Manchester Metropolitan University

Artificial Intelligence Principles

6G7V0011 - 1CWK100

Dr. Peng Wang

Email: p.wang@mmu.ac.uk

Department of Computing and Mathematics

Tuesday, Nov. 5th, 2024

Outline

Outline

Review

Manchester Metropolitan University

- Artificial Intelligence and SOTA techniques
- Agents and Rational agents
- Four types of agent programs
  1. Simple reflex agent
  2. Model-based agent
  3. Goal-based agent
  4. Utility-based agent
- Graph search, Tree (tree-like) search, and DFS as an example
- Breadth First Search

Please go to Page 21 for Week 6's material. I put this all together as a summary of Uninformed Search.

Agent Programs - Goal-based Agent

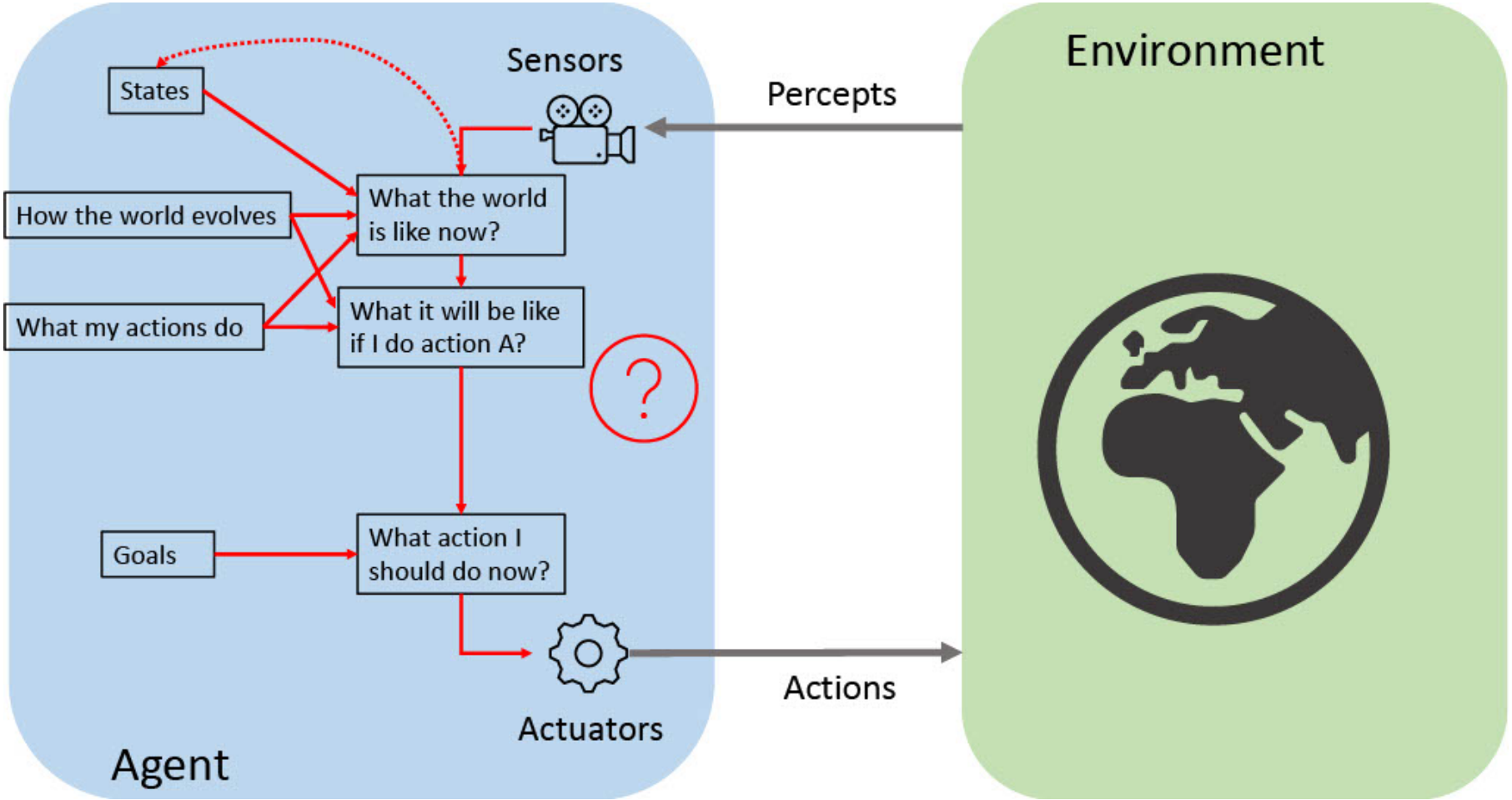


Figure 1: Goal-based agent

Agent Programs - Utility-based Agent

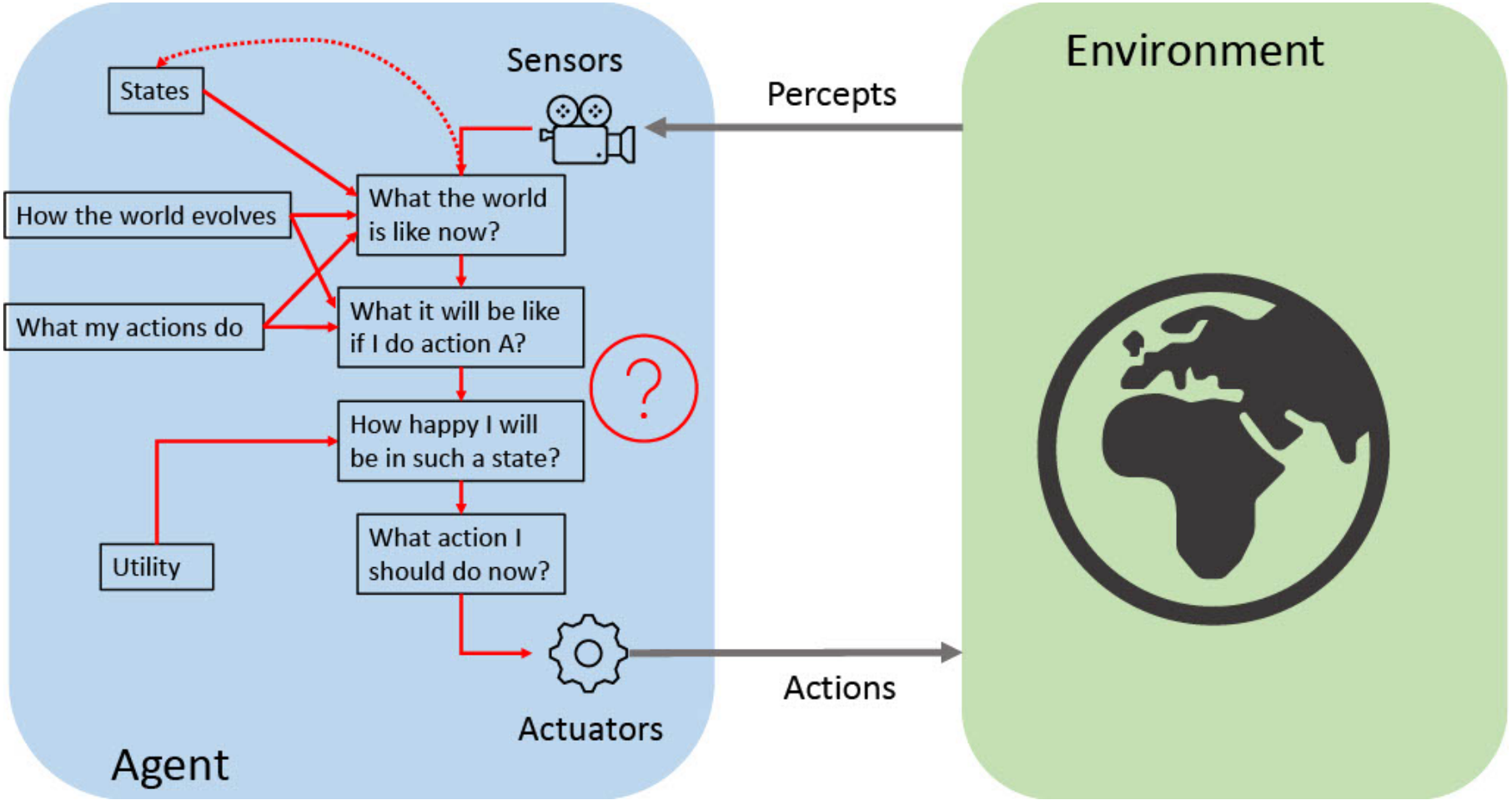


Figure 2: Utility-based agent

Search

Manchester Metropolitan University

In goal-based and utility-based agents, when the goal(s) change, the agent needs to **search** for a new solution.

Search

- **Search** is the computational process undertaken by an agent when no immediately obvious actions are available.
- *Informed search*: Agent can **estimate** the cost from current state to goal(s).
- *Uninformed search*: No such functionality.

*Why **estimate**?*

The cost is from prior knowledge/experience/intuition, etc., but not the truth/fact. You (agent) can only find out by doing (have done) it.

Search happens before action! Alternatively, consider search as a Planning process!

Formulating a Search Problem

Manchester Metropolitan University

1. A set of possible **states** that the **world (agent + environment)** can be in. All the possible states together are called **state space**.
2. The **initial state** (start state) that the agent starts in.
3. A set of *one* or *more* **goal states**.
4. The **actions** that are available to the agent.
5. A **transition model** describes what each action does.
6. An **action cost function**, denoted by Action-Cost(s, a, s'), that gives the numeric cost of applying action a in state s to reach state s'.
7. A sequence of actions forms a **path**.
8. A **solution** is a path from the initial state to a goal state.
9. An **optimal solution** has the lowest **path cost** among all solutions.
10. The state space can be represented as a **graph** in which the vertices are states and the directed edges between them are actions.

State Space Graphs vs. Search Trees

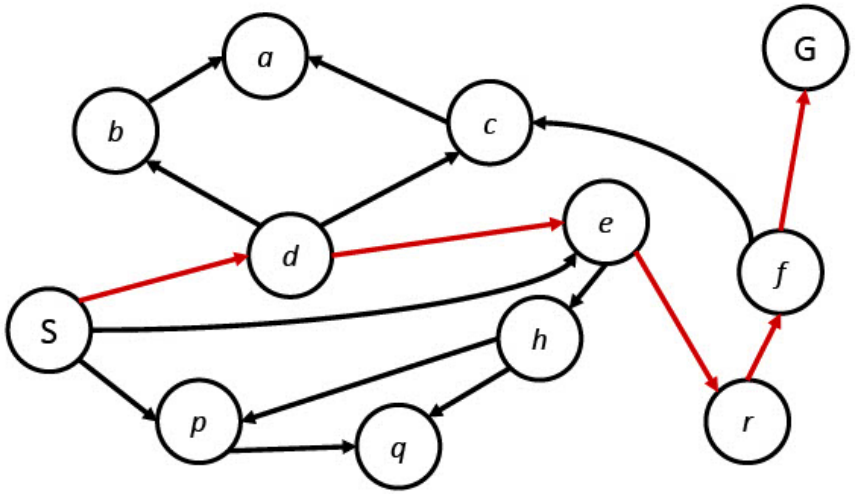


Figure 3: A state space graph

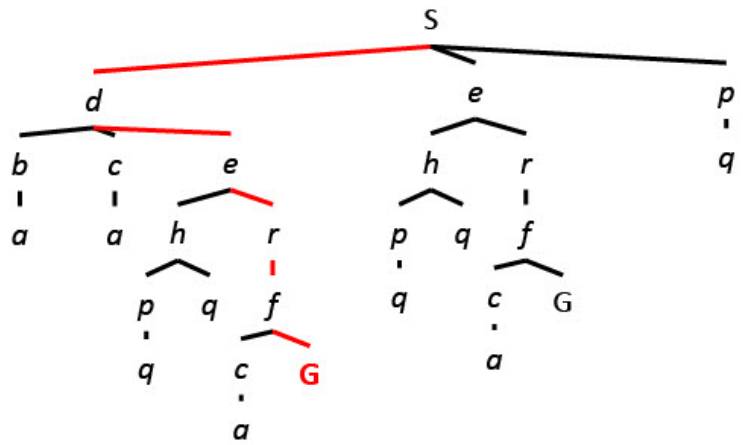


Figure 4: A search tree

- **State space graph**: A mathematical representation of a search problem
- **Search tree**: A 'what if' tree of plans and their outcomes
- We construct both on demand and we construct as little as possible.

State Space Graphs vs. Search Trees

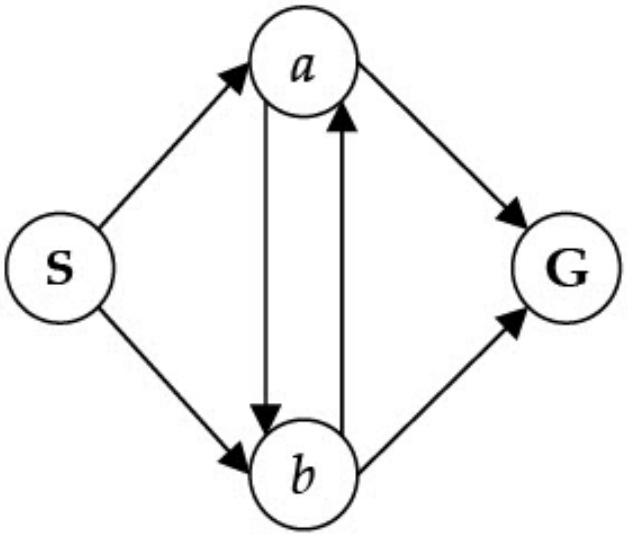


Figure 5: A four-state state space graph

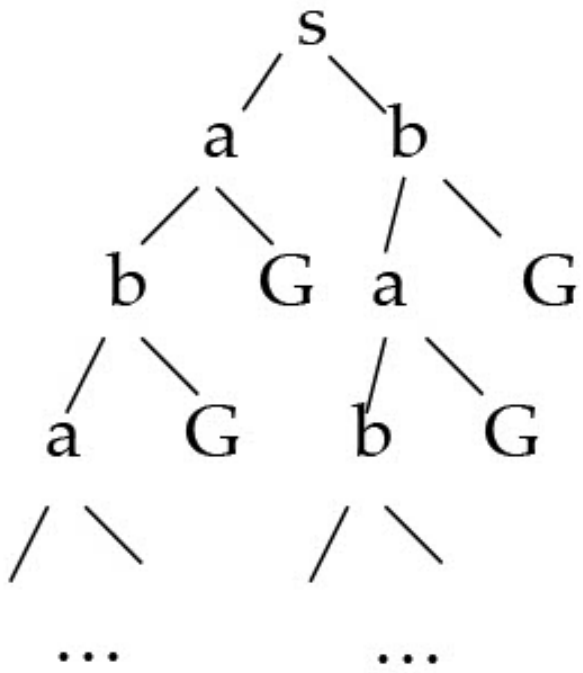
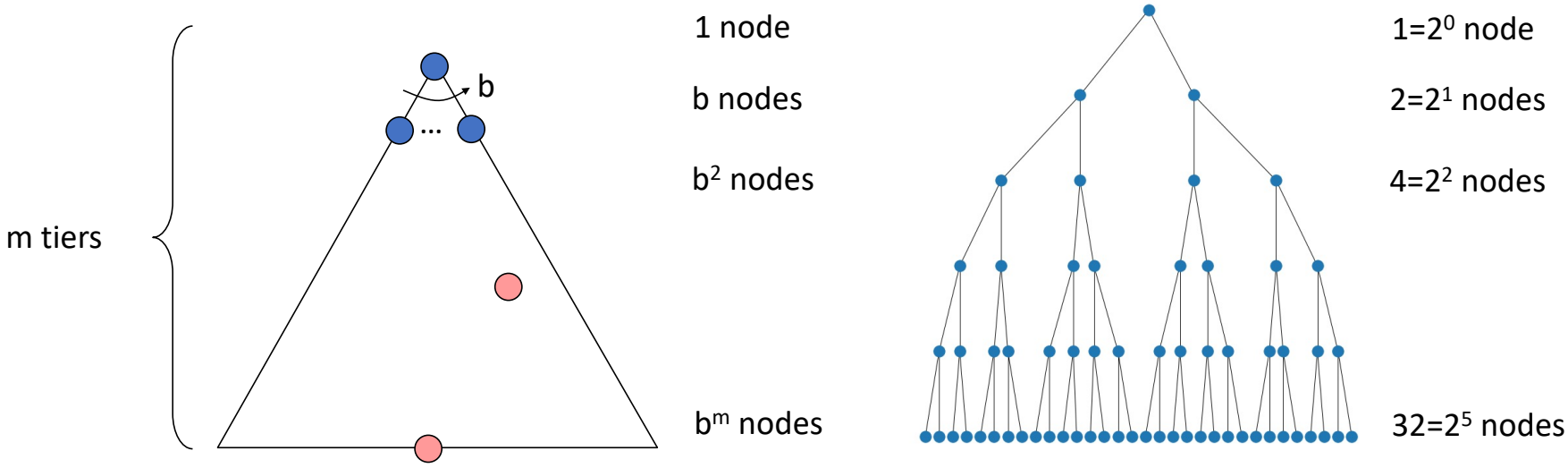


Figure 6: A infinite depth search tree

Note: 4-state graph, a infinite depth tree with cyclic state space, **algorithm design to detect cycle**

Search Algorithm Properties



- **Complete**: Guaranteed to find a solution if one exists?
- **Optimal**: Guaranteed to find the least cost path?
- **Time** complexity?
- **Space** complexity?

- Key factors of a search tree
  1. b is the branching factor
  2. m is the maximum depth
  3. solutions at various depths

Node number: $b^0 + b^1 + b^2 + \cdots + b^m = \sum_{i=0}^{m} b^i \backsim \mathcal{O}(b^m)$

Breath First Search (BFS) - graph I

Manchester Metropolitan University

Algorithm 1: Pseudocode of Breath First Search

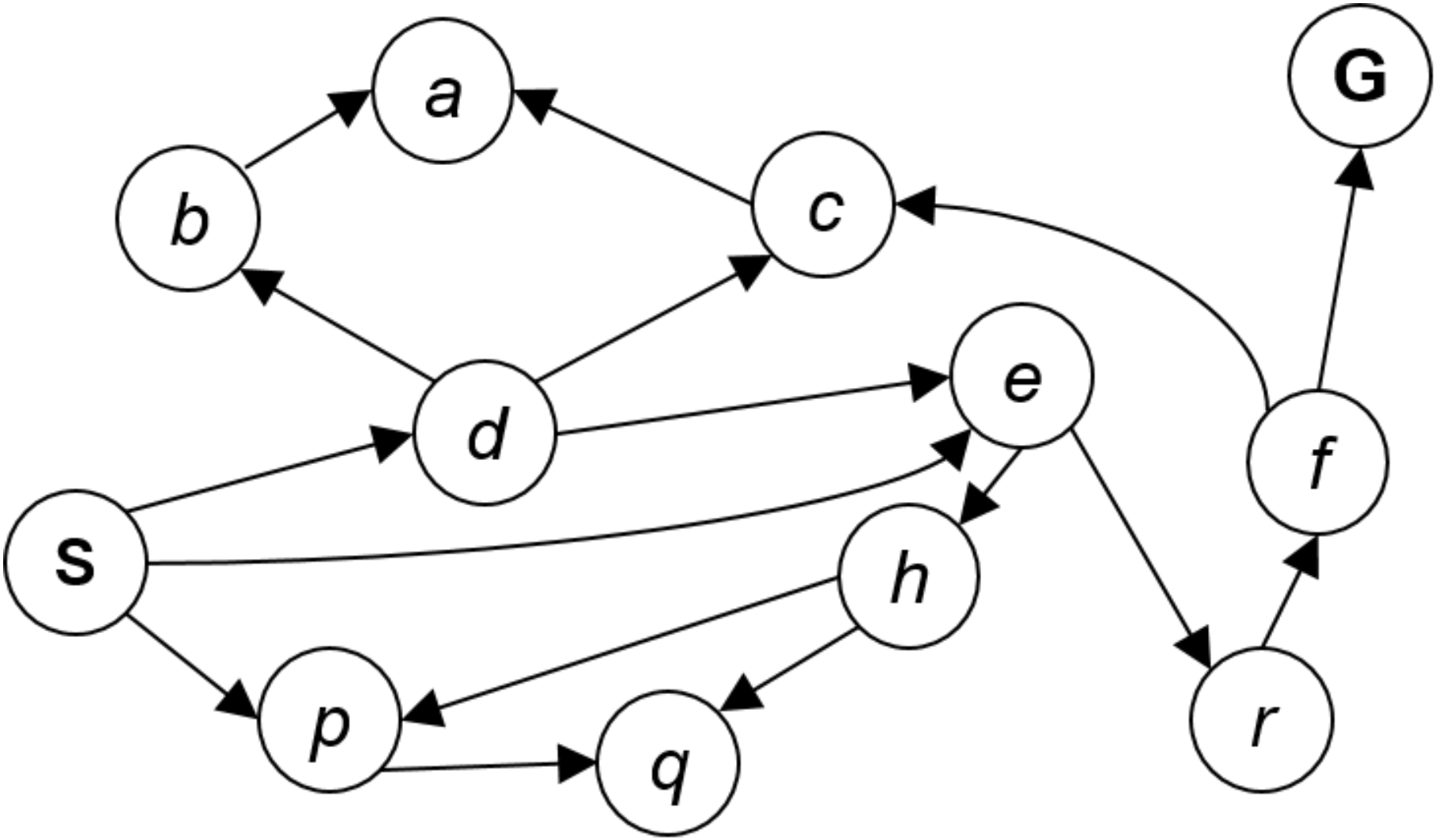
Input: Initial state **s**, goal state **g**
Output: A *node* helps to retrieve a solution (path) $\mathcal{P}$, or failure

```
1:  node ← with s as state
2:  if s == g then
3:      return node
4:  end if
5:  O ← node, an FIFO queue
6:  C ← ∅
7:  while O != ∅ do
8:      parent ← the first node in O
9:      C ← parent.state
10:     for child in successor (of the current parent) do
11:         v ← child.state
12:         if v is not in C and child is not in O then
```

Breath First Search (BFS) - graph II

```
13:          if v == g then
14:             return child
15:          end if
16:          add child to the end of 𝒪
17:       end if
18:    end for
19: end while
20: return failure
```

Breadth First Search



Breadth First Search Properties

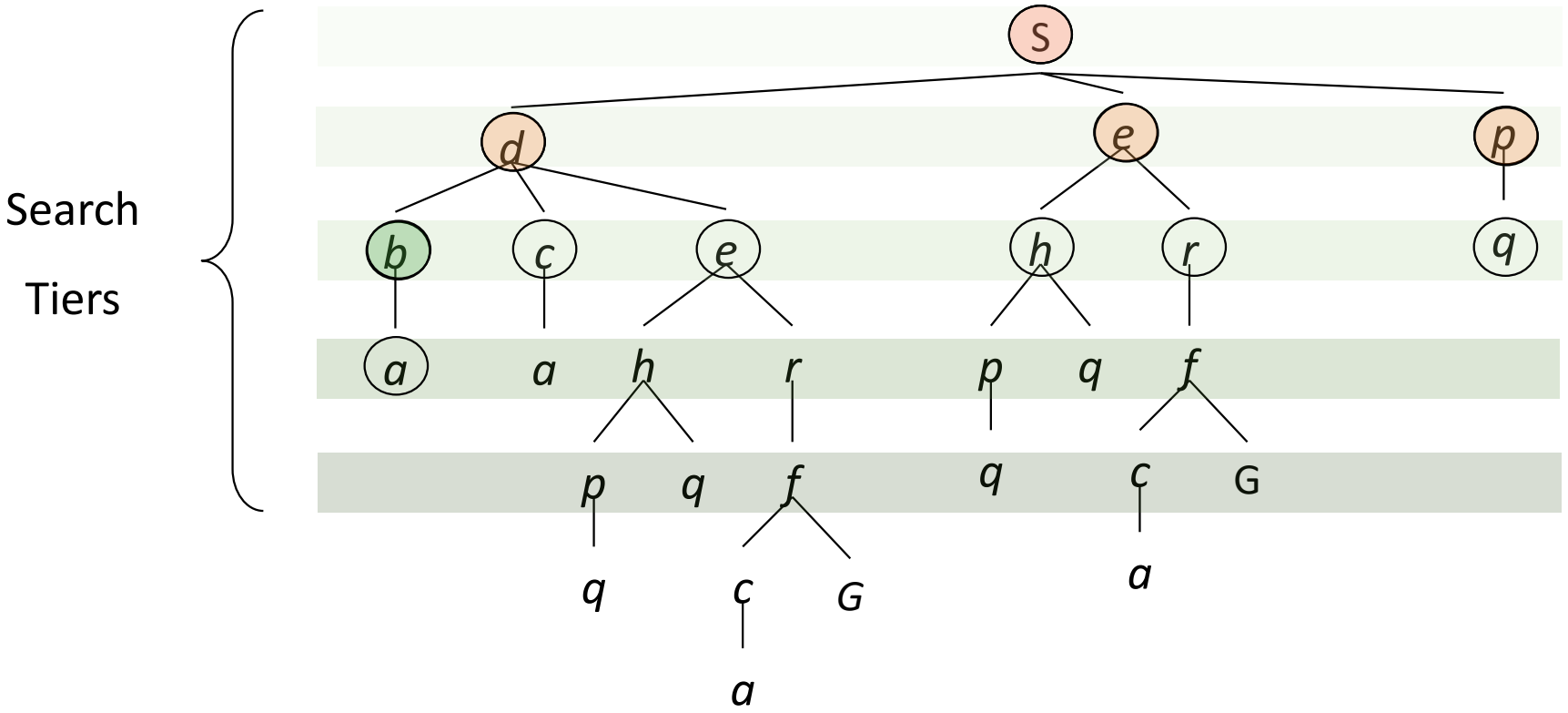
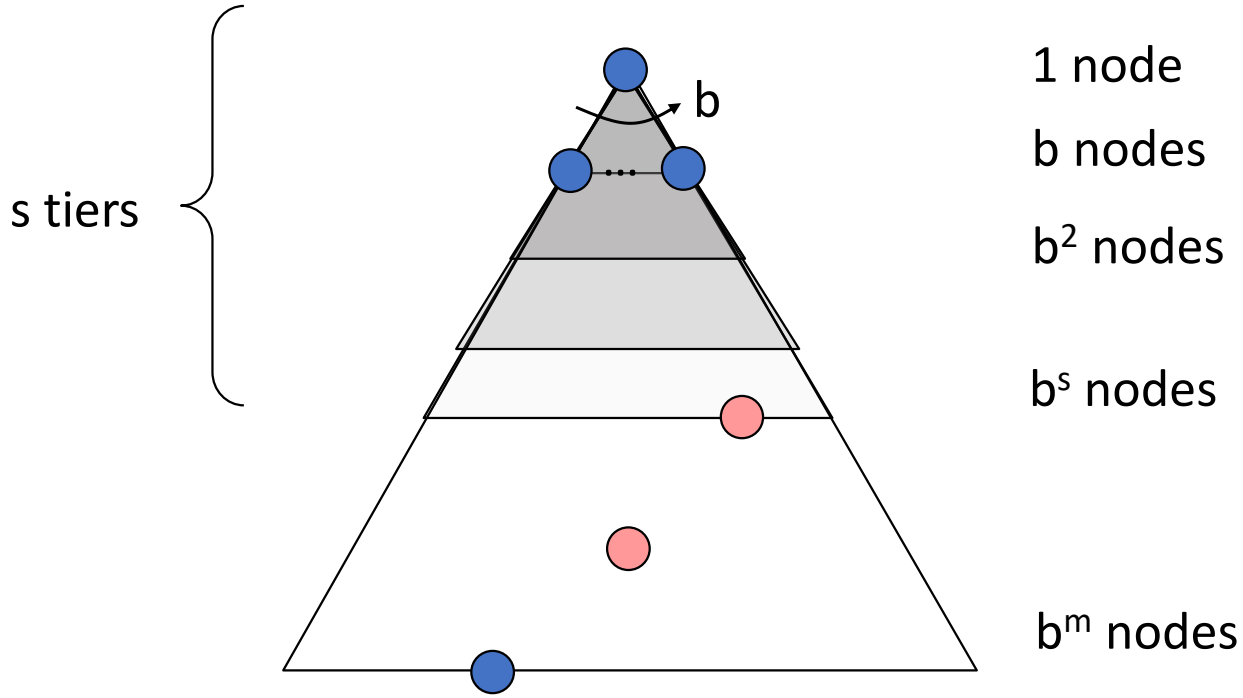


Figure 7: An example of Breadth First Search

Breadth First Search



When search reached the shallowest solution at depth s

- Has processed all nodes above depth s
- Time complexity: $\mathcal{O}(b^s)$
- Space complexity (frontier): $\mathcal{O}(b^s)$
- Complete: s must be finite if a solution exists
- Optimal: When costs of edges are equal c.

Depth First Search (DFS) - graph I

Manchester Metropolitan University

Algorithm 2: Pseudocode of Depth First Search

Input: Initial state **s**, goal state **g**.
Output: A *node* helps to retrieve a solution (path) $\mathcal{P}$, or failure

```
1:  node ← with s as state
2:  O ← node, an LIFO queue
3:  C ← ∅
4:  while O != ∅ do
5:      parent ← the last node in O
6:      v ← parent.state
7:      if v == g then
8:          return parent
9:      end if
10:     C ← parent.state
11:     for child in successor (of the current parent) do
12:         v ← child
```

Depth First Search (DFS) - graph II

Manchester Metropolitan University

```
13:       if v is not in C and child is not in O then
14:          add child to O
15:       end if
16:    end for
17: end while
18: return failure
```

Depth First Search

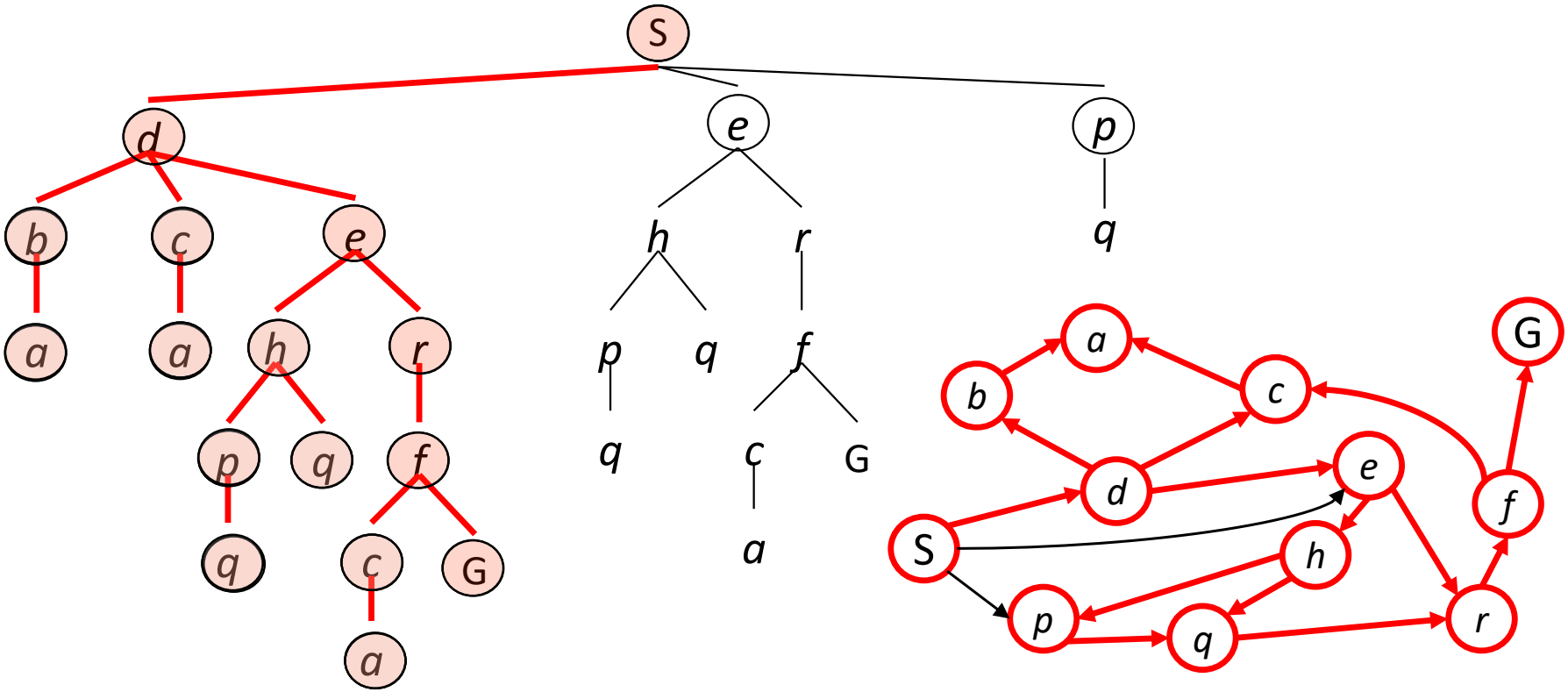
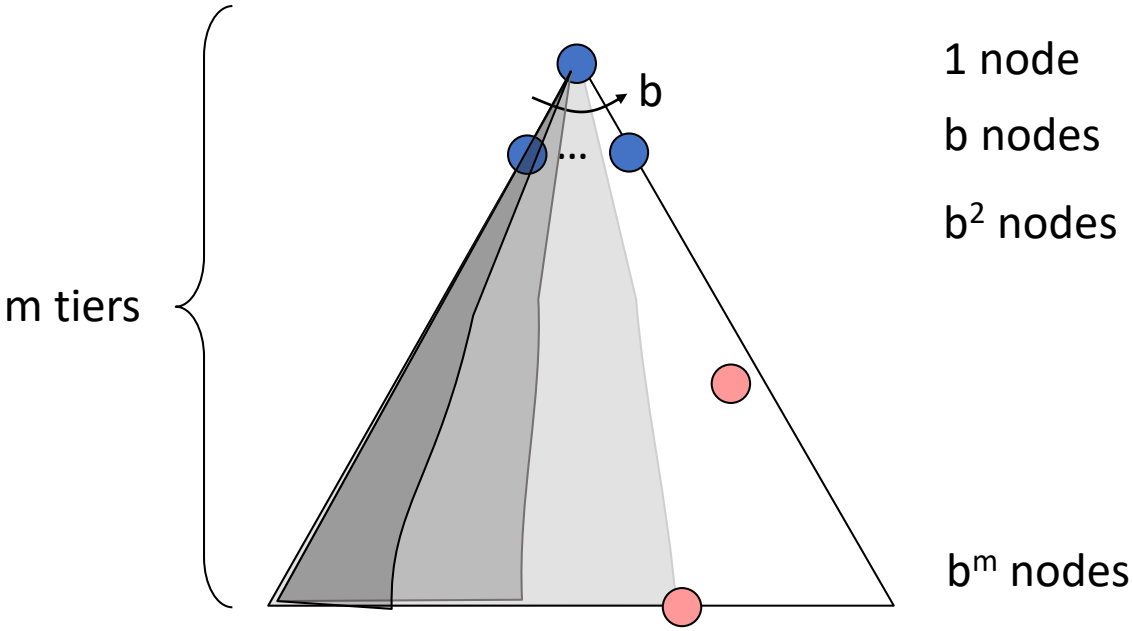


Figure 8: An example of depth first search

Depth First Search Properties



Reached the cheapest solution with depth s

- Has processed all nodes on the left (left prefix)
- Could have processed the whole tree
- Time complexity: $\mathcal{O}(b^m)$ with finite m
- Space complexity (frontier): $\mathcal{O}(bm)$ (only siblings along path)
- Complete: Yes without cycle
- Optimal: No, 'leftmost'
- Action coat: 0

Uniform Cost Search (UCS)

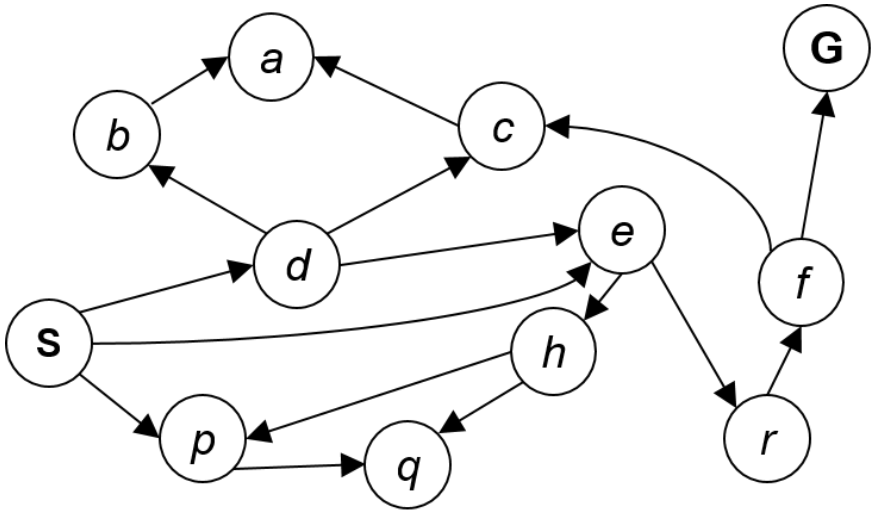


Figure 9: Same cost for each edge

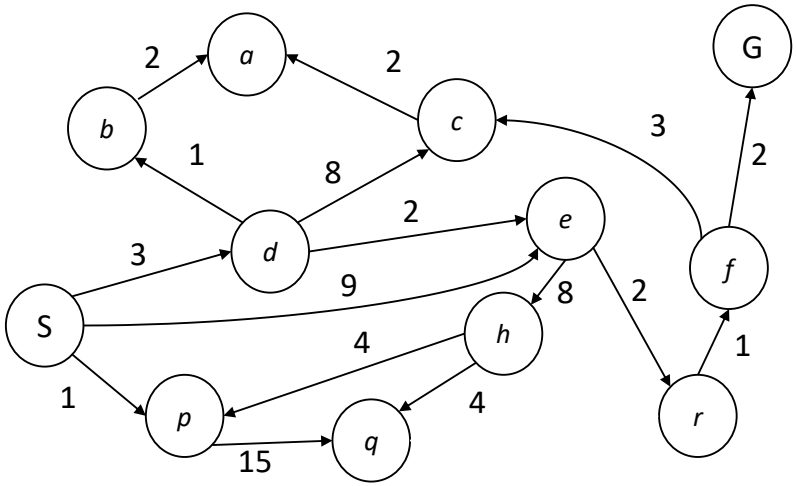


Figure 10: Different cost for each edge

- Breadth first search considers number of actions to reach a node (goal)
- When each edge comes with a cost (different), breadth first search can be tricky
- **Try the codes with '8n' for breadth first search, tell us why it is tricky?**

Uniform Cost Search (UCS) - graph I

Manchester Metropolitan University

Algorithm 3: Pseudocode of Uniform Cost Search

Input: Initial state **s**, goal state **g**, a evaluation function f
Output: A *node* helps to retrieve a solution (path) $\mathcal{P}$, or failure

1: *node* $\leftarrow$ with **s** as state
2: $\mathcal{O} \leftarrow$ *node*, an priority queue % *node* with the least cost.
3: $\mathcal{C} \leftarrow \emptyset$
4: **while** $\mathcal{O}$!= $\emptyset$ **do**
5: *parent* $\leftarrow$ the best node in $\mathcal{O}$ % based on the evaluation function f.
6: **if** *parent*.state == **g then**
7: **return** *parent*
8: **end if**
9: **del** *parent* from O
10: $\mathcal{C} \leftarrow$ *parent*
11: **for** *child* **in** successor (of the current *parent*) **do**

Uniform Cost Search (UCS) - graph II

Manchester Metropolitan University

```
12:        v ← current child
13:        if v is not in C and child is not in O then
14:           add child to O        % found a new node.
15:        else if v is not in C and child is in O then
16:           if current pathcost of child < previous pathcost of child then
17:              add current child to O
18:           end if
19:        end if
20:     end for
21:  end while
22:  return failure
```

- openset $\mathcal{O}$ is a priority queue, i.e., the node with the minimum (maximum) path cost will always pop out first, regardless when it is pushed in.
- closedset $\mathcal{C}$ is a dictionary with the node state (location) as key, which ensures a node is unique.

Uniform Cost Search (UCS) - graph III

Manchester Metropolitan University

- a solution *node* helps to retrieve the path $\mathcal{P}$ from a tree after the goal is reached.

Uniform Cost Search (UCS)

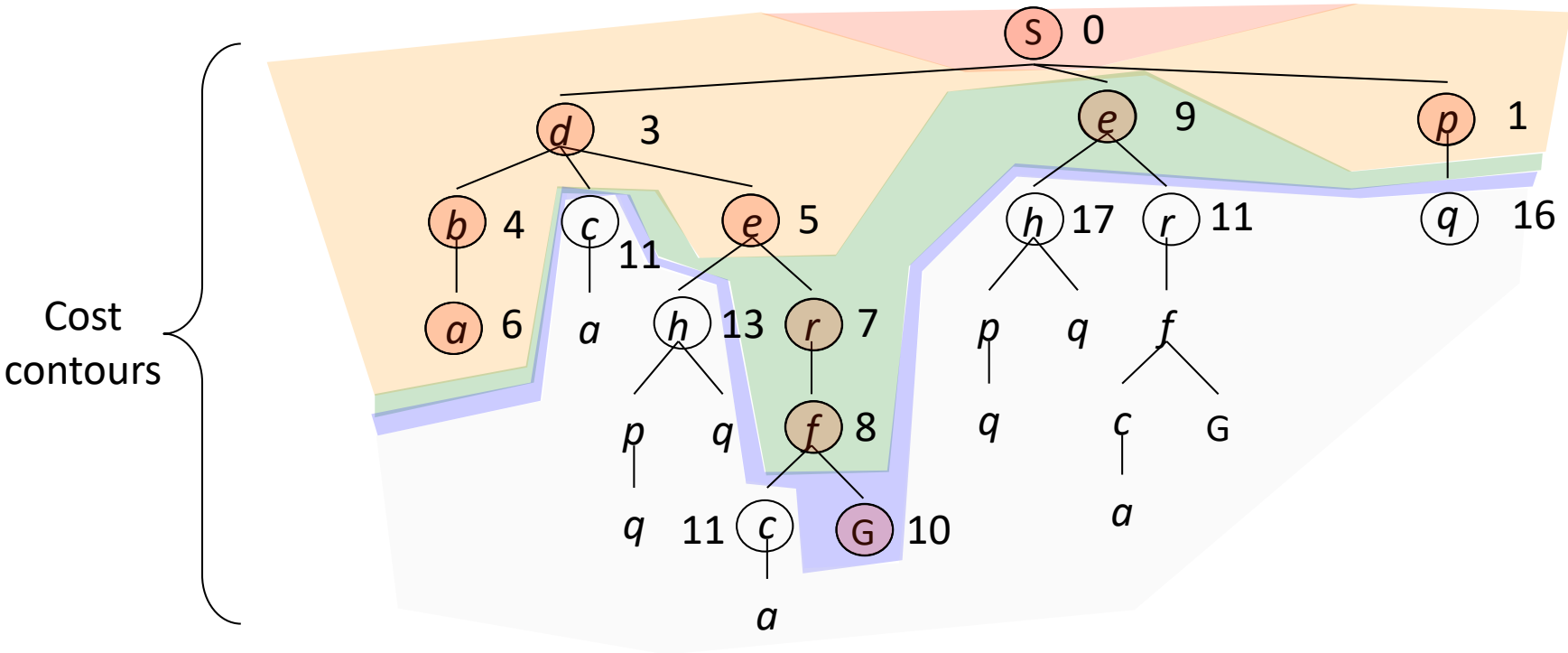
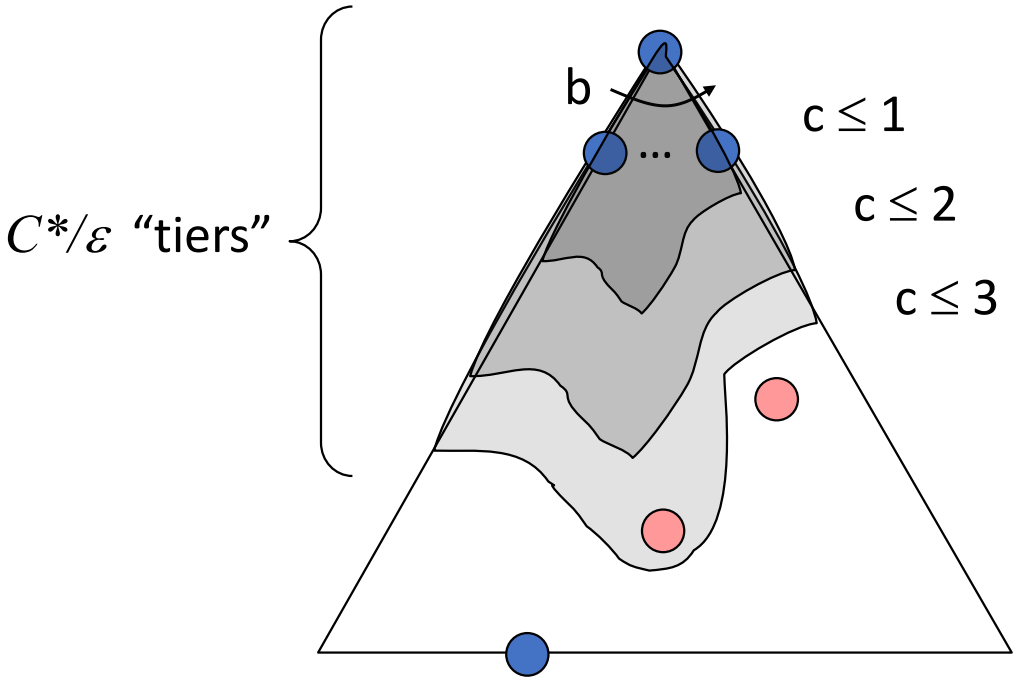


Figure 11: Search contour of UCS

- Think about the contour of breadth first search (later depth first search and else)

Uniform Cost Search Properties



Reached the cheapest solution with cost C^*, and the minimum edge cost is ϵ

- Has processed all nodes with cost less than C^*
- The 'effective depth' (deepest depth) is $\backsim C^*/\epsilon$
- Time complexity: $\mathcal{O}(b^{C^*/\epsilon})$
- Space complexity: $\mathcal{O}(b^{C^*/\epsilon})$
- Complete: yes if the best solution has a finite cost and minimum edge cost is positive
- Optimal: Yes

BFS vs. UCS vs. DFS

Manchester Metropolitan University

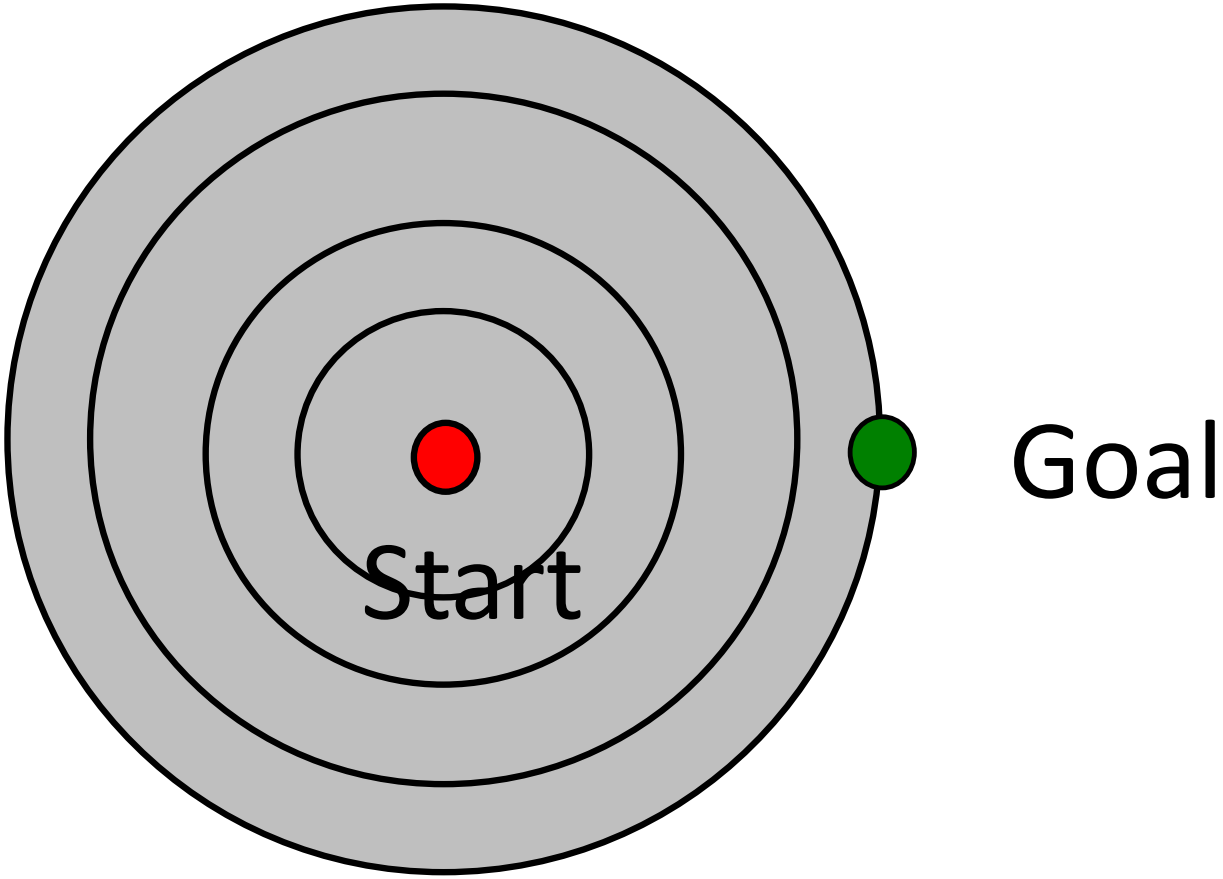


Figure 12: Search Contour of BFS

- Search in all directions
- Edge costs are equal

BFS vs. **UCS** vs. DFS

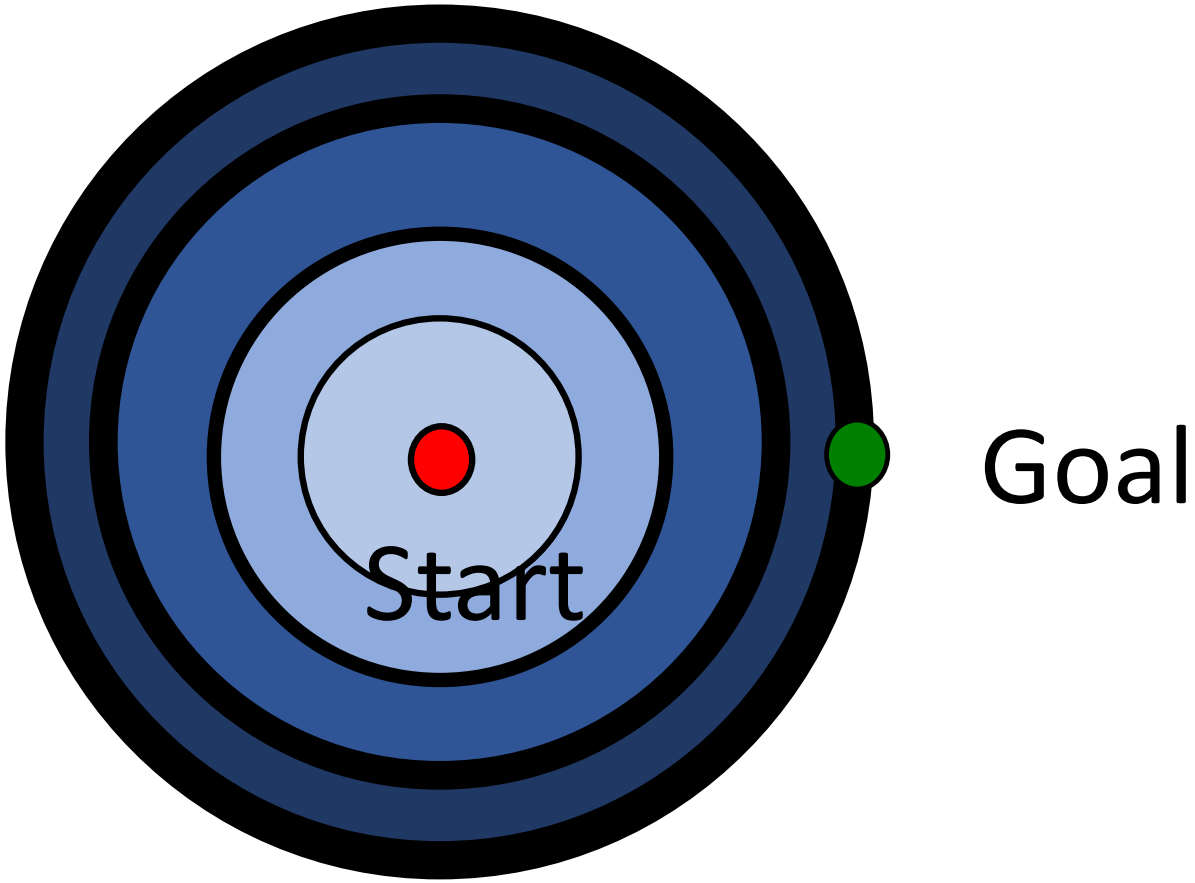


Figure 13: Search Contour of UCS

- Search in all directions
- Edge costs are different (circle linewidth)

BFS vs. UCS vs. **DFS**

Manchester Metropolitan University

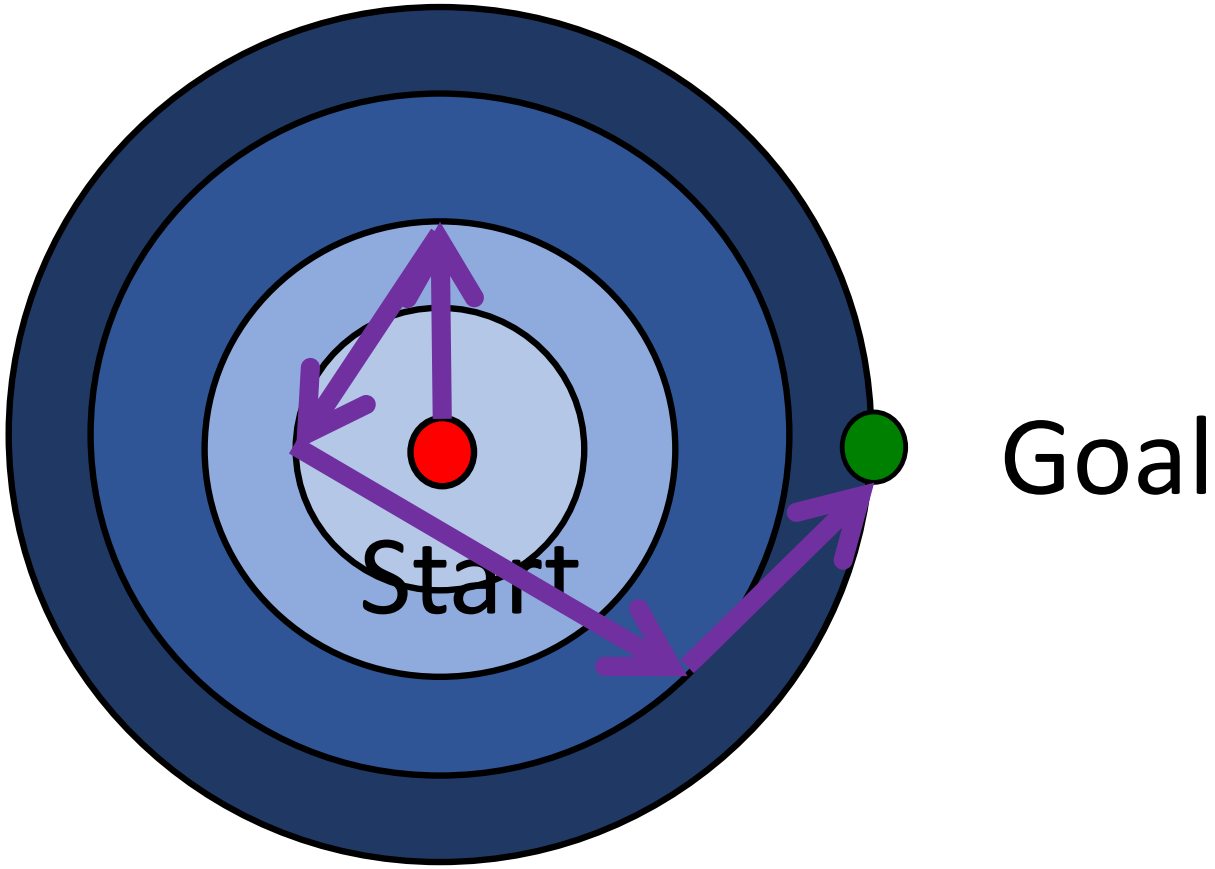


Figure 14: Search Contour of DFS

- Search in a specific direction (depth)
- Edge costs are not prioritised, not as much as depth

Summary

Manchester Metropolitan University

We have learned:

- How to formulate a search problem
- Breadth First Search
- Uniform Cost Search
- Depth First Search
- Pros and Cons of each algorithm