

UCS, DLS and IDS

Last week BFS found a route that cost 60 km too much, and DFS wandered off. Today: a search that minimises cost, and one that has BFS's guarantees on DFS's memory budget.

01 COLD OPEN 0–4 min

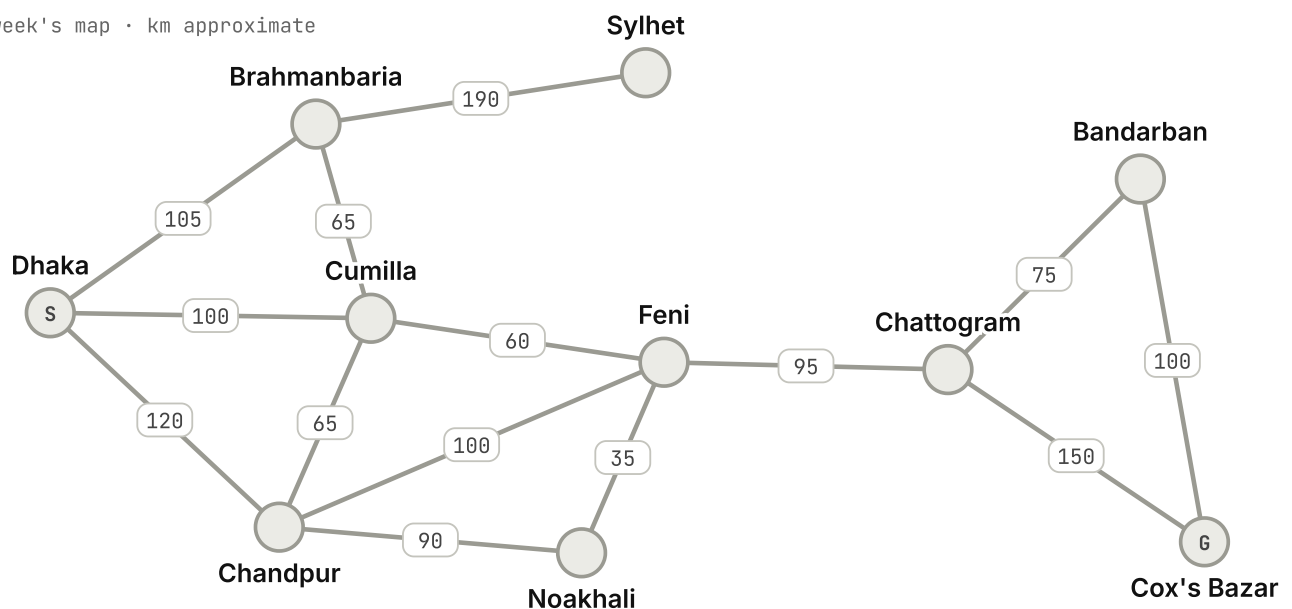
60 km of wasted diesel

Last week's BFS sent us Dhaka → Chandpur → Feni → Chattogram → Cox's Bazar: **465 km**. Via Cumilla it is **405 km**.

Same number of hops, so BFS could not tell them apart. An ambulance, a Pathao rider or a cargo truck pays for *kilometres* (or minutes, or taka), not for hops.

Today's question: if roads have different costs, which city should a search look at next?

last week's map · km approximate



The one that is **cheapest to reach so far**: Cumilla (100 km), not the alphabetically first or the first discovered. That single rule is uniform-cost search.

Uniform-cost search: cheapest first

```

function UNIFORM-COST-SEARCH(problem)
  frontier  $\leftarrow$  priority queue ordered by g, holding  $s_0$  ( $g = 0$ )
  explored  $\leftarrow \emptyset$ 
  loop
    if frontier is empty: return failure
    node  $\leftarrow$  POP-LOWEST-g(frontier)
    if IS-GOAL(node.state): return SOLUTION(node)    # test on POP
    add node.state to explored
    for each child of node:
      if child.state not in explored or frontier: insert child
      else if it is in frontier with a higher g: replace that entry

```

$g(n)$ = cost of the path from the start to n . Adapted from Russell & Norvig's uniform-cost search.

TWO CHANGES FROM BFS

1. The FIFO queue becomes a **priority queue on $g(n)$** .
 2. The goal test moves to when a node is **popped**, not when it is generated.
- (Plus: if a cheaper path to a frontier state turns up, keep the cheaper one.)



POP CULTURE

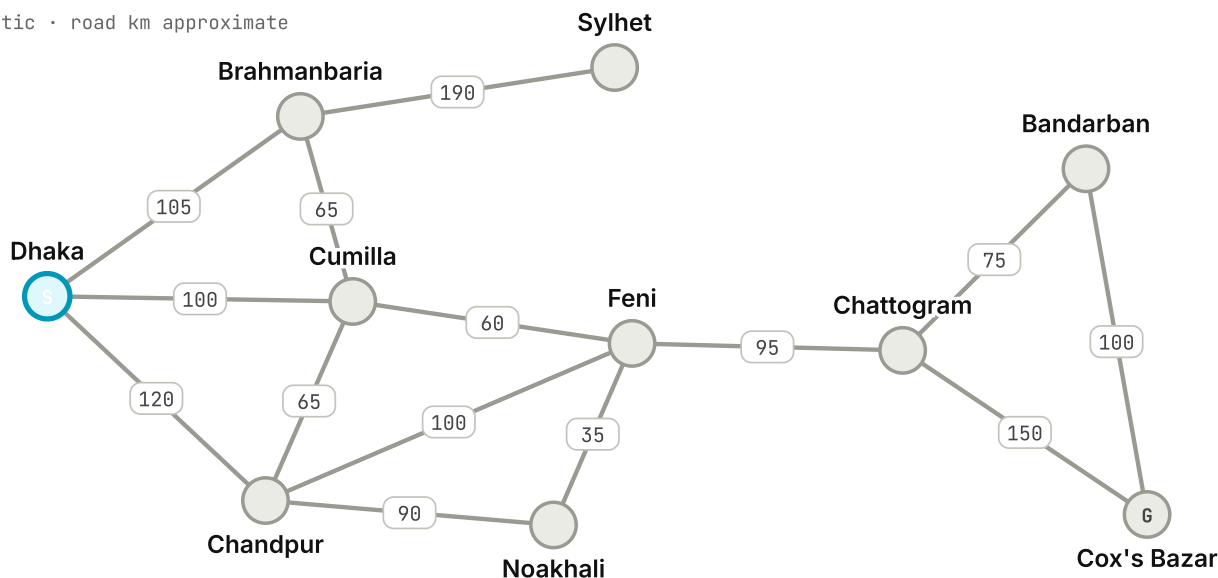
Edsger Dijkstra designed his shortest-path algorithm in about twenty minutes, sitting at a café in Amsterdam in 1956, without pencil and paper. UCS is Dijkstra's algorithm that stops as soon as the goal is popped.

Live lab: UCS on the road to Cox's Bazar

UCS vs BFS: Dhaka → Cox's Bazar

The frontier is now sorted by g (km so far). Watch Cox's Bazar sit on the frontier at 405 km while cheaper cities are popped first.

schematic · road km approximate



● frontier ● explored ● being expanded — parent link ● solution

FRONTIER

priority queue: pop lowest g

DHK 0

EXPLORED SET

never expand these again

empty

init Start: frontier = [Dhaka]

EXPANDED

0

MAX FRONTIER

1

COST

—

HOPS

—

Worked example: UCS by hand

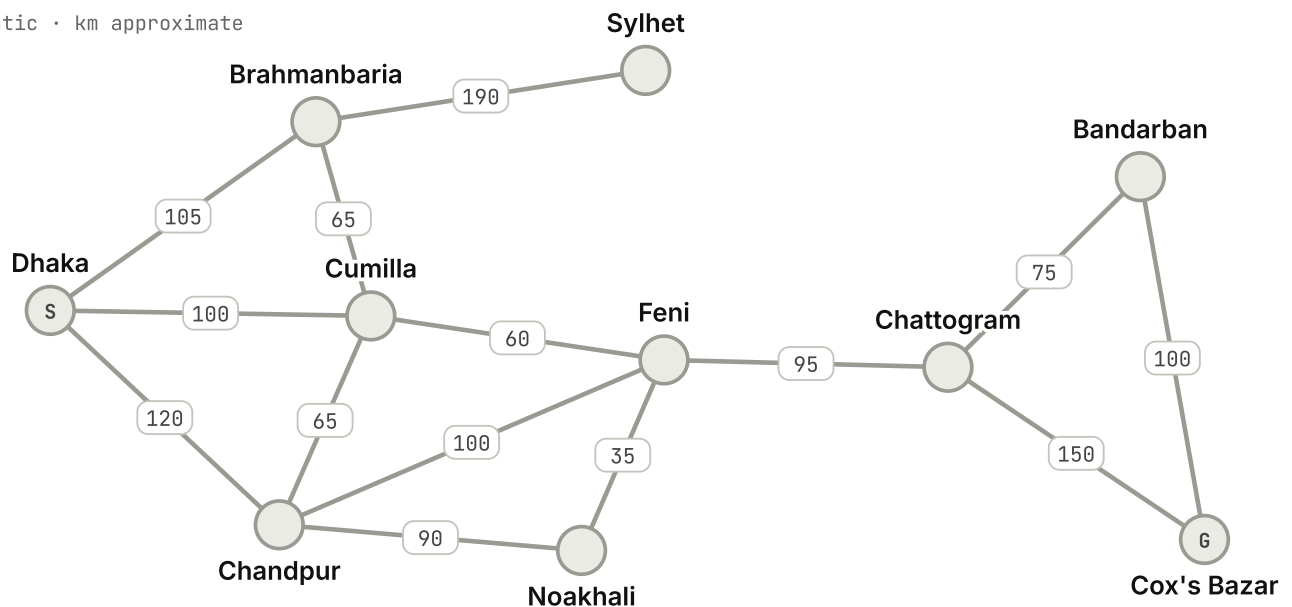
GIVEN

Last week's map (km approximate). Start Dhaka, goal Cox's Bazar. Graph search, ties broken alphabetically.

FIND

The expansion order, the frontier after each expansion, and the route returned.

schematic · km approximate



Method: uniform-cost graph search (Russell & Norvig)

SOLUTION

- Pop **DHK (0)**. Frontier, cheapest first: CUM 100, BRB 105, CHP 120.
Rules: goal test on *pop*; skip explored states; if a state is already on the frontier, keep whichever path is cheaper.
- Pop **CUM (100)** → add FEN 160. BRB via CUM would be 165 > 105 and CHP via CUM 165 > 120: keep the old entries.
- Pop **BRB (105)** → add SYL 295. Pop **CHP (120)** → add NOA 210 (FEN via CHP = 220 > 160, ignored).
- Pop **FEN (160)** → add CTG 255. NOA via FEN = 160 + 35 = **195 < 210**: replace NOA's entry, parent now FEN.
This "decrease-key" is the only bookkeeping UCS adds to BFS.
- Pop **NOA (195)**: nothing new. Pop **CTG (255)** → add BAN 330, **COX 405**. The goal is on the frontier, but we do *not* stop yet.
- Pop SYL (295), then BAN (330): COX via BAN = 430 > 405, ignored. Pop **COX (405)**: goal. Follow the parents:

$$\text{DHK} \xrightarrow{100} \text{CUM} \xrightarrow{60} \text{FEN} \xrightarrow{95} \text{CTG} \xrightarrow{150} \text{COX} = 405 \text{ km}$$

ANSWER

Expansion order DHK, CUM, BRB, CHP, FEN, NOA, CTG, SYL, BAN, COX. Route DHK → CUM → FEN → CTG → COX = 405 km (optimal).

Why cheapest-first is optimal

1. With step costs ≥ 0 , UCS pops nodes in **non-decreasing order of g**. Popping never goes back to something cheaper.
2. When the goal is popped with cost C, every node still on the frontier has $g \geq C$.
3. Any other path to the goal must pass through one of those frontier nodes, so it costs at least C. Nothing cheaper exists: **C is optimal**.



GOTCHA

Test the goal when it is *generated* and step 2 breaks: you stop with an upper bound, not the minimum. In the next problem that mistake returns a 12-unit path when a 7-unit one exists.

EXAM LINE

"UCS is optimal because it expands nodes in order of increasing path cost, so the first time it selects a goal node for expansion, that node's path is the cheapest."



PRO TIP

If every step costs the same, ordering by g is ordering by depth: UCS behaves exactly like BFS.

UCS: complete, optimal, and what it costs

QUESTION	UNIFORM-COST SEARCH
Complete?	Yes, if b is finite and every step costs at least some $\epsilon > 0$
Optimal?	Yes, for any step costs ≥ 0 (with $\epsilon > 0$ for completeness)
Time and space	$O(b^{1+\lceil C^*/\epsilon \rceil})$

C^* = cost of the optimal solution, ϵ = the smallest step cost. C^*/ϵ is the deepest the optimal path could be, if every step were as cheap as possible.



GOTCHA

Why $\epsilon > 0$? With zero-cost steps (a free U-turn, a NoOp action) tree-search UCS can loop forever on a path whose cost never grows.

**PRO TIP**

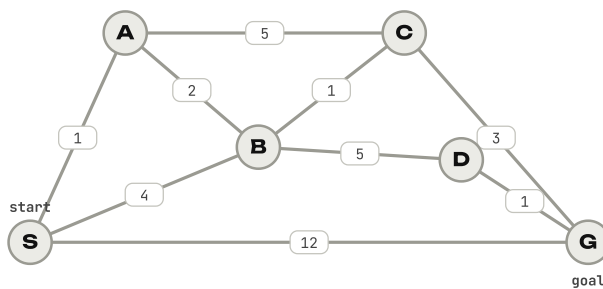
When all steps cost the same, $C^*/\epsilon = d$, and the bound becomes b^{d+1} : BFS with the goal test on pop.

07 **YOUR TURN** 27-31 min

Your turn • 1: UCS, and the road that lies

1

EXAM CQ



Run **UCS graph search** from S to G. Show the sorted frontier after each expansion and the path returned.

Then: what would you return if you tested for the goal as soon as it was *generated*?

SOLUTION

1. Pop **S (0)** → frontier: A 1, B 4, **G 12**. (A goal-on-generation search would stop here with cost 12.)
2. Pop **A (1)** → C 6; B via A = 3 < 4: update. Frontier: B 3, C 6, G 12.
3. Pop **B (3)** → D 8; C via B = 4 < 6: update. Frontier: C 4, D 8, G 12.
4. Pop **C (4)** → G via C = 7 < 12: update. Frontier: G 7, D 8.
5. Pop **G (7)**: goal. Path $S \rightarrow A \rightarrow B \rightarrow C \rightarrow G$, cost $1 + 2 + 1 + 3 = 7$.

BFS returns $S \rightarrow G$ (one hop, cost 12). Goal-on-generation UCS also returns 12. Only the correct UCS finds 7.

ANSWER

Expansion order S, A, B, C, G. Path $S \rightarrow A \rightarrow B \rightarrow C \rightarrow G$, cost 7. Testing on generation (or BFS) gives $S \rightarrow G$, cost 12.

08 **THINK DEEPER** 31-34 min

Critical thinking • when

cheapest-first lies

THINK DEEPER

A map app adds a **negative cost** for roads that pass a sponsor's shop (-10 'reward'). Does UCS still return the cheapest route?

No. UCS's whole argument is "costs only grow along a path". A negative step breaks it: a node popped early ($S \rightarrow A$ at cost 1) can later be reached more cheaply ($S \rightarrow B \rightarrow A$ at $5 - 10 = -5$), but UCS has already closed it.

A STRONG ANSWER MENTIONS

- UCS finalises a node when it is popped, assuming nothing later can make it cheaper.
- A negative edge can lower the cost of an already-popped node: e.g. $S \rightarrow A = 1$ popped first, but $S \rightarrow B = 5$ then $B \rightarrow A = -10$ gives -5 .
- With a negative cycle there is no cheapest route at all (go round forever). Use Bellman–Ford, or keep all costs ≥ 0 .

THINK DEEPER

The hospital-robot problem has a free action *Wait* (cost 0). What can go wrong for UCS, and how do you fix it?

An endless chain of Waits has the same g , so UCS can keep popping them. Give Wait a small cost (time is never free), or use the explored set so a repeated state is never expanded twice.

A STRONG ANSWER MENTIONS

- Zero-cost actions let g stay the same forever: Wait, Wait, Wait...
- Tree-search UCS can loop without making progress; completeness needs every step $\geq \epsilon > 0$.
- Fix: graph search with an explored set, or give Wait a small positive cost.

09 **CONCEPT** 34–38 min

Depth-limited search: DFS on a leash

```
function DLS(node, ℓ)
  if IS-GOAL(node.state): return SOLUTION(node)
  if ℓ = 0: return cutoff           # the leash
  cut ← false
  for each child of node:
    result ← DLS(child, ℓ - 1)
    if result = cutoff: cut ← true
    else if result ≠ failure: return result
  return cut ? cutoff : failure
```

DFS that treats nodes at depth ℓ as if they had no children. Infinite paths and endless loops are no longer a problem.

TWO KINDS OF "NO"

failure: searched everything, there is no solution.

cutoff: no solution *within* ℓ ; one may exist deeper.

PRO TIP

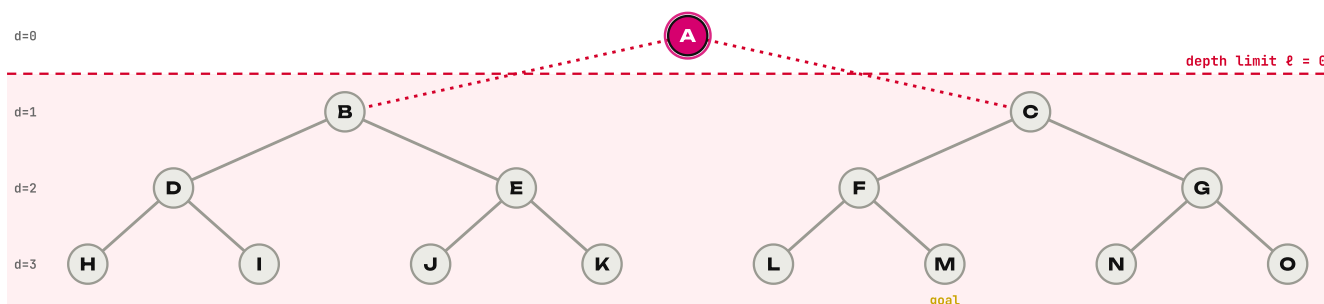
Choosing ℓ : use what you know. Every city on our map reaches every other in at most **5 roads** (Sylhet to Bandarban is the longest), so $\ell = 5$ can never miss a route. That number is the map's *diameter*.

10 **LIVE LAB** 38-45 min

Live lab: DLS and IDS on a tree

Depth-limited and iterative deepening search

Click any node to make it the goal (or pick "none"). Step through and watch the stack: it never holds more than one path.



STACK (CURRENT PATH)

root on the left, top on the right

A

ROUND

$\ell = 0$

VISITS SO FAR

1

MAX STACK

1

BFS NEEDS

13 visits

RESULT

...

$\ell=0$ Round $\ell = 0$: visit A (depth 0): at the limit, children not expanded (cutoff).

11 **WORKED EXAMPLE** 45-49 min

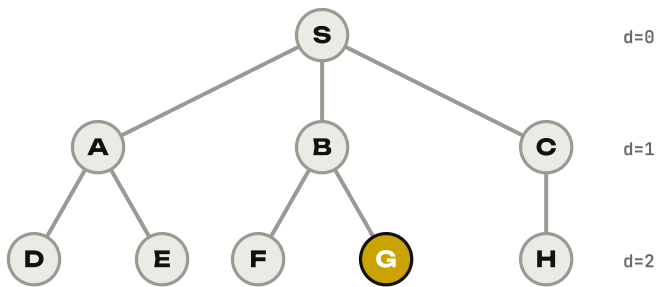
Worked example: IDS round by round

GIVEN

The tree below, goal G. Children are visited left to right; goal test when a node is visited.

FIND

The visit order in each round of IDS, the path found, and the total visits compared with BFS.



Method: iterative deepening = DLS with $\ell = 0, 1, 2, \dots$ (Russell & Norvig)

SOLUTION

1. $\ell = 0$: visit S. Not the goal, and S has children below the limit $\rightarrow$ **cutoff**. Try again deeper.
2. $\ell = 1$: S, A, B, C. A, B, C sit on the limit and have children $\rightarrow$ **cutoff**.
3. $\ell = 2$: S, A, D, E, B, F, **G**: goal. The stack at that moment is the path:

$$S \rightarrow B \rightarrow G$$

4. Count: $1 + 4 + 7 = 12$ **visits**. BFS would visit S, A, B, C, D, E, F, G = 8. The repeats are the shallow levels, which are the *small* ones.
| Largest stack: 3 nodes (one path). BFS's queue peaks at 5 (D, E, F, G, H after C is popped).

ANSWER

Rounds: [S] · [S A B C] · [S A D E B F G]. Path $S \rightarrow B \rightarrow G$. 12 visits for IDS, 8 for BFS.

12 **CONCEPT** 49-53 min

Isn't repeating everything wasteful?

Level i is regenerated once for every round from i to d , so $d - i + 1$ times. Counting generated nodes (root excluded):

$$N_{IDS} = db + (d-1)b^2 + \dots + 2b^{d-1} + 1 \cdot b^d$$

B = 10, D = 5

NODES GENERATED

BFS: $10 + 100 + 1,000 + 10,000 + 100,000$

111,110

IDS: $50 + 400 + 3,000 + 20,000 + 100,000$

123,450 (+11 %)

WHY SO LITTLE?

Most of a tree is its bottom level. With $b = 10$ the last level is 90 % of all nodes, and IDS generates it only **once**. For large d the ratio tends to $b/(b - 1)$.

B	IDS EXTRA WORK
2	$\approx 100 \%$
3	$\approx 50 \%$
10	$\approx 11 \%$



PRO TIP

Memory: one path plus its siblings, **$O(bd)$** . For $b = 10$, $d = 5$: about 50 nodes instead of 100,000.

13 **YOUR TURN** 53-57 min

Your turn • 2: count the cost of deepening

2

EXAM CQ

A uniform tree has branching factor **$b = 3$** , and the only goal is the right-most node at depth **$d = 4$** (worst case for both).

(a) How many nodes do BFS and IDS generate (do not count the root)? (b) What is the extra work as a percentage? (c) Compare the largest number of nodes each must keep in memory.

SOLUTION

1. BFS generates each level once:

$$3 + 9 + 27 + 81 = 120$$

2. IDS generates level i exactly $(d - i + 1)$ times:

$$4(3) + 3(9) + 2(27) + 1(81) = 12 + 27 + 54 + 81 = 174$$

3. Extra work: $174/120 - 1 = \mathbf{45 \%}$ (close to $b/(b - 1) - 1 = 50 \%$).

4. Memory: IDS keeps one path with the waiting siblings, at most $b \cdot d = \mathbf{12}$ nodes. BFS's queue holds a whole level: up to **81**.

ANSWER

BFS 120 nodes, IDS 174 (+45 %). Memory: IDS ≤ 12 nodes, BFS up to 81.

14 **CONCEPT** 57-60 min

Bidirectional search:

dig from both ends

Run one search forward from the start and one backward from the goal; stop when the two frontiers meet.

$$b^{d/2} + b^{d/2} \ll b^d \quad b = 10, d = 6 : 2,000 \text{ vs } 1,000,000$$

- Needs the goal as an explicit **state** (not just a test) and actions you can **run backwards**.
- Memory $O(b^{d/2})$: one of the frontiers must be stored to check the meeting.



POP CULTURE

"Six degrees of separation": LinkedIn finds how you are connected to someone by expanding your contacts and theirs at the same time, and looking for the overlap.



GOTCHA

Checkmate in chess is a *test* (millions of positions pass it), not one state. Searching backward from "all checkmates" is not practical.

15 **CONCEPT** 60-64 min

The whole uninformed family, one table

CRITERION	BFS	UCS	DFS	DLS	IDS	BIDIRECTIONAL
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

^a if b is finite · ^b if every step cost $\geq \epsilon > 0$ · ^c if all step costs are equal · ^d if both directions use BFS. b = branching factor, d = depth of the shallowest solution, m = maximum depth, ℓ = depth limit, C^* = optimal cost.



PRO TIP

Read the IDS column: BFS's guarantees with DFS's memory. That is why it is the default uninformed search when the solution depth is unknown.

16 **THINK DEEPER** 64-67 min

Critical thinking • which

search fits in your phone?



CRITICAL THINKING

A route planner on a phone has **4 GB** free memory. The problem has $b = 10$ and the solution is at depth $d = 12$. Each node takes 1 KB; the phone generates 10^6 nodes per second.

Can BFS do it? Can IDS? What would you tell the app's developer?

SOLUTION

1. **BFS memory:** about $b^d = 10^{12}$ nodes $\times$ 1 KB = **1 PB**. Impossible on any phone.
2. **IDS memory:** $b \cdot d = 120$ nodes $\times$ 1 KB = **120 KB**. Fits easily.
3. **IDS time:** $\approx 1.11 \times 10^{12}$ nodes $\div$ 10^6 per second $\approx 1.1 \times 10^6$ s $\approx$ **13 days**.
4. So IDS fixes memory, not time. Uninformed search cannot do this problem at all. We need knowledge about the problem: a **heuristic** (next week, A*).

ANSWER

BFS runs out of memory ($\approx$ 1 PB). IDS fits ($\approx$ 120 KB) but takes $\approx$ 13 days. Neither is practical: use an informed search.

17 REAL WORLD 67–71 min

Real world: deepening in chess, cheapest-first in networks



CHESS ENGINES

Engines such as Stockfish search iteratively deeper: depth 1, 2, 3, ... until the clock runs out. Each finished round gives a best move, so the engine **always has an answer ready**. That is the "anytime" property, and why IDS beats a single deep DFS under time pressure.



THE INTERNET

Routers running OSPF use Dijkstra's algorithm (UCS without an early stop) to compute the cheapest path to every other router, with link costs set by bandwidth. Your packet to a server in Singapore followed it.

THINK DEEPER

An ambulance dispatcher in Dhaka uses UCS with travel *time* as cost. At 5 PM the costs change every minute. What goes wrong, and what would you do?

The route was optimal for the costs at 5:00 PM. By 5:20 the ambulance is somewhere else and Mohakhali is jammed. Real systems **re-plan from the current position** and use **time-dependent** costs (predicted traffic when you reach each road).

A STRONG ANSWER MENTIONS

- UCS assumes fixed step costs during the search; the answer can be stale by the time the ambulance arrives.
- Re-plan periodically from the current position (closed loop).
- Use predicted, time-dependent costs (traffic at the time you reach each road), not the costs at departure.

18 POP QUIZ 71-76 min

Pop quiz

1. Uniform-cost search always expands the frontier node with the lowest...

- A. depth
- B. heuristic $h(n)$
- C. path cost $g(n)$ ✓**
- D. number of children

UCS orders its priority queue by $g(n)$, the cost of the path from the start. $h(n)$ enters next week with greedy search and A^* .

2. Why does UCS apply the goal test when a node is *popped*, not when it is generated?

- A. It is faster
- B. A cheaper path to the goal may still be waiting on the frontier ✓**
- C. Generated nodes have no parent yet
- D. It saves memory

When the goal is generated, its g is only an upper bound. Only when it is popped do we know nothing cheaper is left (every remaining g is at least as large).

3. Depth-limited search with $\ell = 3$ returns *cutoff*. This means...

- A. There is no solution
- B. A solution exists at depth 3
- C. Nodes below depth 3 were not searched, so a solution may exist deeper ✓**
- D. The tree has depth 3

Cutoff $\neq$ failure. Failure means the whole space was searched; cutoff means the limit hid part of it.

4. Uniform tree, $b = 10$, $d = 5$. IDS generates about how many more nodes than BFS?

A. About 11 % more ✓

B. About 2× as many

C. About 5× as many

D. Exactly as many

123,450 vs 111,110. The bottom level holds ~90 % of the tree and IDS generates it only once.

5. The memory IDS needs grows like...

A. $O(b^d)$

B. $O(bd)$ ✓

C. $O(d)$

D. $O(b^{d/2})$

It is depth-first inside each round: one path of length d plus the siblings waiting at each level.

19 RECAP 76-80 min

Three things to remember

01

UCS pops the lowest $g(n)$ and tests the goal on pop. Optimal for costs ≥ 0 ; complete if every step $\geq \epsilon > 0$.

02

DLS is DFS with a depth limit ℓ : *cutoff* (maybe deeper) $\neq$ *failure* (none at all). Complete only if $\ell \geq d$.

03

IDS runs DLS for $\ell = 0, 1, 2, \dots$: complete and optimal like BFS (equal costs), $O(bd)$ memory like DFS, $\approx 11\%$ extra work for $b = 10$.

HW

BEFORE NEXT CLASS

Use the lab's binary tree (A at the root, H–O the leaves, children left to right) with goal **K**. Write the IDS visit order for every round, count the total visits and compare with BFS. Check yourself with the sim.

Class Test 1 is tomorrow, last 20 minutes: problem formulation, BFS, DFS, UCS, DLS, IDS (traces and the comparison table) and the basics of local search. Bring a pen and a calculator.

SOLUTION

1. $\ell = 0$: A · $\ell = 1$: A B C · $\ell = 2$: A B D E C F G · $\ell = 3$: A B D H I E J **K**.
2. Total visits $1 + 3 + 7 + 8 = \mathbf{19}$. BFS visits A B C D E F G H I J K = **11**.
3. Path A $\rightarrow$ B $\rightarrow$ E $\rightarrow$ K; largest stack 4 nodes.

ANSWER

19 visits for IDS vs 11 for BFS; path A $\rightarrow$ B $\rightarrow$ E $\rightarrow$ K.

S. M. Sakeef Sani

Adjunct Lecturer, CSE · Northern University Bangladesh

smsakeefsani.hbitcorp.com