



Search (I)

Mingsheng Long

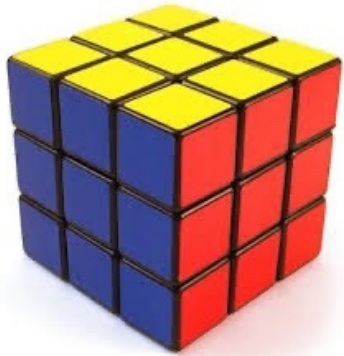
Tsinghua University

Spring 2023

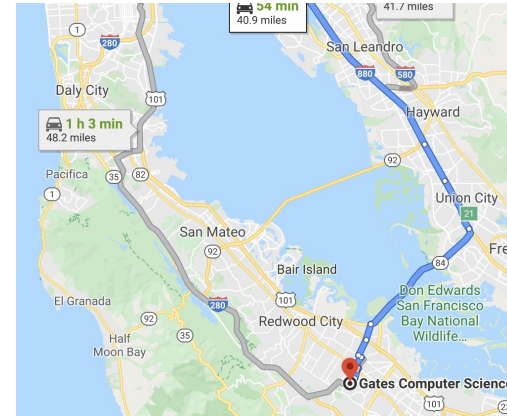
Outline

- **Search Problems**
- General Search
 - Tree Search, Graph Search
- Uninformed Search
 - Depth-First (DFS), Breadth-First (BFS), Uniform-Cost (UCS)
- Informed (Heuristic) Search
 - A* Tree Search
 - A* Graph Search

Search Problems in Real Applications



Puzzle solving



Route finding



Robot motion planning

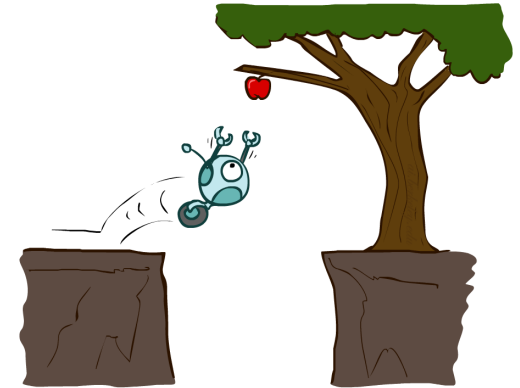


Multi-robot systems

Reflex Agents vs. Planning Agents

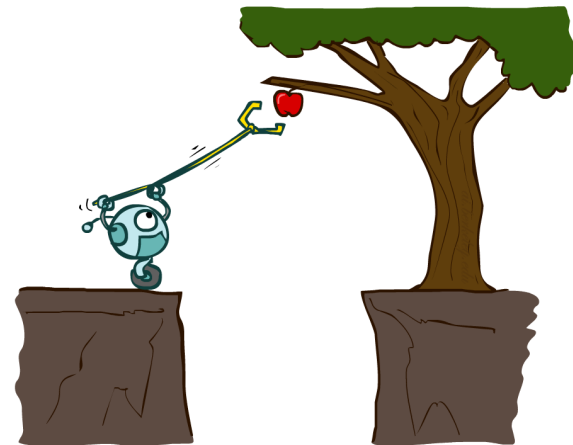
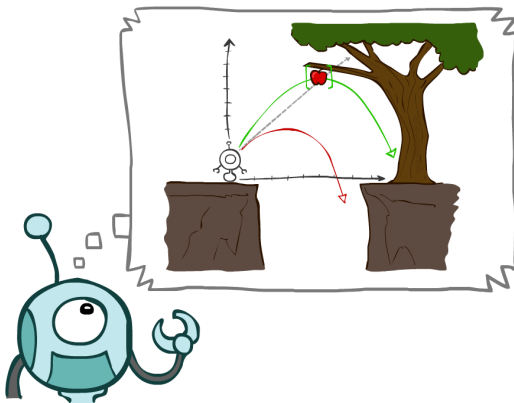
- Reflex Agents:

percept $x \rightarrow f \rightarrow$ single action y



- Planning Agents:

percept $x \rightarrow f \rightarrow$ action sequence $(a_1, a_2, a_3, a_4, \dots)$



A model of how the world evolves in response to actions

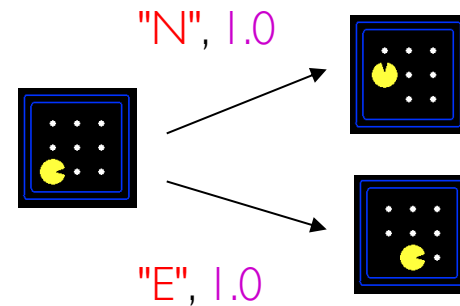
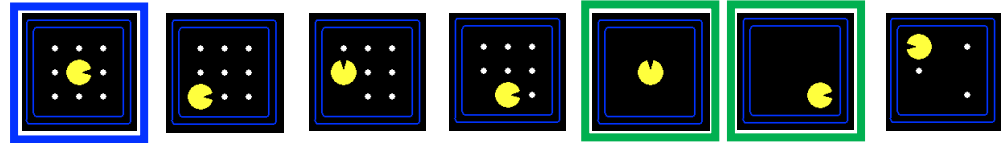
Action sequences to achieve a definite goal

Search Problems

- A **search problem** consists of:

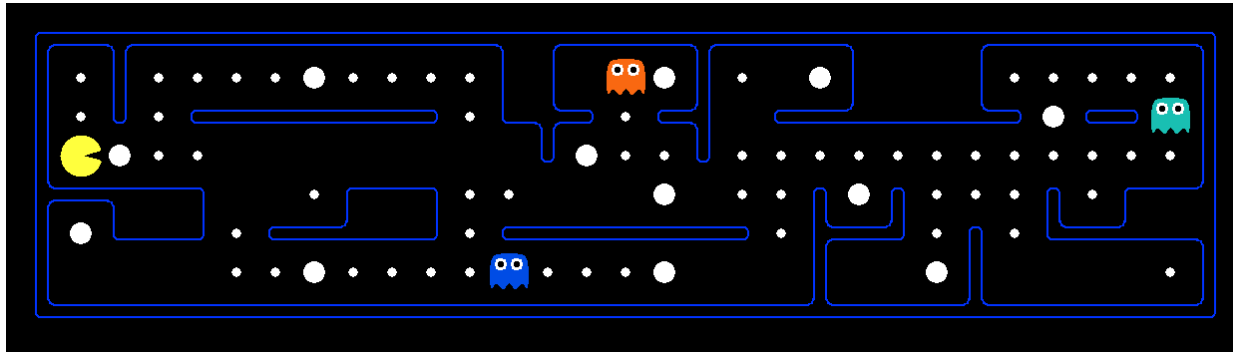
Pac-Man (吃豆人), 1980

- A **state space** S
- An **initial state** s_0
- **Actions** $A(s)$ in each state
- **Transition model** $\text{Result}(s, a)$
 - A **successor function**
- A **goal test** $G(s)$
- **Action cost** $c(s, a, s')$



- A **solution** is an action sequence that reaches a goal state
- An **optimal solution** has the least cost among all solutions

State Space



The **world state** includes every last detail of the environment

A search state **abstracts** away details not needed to solve the problem

- Problem: **Pathing**

- States: (x,y) location
- Actions: NSEW
- Transition: update location only
- Goal test: is $(x,y)=\text{END}$

- Problem: **Eat-All-Dots**

- States: $\{(x,y), \text{dot Booleans}\}$
- Actions: NSEW
- Transition: update location and possibly a dot Boolean
- Goal test: dots all false

State Space

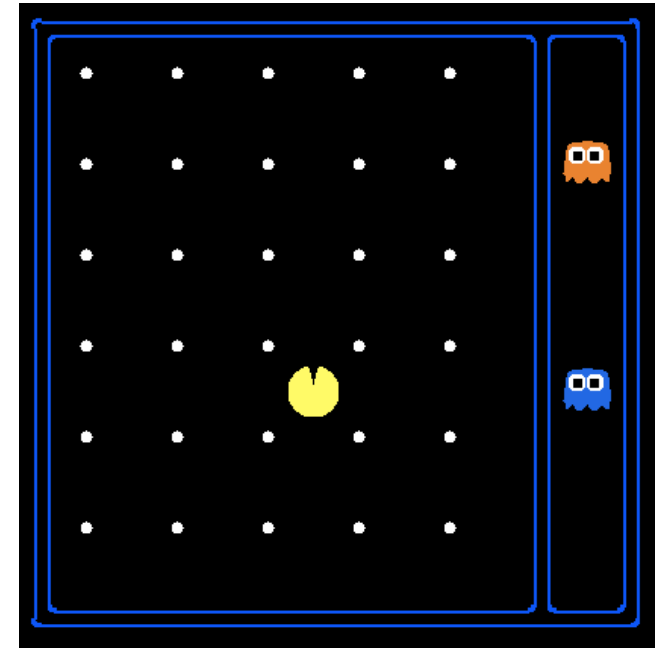
- World state:

- Agent positions: 120
- Food count: 30
- Ghost positions: 12
- Agent facing: NSEW

- How many

- World states: $120 \times (2^{30}) \times (12^2) \times 4$
- States for pathing: 120
- States for eat-all-dots: $120 \times (2^{30})$

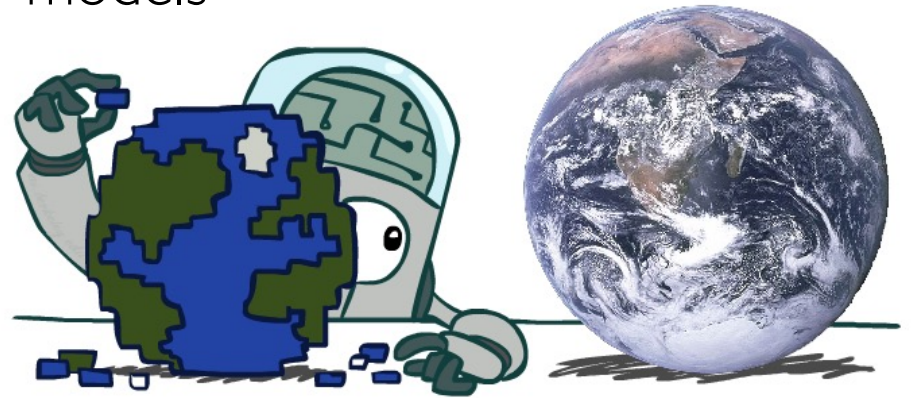
- The **size** of the search space depends on the problem being solved



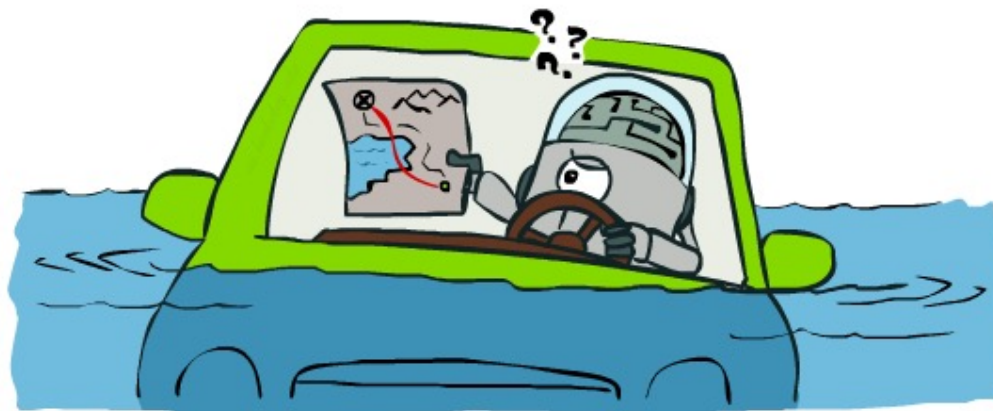
Usually **exponential**
(NP-hard)

Search Models

- A search problem is reasonable but **not the real thing**
- It is a **model**, an **abstract mathematical description** of real problems
- Search operates over models of the world
 - The agent doesn't actually try all the plans out in the real world
 - Planning is all **in simulation (rollouts)**
 - Search is only as good as your models
- All models are **wrong...**
- But some are **useful** 😊



Gone Wrong with Search Models

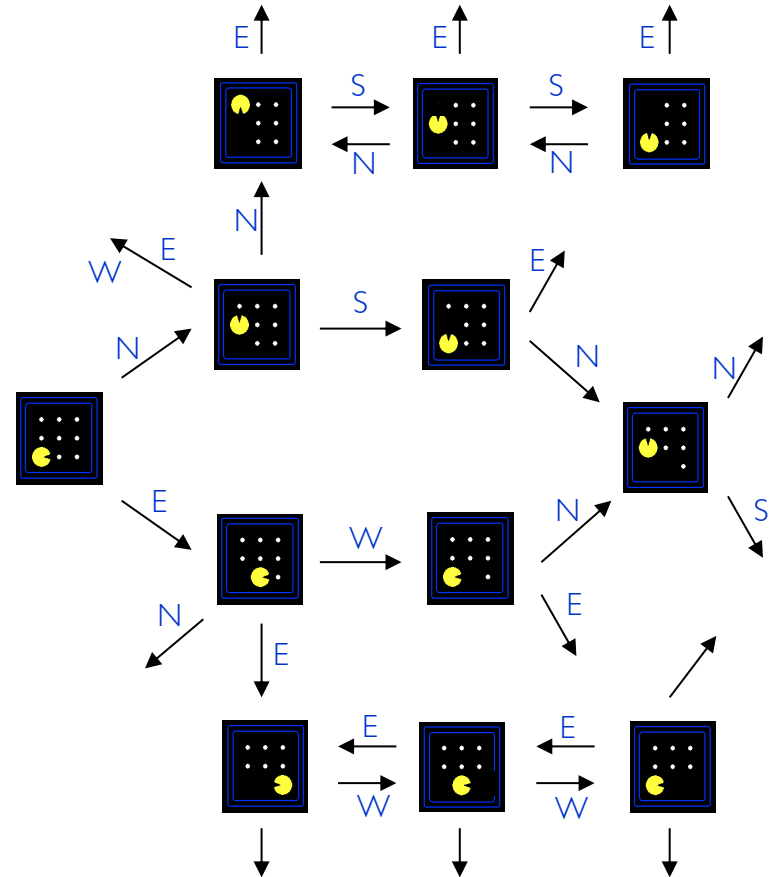


Outline

- Search Problems
- **General Search**
 - Tree Search, Graph Search
- Uninformed Search
 - Depth-First (DFS), Breadth-First (BFS), Uniform-Cost (UCS)
- Informed (Heuristic) Search
 - A* Tree Search
 - A* Graph Search

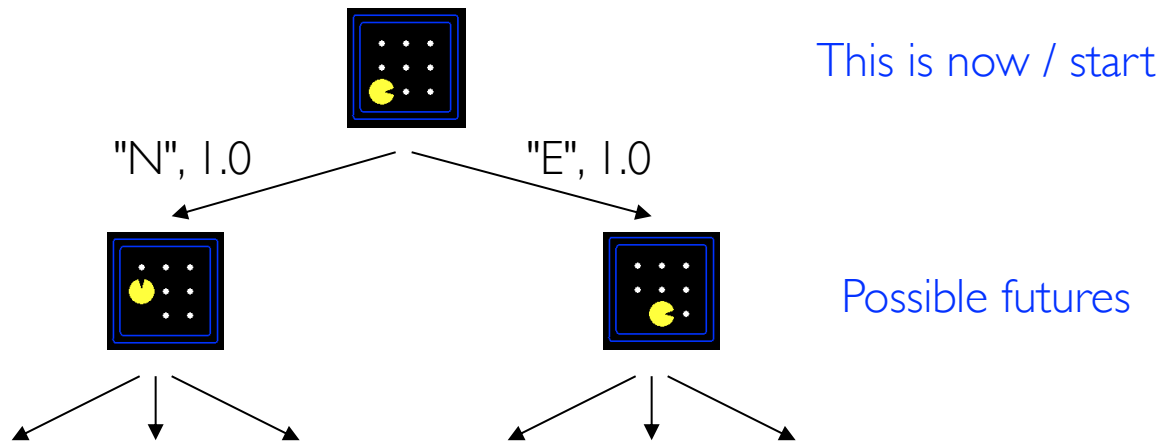
State Space Graph

- **State space graph**: A mathematical representation of a search problem
 - **Nodes** are states
 - **Arcs** represent transitions
 - The goal test is a set of **goal nodes** (maybe only one)
- Each state occurs **only once**
- We can rarely build the **full graph** in memory
 - It is **too big** but is a **useful idea**



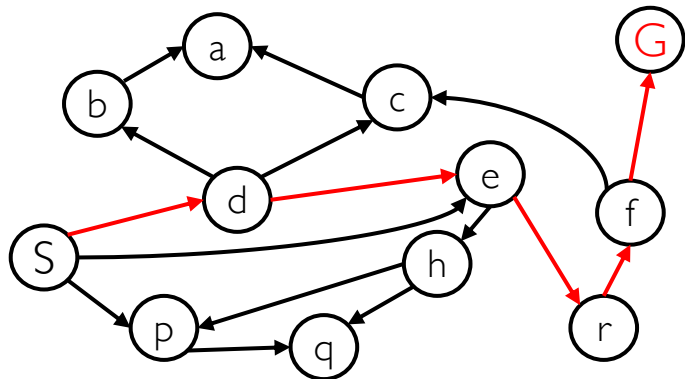
Search Tree

- **Search tree**: A "what if" tree of plans and their outcomes
 - The start state is the **root node**
 - **Children** correspond to successors
 - **Nodes** show states, but correspond to **plans** that achieve those states
- For most problems, we can **never** actually build the **whole tree**

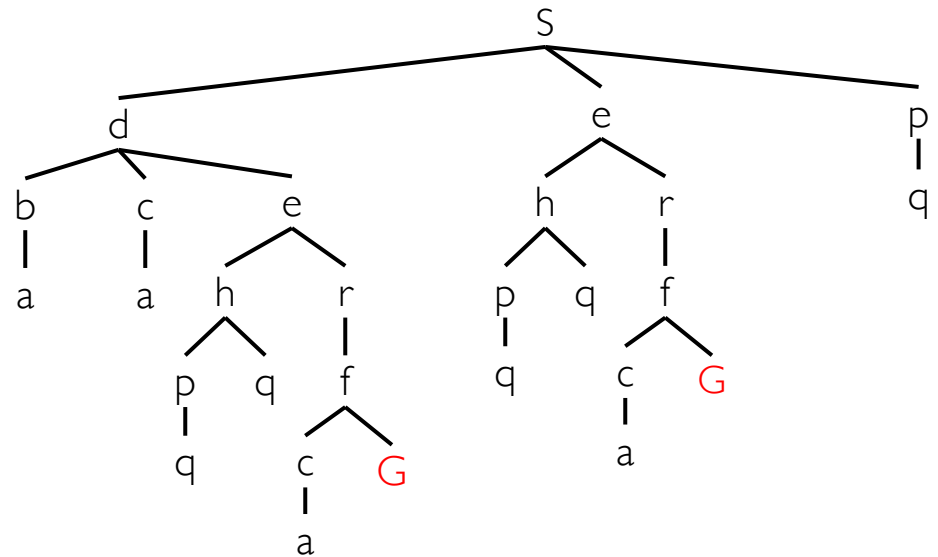


State Space Graph vs. Search Tree

- Each **node** in the search tree is an entire **path** in the state space graph.
- We construct the tree **on demand** – and we construct **as little as possible**.



How big is this search tree?



General Tree Search

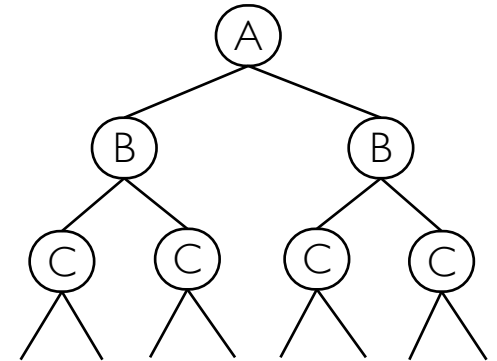
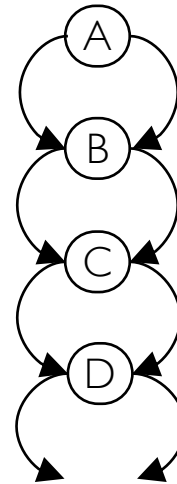
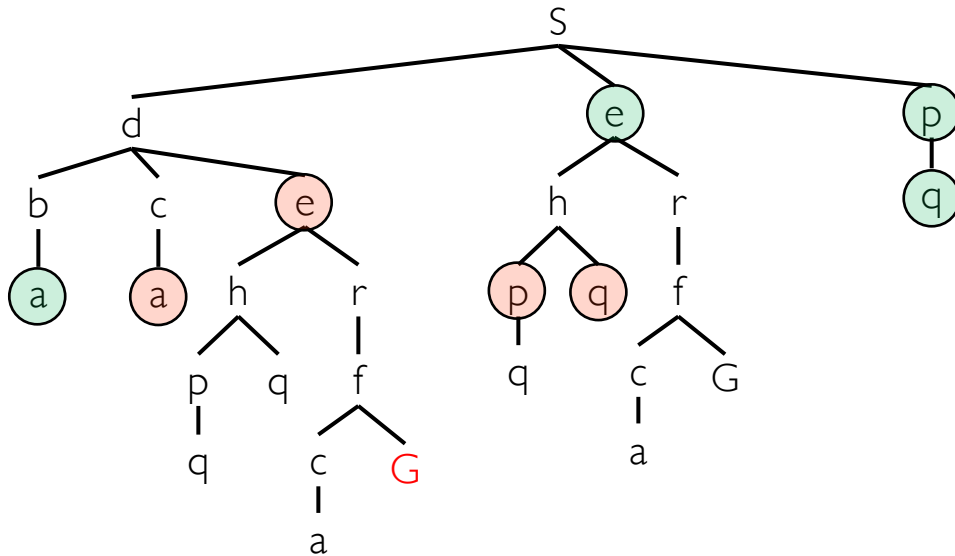
1. **function** TREE-SEARCH(*problem*) **returns** a solution, or *failure*
2. initialize the *frontier* as a specific work list (stack, queue, priority queue)
3. add initial state of *problem* to *frontier* (1). Initialization
4. **loop do**
5. **if** the *frontier* is empty **then**
6. **return** *failure*
7. choose a *node* and remove it from the *frontier*
8. **if** the *node* contains a goal state **then**
9. **return** the corresponding solution
10. (2). Selection
11. **for each** resulting *child* from node
12. add *child* to the *frontier* (3). Expansion

Frontier: leaf nodes for expansion

Strategy: How to choose a leaf node to expand

Tree Search: Extra Work

- Repeated states and redundant paths can cause a tractable problem to become intractable.



In BFS, for example, we should not bother expanding the red circled nodes

Failure to detect repeated states can
cause exponentially more work

General Graph Search

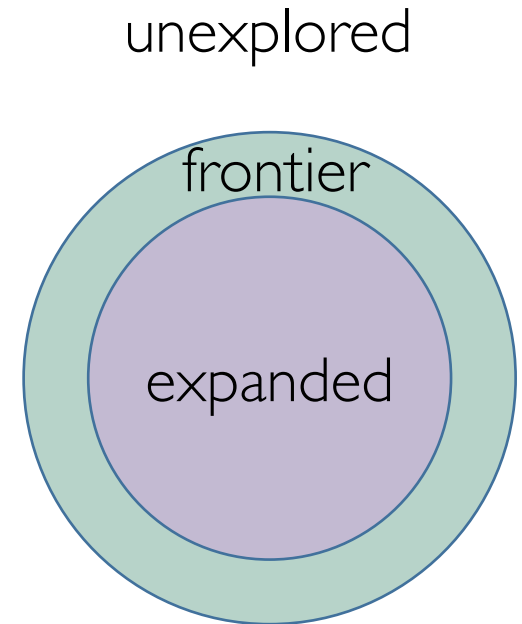
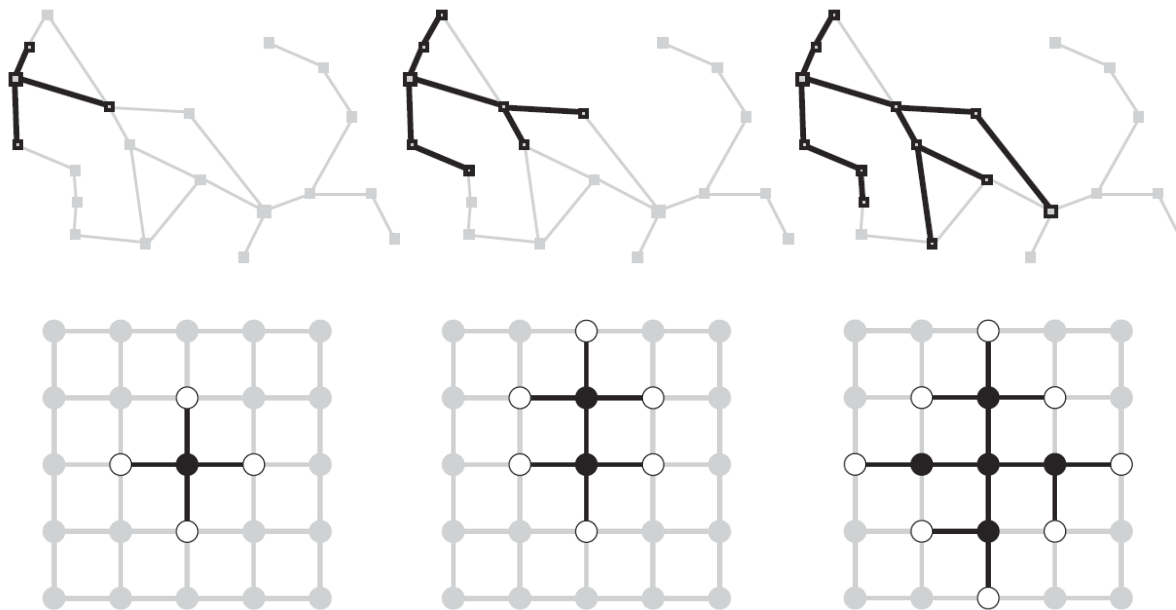
1. **function** GRAPH-SEARCH(*problem*) **returns** a solution, or *failure*
2. **initialize** the *explored set* to be empty
3. **initialize** the *frontier* as a specific work list (stack, queue, priority queue)
4. **add** initial state of *problem* to *frontier* (1). Initialization
5. **loop do**
6. **if** the *frontier* is empty **then**
7. **return** *failure*
8. **choose** a *node* and remove it from the *frontier*
9. **if** the *node* contains a goal state **then**
10. **return** the corresponding solution (2). Selection
11. **add** the *node* state to the *explored set*
12. **for each** resulting *child* from node
13. **if** the *child* state is not already in the *frontier* or *explored set* **then**
14. **add** *child* to the *frontier* (3). Expansion

Strategy: How to choose a leaf node to expand

Idea: never expand a state twice

Graph Search: Frontier Separation

- This graph search algorithm **overlays a growing tree on a graph**
- **Frontier** separates **expanded** region from **unexplored** region of the state-space graph

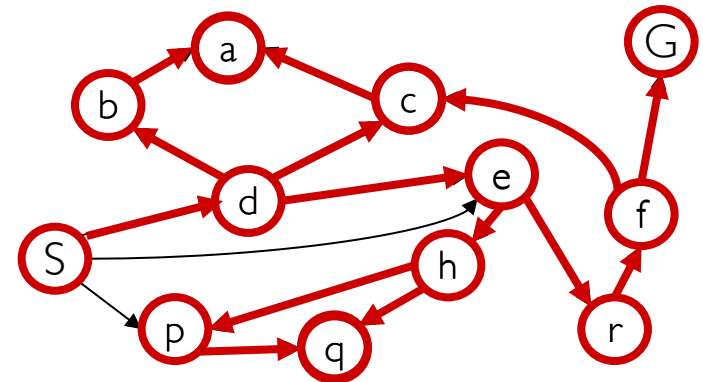
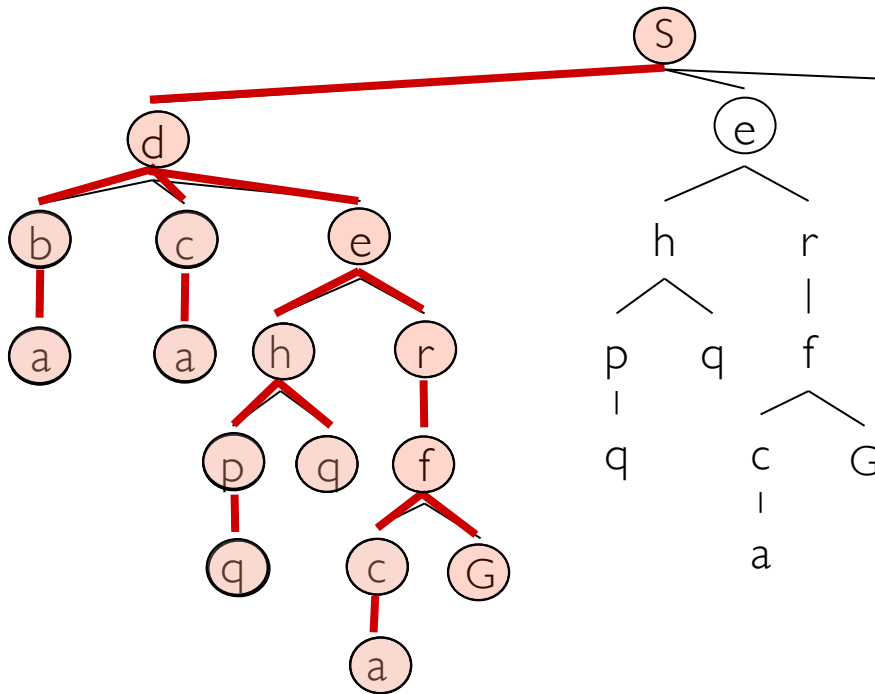
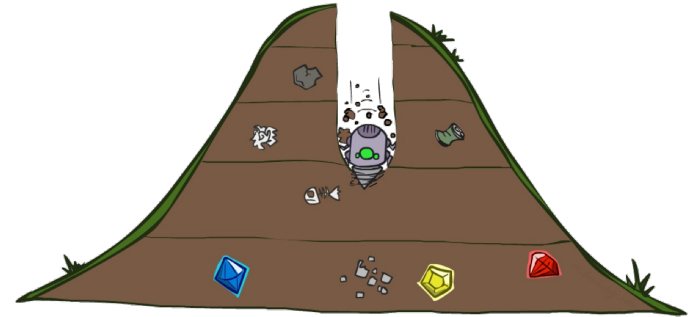


Outline

- Search Problems
- General Search
 - Tree Search, Graph Search
- **Uninformed Search**
 - Depth-First (DFS), Breadth-First (BFS), Uniform-Cost (UCS)
- Informed (Heuristic) Search
 - A* Tree Search
 - A* Graph Search

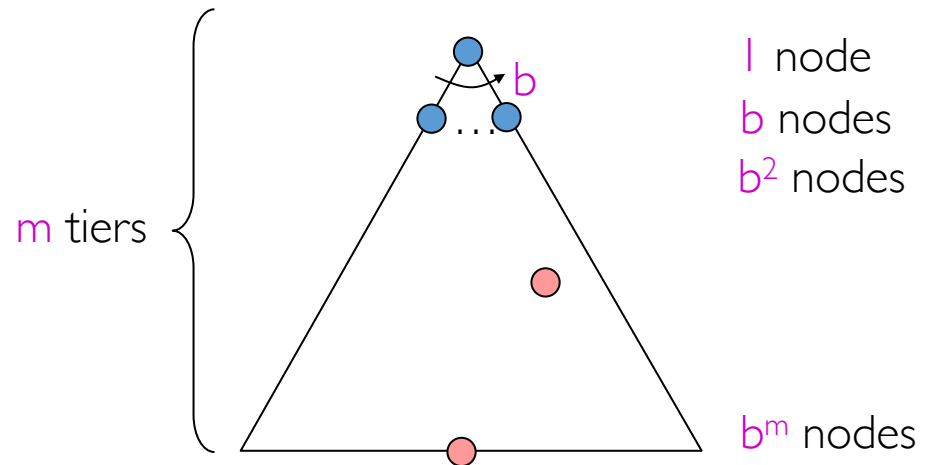
Depth-First Search (DFS)

- **Strategy**: expand a **deepest** node first
- **Implementation**: Frontier is a **LIFO stack**



Search Algorithm Properties

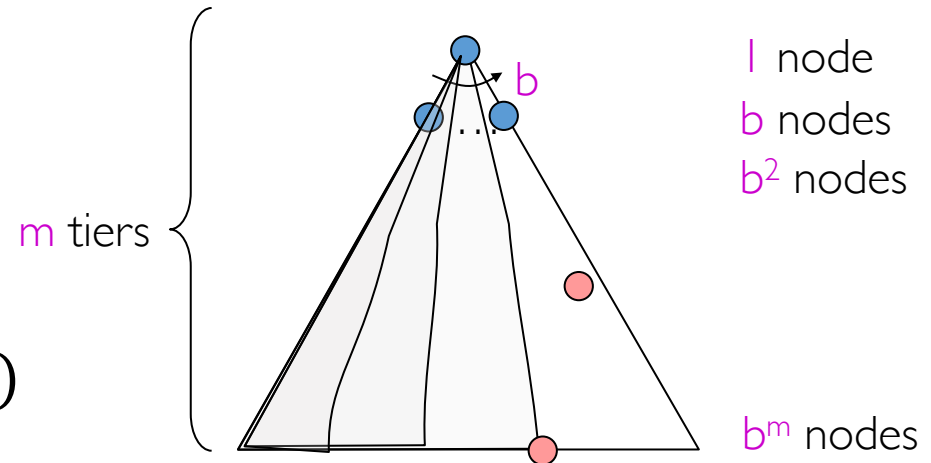
- Properties
 - **Complete**: Guaranteed to find a solution if one exists
 - **Optimal**: Guaranteed to find the least cost path
 - Time complexity
 - Space complexity
- Cartoon of search tree:
 - b is the branching factor
 - m is the maximum depth
 - **Solutions** at various depths
 - Number of nodes in entire tree



$$1 + b + b^2 + \dots + b^m = O(b^m)$$

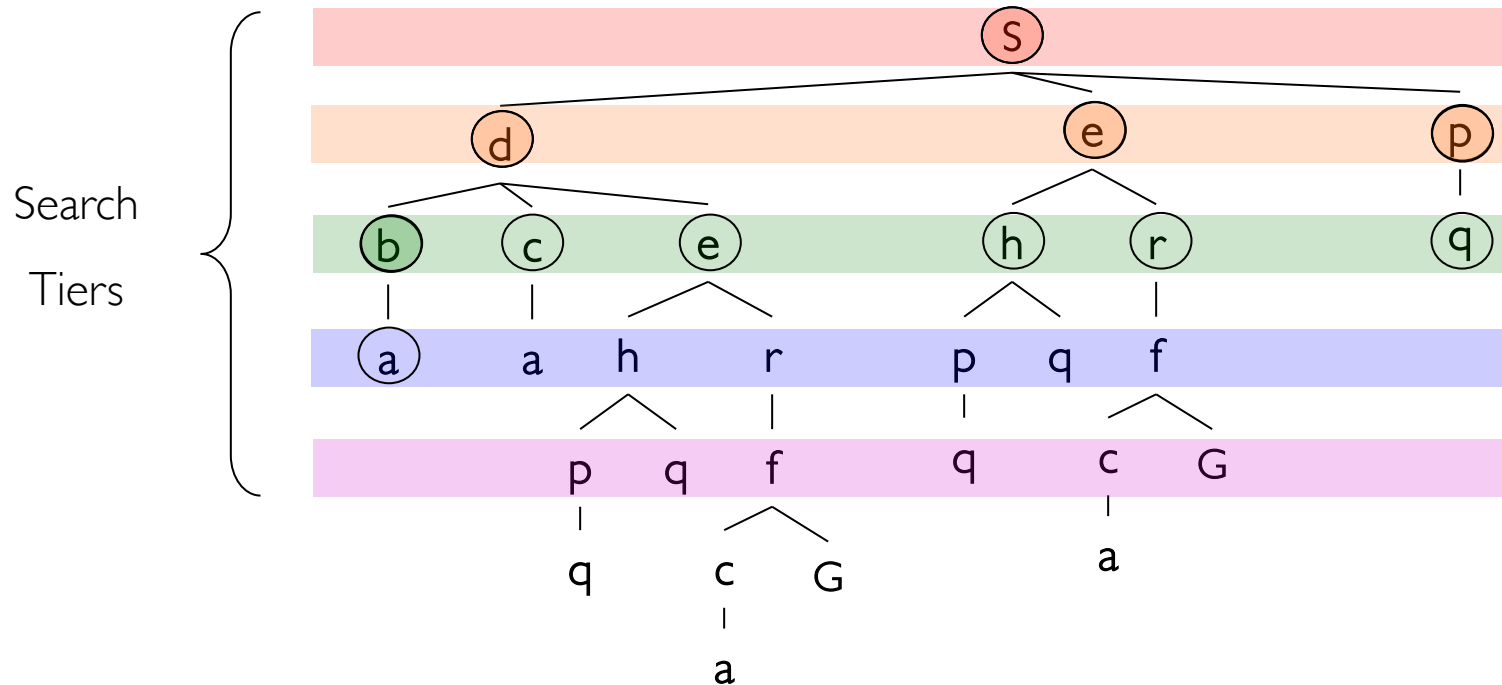
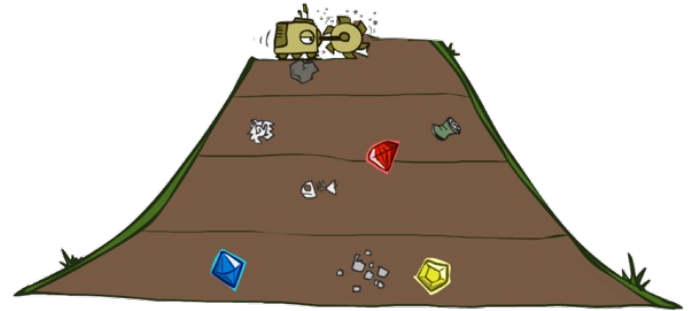
Depth-First Search Properties

- Time complexity: what nodes DFS expand?
 - Some left prefix of the tree
 - Could process the whole tree
 - If m is finite, takes time $O(b^m)$
- Space complexity
 - Only siblings on path, so $O(bm)$
- Is it complete?
 - m could be infinite, so only if we prevent cycles (Graph Search)
- Is it optimal?
 - No, it finds the leftmost solution, regardless of depth or cost



Breadth-First Search (BFS)

- **Strategy**: expand a **shallowest** node first
- **Implementation**: Frontier is a **FIFO queue**



Breadth-First Search Properties

- Time complexity: what nodes does BFS expand?

- Processes all nodes above shallowest solution
- Let depth of shallowest solution be d
- Search takes time $O(b^d)$

- Space complexity

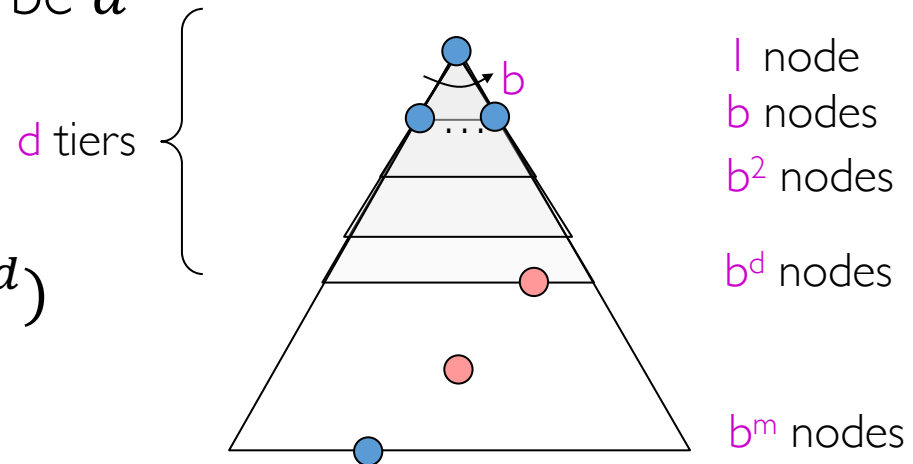
- Has roughly the last tier, so $O(b^d)$

- Is it complete?

- d must be finite if a solution exists, so yes

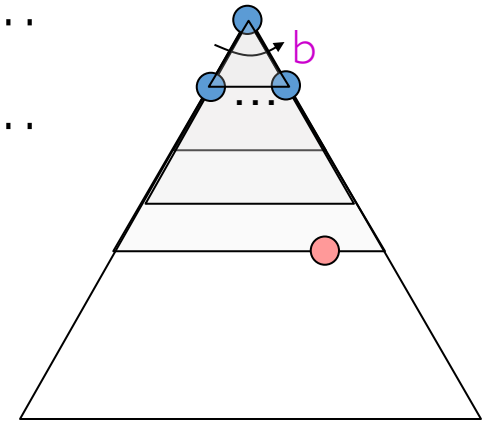
- Is it optimal?

- If costs are equal (e.g., 1)
- If not, Uniform Cost Search (more later)



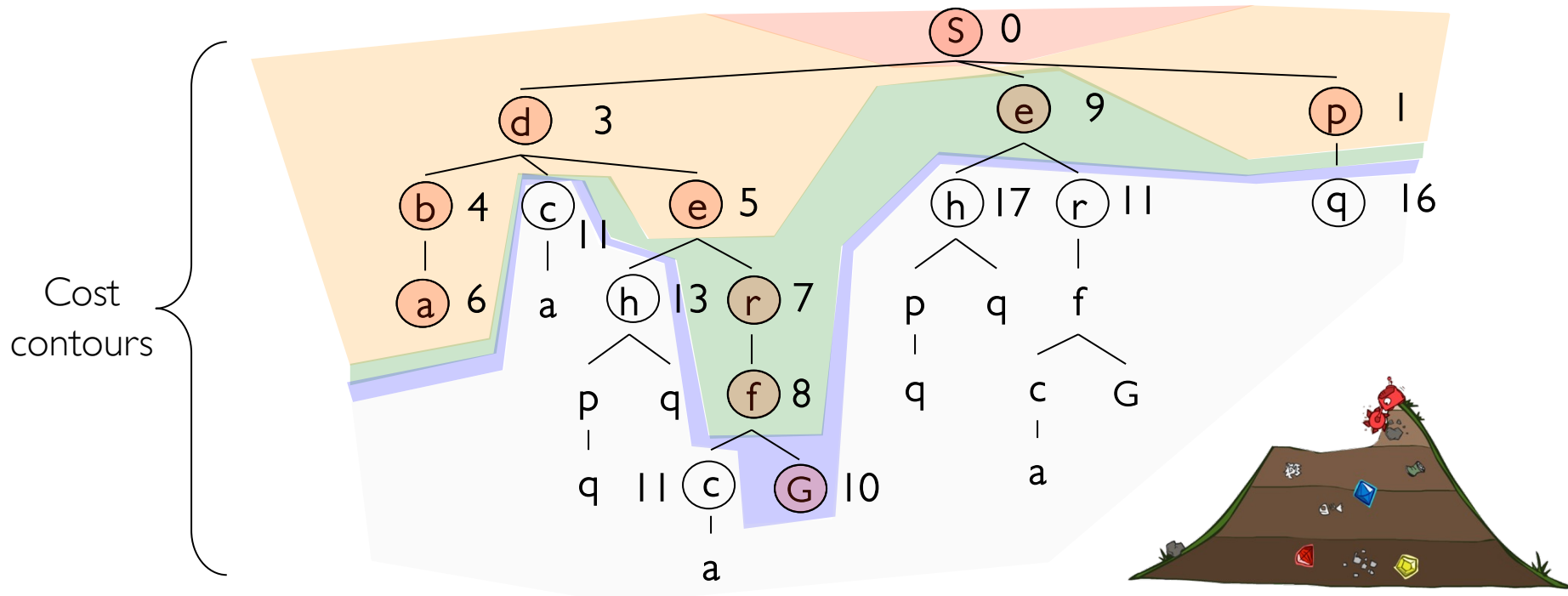
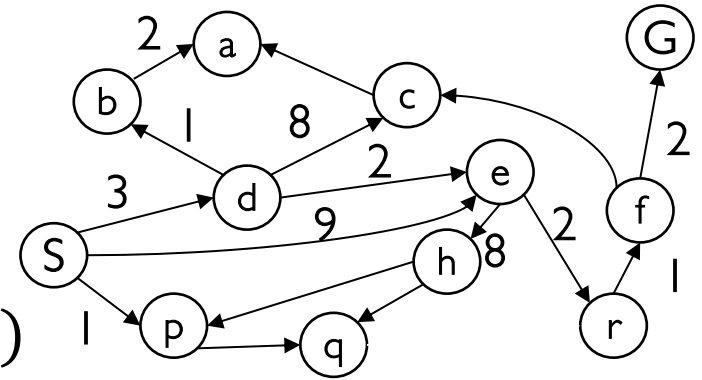
Iterative Deepening Search (IDS)

- **Idea:** get DFS's *space* advantage with BFS's *time* & *shallow-solution* advantages
 - Run a DFS with *depth limit* 1. If no solution...
 - Run a DFS with *depth limit* 2. If no solution...
 - Run a DFS with *depth limit* 3. ...
- Complete? **Yes**. Optimal? **Yes**.
- Isn't that wastefully redundant? Not so bad:
$$O(1) + O(b) + O(b^2) + \dots + O(b^d) = O(b^d)$$
- Time complexity: $O(b^d)$
- Space complexity: $O(bd)$



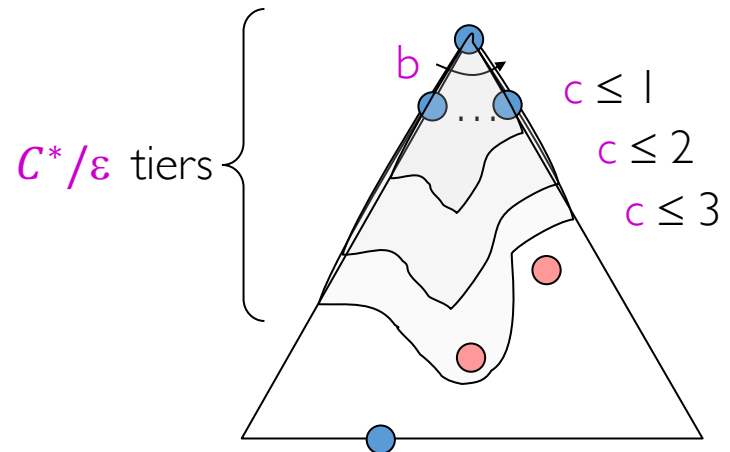
[Dijkstra, 1956] Uniform Cost Search (UCS)

- **Strategy**: expand **lowest** $g(n)$
 - $g(n)$ = cost from root to n
- Frontier is a **priority queue** sorted by $g(n)$



Uniform Cost Search Properties

- Time complexity: what nodes does UCS expand?
 - Processes all nodes with cost less than the **cheapest** solution
 - If that solution costs C^* and arcs cost at least ϵ , then the "effective depth" is roughly C^*/ϵ
 - Takes time $O(b^{C^*/\epsilon})$
- Space complexity
 - Has roughly the last tier, so $O(b^{C^*/\epsilon})$
- Is it **complete**?
 - Assuming C^* is finite and $\epsilon > 0$, **yes**
- Is it **optimal**?
 - **Yes** (Proof via A*)



Uniform Cost (Graph) Search

1. **function** UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or *failure*
2. initialize the *explored set* to be empty
3. initialize the *frontier* as a **priority queue** using **node path-cost** as the **priority**
4. add initial state of *problem* to *frontier* with **path-cost = 0** (1). Initialization
5. **loop do**
6. **if** the *frontier* is empty **then return** *failure*
7. choose a *node* and remove it from the *frontier*
8. **if** the *node* contains a goal state **then**
9. **return** the corresponding solution (2). Selection
10. add the *node* state to the *explored set*
11. **for each** resulting *child* from node
12. **if** the *child* state is not already in the *frontier* or *explored set* **then**
13. add *child* to the *frontier*
14. **else if** the *child* is already in the *frontier* with higher path-cost **then**
15. replace that *frontier* node with *child* (3). Expansion

Key idea: extra check

A shorter path to a *frontier* state is discovered.



Uninformed Search: Summary

- Tree Search

Criterion	Action Costs	Complete?	Optimal?	Time	Space
Depth-First	$= c$	No	No	$O(b^m)$	$O(bm)$
Breadth-First	$= c$	Yes	Yes	$O(b^d)$	$O(b^d)$
Iterative Deepening	$= c$	Yes	Yes	$O(b^d)$	$O(bd)$
Uniform-Cost	$\geq \epsilon$	Yes	Yes	$O(b^{C^*/\epsilon})$	$O(b^{C^*/\epsilon})$

- Graph Search

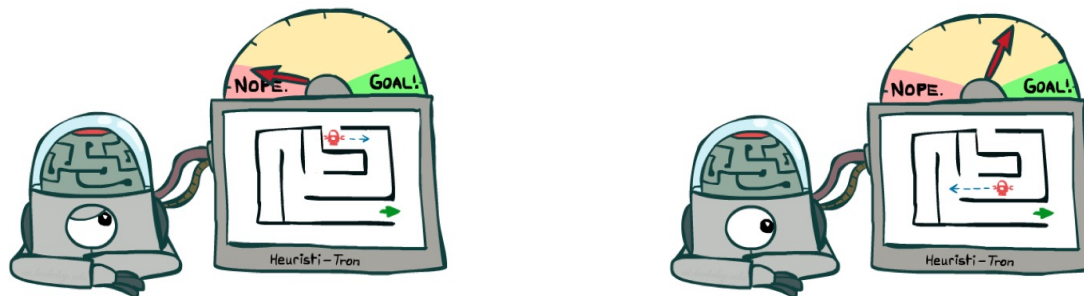
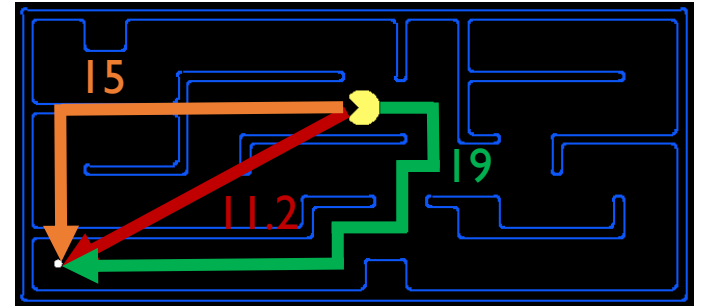
- Depth-first search is **complete** for finite state spaces
 - The space and time complexities are **bounded by the size of the state space**
- All these search algorithms are **the same except for frontier strategies**
 - Can code one implementation that takes a variable queuing object

Outline

- Search Problems
- General Search
 - Tree Search, Graph Search
- Uninformed Search
 - Depth-First (DFS), Breadth-First (BFS), Uniform-Cost (UCS)
- **Informed (Heuristic) Search**
 - A* Tree Search
 - A* Graph Search

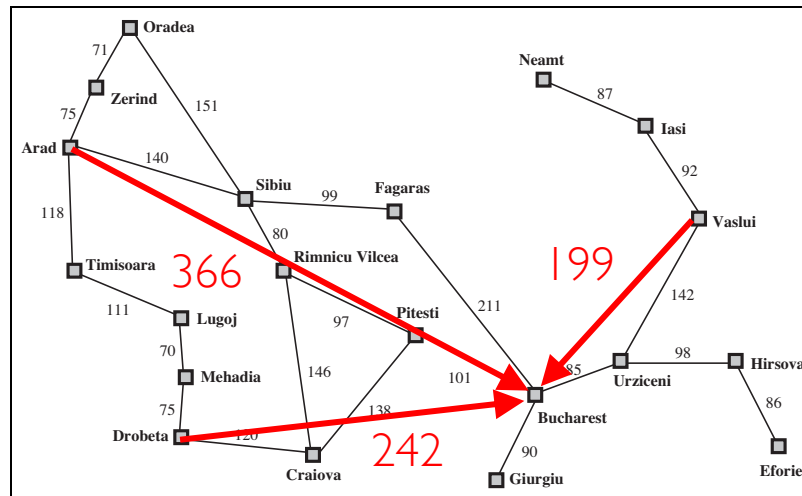
Informed (Heuristic) Search

- Issues of Uniform Cost Search
 - Explores options in **every "direction"**
 - **No information** about goal location
- A **heuristic** is:
 - A function that **estimates** how close a state is to a goal
 - Designed for a particular search problem
 - Examples: **Manhattan distance**, **Euclidean distance** for pathing



Example: Heuristic Function

- Route finding: straight-line distance

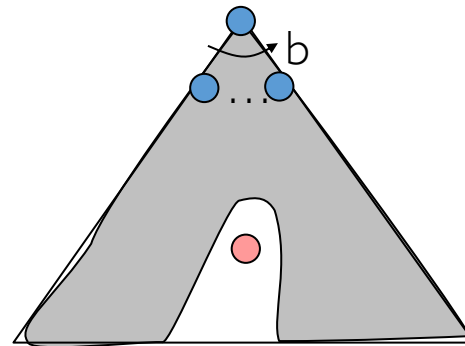
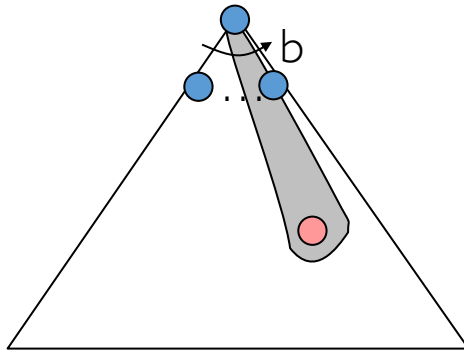


Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Figure 3.22 Values of h_{SLD} —straight-line distances to Bucharest.

Greedy Search

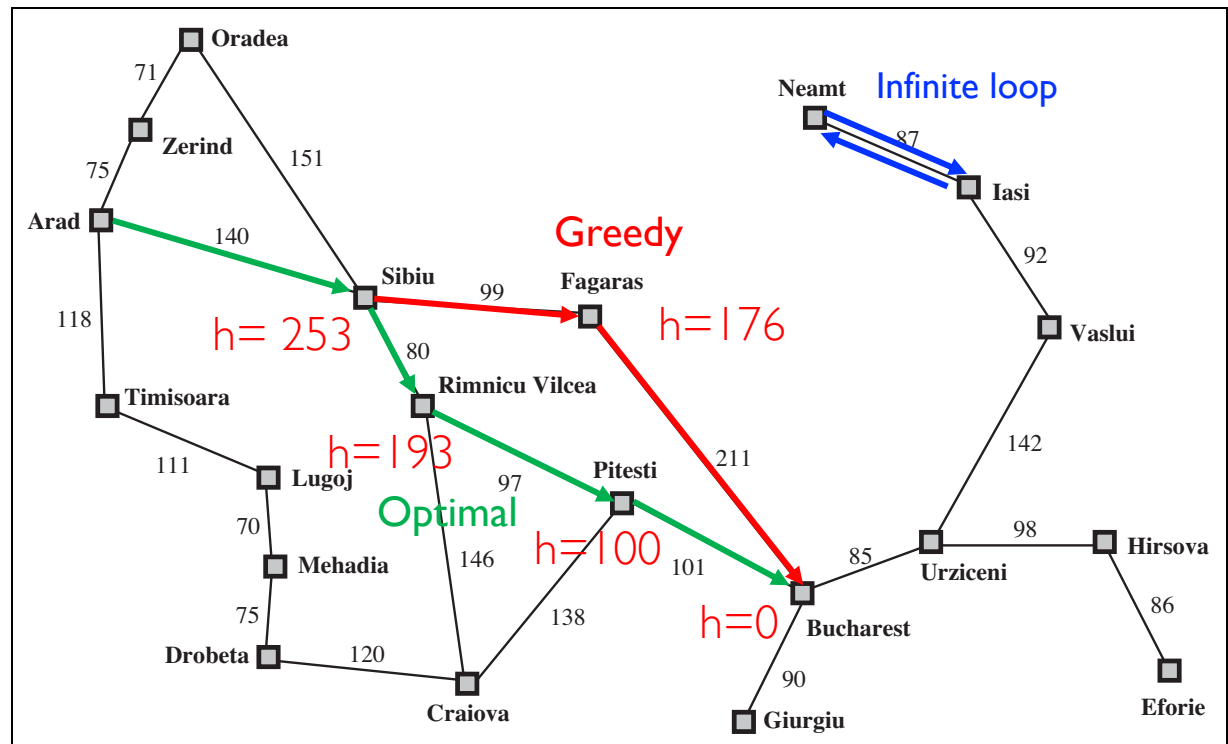
- **Strategy**: expand a node that you think is closest to a goal state
 - $h(n)$ = heuristic of state n
- Frontier is an ascending order priority queue by $h(n)$
- **Best-first** takes you straight to the (wrong) goal
 - A common case
- Worst-case: like a badly-guided DFS



Greedy Search Properties

- Greedy Tree Search
 - Is it **complete**? **No**. Example: **Iasi** → **Fagaras**, leads to infinite loop
- Greedy Graph Search: avoid repeated state
 - Is it **optimal**? **No**.

Sibiu-Fagaras-Bucharest =
 $99 + 211 = 310$
Sibiu-Rimnicu Vilcea-
Pitesti-Bucharest =
 $80 + 97 + 101 = 278$



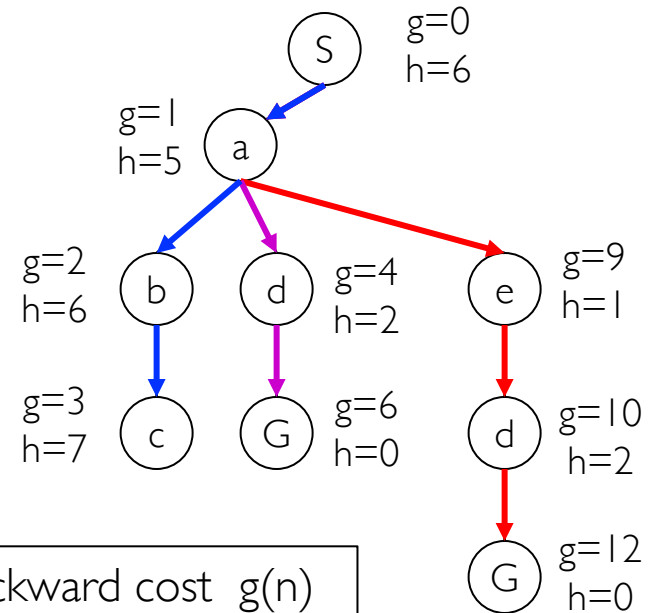
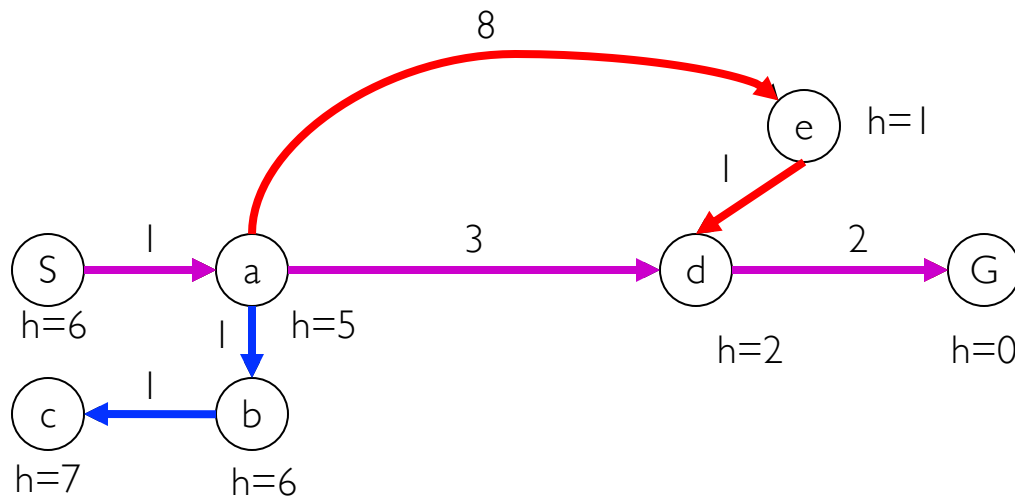
Outline

- Search Problems
- General Search
 - Tree Search, Graph Search
- Uninformed Search
 - Depth-First (DFS), Breadth-First (BFS), Uniform-Cost (UCS)
- Informed (Heuristic) Search
 - **A* Tree Search**
 - A* Graph Search

[Hart/Nilsson/Raphael, 1968]

A* Search

- **Strategy:** Combining UCS and Greedy Search
 - Sorted by $f(n) = g(n) + h(n)$

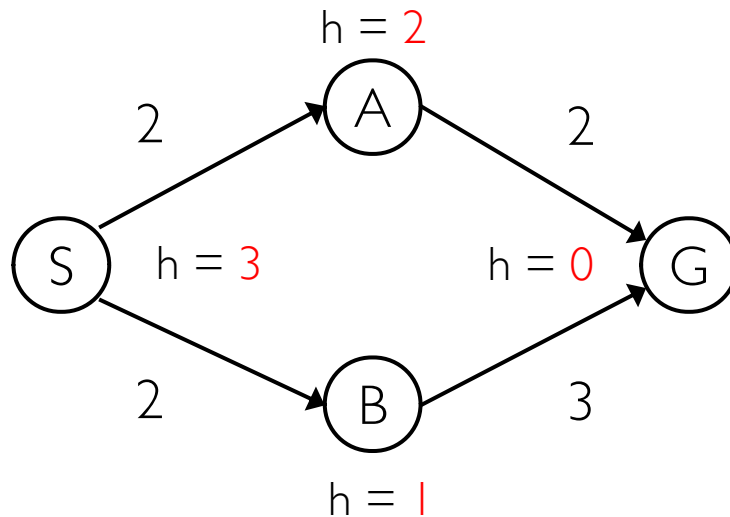


Uniform-cost orders by path cost, or backward cost $g(n)$
Greedy orders by goal proximity, or forward cost $h(n)$
A* Search orders by the sum: $f(n) = g(n) + h(n)$

When Should A* Terminate?

- Should we stop when we enqueue a goal?

State space graph



Queue

S(0+3)

S-B(2+1), S-A(2+2)

S-A(2+2), S-B-G(5+0)

S-A-G(4+0), S-B-G(5+0)

First goal dequeued

First goal
enqueued

No: only stop when we **dequeue** a goal

Is A* Optimal?

- Example: What went **wrong**?

- **Over-estimate** good goal cost

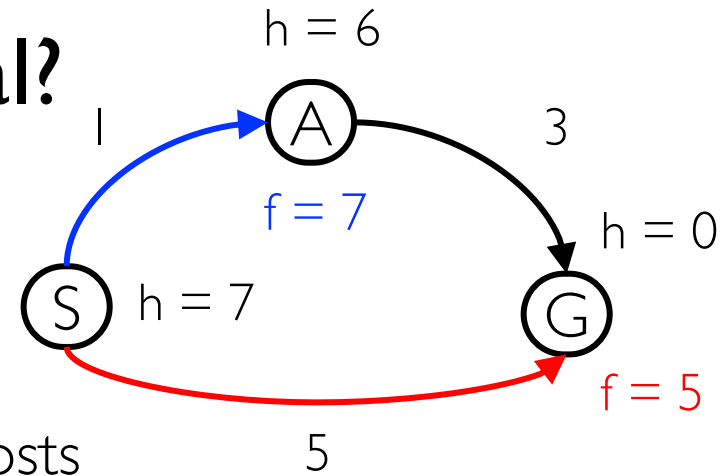
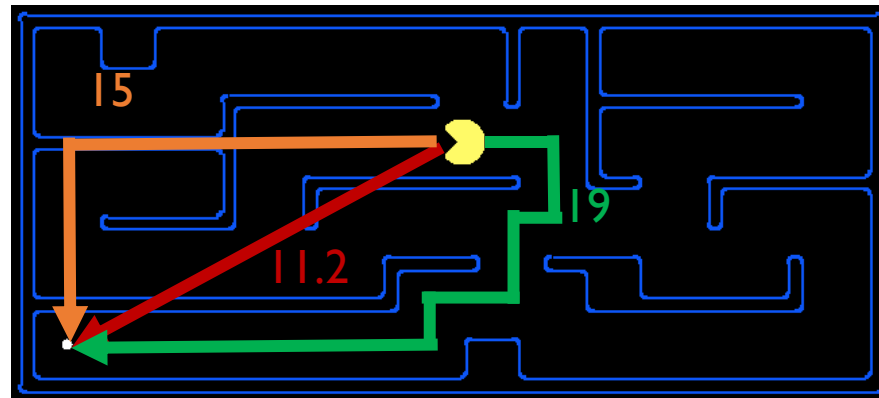
- Need estimates to be less than actual costs

- **Admissible Heuristics**: A heuristic h is **admissible** (可采纳) if:

$$0 \leq h(n) \leq h^*(n)$$

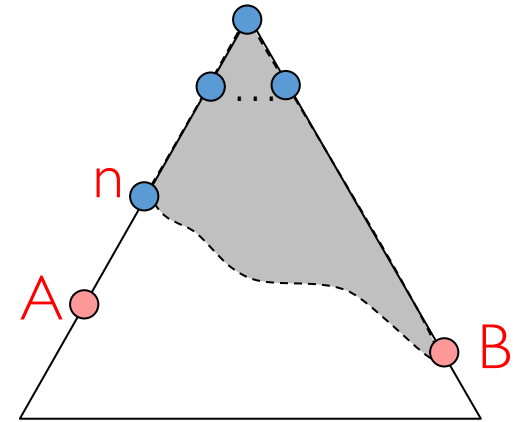
- where $h^*(n)$ is the true cost to a nearest goal

- Example:



Optimality of A* Tree Search

- Assume:
 - A is an optimal goal node, B is a suboptimal goal node
 - h is admissible
- Claim: A will exit the frontier before B
- Proof:
 - Imagine B is on the frontier
 - Some ancestor n of A on the optimal path (maybe A itself) is also on the frontier
 - Claim: n will be expanded before B
 - All ancestors of A expand before B
 - A expands before B



Optimality of A* Tree Search

- Claim: n will be expanded before B

- Proof:

- $f(n)$ is less or equal to $f(A)$:

$$f(n) = g(n) + h(n) \leq g(A) = f(A)$$

Definition of f

Admissibility

$h = 0$ at a goal

- $f(A)$ is less than $f(B)$:

$$f(A) = g(A) < g(B) = f(B)$$

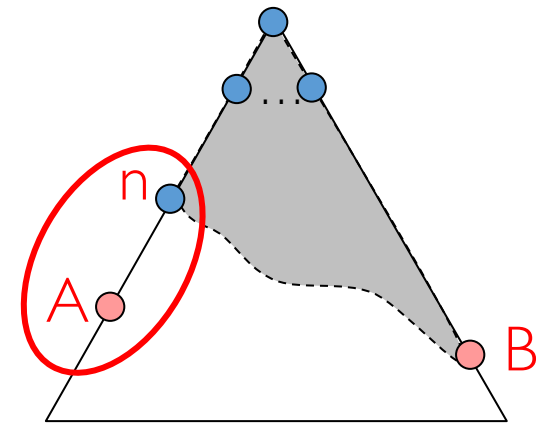
$h = 0$ at a goal

B is suboptimal

$h = 0$ at a goal

- n expands before B :

$$f(n) \leq f(A) < f(B)$$

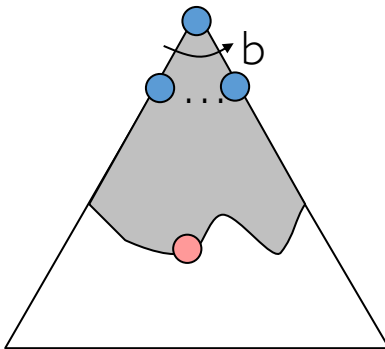


Efficiency of A*

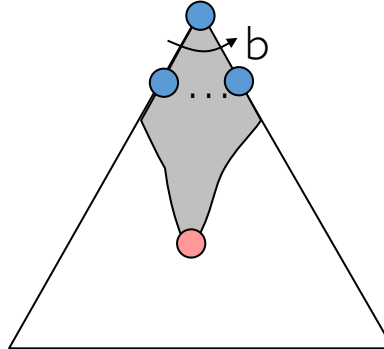
- **Theorem:** efficiency of A*
 - A* explores all states s satisfying
$$g(s) \leq g(s_{\text{goal}}) - h(s)$$
- **Interpretation:** the larger $h(s)$, the better

Key idea: distortion

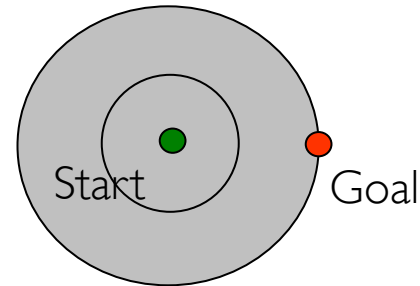
A* **distorts** edge costs to favor goal states



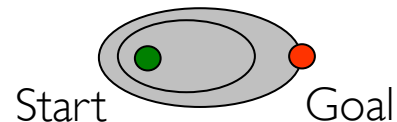
UCS



A*



UCS



A*

Creating Heuristics

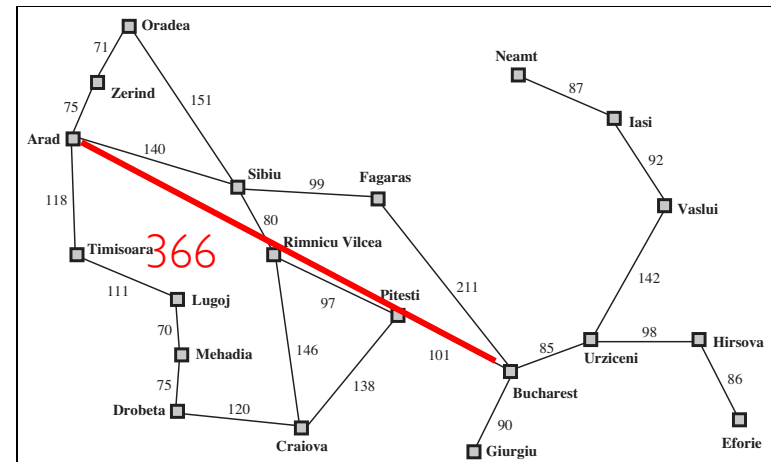
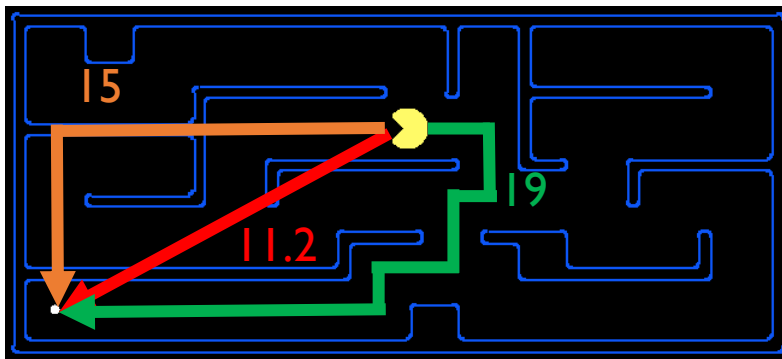
- Admissible heuristics are often solutions to **relaxed problems**

Definition: relaxed problems

Problem P_2 is a **relaxed version** of P_1 if $A_2(s) \supseteq A_1(s)$ for every s

Theorem:

$h_2^*(s) \leq h_1^*(s)$ for every s , so $h_2^*(s)$ is admissible for P_1

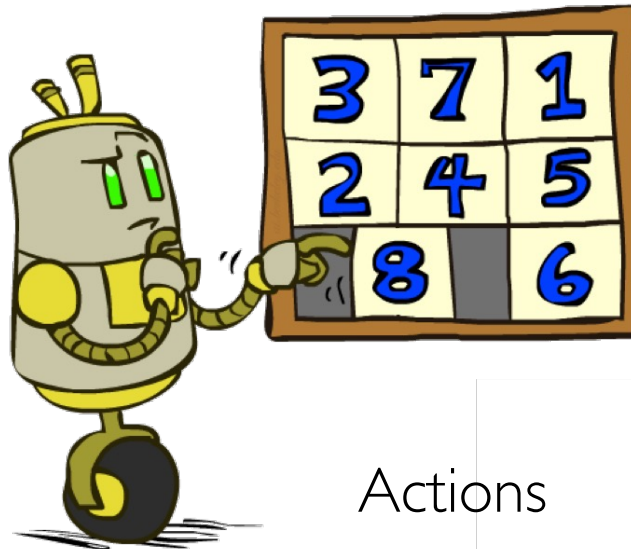


Example: 8 Puzzle

- The 8-puzzle actions:
 - A tile can move from square A to square B if A is horizontally or vertically adjacent to B **and** B is blank.

7	2	4
5		6
8	3	1

Start State



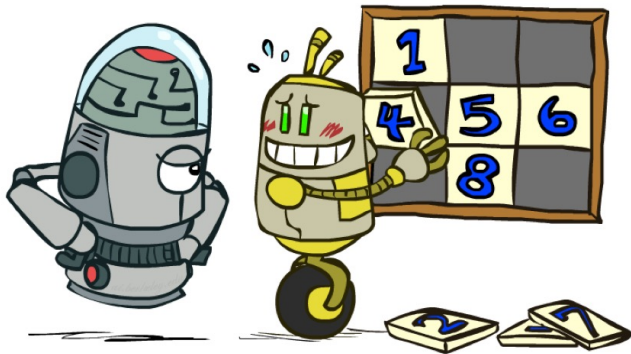
Actions

	1	2
3	4	5
6	7	8

Goal State

Example: 8 Puzzle

- **Relaxation I:** A tile can move **directly** from square A to square B.
 - h_1 = Number of tiles misplaced, e.g. $h_1(\text{start}) = 8$
- **Relaxation II:** A tile can move from square A to square B if A is adjacent to B **and B is blank**.
 - h_2 = Total Manhattan distance, e.g. $h_2(\text{start}) = 18$



7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Example: 8 Puzzle

- As heuristics get closer to the true cost, you will **expand fewer nodes** but usually **do more work per node** to compute the heuristic itself

	Average nodes expanded when the optimal path has...		
	...4 steps	...8 steps	...12 steps
UCS	112	6,300	3.6×10^6
TILES h_1	13	39	227
MANHATTAN h_2	12	25	73

Key Idea: a **trade-off** between quality of estimate and work per node

- Combining heuristics:** If a collection of admissible heuristics $h_1 \cdots h_m$ is available and **none of them dominates** any of the others, then take

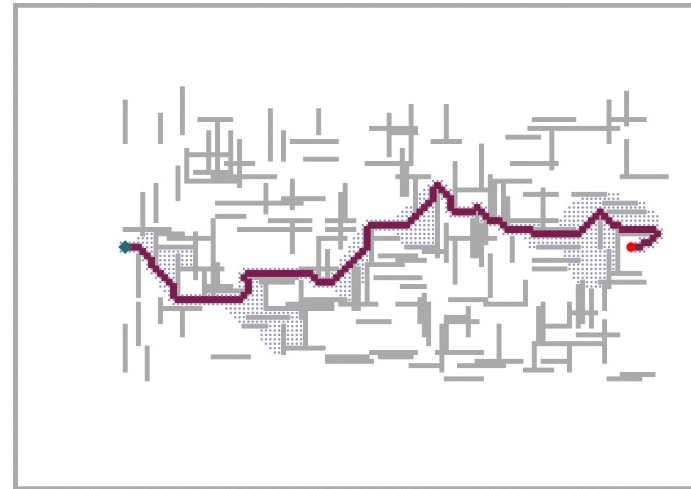
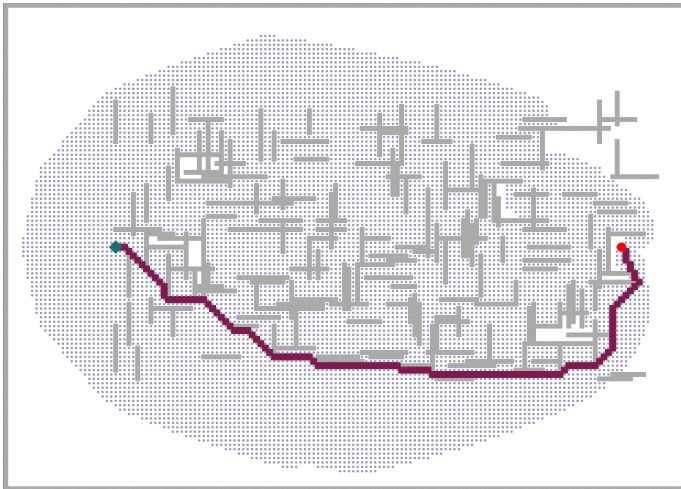
$$h(n) = \max\{h_1(n), \cdots, h_m(n)\}$$

Definition: dominance

$$h_1 \geq h_2 \text{ if} \\ \forall n \ h_1(n) \geq h_2(n)$$

Weighted A* Search

- Evaluate states by various ways $f(n) = g(n) + W * h(n)$
 - A* Search: $W = 1$
 - Uniform-cost search: $W = 0$
 - Greedy best-first search: $W = \infty$
 - Weighted A* search: $1 < W < \infty$



Outline

- Search Problems
- General Search
 - Tree Search, Graph Search
- Uninformed Search
 - Depth-First (DFS), Breadth-First (BFS), Uniform-Cost (UCS)
- Informed (Heuristic) Search
 - A* Tree Search
 - **A* Graph Search**

Uniform Cost (Graph) Search

1. **function** UNIFORM-COST-SEARCH(*problem*) **returns** a solution, or *failure*
2. initialize the *explored set* to be empty
3. initialize the *frontier* as a priority queue using **node path-cost** as the priority
4. add initial state of *problem* to *frontier* with **path-cost = 0** (1). Initialization
5. **loop do**
6. **if** the *frontier* is empty **then return** *failure*
7. choose a *node* and remove it from the *frontier*
8. **if** the *node* contains a goal state **then**
9. **return** the corresponding solution (2). Selection
10. add the *node* state to the *explored set*
11. **for each** resulting *child* from node
12. **if** the *child* state is not already in the *frontier* or *explored set* **then**
13. add *child* to the *frontier*
14. **else if** the *child* is already in the *frontier* with higher **path-cost** **then**
15. replace that *frontier* node with *child* (3). Expansion

Key idea: extra check

A shorter path to a *frontier* state is discovered.



A* Graph Search

1. **function** A-STAR-SEARCH(*problem*) **returns** a solution, or *failure*
2. initialize the *explored set* to be empty
3. initialize the *frontier* as a priority queue using $f(n) = g(n) + h(n)$ as the priority
4. add initial state of *problem* to *frontier* with priority $f(S) = 0 + h(S)$ (1). Initialization
5. **loop do**
6. **if** the *frontier* is empty **then return** *failure*
7. choose a *node* and remove it from the *frontier*
8. **if** the *node* contains a goal state **then**
9. **return** the corresponding solution (2). Selection
10. add the *node* state to the *explored set*
11. **for each** resulting *child* from node
12. **if** the *child* state is not already in the *frontier* or *explored set* **then**
13. add *child* to the *frontier*
14. **else if** the *child* is already in the *frontier* with higher $f(n)$ **then**
15. replace that *frontier* node with *child* (3). Expansion

Key idea: extra check

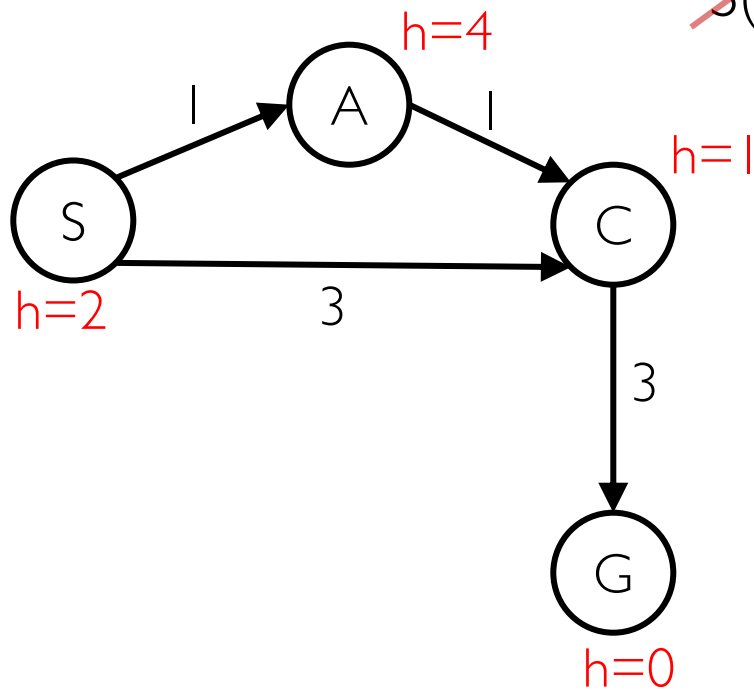
A path with shorter estimation to a frontier state is discovered



A* Graph Search Gone Wrong

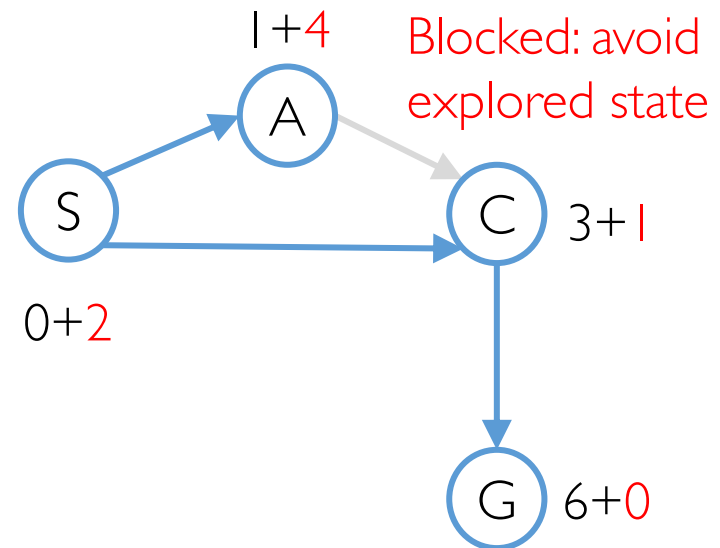
State space graph

Queue



~~$S(0+2)$~~ , ~~$S-C(3+1)$~~ , ~~$S-A(1+4)$~~ , $S-C-G(6+0)$

Search Tree



Consistency of Heuristics

- **Admissibility** (可采纳性): heuristic cost \leq actual cost to goal
- **Consistency** (一致性): heuristic "arc" cost \leq actual cost for each arc

$$h(n) - h(n') \leq c(n, a, n')$$

- Consequences of consistency

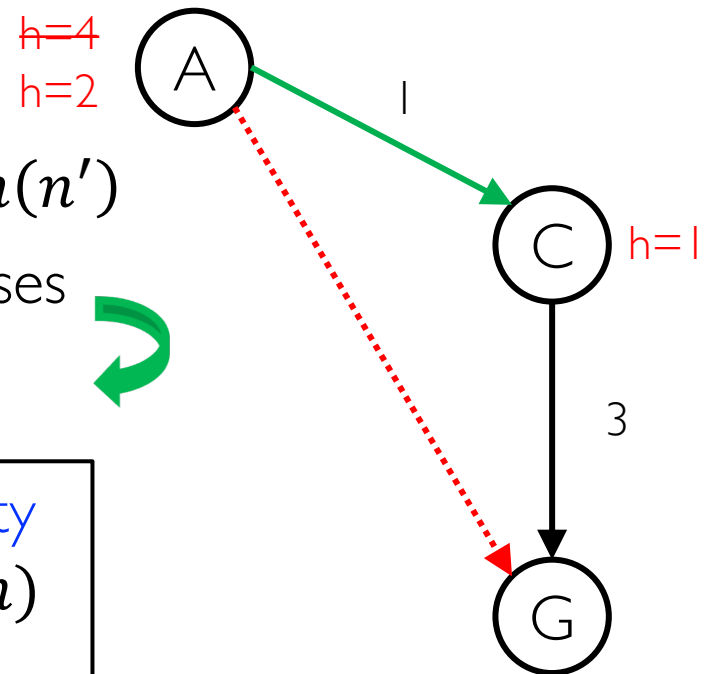
- Triangle inequality

$$h(n) \leq c(n, a, n') + h(n')$$

- The f value along a path never decreases
- A* graph search is **optimal**

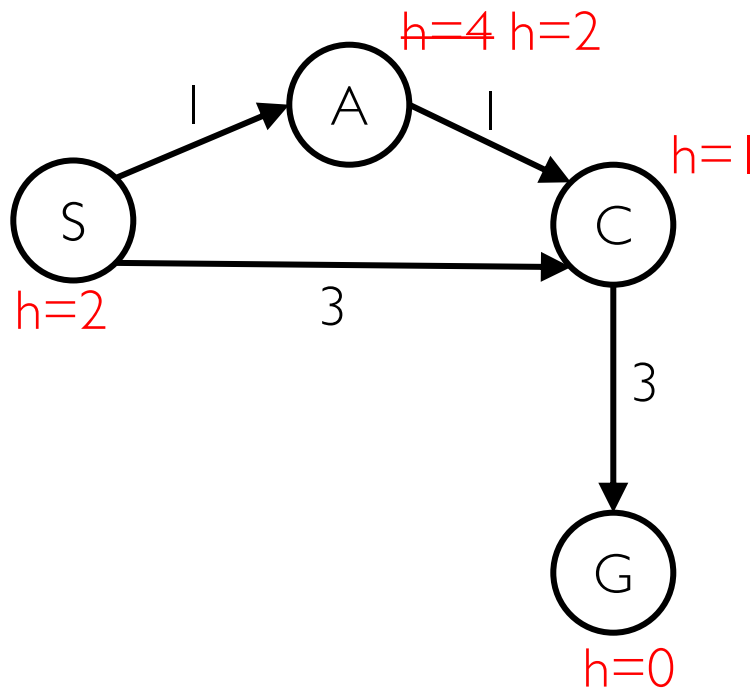
Theorem: consistency implies admissibility

If a heuristic $h(n)$ is consistent, then $h(n)$ is admissible.



A* Graph Search Gone Right

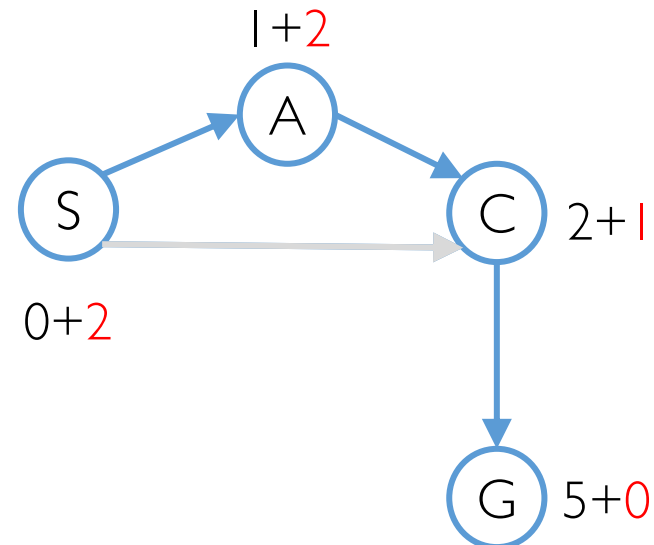
State space graph



Queue

~~$S(0+2)$~~ , ~~$S-A(1+2)$~~ , ~~$S-A-C(2+1)$~~
 $S-A-C-G(5+0)$

Search Tree



Optimality of A* Graph Search

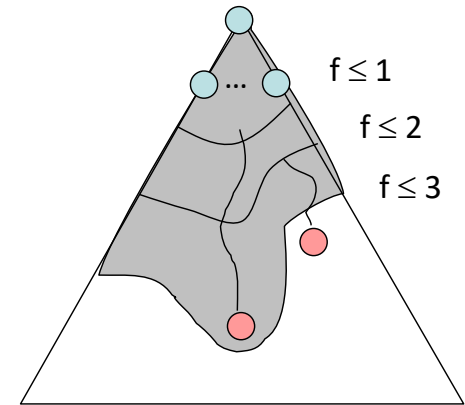


- Step I: if $h(n)$ is consistent, then the values of $f(n)$ along any path are nondecreasing.

$$f(n') = g(n') + h(n') = g(n) + c(n, a, n') + h(n') \geq g(n) + h(n) = f(n)$$

- Step II: whenever A* selects any node n for expansion, the optimal path to that node has been found.

- Intuition: replace g in UCS with f
- A node n with minimum $f(n)$ corresponds to a path to n with minimum $g(n)$, because $h(n)$ are always the same



- Step III: the first goal node selected for expansion must be an optimal solution because f is the true cost for goal nodes which have $h = 0$

Applications of A* Search

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- Database query optimization
- ...



Example: pathfinding in games

A* Search: Summary

- A* search uses both **backward** costs and (estimates of) **forward** costs
- A* search is optimal with admissible / consistent heuristics
 - **Admissibility** guarantees optimality of **A* Tree Search**
 - **Consistency** guarantees optimality of **A* Graph Search**
- Implementation: replace path cost in UCS with $f = g + h$
- Heuristic design is key: use **relaxed problem** or **learn from experience**

Thank You

Questions?

Mingsheng Long

mingsheng@tsinghua.edu.cn

<http://ise.thss.tsinghua.edu.cn/~mlong>

答疑：东主楼11区413室

[Some slides adapted from Dan Klein and Pieter Abbeel]