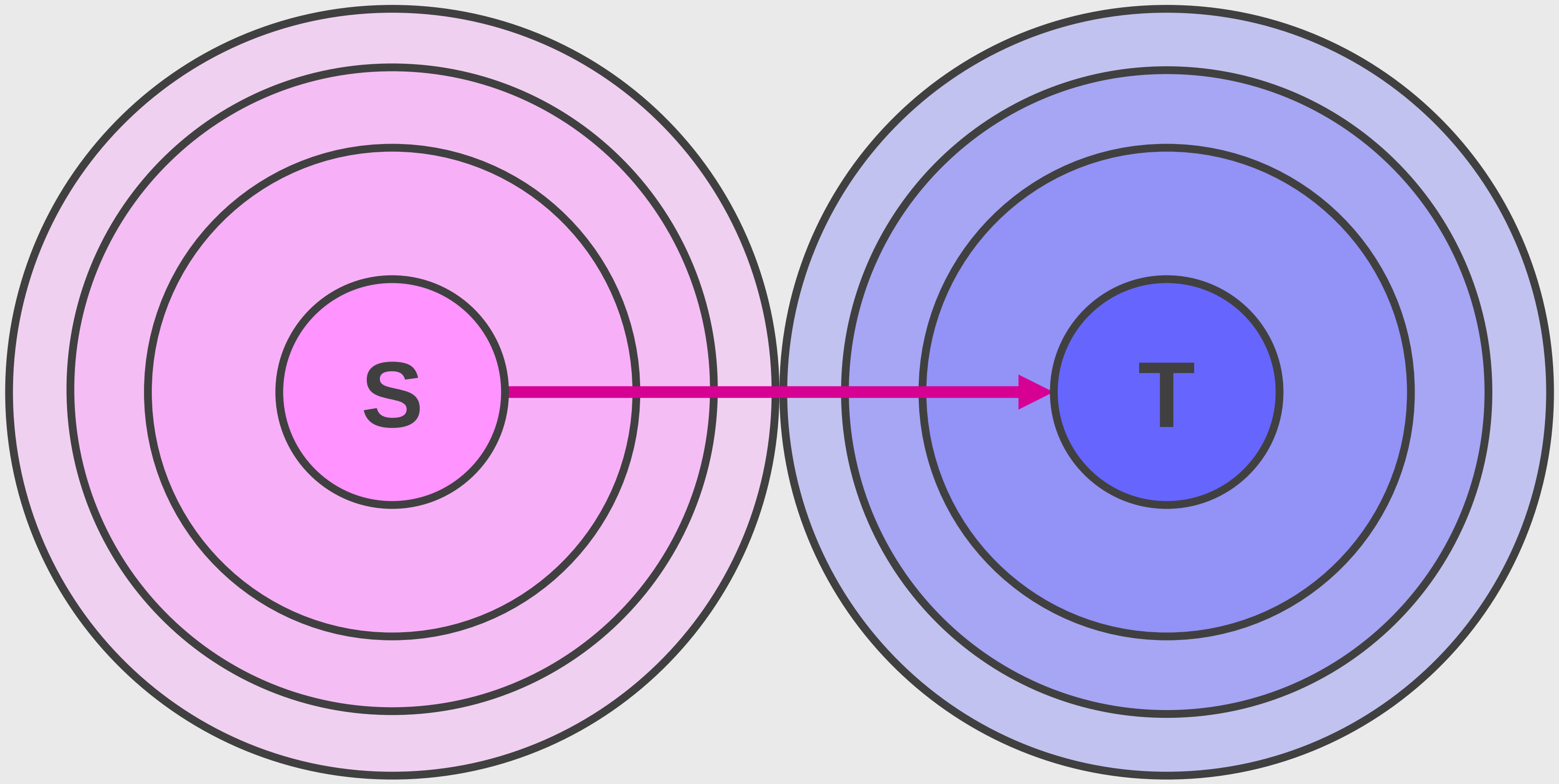
- BFS **will exhaust the memory in minutes**.





6:43 PM

Graphics inspiration:
https://qiao.github.io/PathFinding.js/visual/

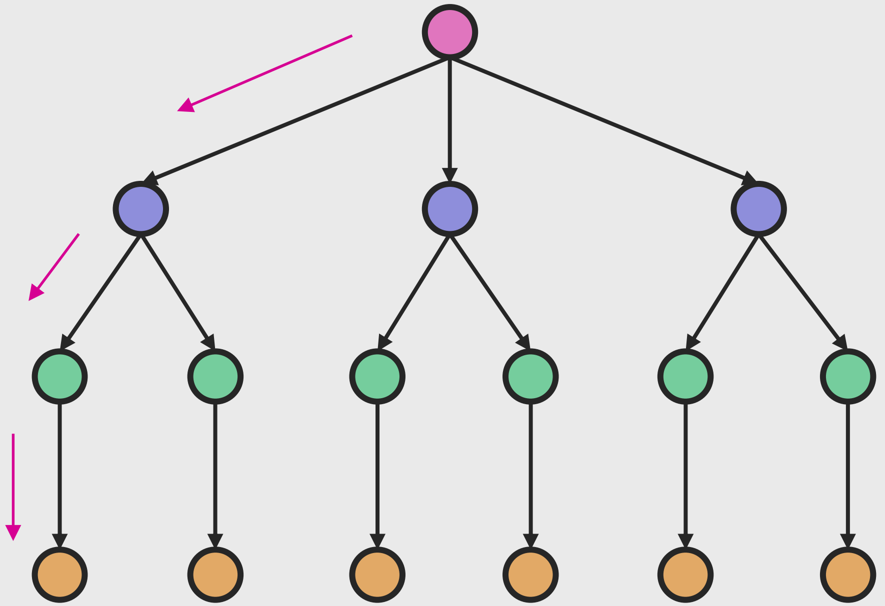


Bi-Directional Breadth-First Search

Graphics inspiration:
https://qiao.github.io/PathFinding.js/visual/

Depth-First Search

- **Depth-First is iterative-deepening**
  - First performs a DFS to depth one. Than starts over executing DFS to depth two and so on.
- Implemented by LIFO stack
- Space Complexity is linear in the maximum search depth.
- DFS generate the same set of nodes as BFS
- Time Complexity is $O(b^d)$
- **The first solution DFS found may not be the optimal one**.
- **On infinite (branch) tree DFS may not terminate**.



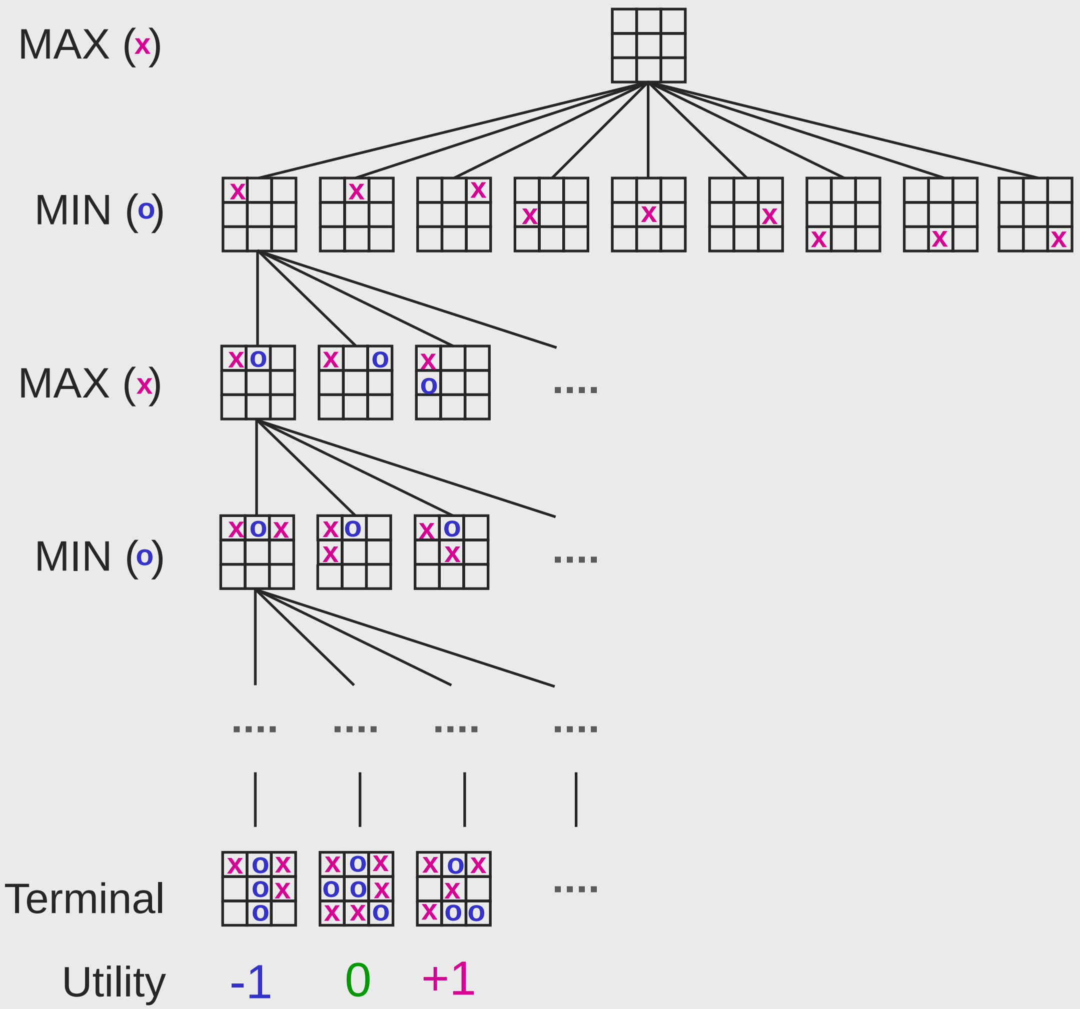
Minimax

- We consider games with two players
- **Zero-Sum games:** One person's gains are the result of another person's losses (so called).
- The minimax algorithm is a specialized search algorithm which **returns the optimal sequence of moves for a player in a zero-sum game**.
- In the game tree that results from the algorithm, **each level represents a move by either of two players, say A and B**.

Minimax: Example

- Tic-Tac-Toe
  - Player A: MAX (**x**)
  - Player B: MIN (**o**)

  - Zero Sum:
    - If MAX wins gets **+1**
    - If MIN wins gets **-1**
    - Net Sum = 0

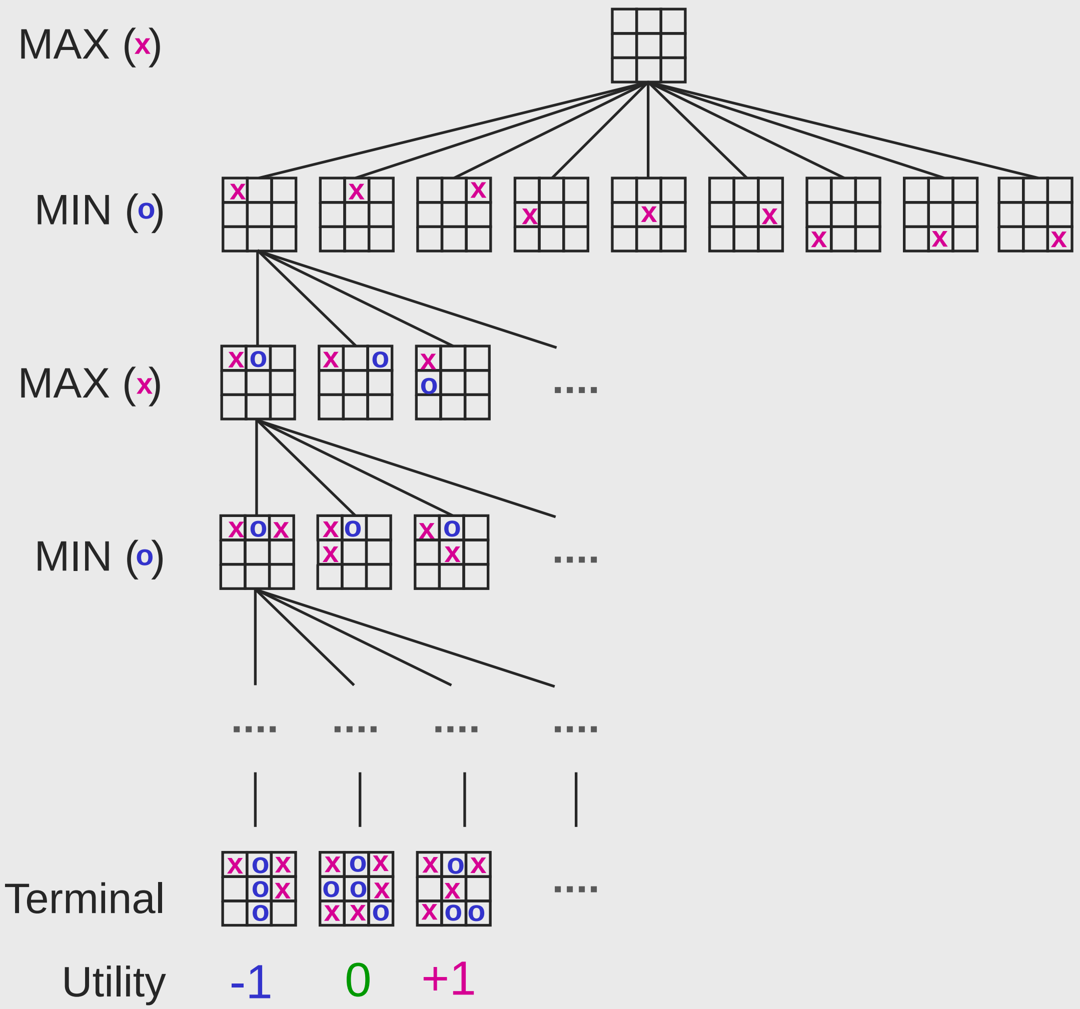


Minimax

- The minimax algorithm explores the entire game tree using a depth-first search.

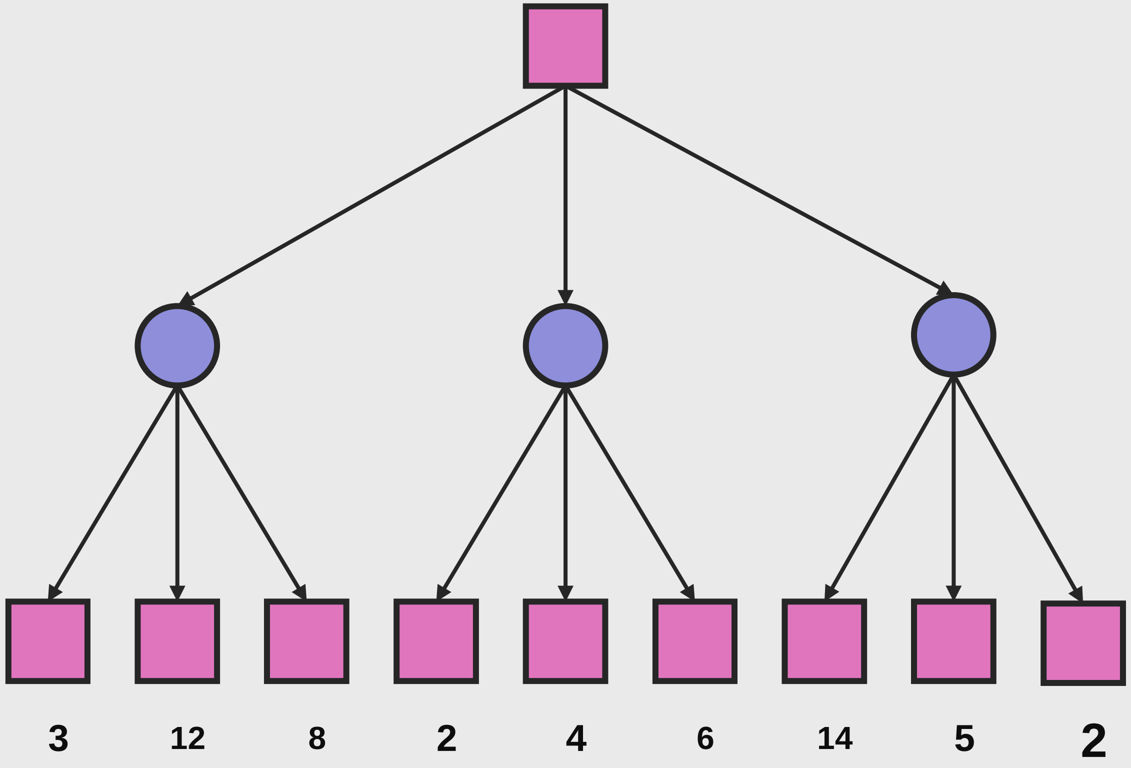
- **At each node** in the tree where **player-A** has to move. The **player-A** would like to play the move that **maximises** the payoff.
- **Player-A** will assign the maximum score amongst the children to the node where Max makes a move.
- Similarly, **player-B** will minimize the payoff to A-player.

- The maximum and minimum scores are taken at alternating levels of the tree, since A and B alternate turns.



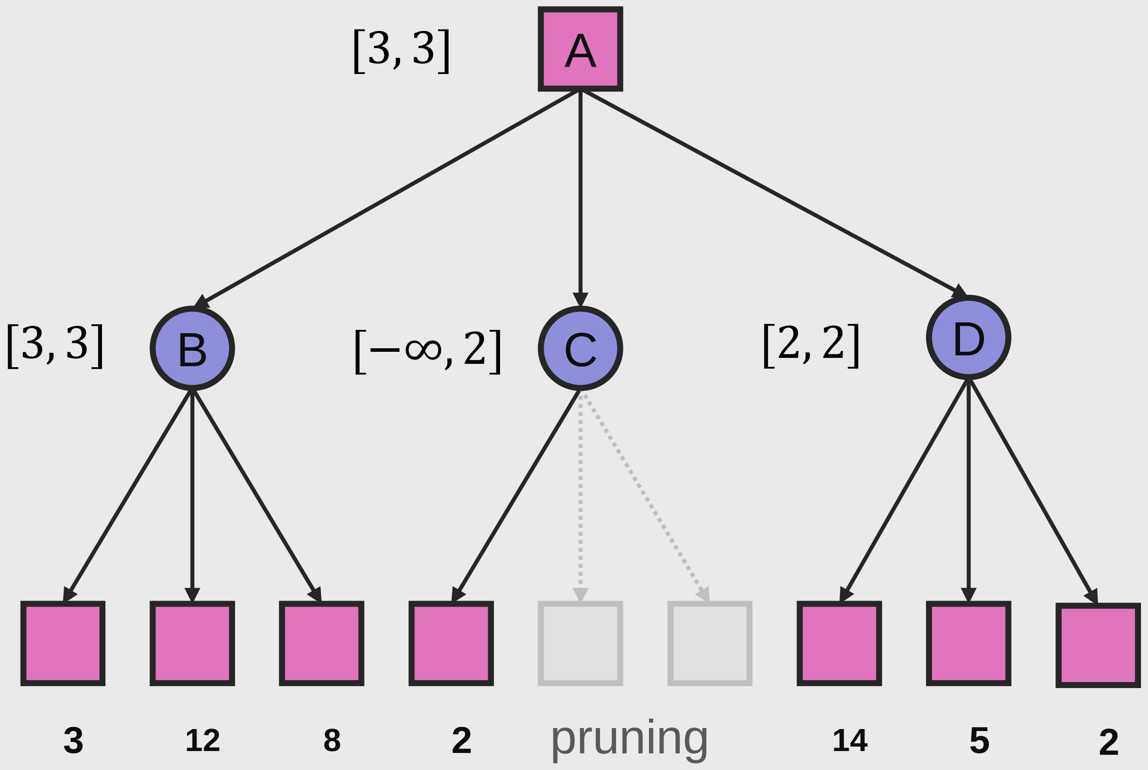
Alpha-Beta Pruning

- Alpha-beta pruning improve the efficiency of Minimax search and reduces the number of state to examine in a game tree.
- It **prunes the branches** that will not influence decision of a node.



Alpha-Beta Pruning

- Initialise $[\alpha = -\infty, \beta = +\infty]$ to the MAX (root node A) and explore its child
- Leaf of B is 3. Set $[\alpha = -\infty, \beta = 3]$ since B is MIN node and it will play **at most** 3. That is beta is the ***minimum upper bound*** of possible solutions
- Explore other child of B to see if any other child has less than 3.
- Last child of B has 8. Set B with $[\alpha = 3,\ \beta = 3]$.
- Root (MAX node A) can play **at least** 3. Set $[\alpha = 3, \beta = +\infty]$. Explore other child to see if any child has a grater value than 3. That is alpha is the ***maximum lower bound*** of possible solutions
- MIN node C has 2. Hence, its other child are pruned since C will not play more than 2 and node B has 3. Hence, A will NOT play C.
- Similarly explore other child of A to check if it can play more than 3.



Systematic Search

- Brute-force and Minimax systematically search the **whole search space.**
  - **Limitation –** Sometimes however it is not feasible to search the whole search space - it's just too big!
  - **Solution –** Use heuristic search (non-systematic search)

Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 4/10: Search and Reasoning

Part 3
Non-Systematic Search

DR VARUN OJHA

Department of Computer Science

University of Reading

Non-Systematic Search

Heuristic Search

Heuristic Search: Principles

Strategy – rather than trying all possible search paths, focus on paths that seem to be getting us closer to the goal state.

Limitation – generally can't be sure that the goal state is really near.

Advantage – might be able to have a good guess based on some heuristics.

Evaluation function – evaluation function that ranks nodes in the search tree according to some criteria (for example, how close we are to the target). This function provides a quick way of guessing.

Heuristic Search: Properties

1. It must provide an **accurate estimator** of the cost to reach a goal.
2. It must be **cheap to compute**.
3. It always must be a **lower bound on actual solution cost**.

Dijkstra Algorithm

- It find the shortest path between two nodes in a graph
- Steps:
  1. Initially all nodes are marked *unvisited* and assigned value ∞
  2. Start with assigning initial node with values 0
  3. Visit other unvisited node assign smallest tentative distance from initial node mark them visited. And **REPEAT**

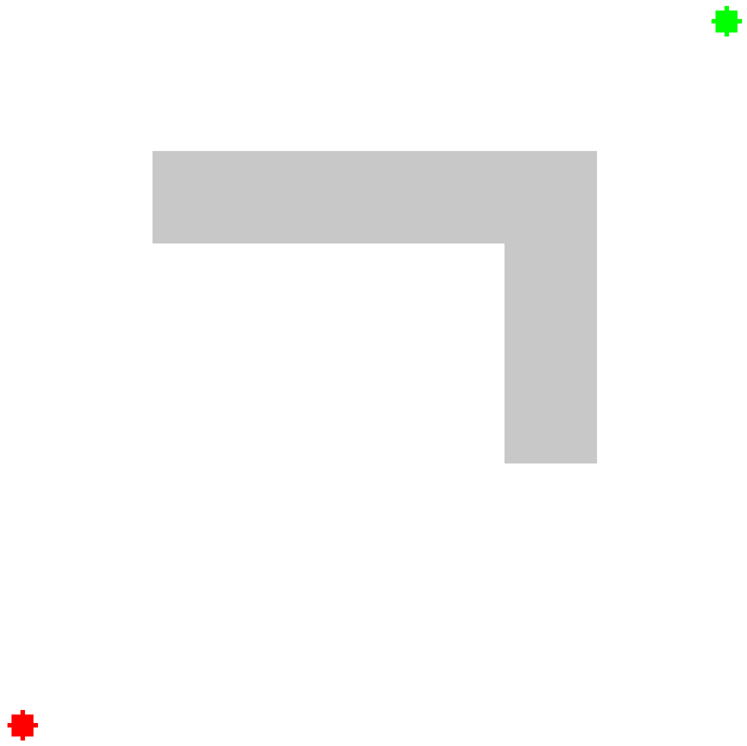
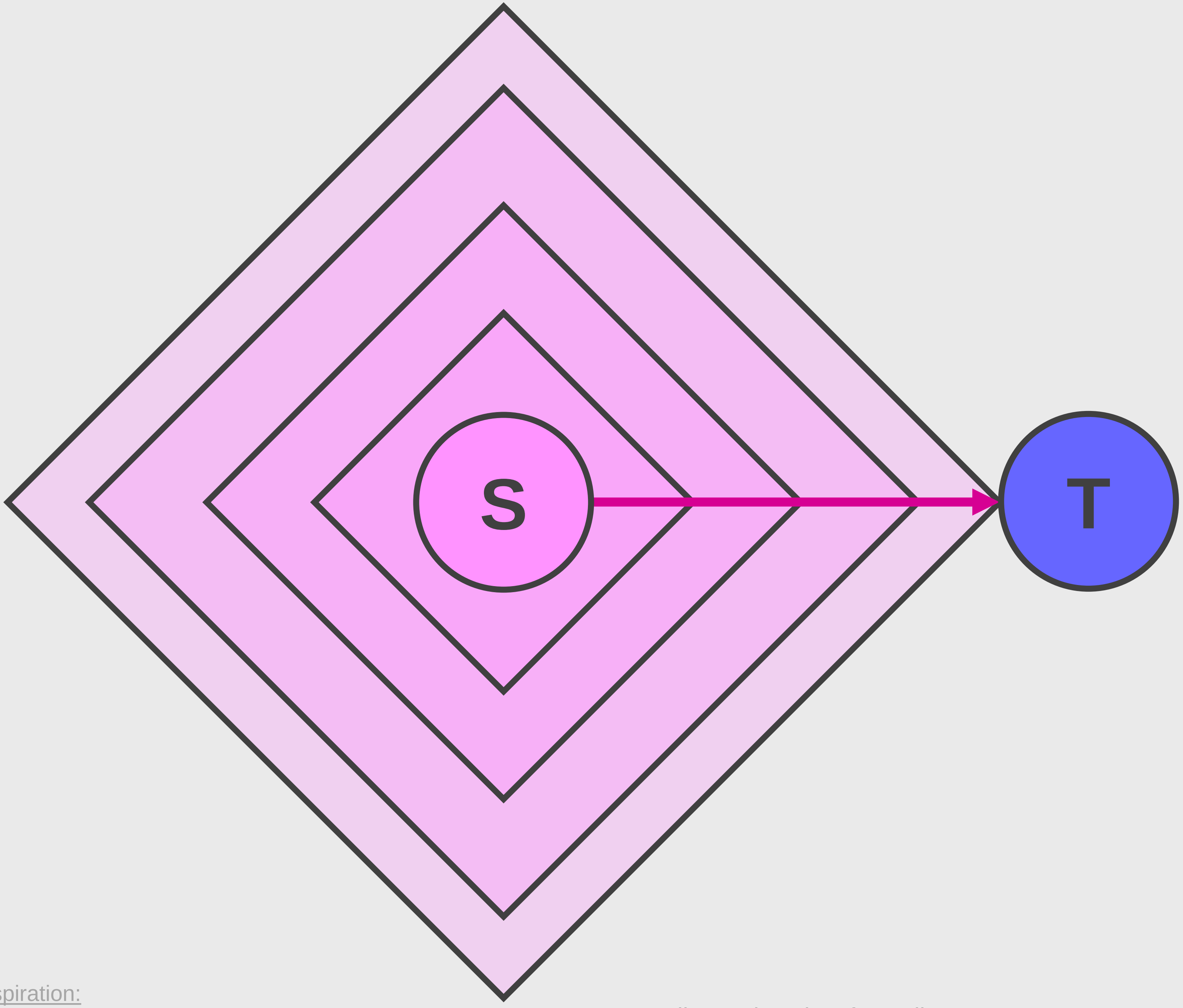
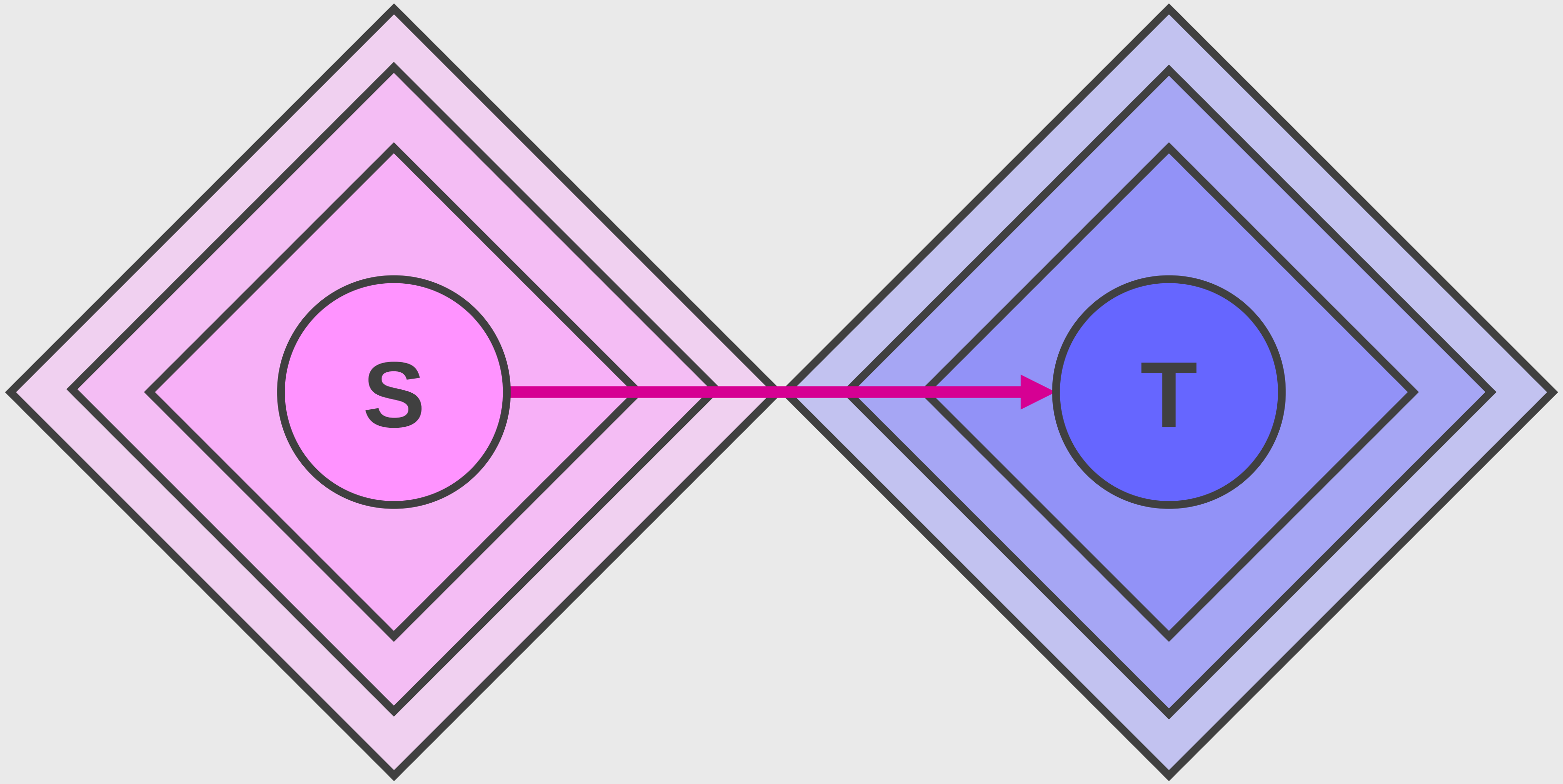


Illustration source: https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm



Graphics inspiration:
https://qiao.github.io/PathFinding.js/visual/



Bi-Directional Dijkstra Search

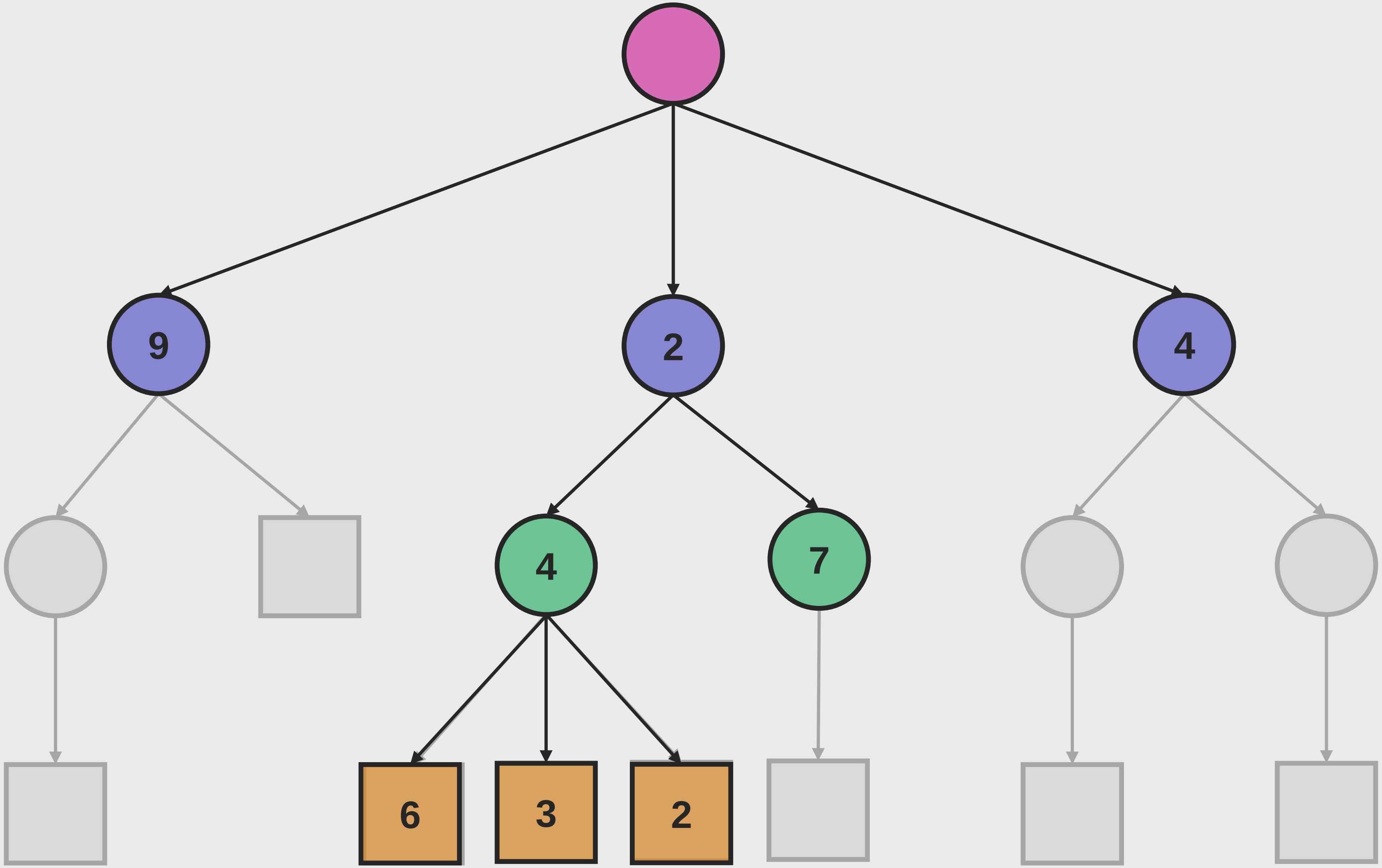
Best-First Search

- The search is similar to Breadth First Search, but **instead of taking the first node it always chooses a node with the best score**, according to an evaluation function.

- If we create a good evaluation function, **best first search may drastically cut down the amount of search time**.

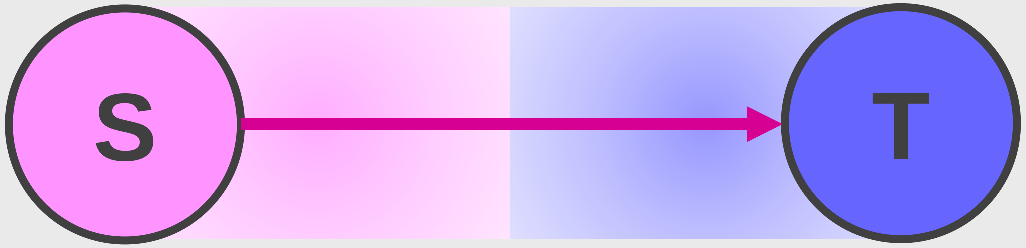
- It is a Greedy algorithm. It uses a **heuristic to evaluate the path**.

6:43 PM



Best-First Search

Graphics inspiration:
https://qiao.github.io/PathFinding.js/visual/



Bi-Directional Best-First Search

Graphics inspiration:
https://qiao.github.io/PathFinding.js/visual/

A* Algorithm

- A* is a variant of Best-First search. Since Best-First search only accounts for heuristic and **the cheapest cost of the path from a start state to the current state**. So, we may find a solution but it may be **not a very good solution**.

- A* attempts to find a solution which **minimizes** the total cost of the solution path.

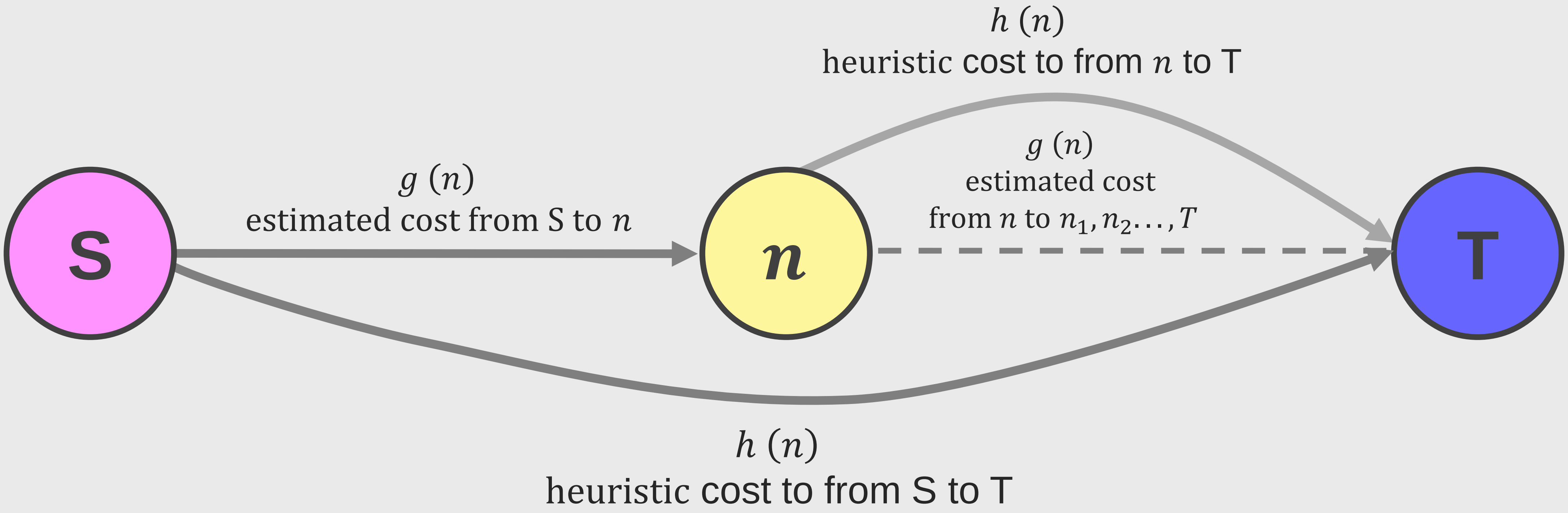
- This algorithm combines advantages of Breadth-First search with advantages of best first search.

Best-First Search

$$f(n) = h\,(n)$$

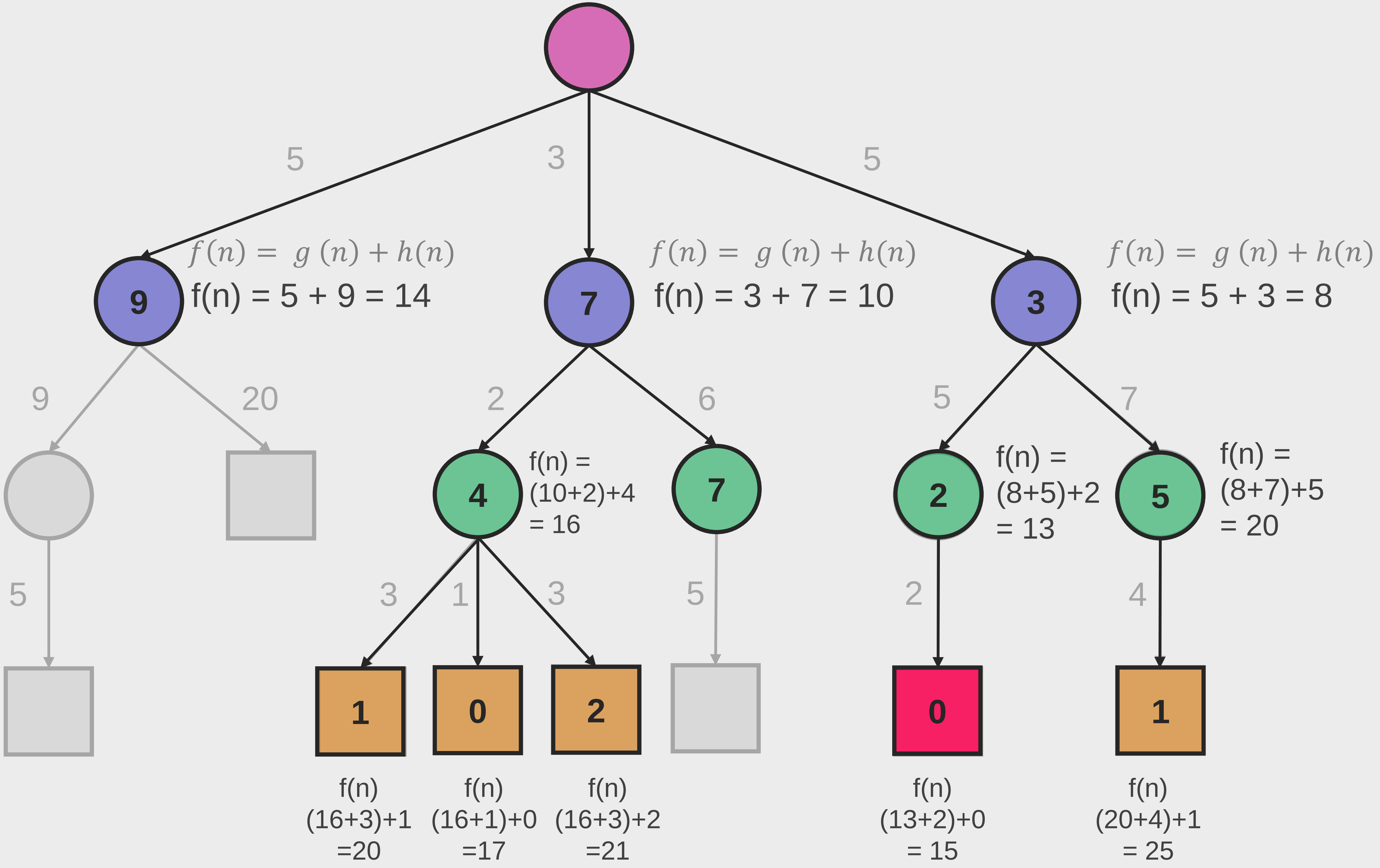
A* Algorithm

$$f(n) = \; g\,(n) + h(n)$$



Admissibility of a heuristic $h(n)$

- A heuristic $h(n)$ is **admissible if it never overestimate** the cost to the Goal. That is $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost from a state n to the Goal.
- Admissible heuristics can be measured as:
  - $h(n) = 0$ (set to zero)
  - $h(n) = \sqrt{(n_x - T_x)^2 + \left(n_y - T_y\right)^2}$ (straight line)



These are suboptimal

Goal reached!

Is it optimal?

Path Finding Example

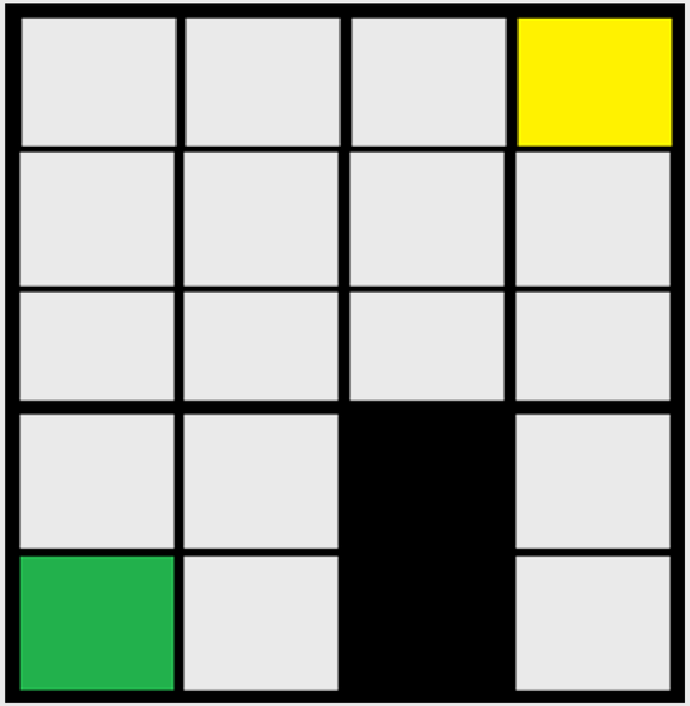
Example adapted from: https://brilliant.org/wiki/a-star-search/ (Accessed on 31 Jan 2021)

S Initial State

T Goal State

$$F(n) = G\,(n) + H(n)$$

$$H(n) = \sqrt{(n_x - T_x)^2 + \left(n_y - T_y\right)^2}$$



Path Finding Example

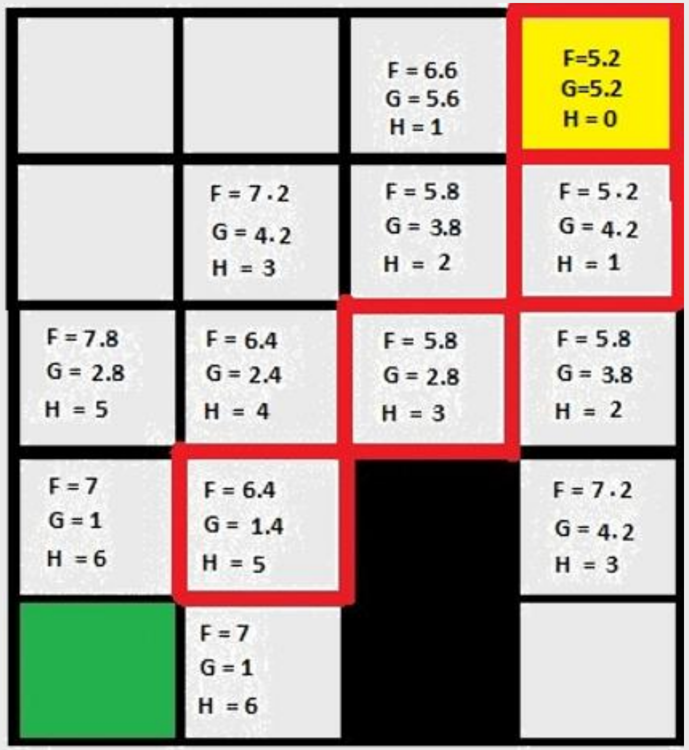
Example adapted from: https://brilliant.org/wiki/a-star-search/ (Accessed on 31 Jan 2021)

S Initial State

T Goal State

$$F(n) = G\,(n) + H(n)$$

$$H(n) = \sqrt{(n_x - T_x)^2 + (n_y - T_y)^2}$$




S
T

Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 4/10: Search and Reasoning

Part 4
Reasoning

DR VARUN OJHA

Department of Computer Science

University of Reading

Probability

Bayes Theorem

likelihood

prior

$$P(H|E) = \underbrace{\frac{P(E \mid H)\, P(H)}{P(H)P(E \mid H) + P(\neg H)P(E \mid \neg H)}}$$

posterior

normalising constant

Where H and E are events

$P(H \mid E)$ is a conditional probability, the likelihood of H given E is true.

$P(E \mid H)$ is a conditional probability the likelihood of E given H is true.

$P(H)$ and $P(E)$ are probabilities of observing H and E

Bayesian Inference (Sequential)

likelihood

prior

$$P(H \mid E_1, E_2) = \frac{P(E_1 \mid H)P(E_2 \mid H)P(H)}{P(E_1)P(E_2)}$$

posterior

normalising constant

Where H and E_i are events

$P(H \mid E_i)$ is a conditional probability, the likelihood of H given E_i is true.

$P(E_i \mid H)$ is a conditional probability the likelihood of E_i given H is true.

$P(H)$ and $P(E_i)$ are probabilities of observing H and E_i

Probabilistic Reasoning

Fact: You return home and the **door** is open

Reason: Is it a **family** person?

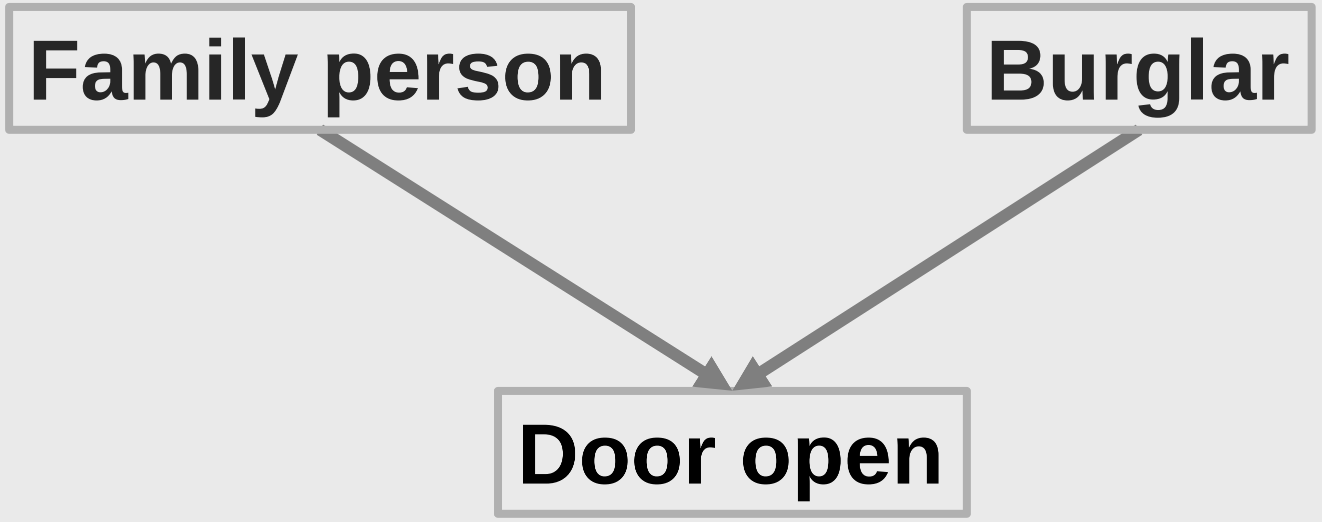
Reason: Is it a **Burglar**?

Who opens the door? Is something stolen? …

How do we represent these relations?

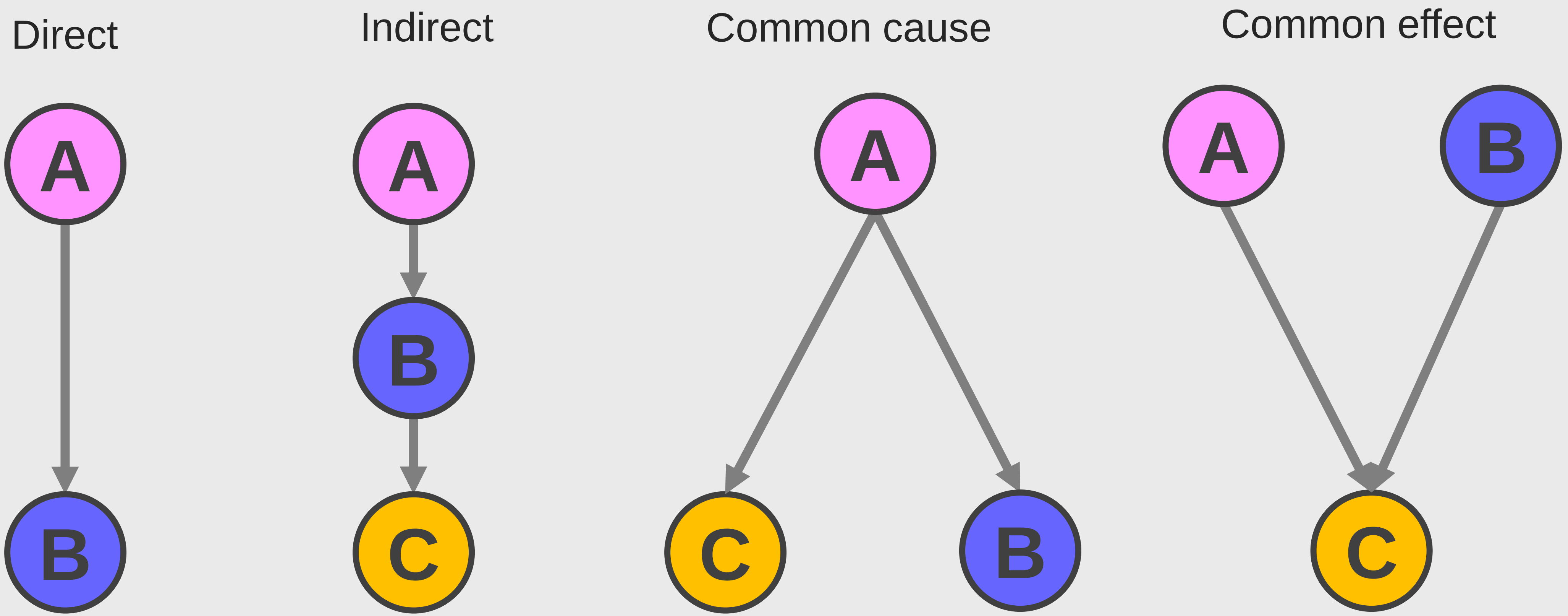
Belief Network

Causal relationship are represented in a direct acyclic graph (DAG) and arrows represent relationship.



Probabilistic Relationships

6:43 PM



$P(B \mid A)$

$P(B \mid A)$
$P(C \mid B)$

C is independent of A given B

$P(B \mid A)$
$P(C \mid A)$

$P(C \mid A, B)$

Joint Probabilities

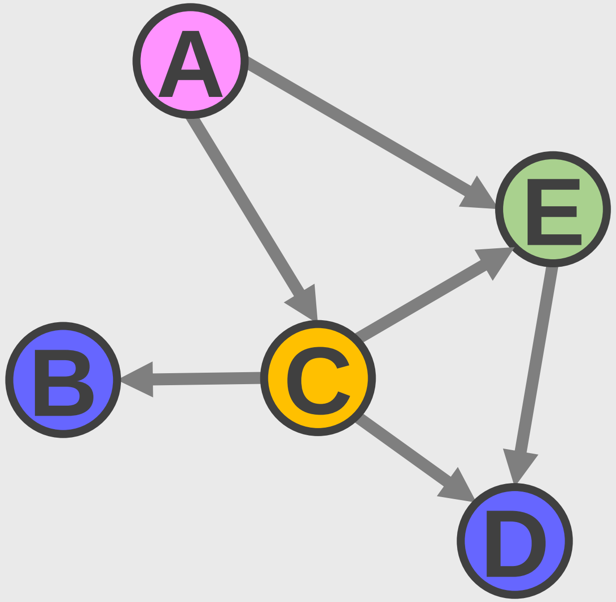
What is the probability that event A and B together (e.g., cloud and sun appearing together).

$$P\ (A, B) = P(A\ |B)\ P(B)$$

$$P\ (A, B) = P(B\ |A)\ P(A)$$

Bayesian Belief Network

$$P\,(A, B, C, D, E) = P(A)\,P(B|C)\,P(C|A)P(D|C, E)P(E|A, C)$$



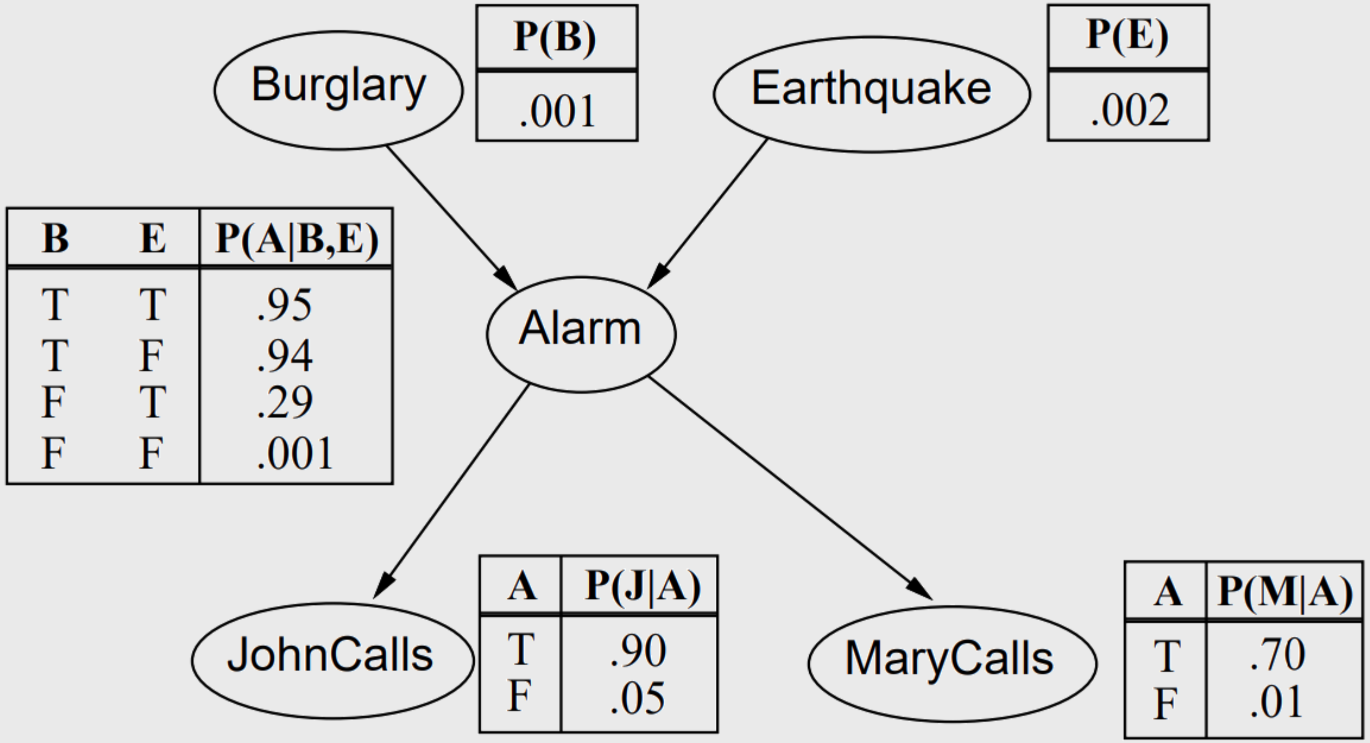
In General, we can write

$$\begin{aligned} P\,(x_1, x_2, \ldots, x_n) &= P(X_1 = x_1 \wedge \cdots \wedge X_n = x_n) \\ &= \prod_{i=1}^{n} P(x_i\,|\mathrm{parent}(X_i)) \end{aligned}$$

Goal: is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e. $\boldsymbol{P}[\,\boldsymbol{Cause}\,|\boldsymbol{Evidance}]$

Example Problem

Example adapted from:
Bayesian networks, Ch 14, Artificial Intelligence: A Modern Approach, Peter Norvig and Stuart J. Russell



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

Variables: Burglar, Earthquake, Alarm, JohnCalls, MaryCalls

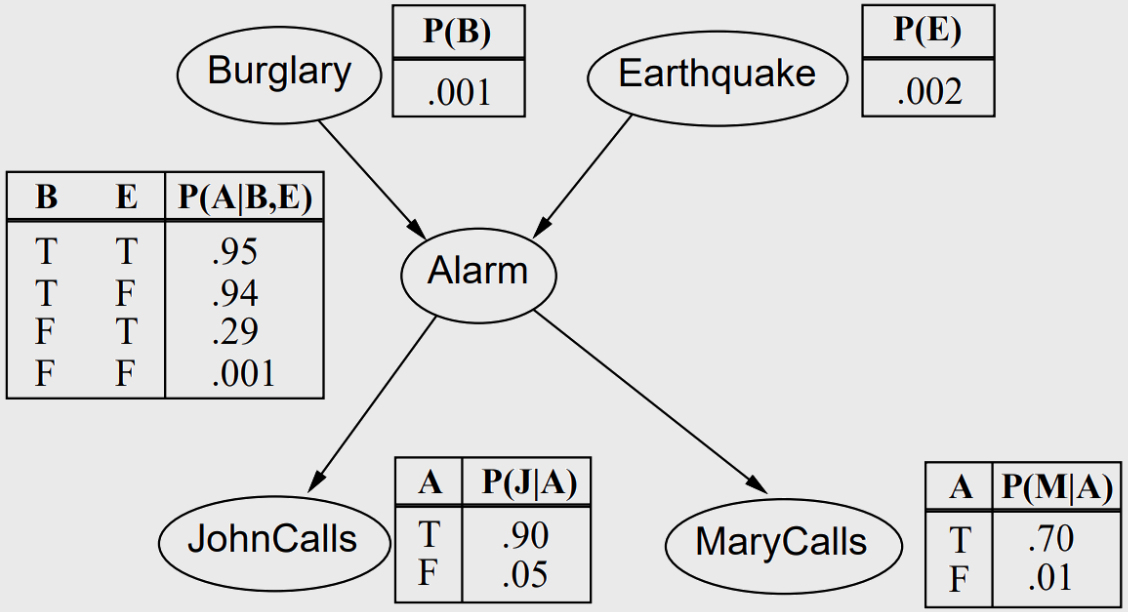
Network topology reflects "causal" knowledge:

- A burglar can set the alarm off
- An earthquake can set the alarm off
- The alarm can cause Mary to call
- The alarm can cause John to call

Note: I keep calling/writing BNN – but it should be BBN – Bayesian Belief Network

Example Problem

Example adapted from:
Bayesian networks, Ch 14, Artificial Intelligence: A Modern Approach, Peter Norvig and Stuart J. Russell



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

If we assume:
Burglar (B) = True
Earthquake (E) = True
Alarm (A) = True
JohnCalls (J) = True
MaryCalls (M) = False

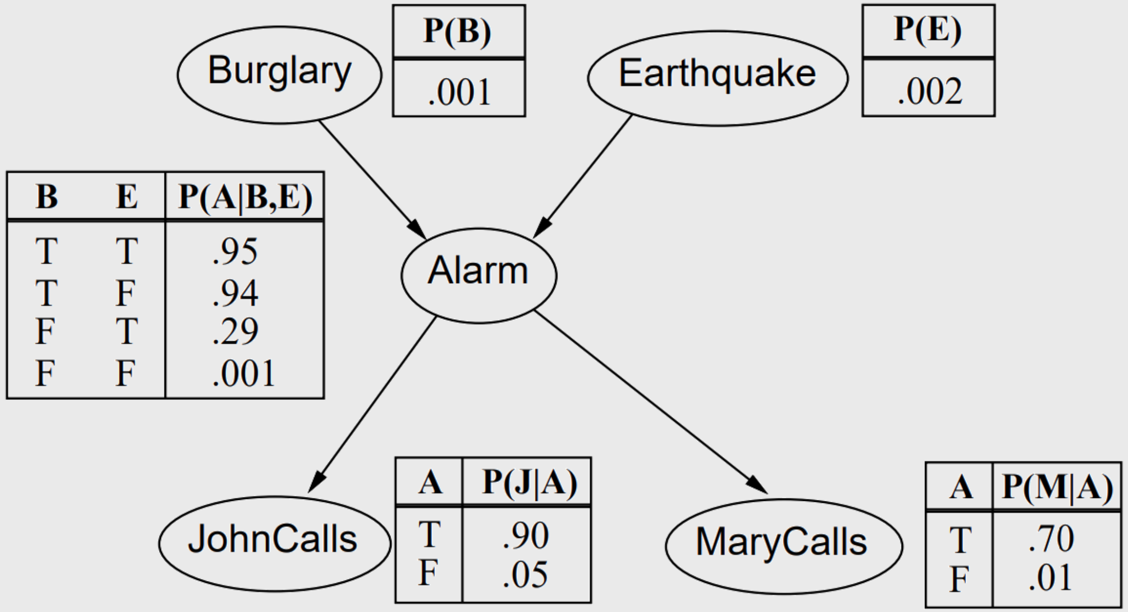
From this Bayesian Belief Network (BNN), we have the following probability:

$$P\ (B = T, E = T, A = T, J = T, M = F)$$

$$P\ (B = T, E = T, A = T, J = T, M = F) =$$
$$P(B{=}T)P(E{=}T)P(A{=}T|B{=}T,E{=}T)P(J{=}T|A{=}T)P(M{=}F|A{=}T)$$

Example Problem

Example adapted from:
Bayesian networks, Ch 14, Artificial Intelligence: A Modern Approach, Peter Norvig and Stuart J. Russell



If we assume:
Burglar (B) = True
Earthquake (E) = True
Alarm (A) = True
JohnCalls (J) = True
MaryCalls (M) = False

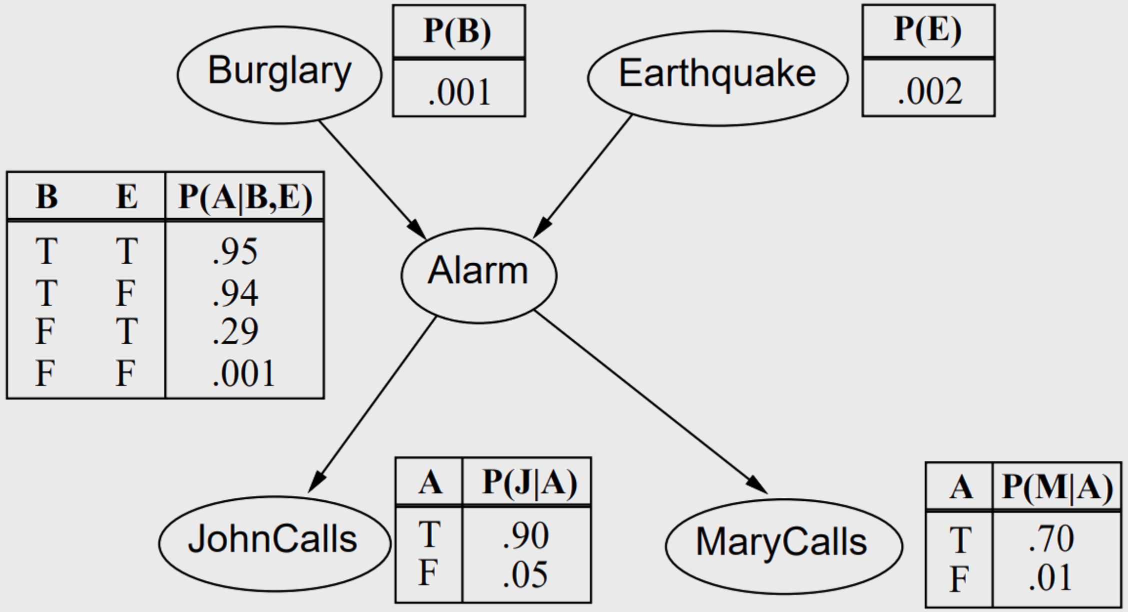
From this Bayesian Belief Network (BNN), we have the following probability:

$P\ (B = T, E = T, A = T, J = T, M = F)$

$$P\ (B = T, E = T, A = T, J = T, M = F) =$$
$$P(B{=}T)P(E{=}T)P(A{=}T|B{=}T,E{=}T)P(J = T|\ A{=}T)P(M = F|\ A{=}T)$$

Example Problem

Example adapted from:
Bayesian networks, Ch 14, Artificial Intelligence: A Modern Approach, Peter Norvig and Stuart J. Russell



P(B)
.001

P(E)
.002

B	E	P(A|B,E)
T	T	.95
T	F	.94
F	T	.29
F	F	.001

A	P(J|A)
T	.90
F	.05

A	P(M|A)
T	.70
F	.01

If we assume:
Burglar (B) = True
Earthquake (E) = True
Alarm (A) = True
JohnCalls (J) = True
MaryCalls (M) = False

From this Bayesian Belief Network (BNN), we have the following probability:

$$P\ (B = T, E = T, A = T, J = T, M = F)$$

We are interested in answering the prediction questions like:

- probability of Alarm going off $P\ (A = T\ )$
- probability of $P\ (John\ Calls\ |Alarm = T\ )$

Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 4/10: Search and Reasoning

Part 5
Practical Exercise

DR VARUN OJHA

Department of Computer Science

University of Reading