

States, actions, goals: formulating problems

Before a computer can solve a problem, someone has to write it down properly. Today we learn to write it down, then take our first steps searching it.

01 COLD OPEN 0–4 min

A bomb, a fountain and two jugs

Die Hard with a Vengeance (1995). Simon's riddle: a **5-gallon** jug, a **3-gallon** jug, a fountain. Put **exactly 4 gallons** on the scale or the bomb goes off.

McClane and Zeus shout at each other for a full minute. A computer would not shout. It would **try things systematically** until something works. That is search, and it only works if the problem is written down precisely first.

Today's version: 4 L and 3 L jugs, and you need exactly 2 L. How many pours at minimum?



POP CULTURE

Same idea, different costume: Google Maps getting you from Mirpur 10 to Motijheel, a Rubik's cube solver, the fastest route through your Among Us tasks. Different problems, one recipe: **states, actions, a goal, a cost.**

6 actions, and no plan with 5 exists. By the end of the lecture you will be able to prove it, because a computer checked every possibility.

02 CONCEPT 4–8 min

The problem-solving agent

A goal-based agent that plans a whole sequence of actions before moving works in four phases:

1

Goal formulation

"Be in Cox's Bazar by tonight"

2

Problem formulation

Which states and actions to consider

3

Search

Simulate action sequences until one reaches the goal

4

Execution

Carry out the plan, step by step



GOTCHA

Executing the plan without looking is **open-loop** control. It is safe only if the environment is **fully observable, deterministic and known** (last week's properties). A flooded road on the way breaks it.

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept)
  state ← UPDATE-STATE(state, percept)
  if plan is empty:
    goal ← FORMULATE-GOAL(state)
    problem ← FORMULATE-PROBLEM(state, goal)
    plan ← SEARCH(problem)
    if plan = failure: return NoOp
  action ← FIRST(plan); plan ← REST(plan)
  return action
```

Adapted from Russell & Norvig's simple problem-solving agent. Everything today lives in

`FORMULATE-PROBLEM`; Weeks 2–4 live in `SEARCH`.

03 **CONCEPT** 8–13 min

A problem, in five parts

PART	NOTATION	DHAKA → COX'S BAZAR
1 · Initial state	<code>s₀</code>	In(Dhaka)
2 · Actions	<code>ACTIONS(s)</code>	{Go(Cumilla), Go(Chandpur), Go(Brahmanbaria)}
3 · Transition model	<code>RESULT(s, a)</code>	RESULT(In(Dhaka), Go(Cumilla)) = In(Cumilla)
4 · Goal test	<code>IS-GOAL(s)</code>	s = In(Cox's Bazar)
5 · Path cost	$\sum c(s, a, s')$	sum of road km (or minutes, or taka)

Parts 1–3 together define the **state space**: every state reachable from s_0 by some sequence of actions.

SOLUTION

A **sequence of actions** from the initial state to a goal state.

OPTIMAL SOLUTION

A solution with the **lowest path cost** among all solutions.



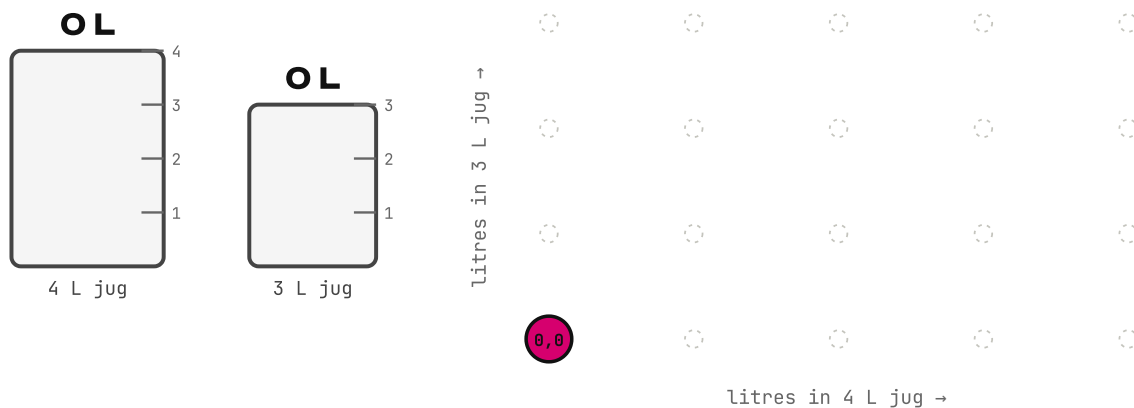
PRO TIP

A solution is a *plan* (actions), not a single final state. "Cox's Bazar" is not an answer; "Cumilla, Feni, Chattogram, Cox's Bazar" is.

04 **LIVE LAB** 13-18 min

Live lab: explore the water-jug state space

Water-jug state space State = (litres in 4 L jug, litres in 3 L jug). Goal: exactly 2 L in the 4 L jug.



STATE

(0, 0)

ACTIONS TAKEN

0

STATES FOUND

1 / 14

Click an action. Every new state you reach appears in the graph.

Click actions and watch the **state graph** grow: each new (x, y) you reach becomes a node.

Then press **Solve with BFS**: it explores level by level and returns the shortest plan.



PRO TIP

Find all 14 states. Why are 6 grid points never reachable?

05 **WORKED EXAMPLE** 18-23 min

Worked example: formulating the water jug

GIVEN

A 4 L jug and a 3 L jug, no markings, a tap with unlimited water, a drain. You can fill a jug, empty it, or pour one into the other until the source is empty or the target is full.

FIND

All five parts of the formulation, and an optimal solution.

Method: Rich & Knight's production rules; Russell & Norvig's five-part template

SOLUTION

1. **States:** (x, y) = litres in the 4 L and 3 L jugs, $x \in \{0, \dots, 4\}$, $y \in \{0, \dots, 3\}$: 20 pairs. **Initial:** $(0, 0)$.
2. **Actions:** Fill4, Fill3, Empty4, Empty3, Pour4 $\rightarrow$ 3, Pour3 $\rightarrow$ 4, only if they change s. $\text{ACTIONS}((0, 0)) = \{\text{Fill4}, \text{Fill3}\}$.
3. **Transition model** (pouring stops when the source is empty or the target full):
$$\text{RESULT}((x, y), \text{Pour4} \rightarrow 3) = (x - t, y + t), t = \min(x, 3 - y)$$
$$\text{RESULT}((x, y), \text{Pour3} \rightarrow 4) = (x + t, y - t), t = \min(y, 4 - x)$$
4. **Goal test:** $x = 2$. **Path cost:** 1 per action.
5. **BFS (unit costs $\rightarrow$ optimal):** $(0,0) \rightarrow \text{Fill4} \rightarrow (4,0) \rightarrow \text{Pour} \rightarrow (1,3) \rightarrow \text{Empty3} \rightarrow (1,0) \rightarrow \text{Pour} \rightarrow (0,1) \rightarrow \text{Fill4} \rightarrow (4,1) \rightarrow \text{Pour} \rightarrow (2,3)$. No 5-step plan exists.

ANSWER

6 actions: Fill4, Pour, Empty3, Pour, Fill4, Pour. Only 14 of the 20 pairs are reachable.

06 CONCEPT 23-27 min

Abstraction: what to leave out

The real trip to Cox's Bazar involves the bus company, the weather, the tea stop and 400 km of potholes. Our state is one word: **In(city)**.

Abstraction = removing detail. It is *valid* if every abstract solution expands into a real one, *useful* if each abstract action is easier than the original problem.

The state space is then a **graph**: nodes are states, edges are actions, weights are step costs. Search = finding a path in that graph *without drawing all of it*.

THINK DEEPER

Why is the state for route finding just 'which city I am in', and not 'the whole map'?

The map is fixed knowledge that the **transition model** uses. The state only needs what actions change: your location. Copying the map into every node costs memory and adds nothing, unless the map itself can change (a road flooded in monsoon); then that part moves into the state.

A STRONG ANSWER MENTIONS

- The map does not change when you act: it belongs in the transition model, not the state.
- The state holds only what actions change and what the goal test needs.
- If roads can close (floods, hartal), road status becomes part of the state.

07 LIVE LAB 27-32 min

Live lab: the 8-puzzle

8-puzzle State = where every tile is. Actions = move the blank Up / Down / Left / Right.



MOVES

0

BRANCHING
HERE

b = 2

MISPLACED
TILES

0

INVERSIONS

**0 → even:
solvable**

Arrangements	$9! = 362,880$
Reachable from goal	$9!/2 = 181,440$
Branching factor	2, 3 or 4 (avg ≈ 2.67)
Hardest instance	31 moves (optimal)

Click a tile next to the blank to slide it.

States: location of each of the 8 tiles and the blank.

Initial: any state (but see parity!).

Actions: move the *blank* Left, Right, Up, Down. ACTIONS(s) depends on where the blank is.

Goal test: matches 1-8 in order, blank last.

Path cost: 1 per move.



POP CULTURE

In the 1880s Sam Loyd offered \$1,000 for solving the 15-puzzle with 14 and 15 swapped. Safe money: it is on the wrong side of the parity split.

Sundarbans crossing (Life of Pi edition)

Three **bawalis** (wood-cutters) and three **tigers** must cross a khal in a boat that holds at most two. Anyone can row (it is a puzzle). If on either bank the tigers ever outnumber the bawalis present there, lunch is served.

The classic "missionaries and cannibals", with better scenery.

State: (B, T, boat) = bawalis and tigers on the *start* bank, boat side (L/R). The other bank is implied.

Initial: (3, 3, L). **Goal:** (0, 0, R).

Actions: the boat carries {1B, 2B, 1T, 2T, 1B+1T}, only if the result is safe on both banks.

Cost: 1 per crossing.

ALL (B, T, BOAT)

4 × 4 × 2 = 32

SAFE

20

REACHABLE

16

(3, 3, L) → 2T → (3, 1, R) → 1T back → (3, 2, L) → 2T → (3, 0, R) → 1T back →
 (3, 1, L) → 2B → (1, 1, R) → 1B+1T back → (2, 2, L) → 2B → (0, 2, R) → 1T back
 → (0, 3, L) → 2T → (0, 1, R) → 1B back → (1, 1, L) → 1B+1T → (0, 0, R)

11 crossings, and BFS proves none shorter exists.

Your turn • 1: eight queens, two ways

1 EXAM CQ

Place 8 queens on a chessboard so that no two attack each other (same row, column or diagonal).

- Give an **incremental** formulation and estimate how many action sequences a naive version must consider.
- Improve it so the state space is tiny.
- Give a **complete-state** formulation and its number of states. Which formulation suits which kind of search?

SOLUTION

- Naive incremental.** State: 0–8 queens anywhere; initial: empty board; action: add a queen to any empty square; goal: 8 non-attacking queens.

$$64 \times 63 \times \cdots \times 57 \approx 1.8 \times 10^{14} \text{ sequences}$$

- Smart incremental.** Action: add a queen to the leftmost empty column, on a square no queen attacks. Only **2,057** states: 10^{11} times smaller.
- Complete-state.** 8 queens, one per column; action: move a queen within its column (56 successors); goal: no attacks.

$$8^8 = 16,777,216 \quad \left(\binom{64}{8} \approx 4.4 \times 10^9 \text{ without the column rule}\right)$$

- Smart incremental suits DFS/backtracking. Complete-state suits **local search** (Week 3): always a full board, repair conflicts.

ANSWER

$\approx 1.8 \times 10^{14}$ naive sequences $\rightarrow$ 2,057 smart states; complete-state $8^8 \approx 1.7 \times 10^7$. 92 solutions.

Why state-space size matters: b^d

If every state has b successors (branching factor) and the shallowest goal is at depth d , a level-by-level search generates up to

$$1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} - 1}{b - 1} = O(b^d)$$

D	NODES (B = 10)	TIME	MEMORY
4	11,111	11 ms	11 MB
6	$\approx 10^6$	1.1 s	1.1 GB
8	$\approx 10^8$	2 min	111 GB
10	$\approx 10^{10}$	3 hours	11 TB
12	$\approx 10^{12}$	13 days	1 PB

Assumes 10^6 nodes per second and 1 KB per node.

PROBLEM	STATES
Water jug (4, 3)	14
Sundarbans crossing	16
8-queens (smart)	2,057
8-puzzle	181,440
15-puzzle	$\approx 1.05 \times 10^{13}$
Rubik's cube	$\approx 4.3 \times 10^{19}$


GOTCHA

Memory runs out long before patience does. That single fact drives next week (IDS).

11 **WORKED EXAMPLE** 43-48 min

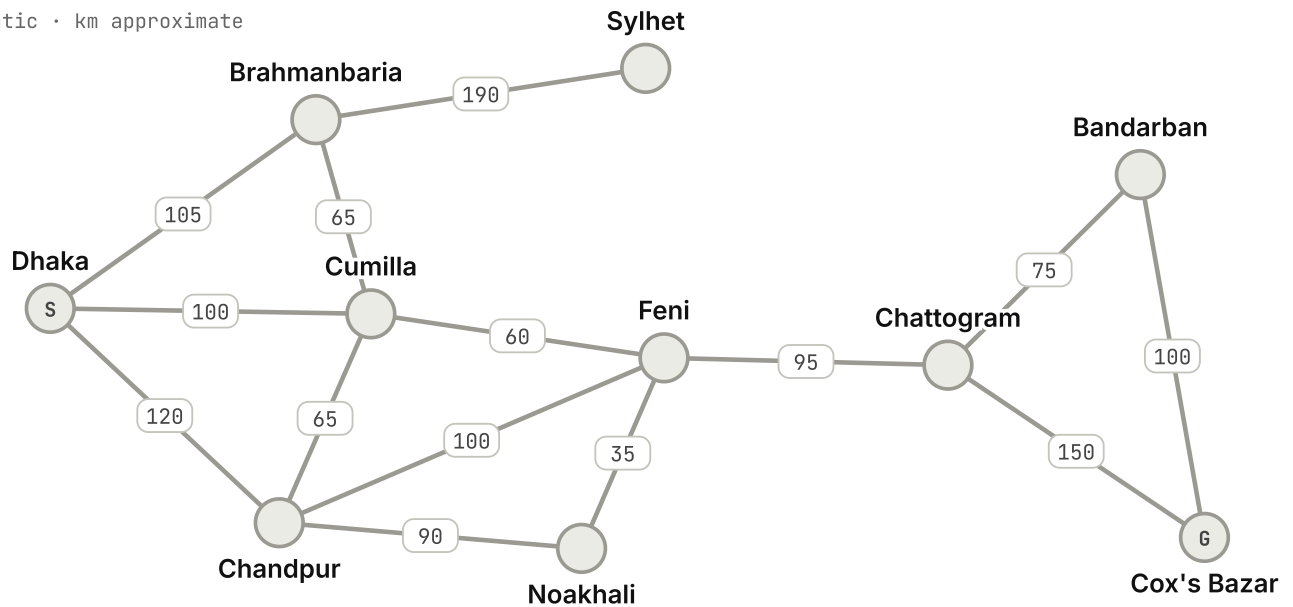
Worked example: route finding, BFS by hand

GIVEN

The map below (km approximate). Start Dhaka, goal Cox's Bazar.

FIND

BFS expansion order, and the route returned with its cost.



Method: graph-search BFS (Russell & Norvig)

SOLUTION

1. State $In(city)$, cost = road km (slide 3). **Rules:** alphabetical; skip explored or queued children; goal test on *pop*.
2. Pop **Dhaka** → [BRB, CHP, CUM]. Pop **Brahmanbaria** → +SYL (CUM already queued) → [CHP, CUM, SYL].
3. Pop **Chandpur** → +FEN, +NOA → [CUM, SYL, FEN, NOA]. *Feni's parent = Chandpur*: this decides the route.
4. Pop CUM, SYL (nothing new). Pop **Feni** → +CTG. Pop NOA. Pop **Chattogram** → +BAN, +COX. Pop BAN. Pop **COX**: goal.
5. Follow parents back from the goal:

$$DHK \xrightarrow{120} CHP \xrightarrow{100} FEN \xrightarrow{95} CTG \xrightarrow{150} COX = 465 \text{ km}$$

ANSWER

DHK, BRB, CHP, CUM, SYL, FEN, NOA, CTG, BAN, COX. Route: 4 hops, 465 km.

12 **LIVE LAB** 48-54 min

Live lab: BFS vs DFS on

the road to Cox's Bazar

BFS vs DFS: Dhaka → Cox's Bazar

Step through the search. Watch how the frontier data structure alone changes which route you get.

schematic · road km approximate

frontier

explored

being expanded

parent link

solution

FRONTIER

FIFO queue: pop the front (left)

DHK

EXPLORED SET

never expand these again

empty

init

Start: frontier = [Dhaka]

EXPANDED

0

MAX FRONTIER

1

COST

—

HOPS

—

13 **CONCEPT** 54–57 min

Judging a search strategy: four questions

CRITERION	BFS (FIFO)	DFS (LIFO)
Complete? finds a solution if one exists	Yes, if b is finite	Graph search: yes in finite spaces. No in infinite ones (or with loops, as tree search)
Optimal? finds the cheapest	Yes, if all step costs are equal	No

CRITERION	BFS (FIFO)	DFS (LIFO)
Time	$O(b^d)$	$O(b^m)$
Space	$O(b^d)$: keeps a whole level	$O(bm)$: one path plus siblings

```
node = frontier.pop(0) # BFS: oldest first (queue)
node = frontier.pop()  # DFS: newest first (stack)
```

In Python, that one argument is the whole difference between the two algorithms.

b = branching factor (max successors)

d = depth of the shallowest goal

m = maximum depth of any path (may be ∞)



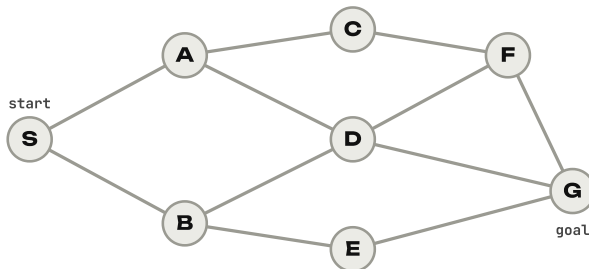
PRO TIP

Next week's question: can we get DFS's $O(bm)$ memory *and* BFS's completeness and optimality? Yes: iterative deepening.

Your turn · 2: trace DFS by hand

2

EXAM CQ



All edges cost 1. Run **graph-search DFS** from S to G. Break ties alphabetically; goal test when a node is popped.

Give the expansion order, the stack after each expansion, and the path returned. Then compare with BFS.

SOLUTION

1. Pop S → push B, A. Stack [B, **A**]
 2. Pop A → push D, C. Stack [B, D, **C**]
 3. Pop C → push F. Stack [B, D, **F**]
 4. Pop F → push G, D (D moves up). Stack [B, G, **D**]
 5. Pop D → push G, B (both move up). Stack [G, **B**]
 6. Pop B → push E. Stack [G, **E**]. Pop E → push G. Stack [**G**]. Pop G: goal.
 7. BFS on the same graph expands S, A, B, C, D, E, F, G and returns S → A → D → G.
- DFS was "lucky" to find G at all quickly, but its path is 7 edges long against BFS's 3. Not optimal.

ANSWER

DFS expansion order: S, A, C, F, D, B, E, G. Path returned: S → A → C → F → D → B → E → G (cost 7). BFS path: S → A → D → G (cost 3).

Critical thinking · is BFS ever wrong?

3 TRAP

A classmate says: "BFS always finds the shortest path, so Google Maps could just run BFS."

Using the Dhaka → Cox's Bazar map, is BFS's route optimal? Under what condition is BFS optimal, and what would you use instead?

SOLUTION

1. BFS returned Dhaka → Chandpur → Feni → Chattogram → Cox's Bazar = 120 + 100 + 95 + 150 = **465 km**.
2. Dhaka → Cumilla → Feni → Chattogram → Cox's Bazar = 100 + 60 + 95 + 150 = **405 km**. Same 4 hops, 60 km shorter.
3. BFS minimises the *number of actions*. That equals minimum cost only when every step costs the same (or cost never decreases with depth).
4. Fix: expand the node with the lowest path cost $g(n)$ first: **uniform-cost search** (next week). It returns 405 km.

Hack that also works: split every road into 5 km pieces so all steps cost the same. BFS is then optimal, but the map grows from 10 nodes to 266. Correct but silly.

ANSWER

No: BFS is optimal only with equal step costs. Here it returns 465 km while 405 km exists. Use UCS for weighted graphs.

Planning with STRIPS: states as sets of facts

STRIPS (Fikes & Nilsson, SRI, 1971) writes each action as three lists: **PRE** facts that must hold, **ADD** facts made true, **DEL** facts made false. Anything not mentioned stays the same.

$$s' = (s \setminus \text{Del}) \cup \text{Add}$$

STRIPS kitchen: one cup of cha State = a set of true facts. An action applies only if all its preconditions are in the state; then delete-list out, add-list in.

CURRENT STATE

KettleEmpty StoveOff

CupEmpty HaveLeaves

HaveMilk HaveSugar

GOAL (ALL MUST HOLD)

○ CupHasTea

○ MilkIn

○ SugarIn

✓ StoveOff

PLAN empty: click actions →

FillKettle

pre KettleEmpty
add KettleFull
del KettleEmpty

LightStove

pre StoveOff
add StoveOn
del StoveOff

Boil

pre KettleFull, StoveOn
add WaterHot
del -
missing: KettleFull, StoveOn

Brew

pre WaterHot, HaveLeaves, CupEmpty
add CupHasTea, KettleEmpty
del CupEmpty, HaveLeaves, WaterHot, KettleFull
missing: WaterHot

AddMilk

pre CupHasTea, HaveMilk
add MilkIn
del HaveMilk
missing: CupHasTea

AddSugar

pre CupHasTea, HaveSugar
add SugarIn
del HaveSugar
missing: CupHasTea

TurnOffStove

pre StoveOn
add StoveOff
del StoveOn
missing: StoveOn

17 **YOUR TURN** 67-70 min

Your turn · 3: a hospital delivery robot

4 DESIGN

A robot carries medicine between rooms of a hospital (Store, Pharmacy, Ward3; Store–Pharmacy and Pharmacy–Ward3 are adjacent). It carries one package at a time.

Initial: $At(R, Store)$, $At(M, Pharmacy)$, $HandEmpty$, plus the Adj facts. Goal: $At(M, Ward3)$.

Write STRIPS operators `Move`, `PickUp`, `Drop` and a plan.

SOLUTION

1. `Move(x, y)` — PRE: $At(R, x)$, $Adj(x, y)$ · ADD: $At(R, y)$ · DEL: $At(R, x)$
2. `PickUp(p, x)` — PRE: $At(R, x)$, $At(p, x)$, $HandEmpty$ · ADD: $Holding(p)$ · DEL: $At(p, x)$, $HandEmpty$
3. `Drop(p, x)` — PRE: $At(R, x)$, $Holding(p)$ · ADD: $At(p, x)$, $HandEmpty$ · DEL: $Holding(p)$
4. **Plan:** `Move(Store, Pharmacy)`, `PickUp(M, Pharmacy)`, `Move(Pharmacy, Ward3)`, `Drop(M, Ward3)`.

Forget DEL $At(R, x)$ and the robot is in two places at once. Forget $HandEmpty$ and it carries every package in the building.

ANSWER

Three operators as above; a 4-action plan reaches $At(M, Ward3)$.

18 REAL WORLD 70–73 min

Real world: planning a spray drone's day



FROM THE STARTUP

An agro-drone spraying a paddy field must cover every cell with enough battery and pesticide. Naive state: (cell, battery, tank, *set of sprayed cells*). A 20×20 field has 400 cells, so the "sprayed set" alone has $2^{400} \approx 2.6 \times 10^{120}$ values.

Real planners **abstract**: fix a lawn-mower (boustrophedon) sweep, so the state becomes (row, battery, tank) and the only decision is *when to return to refill*. Ambulances in Dhaka: state (junction, time of day), step costs from live traffic, not km.

THINK DEEPER

Our formulation assumed a fully observable, deterministic, known world. Is the real world ever like that?

Rarely. Wind pushes the drone, a lorry blocks Jatrabari. But the formulation is still the right **first model**: plan, execute a step, **observe, re-plan if reality diverged**. When uncertainty is large, the state becomes a belief (probabilities), which we meet in Week 8.

A STRONG ANSWER MENTIONS

- Almost never exactly: wind, GPS error, traffic jams, road closures.
- The formulation is still useful if the plan is re-checked and re-planned (closed loop).
- When uncertainty dominates, the state must hold beliefs, and actions need contingencies.

19 POP QUIZ 73-78 min

Pop quiz

1. Which of these is *not* one of the five parts of a problem formulation?

- A. Initial state
- B. Transition model RESULT(s, a)

C. The search algorithm ✓

- D. Path cost

The formulation describes the problem. The algorithm (BFS, DFS, UCS...) is chosen afterwards and works on any well-formulated problem.

2. How many states of the 8-puzzle can be reached from a given start state?

- A. $9! = 362,880$

B. $9!/2 = 181,440$ ✓

- C. $8! = 40,320$
- D. $2^9 = 512$

Every move keeps the permutation parity (with the blank's position) fixed, so the $9!$ arrangements split into two halves that never meet.

3. Replace BFS's FIFO queue with a LIFO stack, change nothing else. You now have...

- A. Uniform-cost search
- B. Depth-first search ✓**
- C. Still BFS
- D. A* search

The frontier discipline alone decides the strategy: FIFO $\rightarrow$ breadth-first, LIFO $\rightarrow$ depth-first, priority queue on $g(n)$ $\rightarrow$ uniform-cost.

4. BFS on our Dhaka → Cox's Bazar map returns a 465 km route although a 405 km route exists.

Why?

A. BFS is incomplete

B. BFS minimises the number of steps, not the kilometres ✓

C. The explored set is wrong

D. Alphabetical ordering is illegal

BFS is optimal only when every step costs the same. Both routes have 4 hops; BFS stopped at the first one it found.

5. In STRIPS, when can an action be applied in state s ?

A. When its add list is in s

B. When all its preconditions are true in s ✓

C. When its delete list is empty

D. Always; it just fails silently

Applicable $\Leftrightarrow$ precondition $\subseteq s$. The result is $(s - \text{Del}) \cup \text{Add}$.

20 **RECAP** 78–80 min

Three things to remember

01

A problem = **initial state, ACTIONS, RESULT, goal test, path cost**. A solution is an action sequence; optimal = lowest cost.

02

Formulation and abstraction decide the **state-space size** (8-queens: 10^{14} vs 2,057). Search cost grows like b^d .

03

BFS = FIFO queue, complete, optimal for equal costs, $O(b^d)$ memory. DFS = LIFO stack, $O(bm)$ memory, not optimal.

HW

BEFORE NEXT CLASS

Formulate the Die Hard 3 riddle (5 and 3 gallon jugs, measure exactly 4 in the big jug) in five parts. Find an optimal solution and count the reachable states. **Assignment 1 is published this week**; Class Test 1 next week covers UCS, DLS, IDS and local search, and assumes today's BFS/DFS.

SOLUTION

1. State (x, y) with $x \in \{0, \dots, 5\}$, $y \in \{0, \dots, 3\}$; initial $(0, 0)$; same six actions; goal $x = 4$; cost 1 per action.
2. BFS: Fill5 $\rightarrow (5, 0)$, Pour5 $\rightarrow$ 3 $\rightarrow (2, 3)$, Empty3 $\rightarrow (2, 0)$, Pour5 $\rightarrow$ 3 $\rightarrow (0, 2)$, Fill5 $\rightarrow (5, 2)$, Pour5 $\rightarrow$ 3 $\rightarrow (4, 3)$.
3. Reachable states: 16 of the 24 pairs.

ANSWER

6 actions (McClane's answer). 16 reachable states.

S. M. Sakeef Sani

Adjunct Lecturer, CSE · Northern University Bangladesh

smsakeefsani.hbitcorp.com