CS 486/686
Heuristic Search

Yuntian Deng

Lecture 3

RN 3.5 · PM 3.6–3.7

Learning goals

- Motivate heuristic search
- Trace LCFS, GBFS, A*
- Analyze: space, time, completeness, optimality
- Design and dominate admissible heuristics
- Prune the search space

Which state is closer to the goal?

State A

5	3	
8	7	6
2	4	1

State B

1	2	3
4	5	
7	8	6

Goal

1	2	3
4	5	6
7	8	

B is just one move away. Humans see this instantly — uninformed search does not.

Uninformed vs heuristic

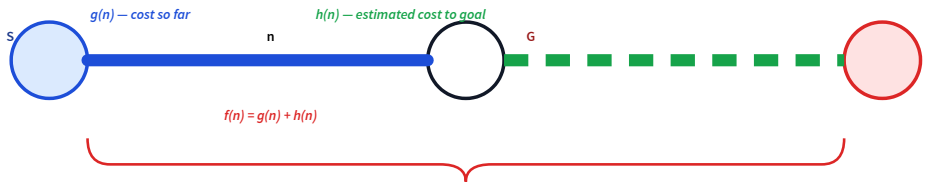
Uninformed

- Treats every state the same
- No idea which is closer to goal

Heuristic

- Estimates distance to goal
- Expands the most promising state

Two quantities at every node n



Solid **blue**: known. Dashed **green**: estimated. **Red**: total estimate.

What makes a good heuristic?

Problem-specific

Non-negative

$h(\text{goal}) = 0$

Cheap to compute

Three algorithms, one knob

Frontier is a priority queue. The three algorithms differ in the priority key.

LCFS

$g(n)$

cheapest-path-so-far

GBFS

$h(n)$

looks-closest-to-goal

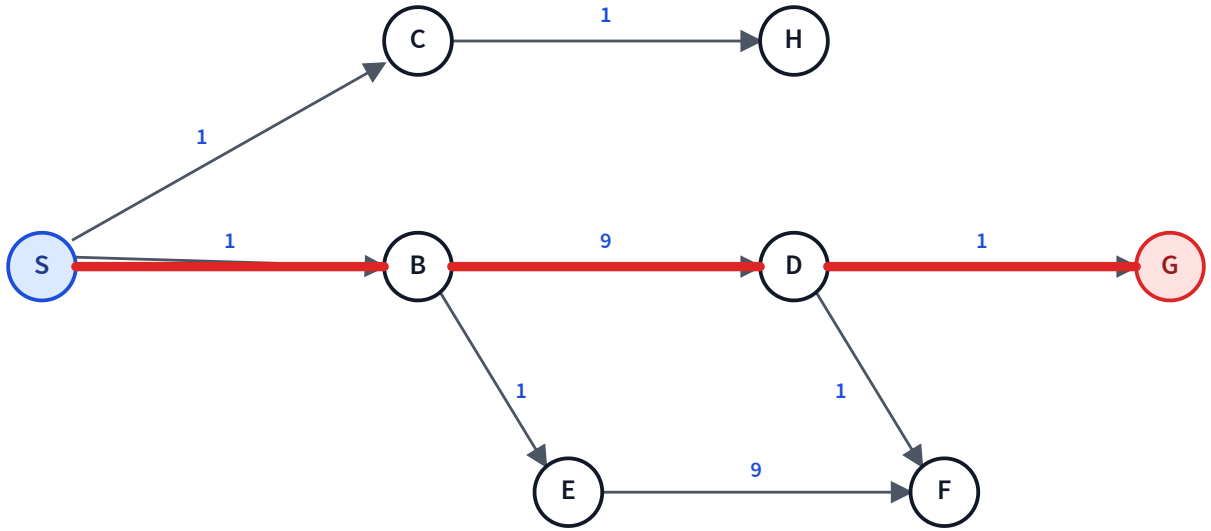
A*

$g(n) + h(n)$

total estimated cost

The trace graph

All four traces use this graph. Edge labels are **costs**. Note that F has two parents (E and D).



Optimal path: S → B → D → G with total cost **11**.

Lowest-cost-first search (LCFS)

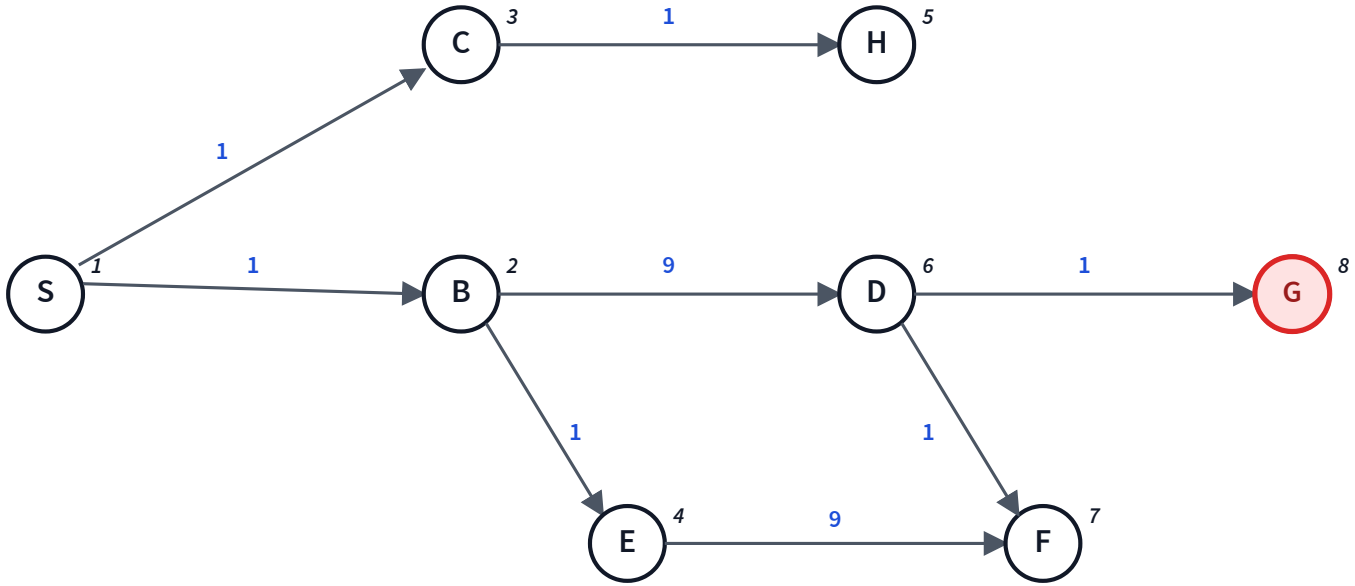
- Frontier = **priority queue**, key = $g(n)$
- Pop the cheapest path so far

a.k.a. **Uniform-cost search (UCS)** in RN, or **Dijkstra's algorithm** in algorithms textbooks.

Trace: LCFS

Path notation SBE : 2 = path S→B→E with cumulative cost 2. Ties broken alphabetically.

```
Frontier:
```



LCFS — properties

Space	$O(b^d)$ *exponential*
Time	$O(b^d)$ *exponential*
Complete?	**Yes** *finite branching, edge cost ≥ ε > 0*
Optimal?	**Yes** *under the same conditions*

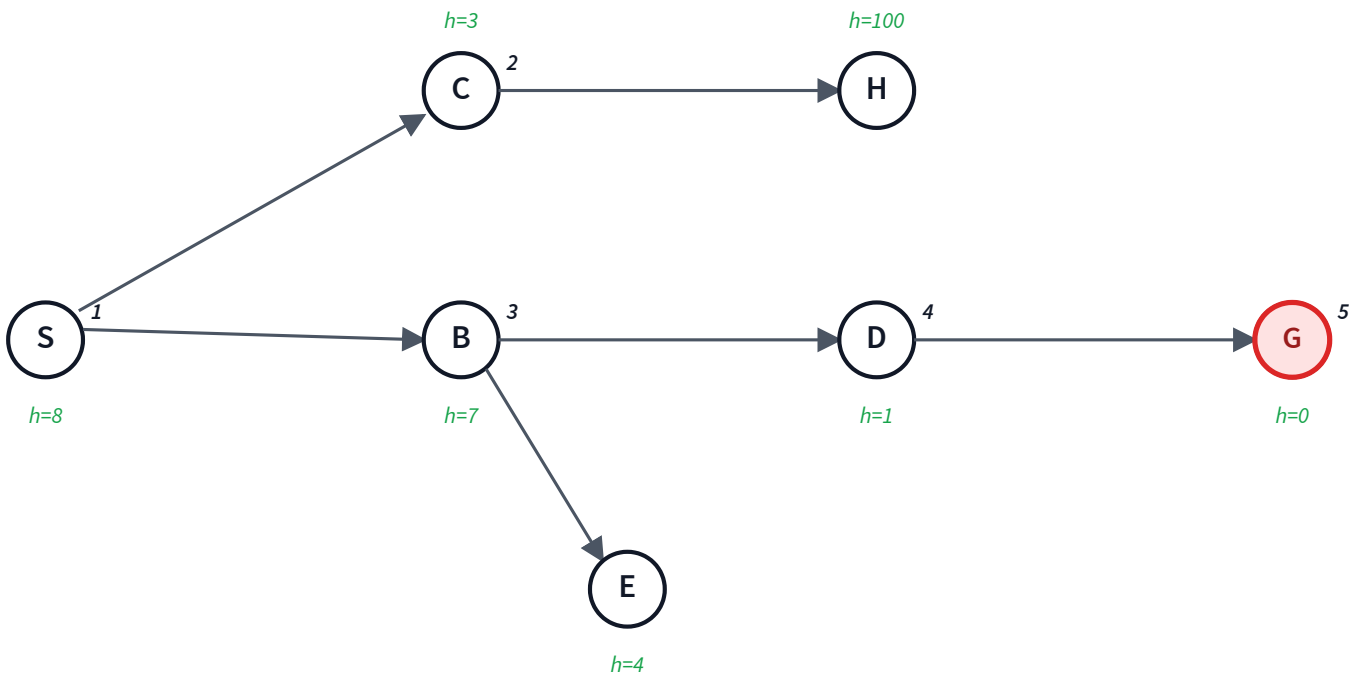
Greedy best-first search (GBFS)

- Frontier = **priority queue**, key = $h(n)$
- Pop the most promising path (lowest h)

Trace: GBFS

Path notation SBD: 1 = path with $h(D) = 1$. Heuristic values shown **in green**.

```
Frontier:
```



GBFS — properties

Space	$O(b^m)$ *exponential*
Time	$O(b^m)$ *exponential*
Complete?	**No** *can get stuck following a misleading heuristic*
Optimal?	**No** *greedy choice may be wrong*

Intuition: what each algorithm prioritises

	LCFS	GBFS
Frontier sorted by	g(n) — cost so far	h(n) — estimated cost to goal
Closest uninformed cousin	**BFS with weights.** When every edge costs 1, LCFS = BFS exactly.	**DFS guided by h.** Chases one direction aggressively.
Search shape	A cost-wavefront that grows *outward* from the start.	A narrow probe that dives toward whatever *looks* closest.

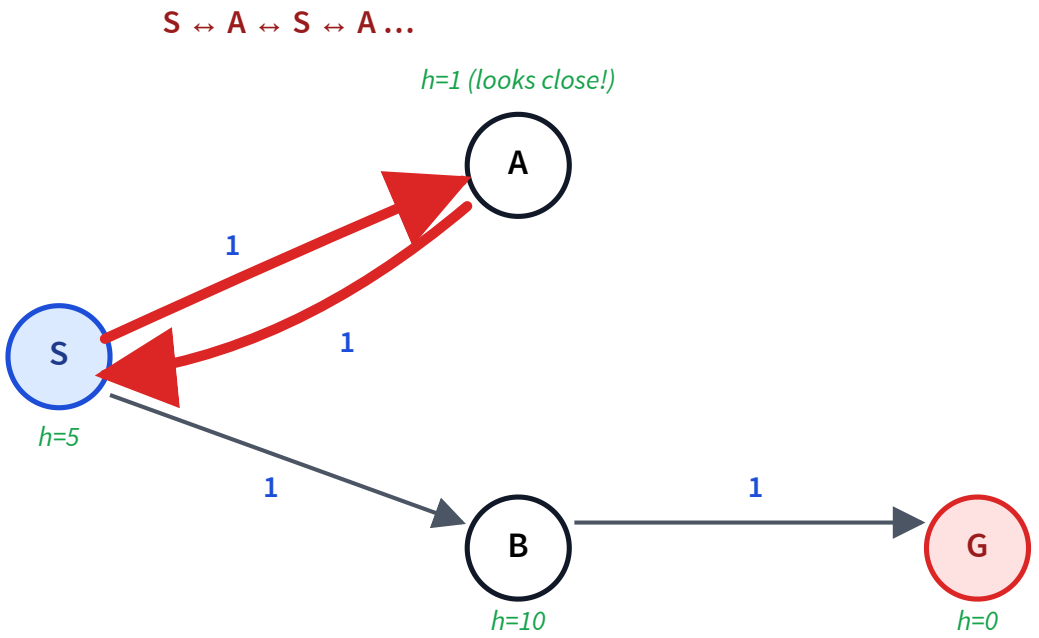
Intuition: what guarantees follow

	LCFS	GBFS
Complete?	**Yes** — wavefront eventually reaches every reachable state.	**No** — low-h cycles can trap it.
Optimal?	**Yes** — first pop of G has the smallest g.	**No** — h alone can prefer a costly "looks-close" detour.

Worst-case time / space: LCFS is $O(b^d)$; GBFS is $O(b^m)$.

GBFS — not complete

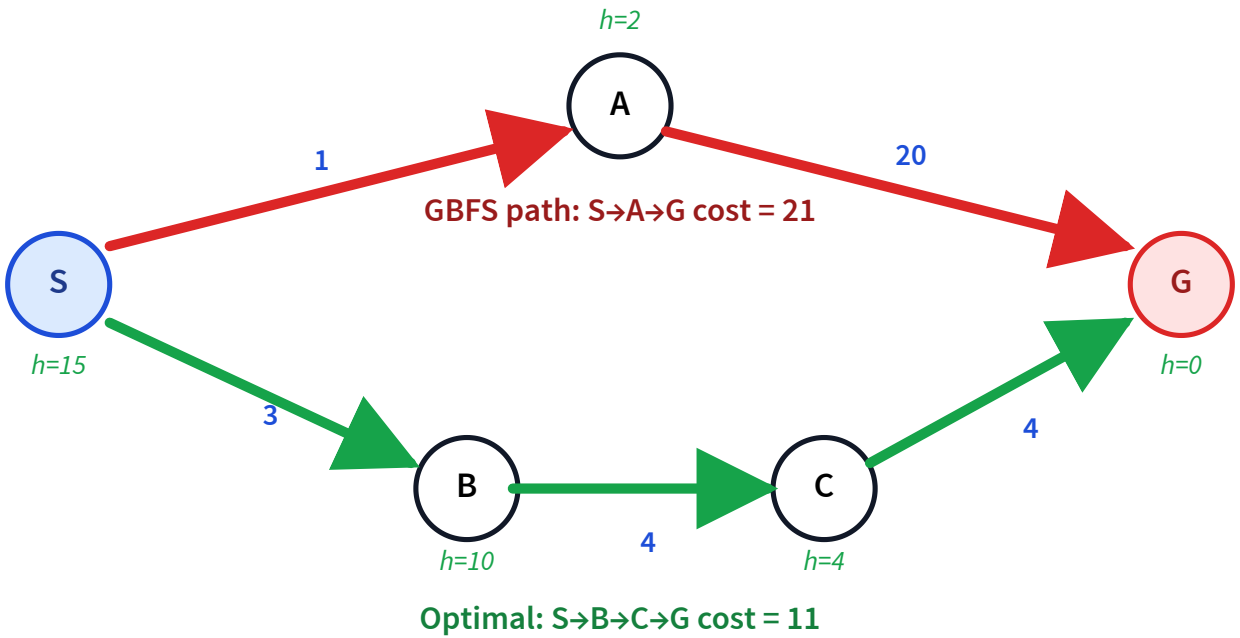
A low-h cycle can trap GBFS forever — it never explores the path that actually reaches G.



Pop order: SA (h=1), SAS (h=5), SASA (h=1), … SB (h=10) is always outranked — G never reached.

GBFS — not optimal

Even when GBFS finds the goal, the path can be far from cheapest.



A* search

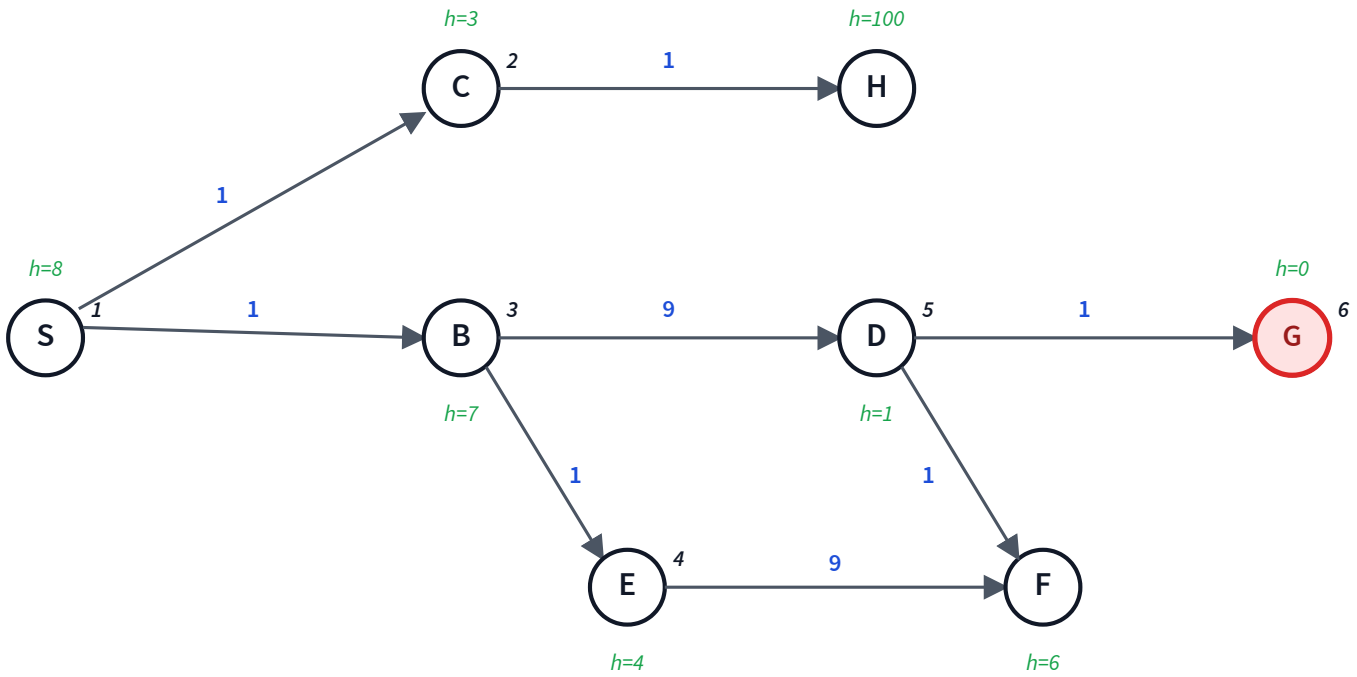
- Frontier = **priority queue**, key = $f(n) = g(n) + h(n)$
- Pop the path with smallest total estimated cost

A* combines LCFS's **cost-so-far** with GBFS's **goal-pull**.

Trace: A*

Path notation SBE: 6 = path with $f = g + h = 6$. **g** in blue, **h** in green.

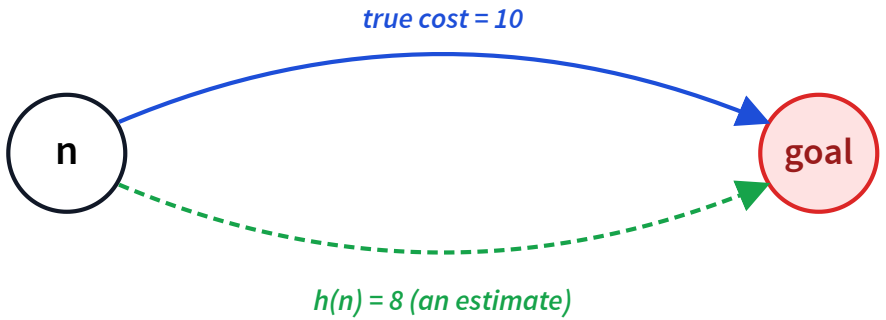
```
Frontier:
```



A* — properties

Space	$O(b^m)$ *worst case; often much better*
Time	$O(b^m)$ *worst case; often much better*
Complete?	**Yes** *under the usual finite-branching condition*
Optimal?	**Yes** *if h is **admissible***

Admissible heuristic



Admissible: $h(n) \leq$ true cost from n to any goal, for every n. *Never over-estimates.*

Theorem. If h is admissible, A* returns an optimal path.

A* optimality — proof sketch

Assume A* returns cost C^*, but the true optimum is $C' < C^*$.

1. Some node N on the optimal path is still on the frontier at termination.
2. Admissibility $\Rightarrow f(N) = g(N) + h(N) \leq C'$.
3. So $f(N) \leq C' < C^*$ — A* would have popped N first. **Contradiction.** ■

A* is optimally efficient

No other optimal algorithm using the same h expands fewer nodes.

Manhattan distance

Sum of grid distances of each tile to its goal position.

Initial

5	3	
8	7	6
2	4	1

Goal

1	2	3
4	5	6
7	8	

tile	1	2	3	4	5	6	7	8
distance	4	3	1	2	2	0	2	2

Manhattan(start) = 4 + 3 + 1 + 2 + 2 + 0 + 2 + 2 = 16

Misplaced tile count

Number of tiles not yet in their goal position.

Initial

5	3	
8	7	6
2	4	1

Goal

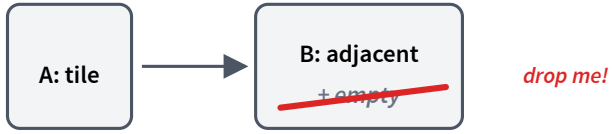
1	2	3
4	5	6
7	8	

Red = misplaced. Green = already at goal.

Misplaced(start) = 7

Both heuristics are **admissible**.

Recipe: relax the problem



1. **Relax** the problem by dropping constraints.
2. **Solve** the relaxed problem (without search).
3. That cost is an **admissible heuristic** for the original.

An easier problem gives a lower bound — safe to use as an estimate.

Quick check: heuristic from relaxation

Q. Drop the "B is empty" constraint: a tile can move from A to B if A and B are adjacent. Which heuristic does this give?

A. Manhattan distance

B. Misplaced tile count

C. Another heuristic not listed

> **A — Manhattan distance.** Without the "B must be empty" constraint, each tile slides freely along the grid to its goal — cost = number of grid steps = Manhattan distance.

Dominating heuristics

Definition. h_2 *dominates* h_1 if $h_2(n) \geq h_1(n)$ for all n, and $h_2(n) > h_1(n)$ for some n.

Theorem. If h_2 dominates h_1, A* with h_2 expands no more nodes than A* with h_1. *Higher = better — as long as still admissible.*

Manhattan vs Misplaced

Q. On our example state, Manhattan = 16 and Misplaced = 7. Which heuristic dominates the other in general?

A. Manhattan dominates Misplaced

B. Misplaced dominates Manhattan

C. Neither dominates the other

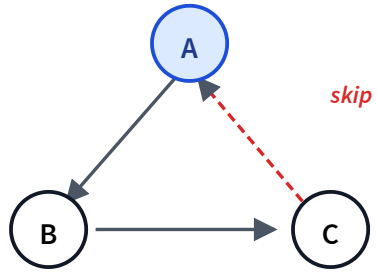
> **A — Manhattan dominates.** Each misplaced tile contributes at least 1 to both heuristics; Manhattan adds the actual grid distance, so Manhattan $\geq$ Misplaced, and strictly $>$ whenever any tile is more than one step from its goal.

Cycle pruning

Don't follow a path that revisits a node already on the current path.

When expanding path $\langle n_0, \ldots, n_k \rangle$:

1. For each neighbour `n` of n_k :
2. If `n` already appears in $\langle n_0, \ldots, n_k \rangle$ → **skip**.
3. Else add $\langle n_0, \ldots, n_k, n \rangle$ to the frontier.



Check is linear in path length. Saves DFS from infinite loops.

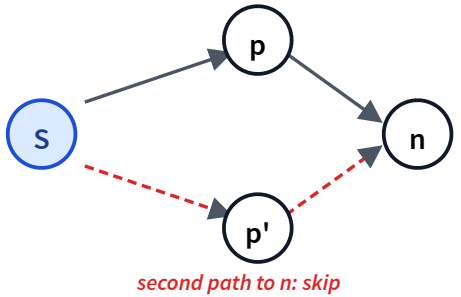
Multi-path pruning

Don't expand a node we've already expanded (across *different* paths). Cycle pruning is a special case.

Maintain an `explored` set, initialized to `{ }`.

Pop path $\langle n_0, \ldots, n_k \rangle$. Then:

1. If $n_k \in$ `explored` → **skip**.
2. Add n_k to `explored`.
3. If n_k is the goal → return the path.
4. Else push each neighbour-extension to the frontier.



Caveats of multi-path pruning

- Repeats still pushed — skipped on pop
- Trades memory (`explored` set) for time
- May sacrifice optimality (see next slides)

LCFS + multi-path pruning

Q. LCFS with multi-path pruning — can it discard the optimal solution?

A. Yes, it can

B. No, never

> **B — No.** First pop of any node already has the smallest g; later paths to it can only be costlier. Safe to prune.

A* + multi-path pruning

Q. A* with admissible h + multi-path pruning — can it discard the optimal?

A. Yes, it can

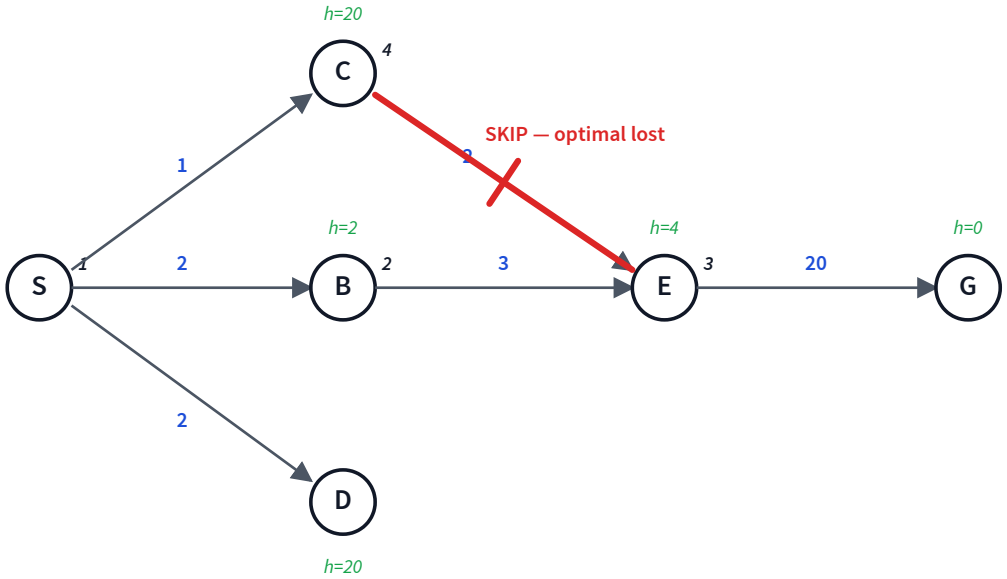
B. No, never

> **A — Yes, it can.** A* orders by $f = g + h$, so the first pop of n need not have the smallest g. A cheaper $g(n)$ found later is then discarded. *Counterexample next.*

A* + multi-path: counterexample

Optimal path is S→C→E→G (cost **23**). With multi-path pruning, A* returns S→B→E→G (cost **25**) instead.

```
Frontier & Explored:
```



Consistent (monotone) heuristic

Definition. h is *consistent* if for every edge $m \to n$:

$$h(m) \leq \text{cost}(m, n) + h(n)$$

i.e., the heuristic decreases by no more than the edge cost as we move toward the goal.

Theorem. If h is consistent, A* with multi-path pruning is **optimal**.

Most natural heuristics (Manhattan, straight-line distance, …) are consistent.

Where A* can mis-prune

Why does **LCFS** need no extra condition, but **A*** does?

LCFS — sorts by $g(n)$:

- first pop of n already has the smallest $g(n)$ — safe.

A* — sorts by $f = g + h$:

- $h(n)$ cancels at n, but differs across prefixes p, p';
- a big h-gap can pop the larger-g path first → n committed to a worse prefix.

How consistency restores the guarantee

Bound h-change along every edge $p \to n$:

$$h(p) - h(n) \leq c(p, n) \;\Leftrightarrow\; h(p) \leq c(p, n) + h(n)$$

— that's exactly **consistency**.

Chain of consequences:

- f is non-decreasing along every path
- first pop of n has the smallest g — LCFS guarantee, restored
- ∴ multi-path pruning is safe

Five search algorithms

Algorithm	Frontier key	Space	Time	Complete?	Optimal?
DFS	LIFO (stack)	$O(bm)$	$O(b^m)$	**No** *cycles*	**No**
BFS	FIFO (queue)	$O(b^d)$	$O(b^d)$	**Yes**	**Yes** *unit cost*
LCFS	$g(n)$	$O(b^d)$	$O(b^d)$	**Yes**	**Yes**
GBFS	$h(n)$	$O(b^m)$	$O(b^m)$	**No**	**No**
A*	$g(n) + h(n)$	$O(b^m)$	$O(b^m)$	**Yes**	**Yes** *admissible* h

Learning goals (recap)

- ✓ Motivate heuristic search
- ✓ Trace LCFS, GBFS, A*
- ✓ Analyze space, time, completeness, optimality
- ✓ Design and dominate admissible heuristics
- ✓ Prune the search space (cycle, multi-path, consistency)