🔍 What is a Search Strategy in AI?

When an AI agent has to **find a solution or reach a goal**, it often needs to search through many possible states or paths. The **search strategy** is the way it decides **which path to explore next**.

There are two main types of search strategies:

1. **Uninformed search (Blind search)** – No clue about the goal; it explores blindly (like Breadth-First Search, Depth-First Search).
2. **Informed search (Heuristic search)** – Has **some knowledge or guess** about where the goal might be and uses that to guide the search.

🧠 Informed Search Strategies – The Basics

Informed search uses extra information (called a **heuristic**) to **make better choices** during the search. This helps it find solutions **faster and more efficiently**.

📌 **Heuristic Function (h(n)):**

- A function that **estimates how close** a node `n` is to the goal.
- It's like a **smart guess** that helps the algorithm pick better paths.

For example, if you're finding a path on a map, a heuristic could be:

> "straight-line distance from current city to the destination."

Key Characteristics of Informed Search Algorithms

1. **Heuristic Function**: Informed search algorithms use a heuristic function h(n) that provides an estimate of the minimal cost from node n to the goal. This function helps the algorithm to prioritize which nodes to explore first based on their potential to lead to an optimal solution.
2. **Efficiency**: By focusing on more promising paths, informed search algorithms often find solutions more quickly than uninformed methods, especially in large or complex search spaces.
3. **Optimality and Completeness**: Depending on the heuristic used, informed search algorithms can be both optimal and complete. An algorithm is complete if it is guaranteed to find a solution if one exists, and it is optimal if it always finds the best solution. For instance, the A* search algorithm is both complete and optimal when the heuristic function is admissible (i.e., it never overestimates the true cost).

What is the Greedy-Best-first search algorithm?

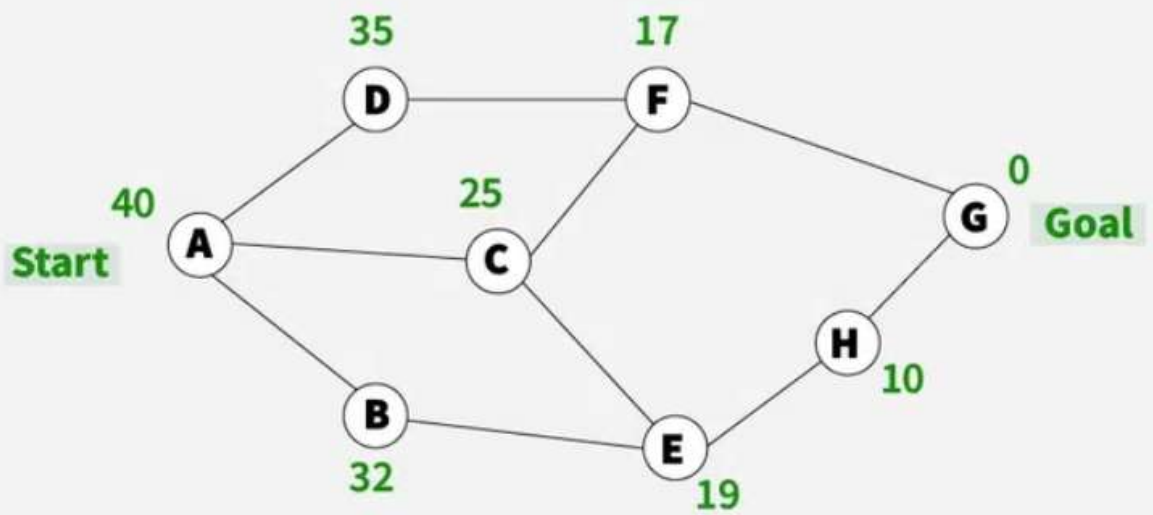
> *Greedy Best-First Search is an AI search algorithm that attempts to find the most promising path from a given starting point to a goal. It prioritizes paths that appear to be the most promising, regardless of whether or not they are actually the shortest path. The algorithm works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.*

The algorithm works by using a heuristic function to determine which path is the most promising. The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths. If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

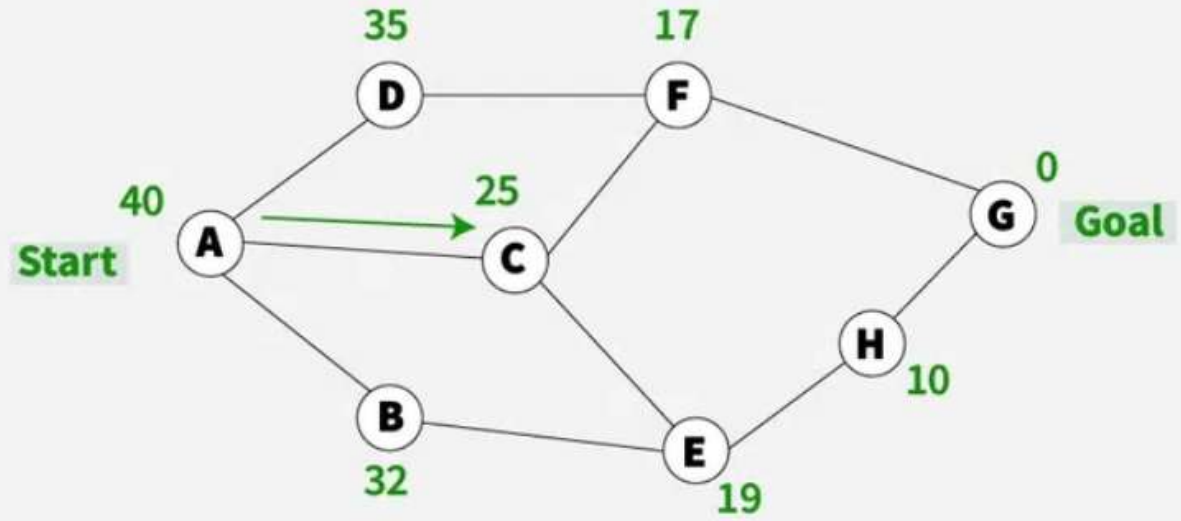
How Greedy Best-First Search Works?

- Greedy Best-First Search works by evaluating the cost of each possible path and then expanding the path with the lowest cost. This process is repeated until the goal is reached.
- The algorithm uses a heuristic function to determine which path is the most promising.
- The heuristic function takes into account the cost of the current path and the estimated cost of the remaining paths.
- If the cost of the current path is lower than the estimated cost of the remaining paths, then the current path is chosen. This process is repeated until the goal is reached.

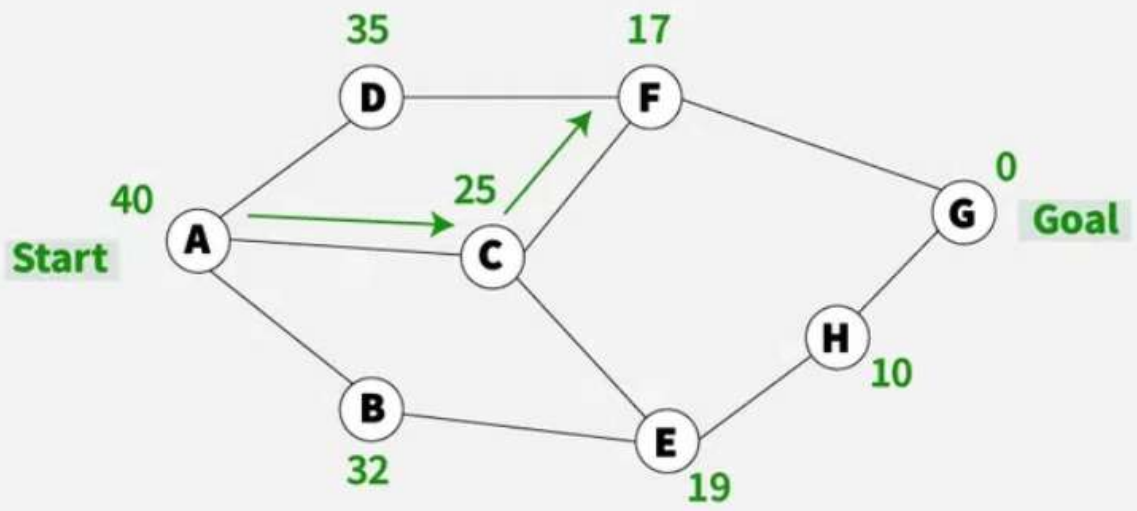
The values in red color represent the heuristic value of reaching the goal node G from current node



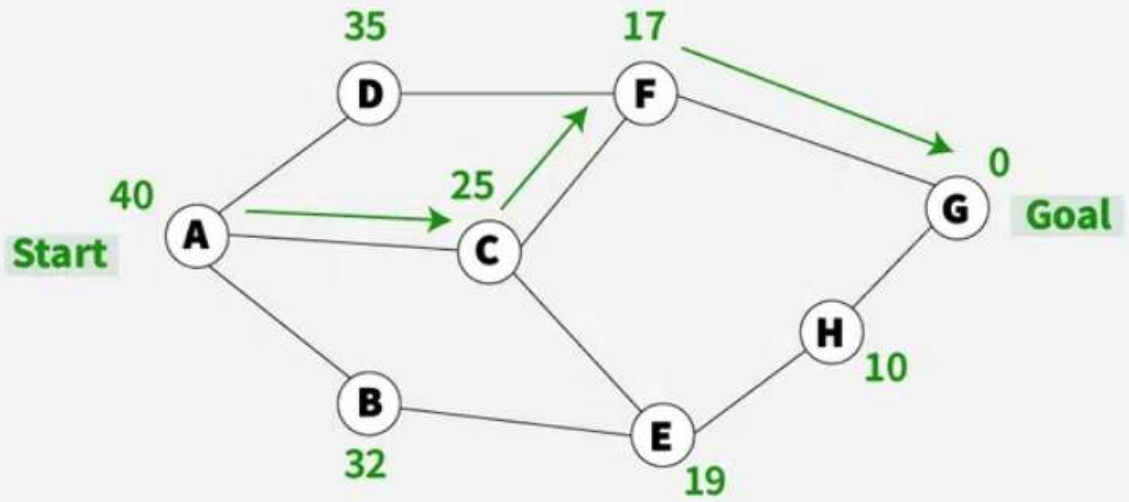
Best-First Search algorithm



Best-First Search algorithm



Best-First Search algorithm



Best-First Search algorithm

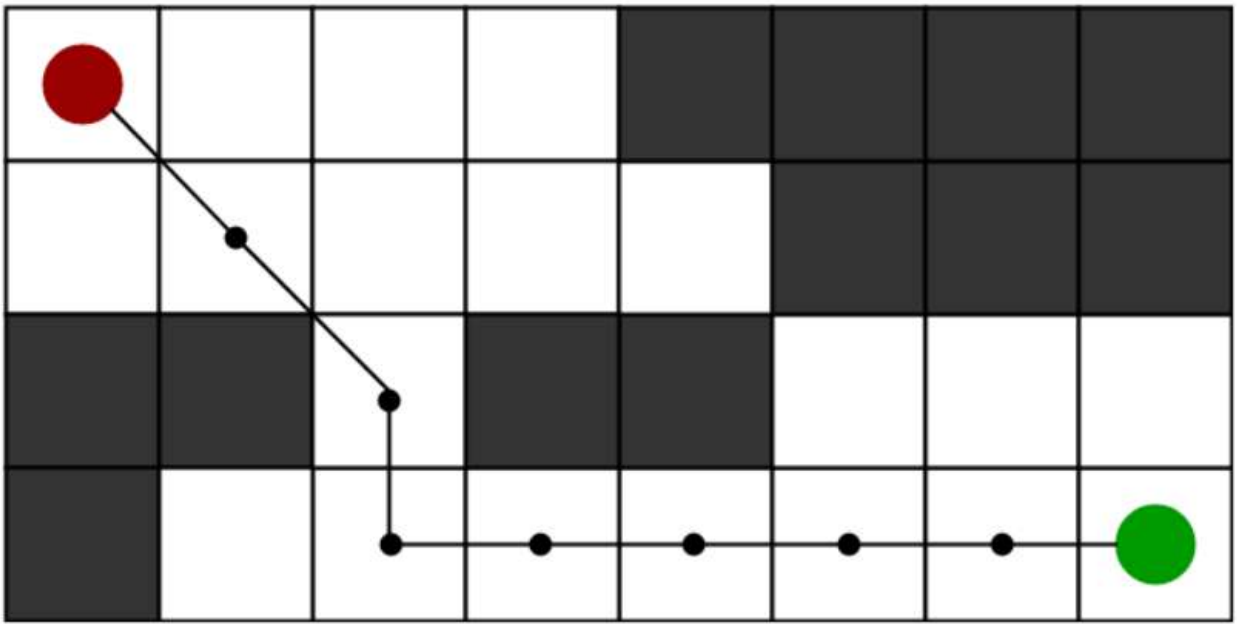
Applications of Greedy Best-First Search:

- **Pathfinding:** Greedy Best-First Search is used to find the shortest path between two points in a graph. It is used in many applications such as video games, robotics, and navigation systems.
- **Machine Learning:** Greedy Best-First Search can be used in machine learning algorithms to find the most promising path through a search space.
- **Optimization:** Greedy Best-First Search can be used to optimize the parameters of a system in order to achieve the desired result.
- **Game AI:** Greedy Best-First Search can be used in game AI to evaluate potential moves and chose the best one.
- **Navigation:** Greedy Best-First Search can be use to navigate to find the shortest path between two locations.
- **Natural Language Processing:** Greedy Best-First Search can be use in natural language processing tasks such as language translation or speech recognisation to generate the most likely sequence of words.

Motivation

To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.

We can consider a 2D Grid having several obstacles and we start from a source cell (colored red below) to reach towards a goal cell (colored green below)



🧠 What is a **Heuristic Function** in AI?

A **heuristic function**, usually written as `h(n)`, is a way for a search algorithm to **"guess" how close** a given state or node `n` is to the goal.

It's like giving the AI a **sense of direction** — a way to estimate how far it still has to go, without actually computing the full path yet.

📌 Why Do We Need It?

In **informed search strategies**, like A* and Greedy Best-First Search, the AI doesn't explore blindly. Instead, it uses `h(n)` to **prioritize smarter choices** and speed up the search.

Without a heuristic, the search might:

- Explore unnecessary paths
- Take more time
- Use more memory

With a good heuristic, it:

- Finds paths faster
- Reduces unnecessary exploration

🚪 Structure of a Heuristic Function

- Input: A **node** (or state) `n`
- Output: A **number** estimating the cost (or steps) to reach the **goal** from `n`

So:

```pgsql
h(n) = estimated cost to reach the goal from node n
```

> 💡 Lower `h(n)` = more promising path

✅ Example: Pathfinding (Map Navigation)

Imagine you're trying to go from **City A to City B**.

- Let's say you are currently at **City C**.
- A possible heuristic: **straight-line distance** from City C to City B.

This doesn't tell you the actual road length (because roads may be curved or blocked), but it's a **good guess** for how far you are.

🔍 What is an **On-line Search Agent?**

An **on-line search agent** is a smart program or robot that **does not know everything about the world at the start**. It has to **explore the environment and learn while doing its job**.

✅ Features:

- It **starts with little or no information** about the surroundings.
- It **takes actions** and **learns from the results**.
- It **thinks and acts step by step**, instead of planning everything in advance.
- It's used when the agent needs to make decisions **right away**, even if it doesn't know everything yet.

🧹 Example:

Think of a robot vacuum in a new house. It doesn't have a map. It **moves around**, bumps into walls or furniture, and **remembers where things are**. Over time, it **builds its own map** while cleaning.

🧠 **Example:**

Think about a robot that's sent into a maze it has never seen before. It doesn't have a map. It has to:

- Move forward,
- Check what's around,
- Decide the next step based on what it just saw,
- Keep doing this until it finds the goal.

So, the robot is **searching and acting at the same time** — this is **on-line searching**.

🌍 What is an **Unknown Environment?**

An **unknown environment** is a place or situation where:

- The agent **does not know what to expect**,
- There might be **hidden obstacles**, or **unseen rules**,
- The agent must **explore and discover** things to figure out how to behave.

🧠 **Example:**

Imagine you are walking in a dark room for the first time:

- You don't know where the furniture is.
- You slowly move, touch things, and try to find the door.
- You learn the layout as you move.

That's an **unknown environment** — because you didn't know it beforehand.