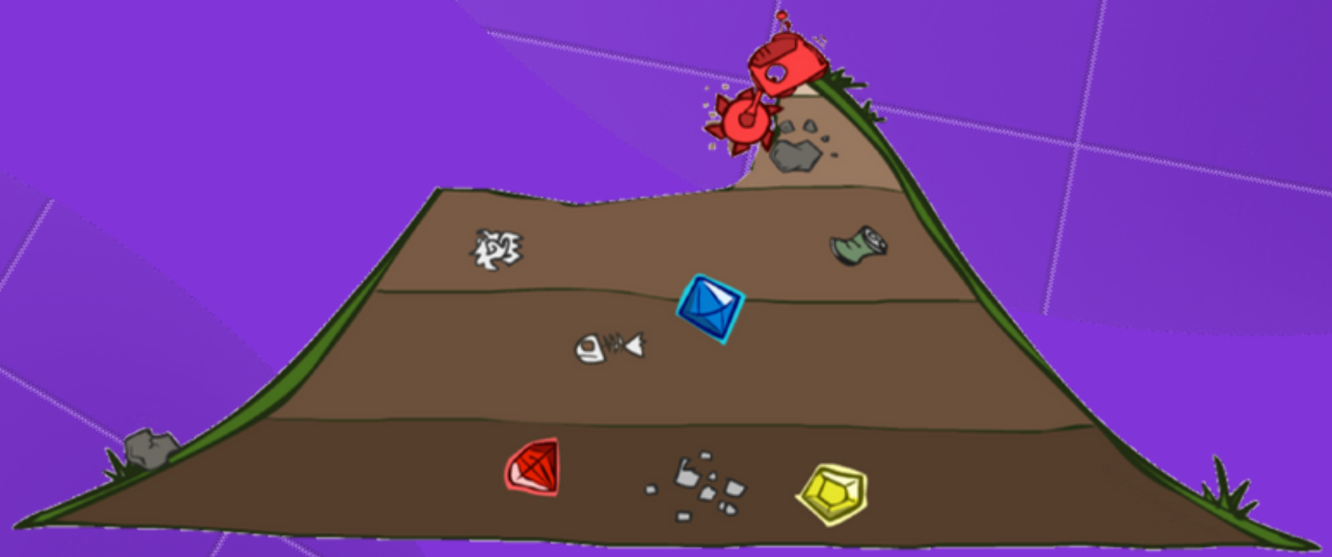
Graph Search
[Uniform Cost Search]

Uniform Cost Search



Introduction to Graph Search Algorithms:

Graph search algorithms are fundamental for solving problems involving nodes and connections. They help navigate graphs efficiently, finding paths and solutions.

Importance of Uniform Cost Search (UCS):

Optimal Paths: UCS ensures finding the shortest path in a weighted graph, considering edge costs.

Dynamic Programming: It builds optimal solutions progressively by breaking them into smaller subproblems.

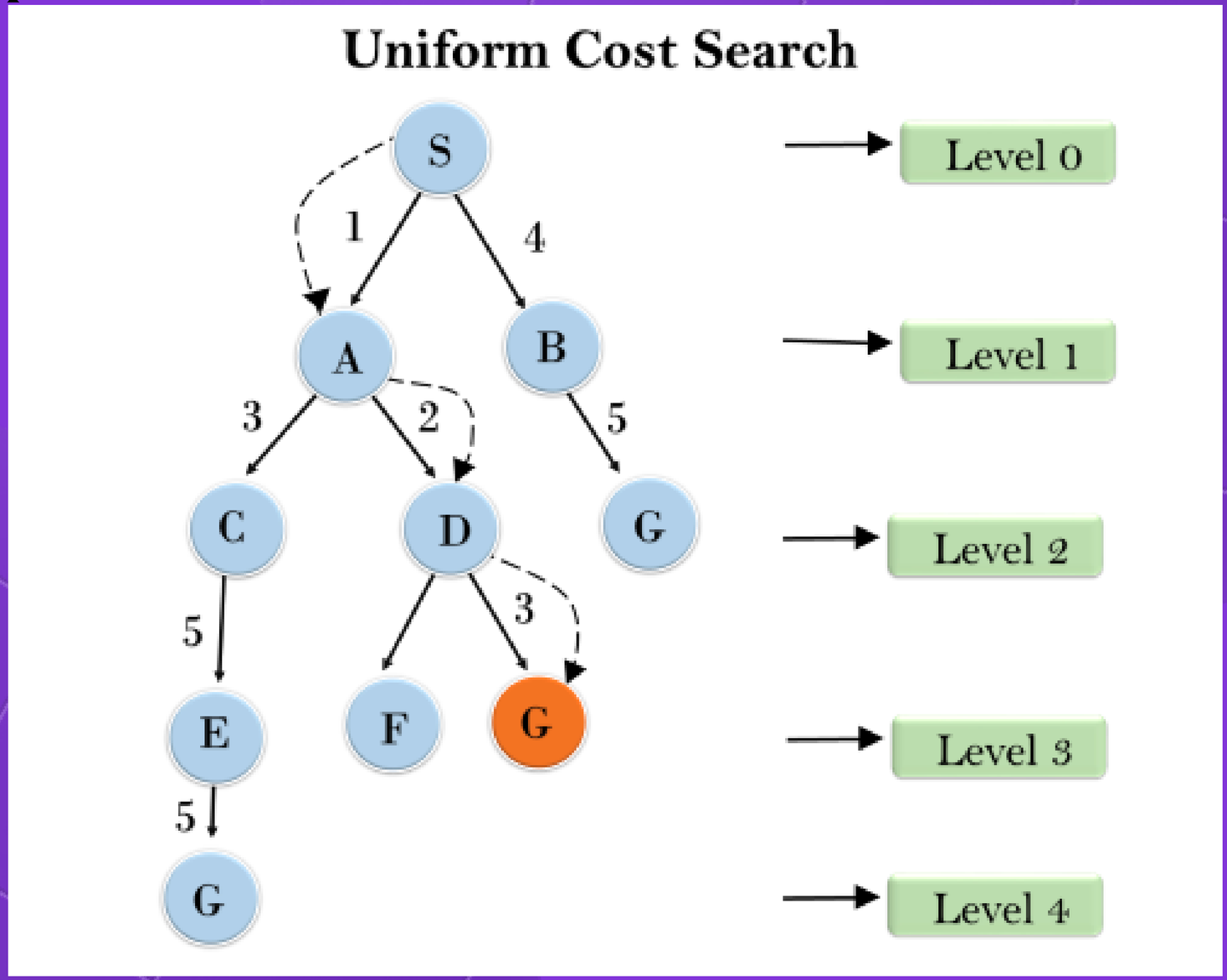
Versatility: Suitable for graphs with diverse edge costs, making it adaptable to various scenarios.

Efficient Exploration: Avoids early expansion of non-optimal paths, improving efficiency.

Navigation: Forms the basis of route planning in navigation systems, accounting for real-time conditions.

Resource Management: Useful in robotics, resource allocation, and decision-making with cost constraints.

Graph Representation



Modeling Real-World Problems with Graphs:

Graphs are versatile tools to model and solve real-world problems by representing relationships, connections, and interactions between entities. They provide a visual and conceptual framework for understanding complex systems and optimizing various scenarios.

Nodes and Edges:

Nodes: Nodes, also known as vertices, are individual entities or points in a graph. Each node can represent a person, location, object, or any relevant entity in the problem domain.

Edges: Edges are connections or links between nodes. They represent relationships, interactions, or pathways between the entities they connect. Edges can be directed (one-way) or undirected (two-way).

Weights or Costs on Edges:

Example: In a transportation network, edges between nodes (cities) can have weights representing distances between them. For a road network, the weights can be actual distances in kilometers, while for a flight network, they might represent flight durations.

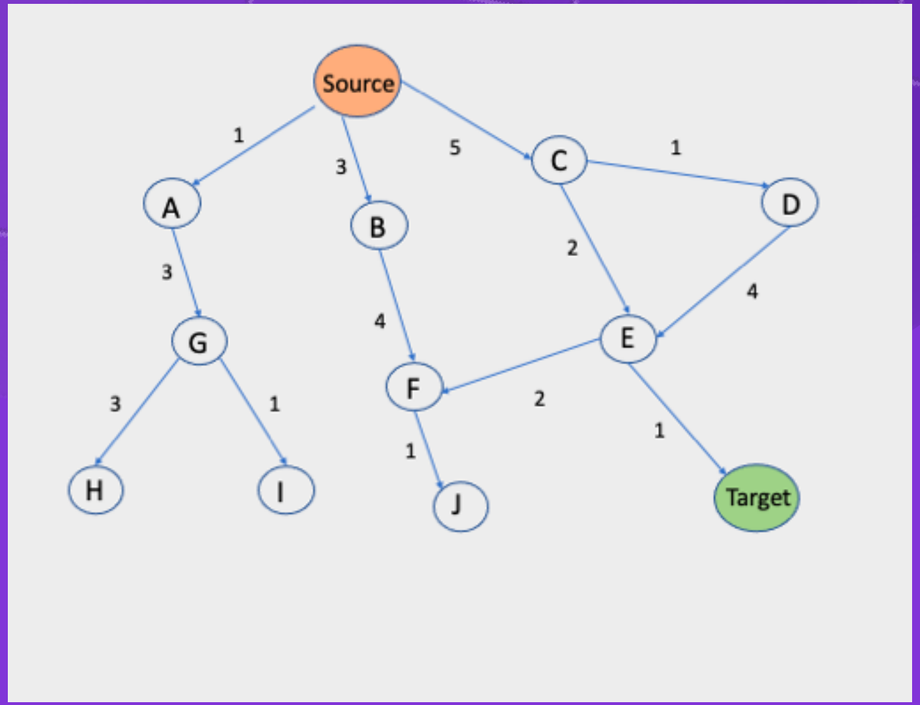
Problem Statement

Need for Finding Optimal Paths Based on Costs (Source and Target Perspective):

Optimal paths based on costs are essential to:

Source: Efficiently utilize resources, minimize delays, and manage budgets.

Target: Ensure quick arrivals, enhance safety, and maximize profits in competitive scenarios.



Uniform Cost Search Overview

Best-First Search Algorithm: UCS is a graph traversal algorithm that operates as a best-first search. It prioritizes exploring paths that have the lowest cumulative cost so far.

Exploring Lowest Cost Paths: Unlike other algorithms, UCS focuses on exploring paths with the smallest accumulated cost. It considers the cost associated with each edge between nodes.

Dynamic Programming: UCS follows the principle of dynamic programming. It gradually constructs the optimal solution by considering smaller subproblems and their solutions.

Edge Weight Utilization: UCS utilizes the weights or costs associated with edges to calculate the cumulative cost of traversing from the start node to each subsequent node.

Algorithm Steps

1. Initialization: Start with a priority queue containing the starting node and its cost, which is initially 0.
2. Main Loop: While the priority queue is not empty, continue the following steps:
3. Node Selection
4. Goal Check:
5. Expansion:
6. Neighbor Addition:
7. Cost Update:
8. Solution Absence Check:
9. Termination:

- Uniform Cost Search ensures that the lowest-cost path to each node is considered first, gradually expanding the search space in a way that leads to an optimal solution.

- This process continues until the goal is reached or all possible paths have been explored.

Priority Queue and Cost Update

Concept of a Priority Queue:

A priority queue is like a line where each person has a number representing their importance. The person with the highest (or lowest) number is served first. It's used in algorithms to handle things in order of importance.

Illustration of Cost Update:

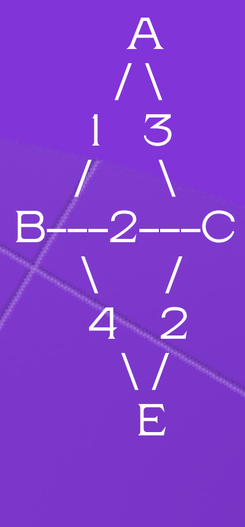
1. Imagine you're finding the cheapest way to travel between cities on a map:

- Start with the first city and cost 0.
- Pick the city with the lowest cost (cheapest path).
- Check its neighbors and update their costs if a cheaper path is found.
- Keep doing this until you reach your destination or all paths are explored.
- So, the priority queue helps you focus on cheaper paths as you move along, making sure you find the most cost-effective way.

Example Problem

Example Problem: Finding Shortest Path

Imagine you want to travel from city A to city E:



Step-by-Step Uniform Cost Search:

Start at A with cost 0.

Step 1: Go to B (Cost: 1), add C (Cost: 3).

Step 2: Go to B (Cost: 1), add E (Cost: 5).

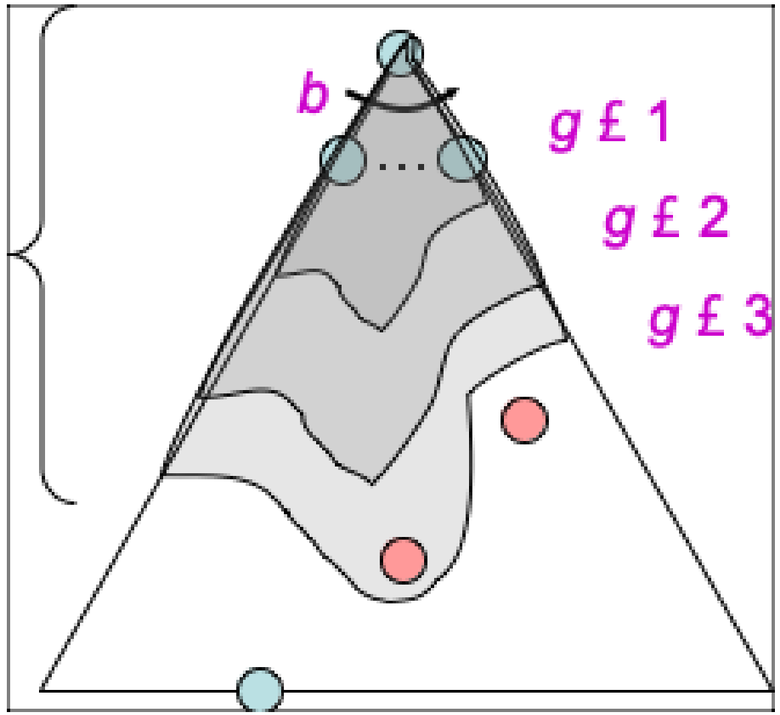
Step 3: Go to C (Cost: 3), update E's cost (Cost: 5).

Step 4: Go to E (Cost: 5).

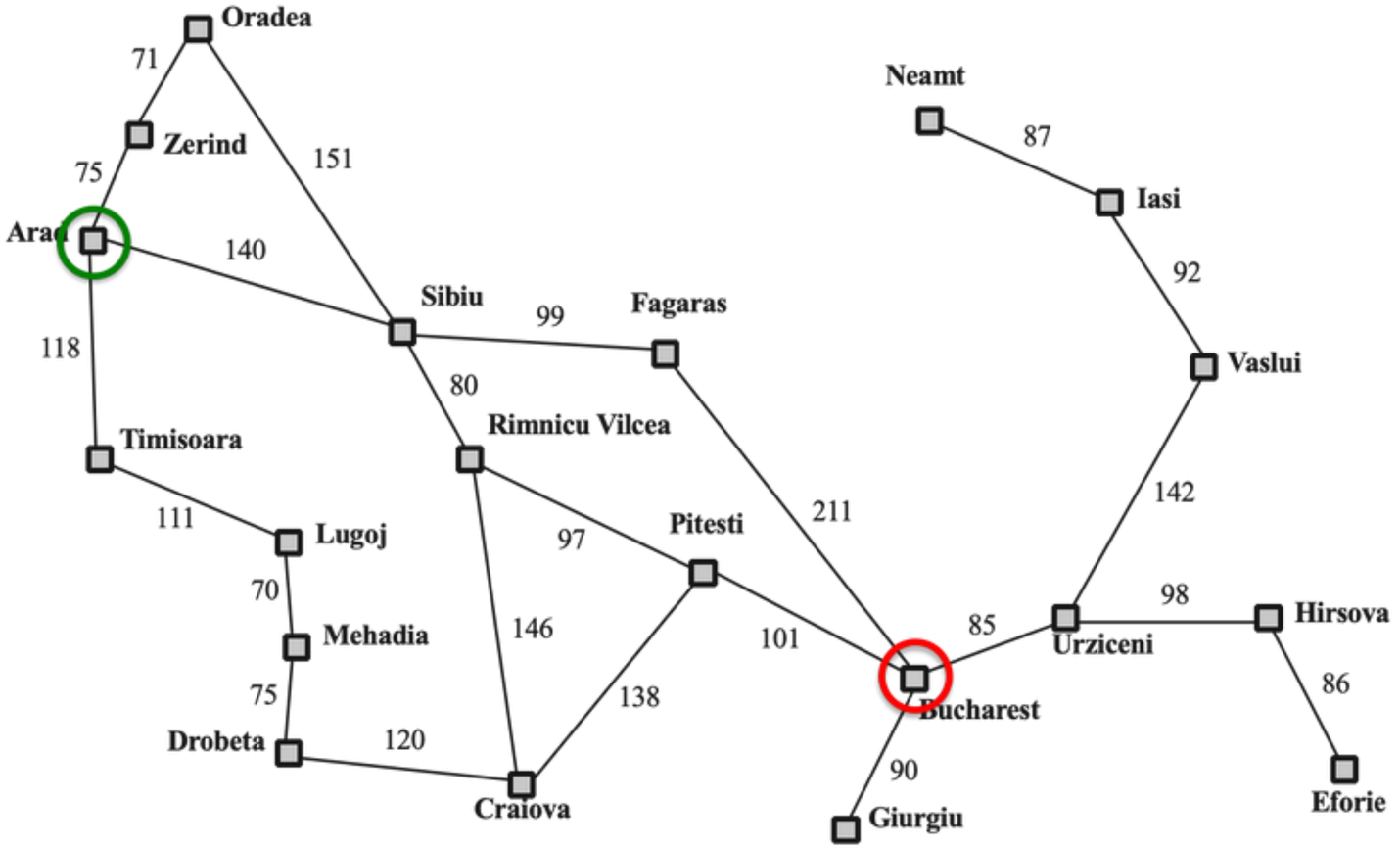
Chosen Path: A -> B -> E (Cost: 5).

Uniform Cost Search picks cheaper paths and found the shortest way from A to E.

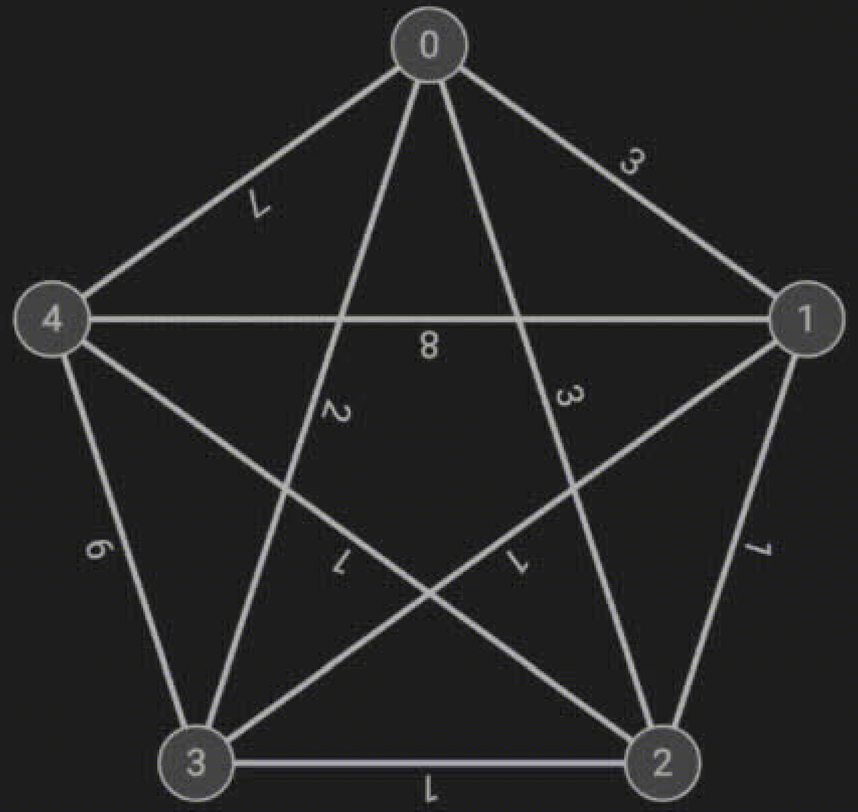
- Uniform Cost Search (UCS) Properties
- What nodes does UCS expand?
- Expands all nodes with cost less than cheapest solution!
- If that solution costs C* and arcs cost at least e, then the
- “effective depth” is roughly C*/e C*/e “tiers”
- Takes time O(bC*/e) (exponential in effective depth)
- How much space does the frontier take?
- Has roughly the last tier, so O(bC*/e)
- Is it complete?
- Assuming C* is finite and e> o, yes!



Example: route-finding in Romania



GraphTracer



Array1DTracer

Search	Frontier	Completeness	Optimality	Time	Space
DFS (Depth-First)	stack	tree search - no (cycle) graph search < yes (finite) / no (infinite)	no	$O(b^m)$	$O(bm)$
BFS (Breadth-First)	queue	yes	no (except when all edge costs same)	$O(b^s)$	$O(b^s)$
Iterative Deepening (BFS result w/ modified DFS algo)	stack (same as DFS)	yes (same as BFS)	no (same as BFS)	$O(b^s)$ (same as BFS)	$O(bs)$ (same as DFS but w/ shortest solution length)
UCS (Uniform Cost)	heap-based PQ (backward cost)	yes (assuming positive edge costs and $\epsilon > 0$)	yes (assuming positive edge costs and $\epsilon > 0$)	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$

b = branching factor (assume finite)

m = max depth of search tree

s = smallest depth of solution (assume finite)

C^* = cost of optimal solution (assume finite)

ϵ = minimum cost between 2 nodes

Uniform Cost Search:

1. Finds optimal paths based on lowest cumulative cost.
2. Prioritizes paths with lower costs.
3. Best for scenarios needing optimal paths, regardless of edge costs.

Breadth-First Search:

1. Explores all neighbors before deeper levels.
2. Works well for finding shortest unweighted paths.
3. Best for graphs without edge costs, like grids.

Dijkstra's Algorithm:

1. Finds shortest paths from a single source to all nodes.
2. Considers cumulative costs, but updates costs based on known paths.
3. Useful for finding shortest paths in weighted graphs with non-negative costs.

Different Approaches:

UCS finds paths based on lowest cost, BFS explores all levels, and Dijkstra's considers cumulative costs while updating based on known paths.

Applications:

UCS is for optimal paths, BFS is for unweighted paths, and Dijkstra's is for weighted paths. Each suits different situations, like navigation, grid-solving, or road networks.

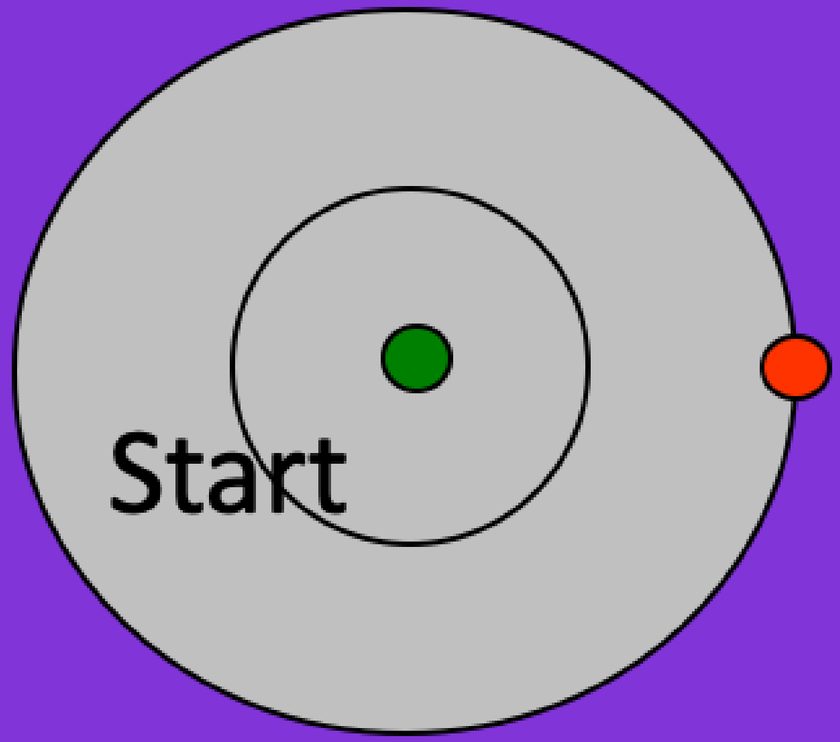
Comparison

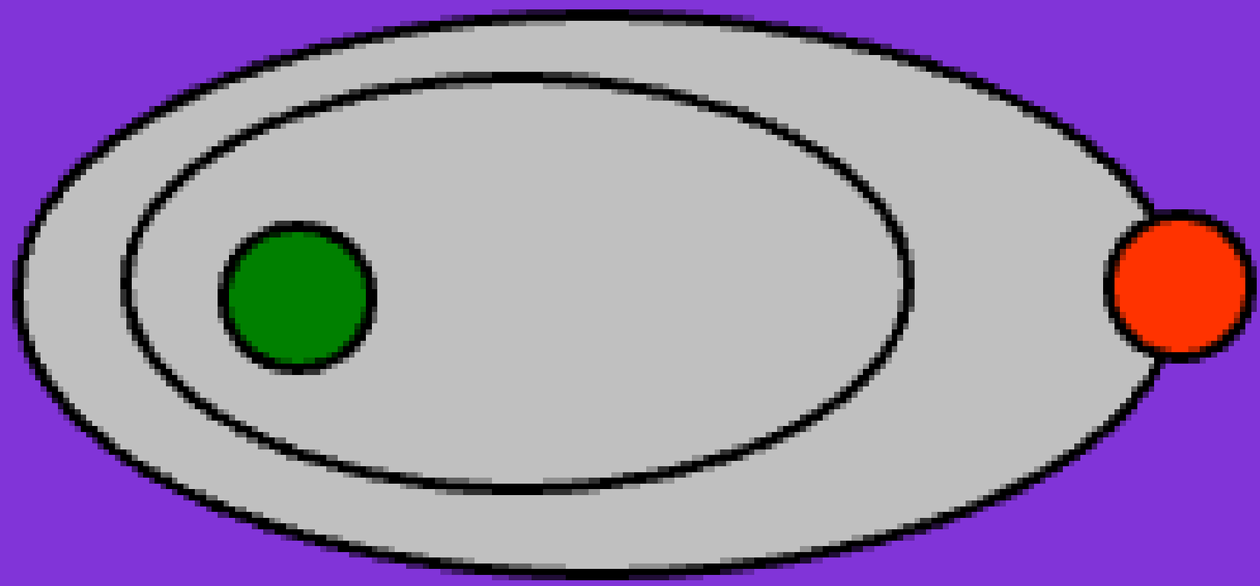
Greedy (h)

Uniform Cost (g)

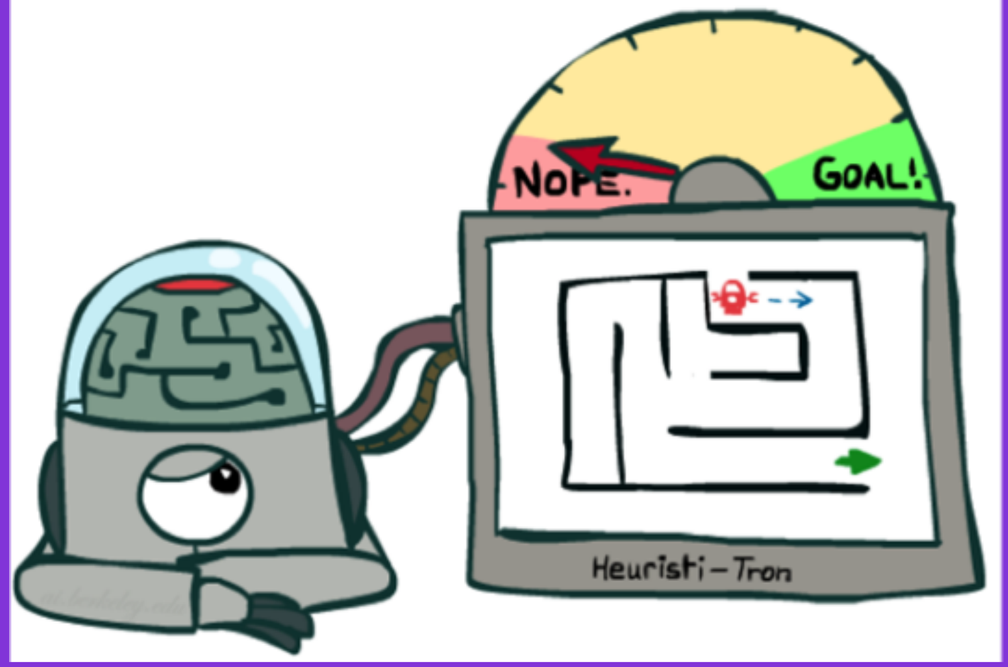
A* (g+h)

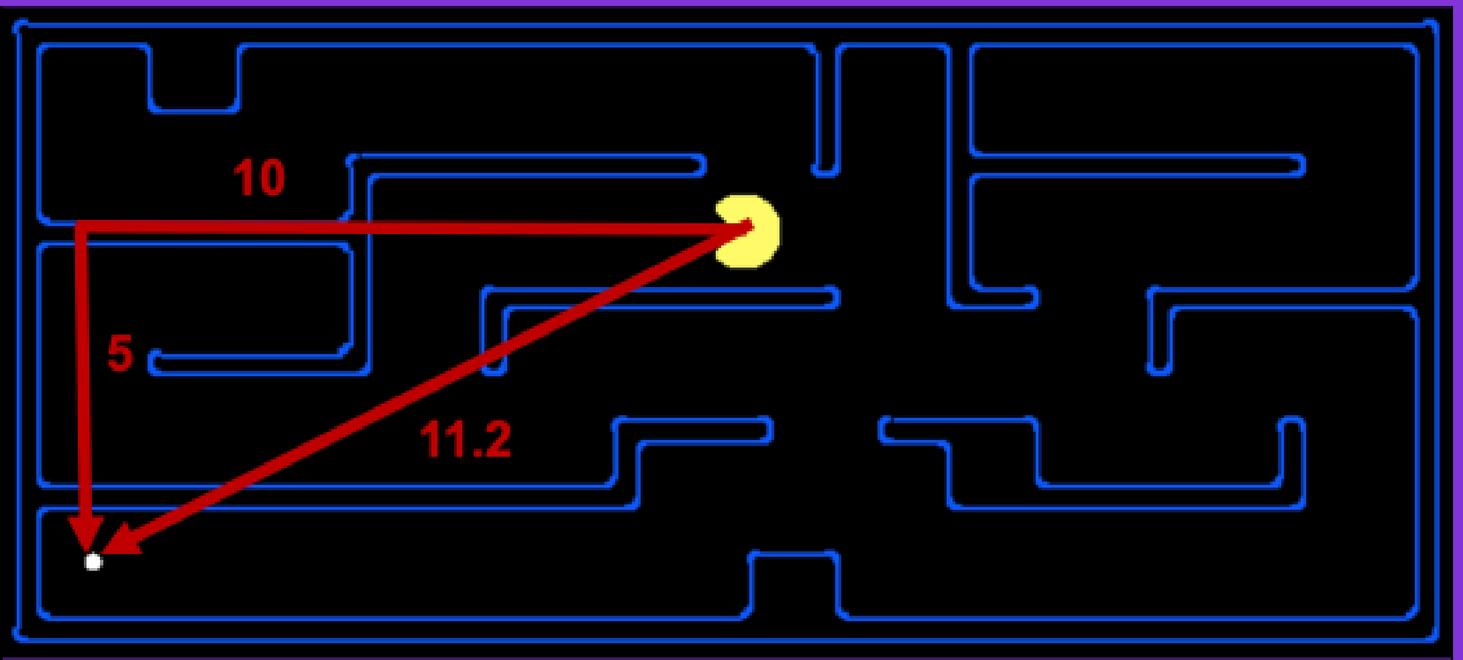
What we would like to have happen
Guide search towards the goal instead of all over the place
Start Goal Goal
Informed Uninformed
Search Heuristics

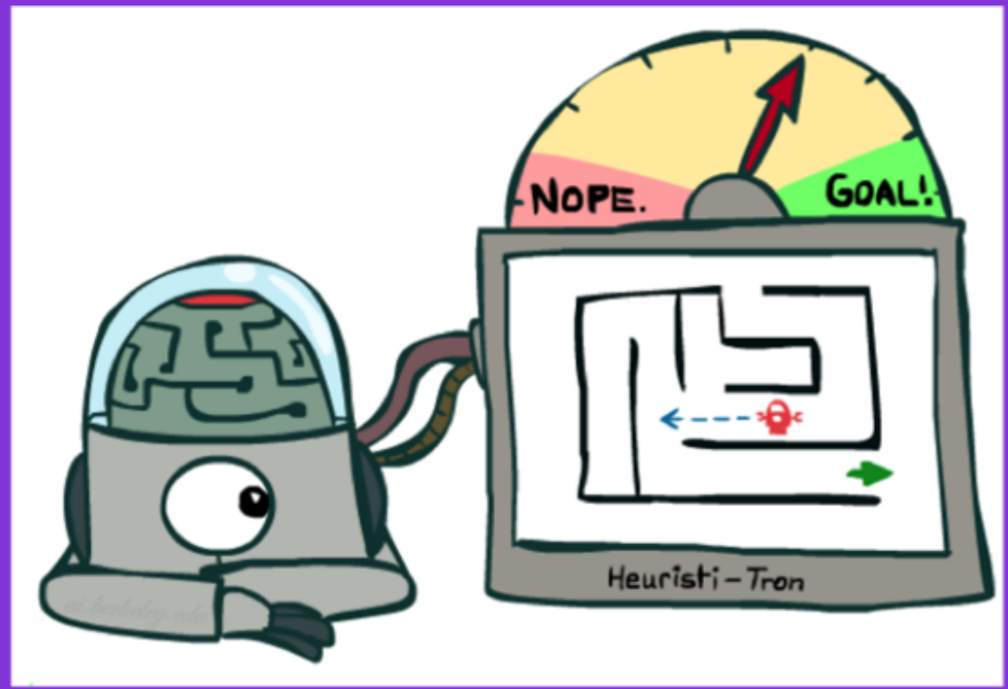




- A heuristic is:
- A function that estimates how close a state is to a goal
- Designed for a particular search problem Pathing?
- Examples:
  Manhattan distance, Euclidean distance for pathing







Applications

- Real-World Applications of Uniform Cost Search:
- Navigation Systems: Helps GPS find quickest routes.
- Resource Allocation: Distributes resources efficiently.
- AI and Games: Characters navigate smartly in games.
- Transportation: Optimizes delivery and routes.
- Robotics: Guides robots through obstacles.
- Network Routing: Efficient data transfer paths.
- Emergency Response: Quick routes for responders.
- Uniform Cost Search makes things efficient and smart in various areas.

Conclusion

Uniform Cost Search is like a treasure map. It helps us find the most efficient paths in various situations. Whether we're navigating through traffic, allocating resources, developing games, or optimizing routes, UCS ensures that we make smart decisions by considering costs. It's a powerful tool that leads us to the best solutions in a wide range of optimization problems.

Thank You