CPSC 322: Introduction to Artificial Intelligence

Search: A* Optimality Branch and Bound, Pruning

Textbook reference: [3.6,3.7.1,3.8.1]

Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

- Midterm time and location

  Time: Friday, Oct 25th, from 6pm to 7pm
  Location: Woodward 2
  (Instructional Resources Centre-IRC) (WOOD) - 2

- If you cannot make this time, please contact me **ASAP**.

- Assignment 1 is due on Sept. 30th at 11:59pm

- My office hours: Fridays from 11am to noon at ICCS 185

Lecture outline

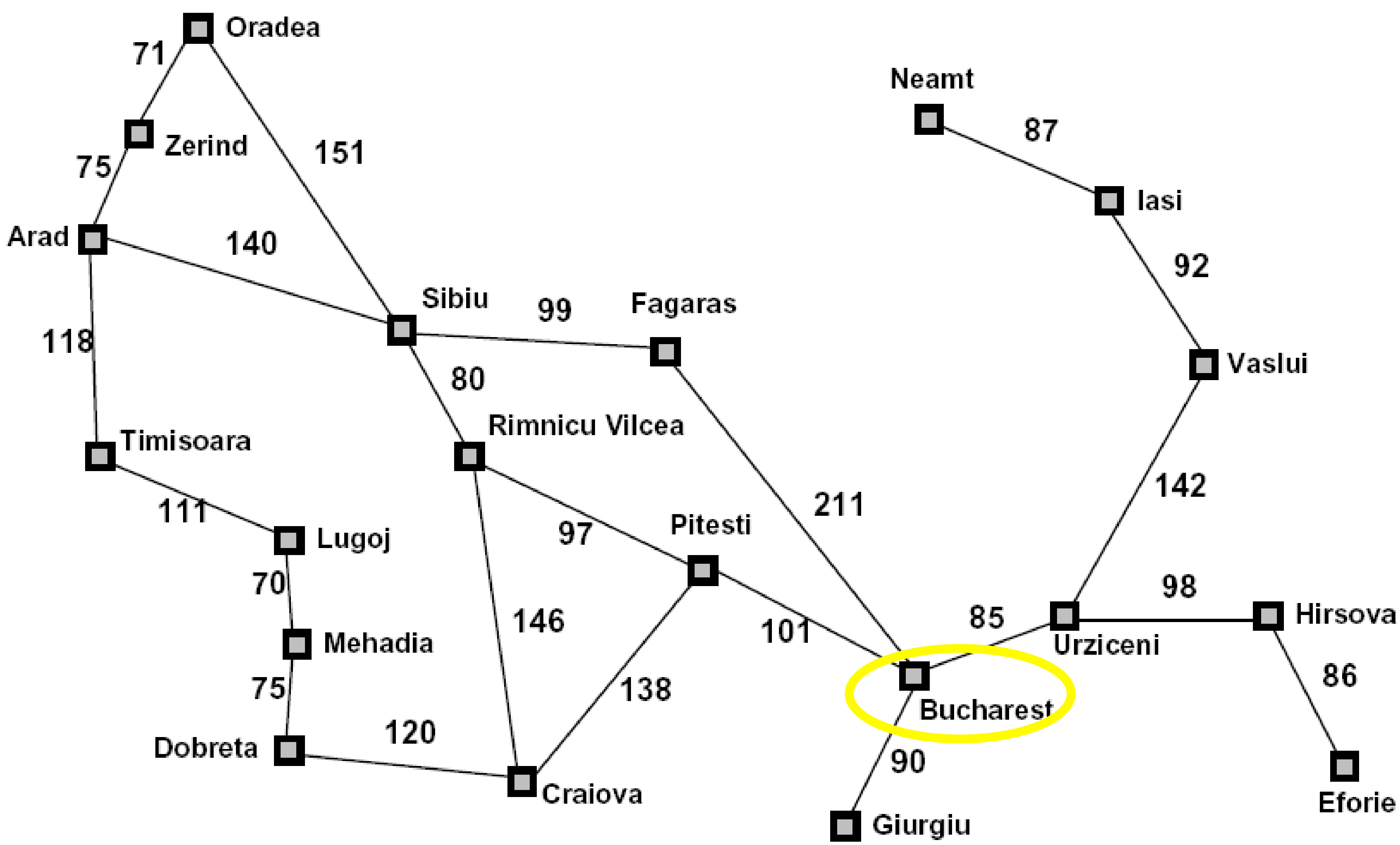
- **Recap from last lecture (~10 mins)**
- A* analysis (~15 mins)
- Branch and bound (~10 mins)
- A* enhancements (~5 mins)
- Class activity (~10 mins)
- Pruning (~15 mins)
- Summary and wrap-up (~5 mins)

Admissible heuristic

A search heuristic is **admissible** if it never overestimates the actual cost of the cheapest path from a node to a goal.

Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is.

Example: Travelling in Eastern Europe



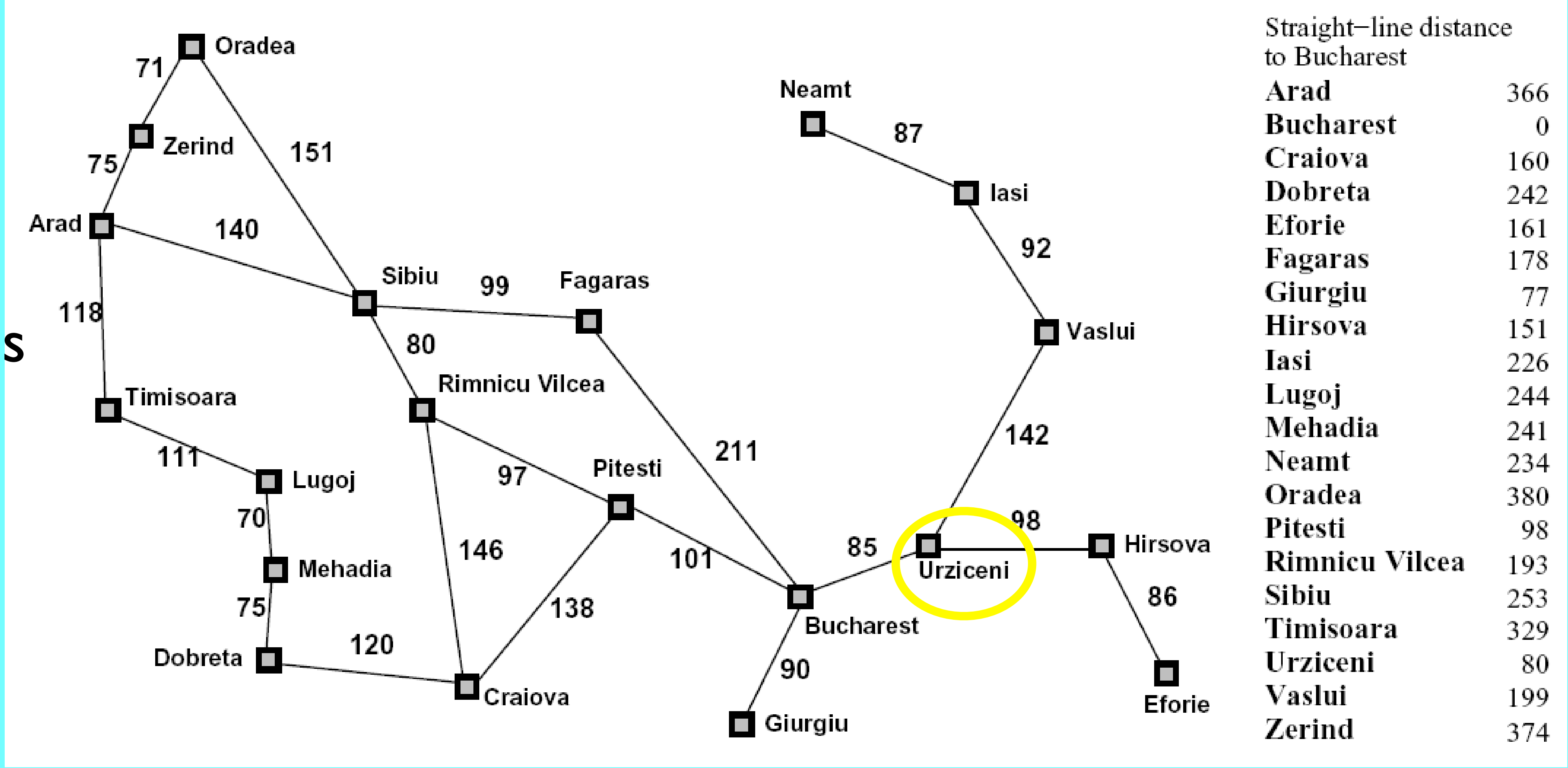
Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Is the given heuristic admissible given the goal is Bucharest?

Example: Travelling in Eastern Europe

Suppose goal = Urziceni
But all we know is straight line distances (sld) to Bucharest.

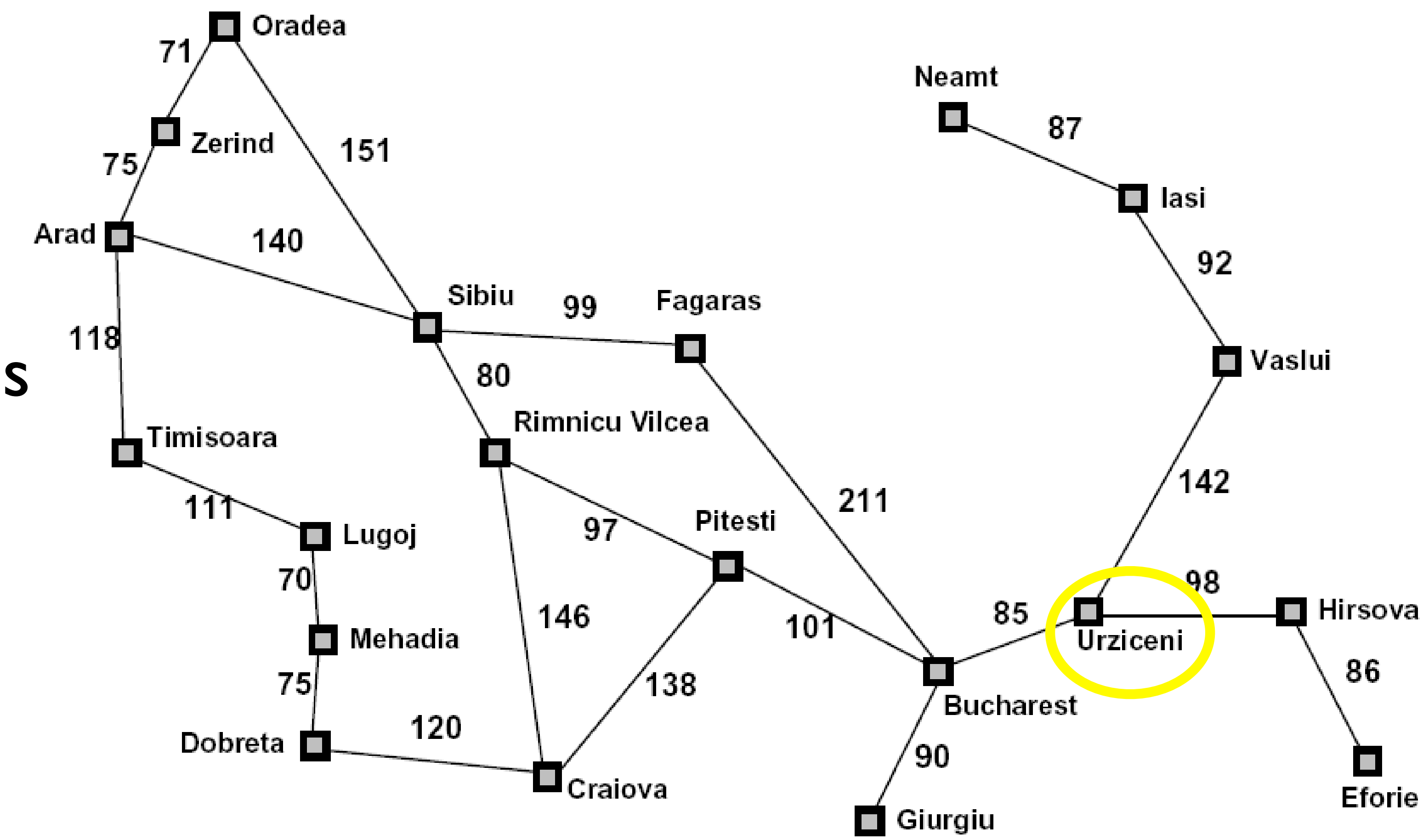


Possible h(n) = sld(n, Bucharest) + cost(Bucharest to Urziceni)
Is this heuristic admissible?

A. Yes
B. No ✅

Example: Travelling in Eastern Europe

Suppose goal = Urziceni
But all we know is straight line distances (sld) to Bucharest.



Straight−line distance to Bucharest

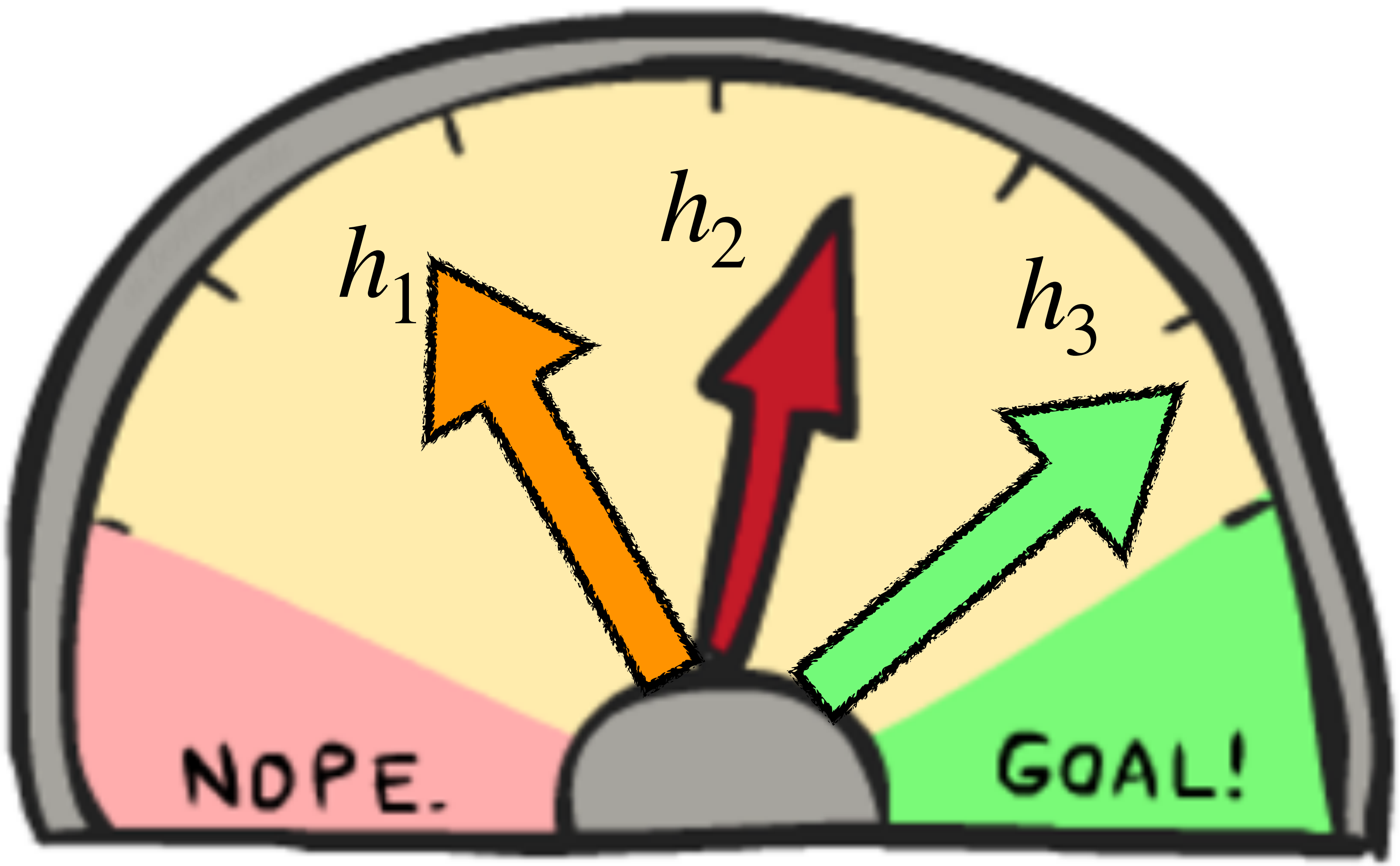
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Possible h(n) = sld(n, Bucharest) + cost(Bucharest to Urziceni)
Is this heuristic admissible?

A. Yes
B. No ✅

Heuristic dominance

All admissible heuristics but the ones closer to the actual cost are more efficient (expand fewer paths).



Actual cost

$h(n) = 0$

Heuristics dominance

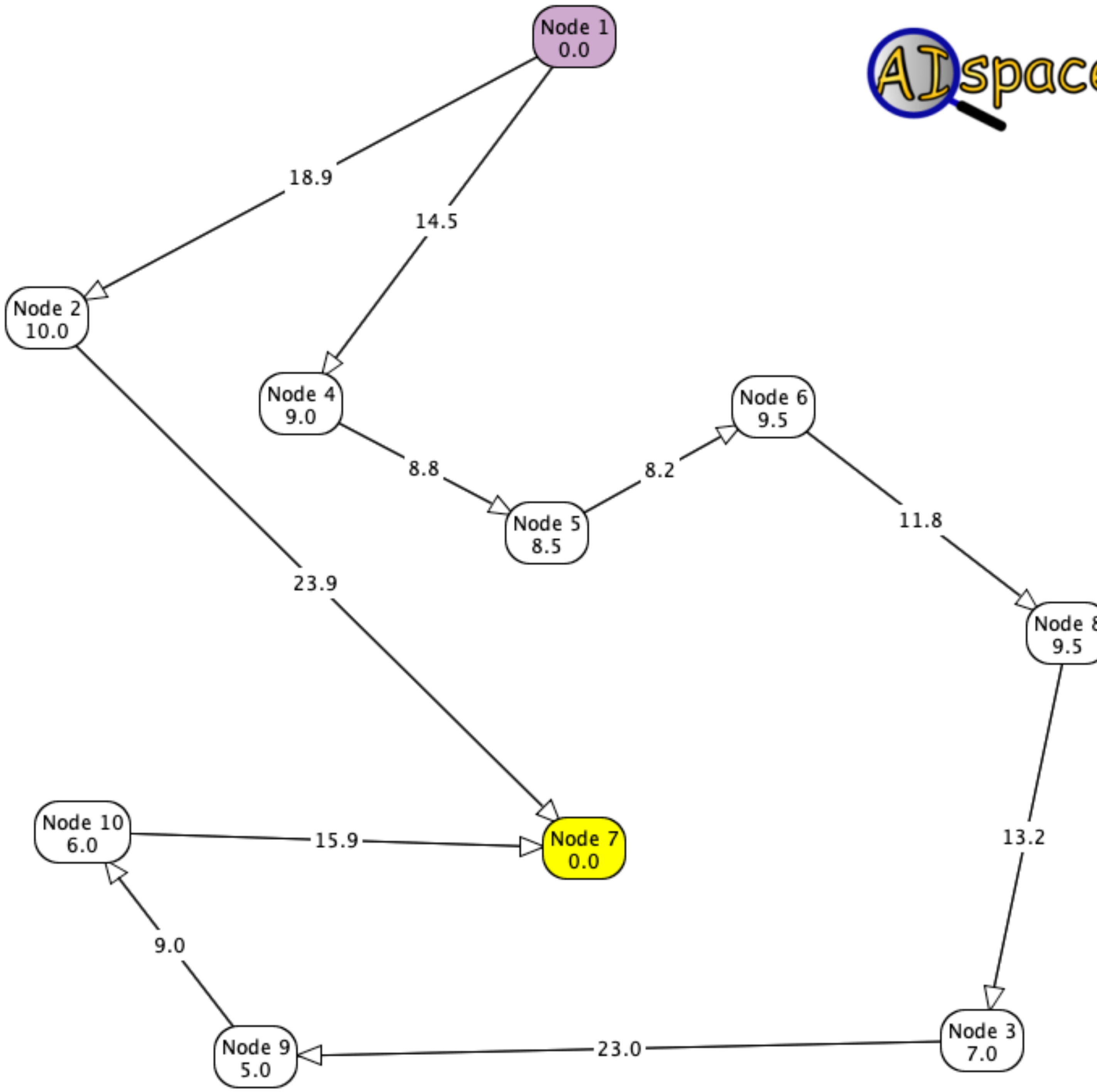
Search costs for the 8-puzzle (average number of paths expanded). Averaged over 100 instances of the 8-puzzle, for various solutions.

$$h_2(n) \geq h_1(n)$$

	$d = 12$	$d = 24$
IDS	3,644,035 paths	too many paths
$A^*(h_1)$	227 paths	39,135 paths
$A^*(h_2)$	73 paths	1,641paths

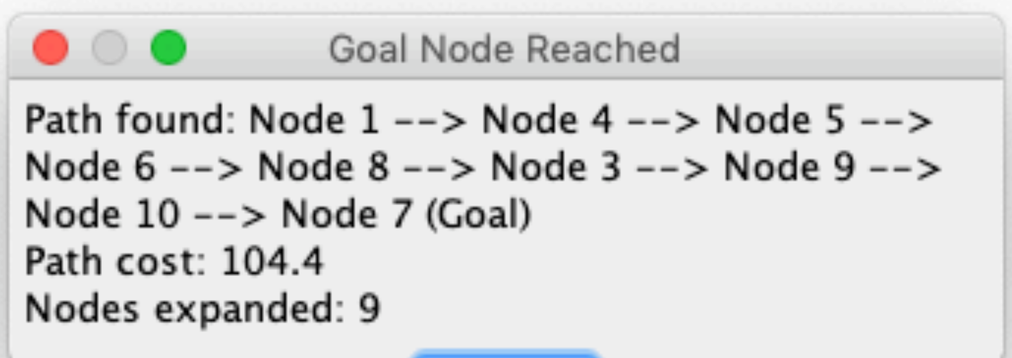
Best-First Search (BestFS)

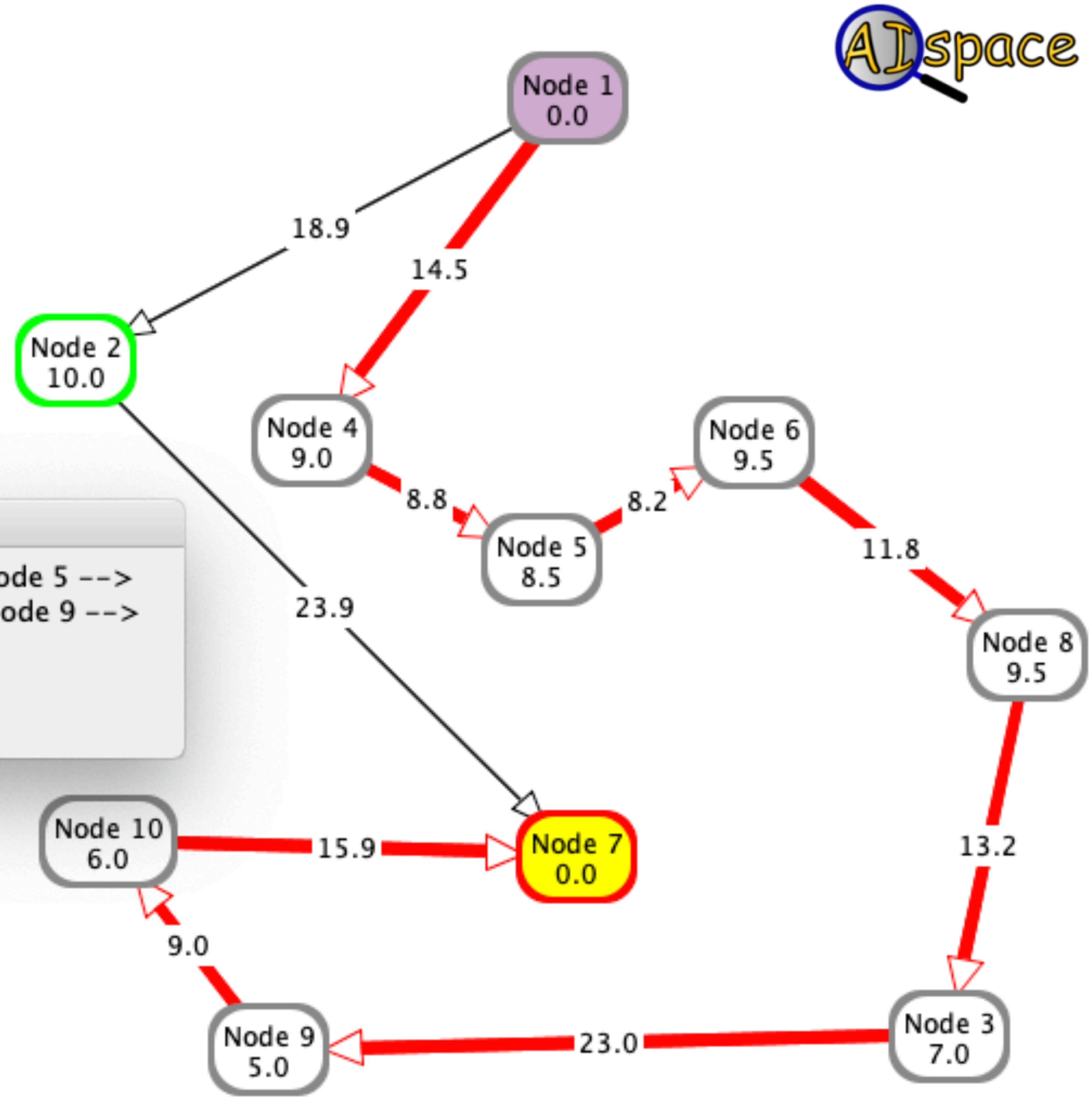
Selects a path on the frontier with minimum h-value



Best-First Search (BestFS)

Selects a path on the frontier with minimum h-value

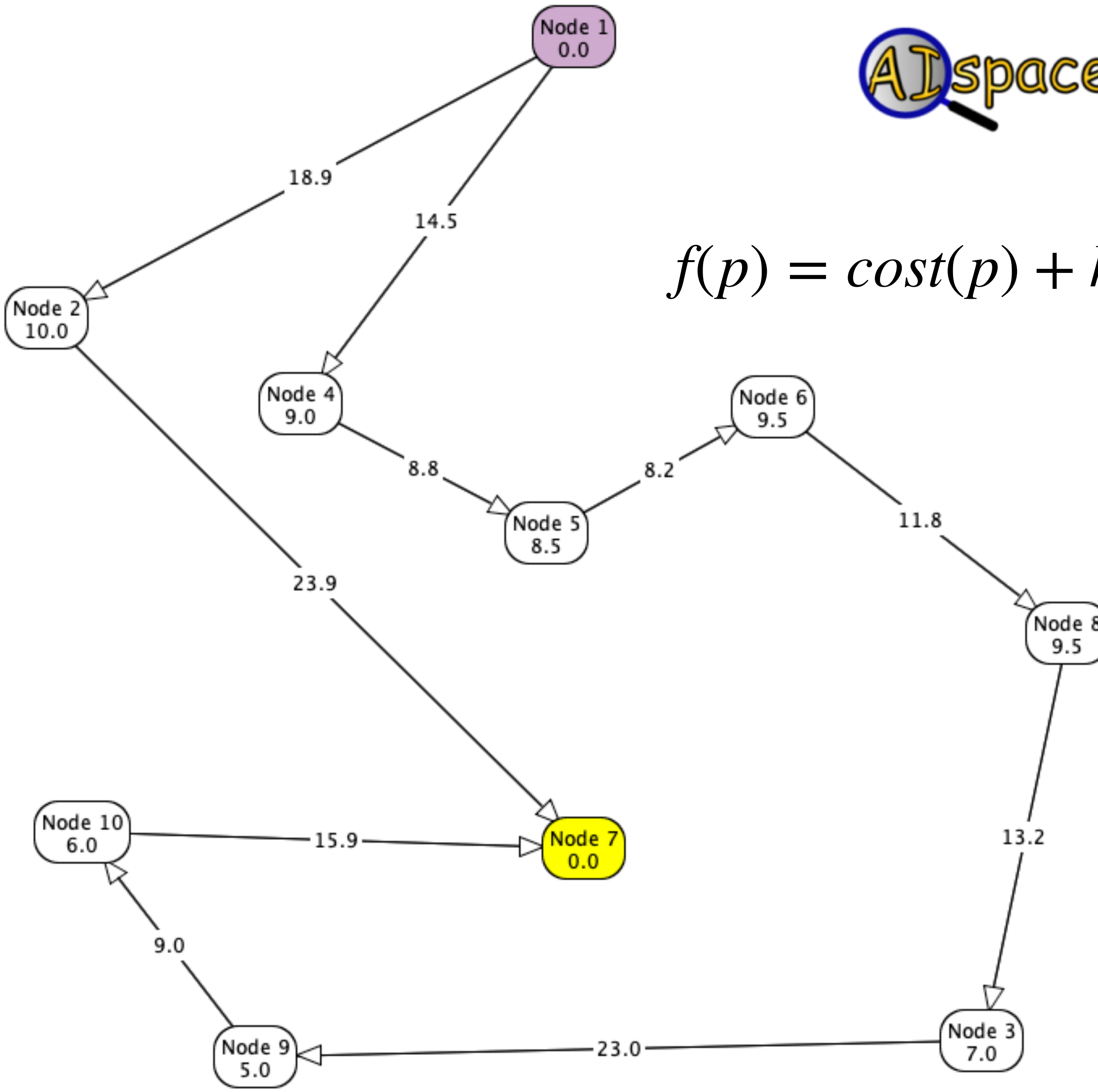




A* search

Selects a path on the frontier with minimum f-value

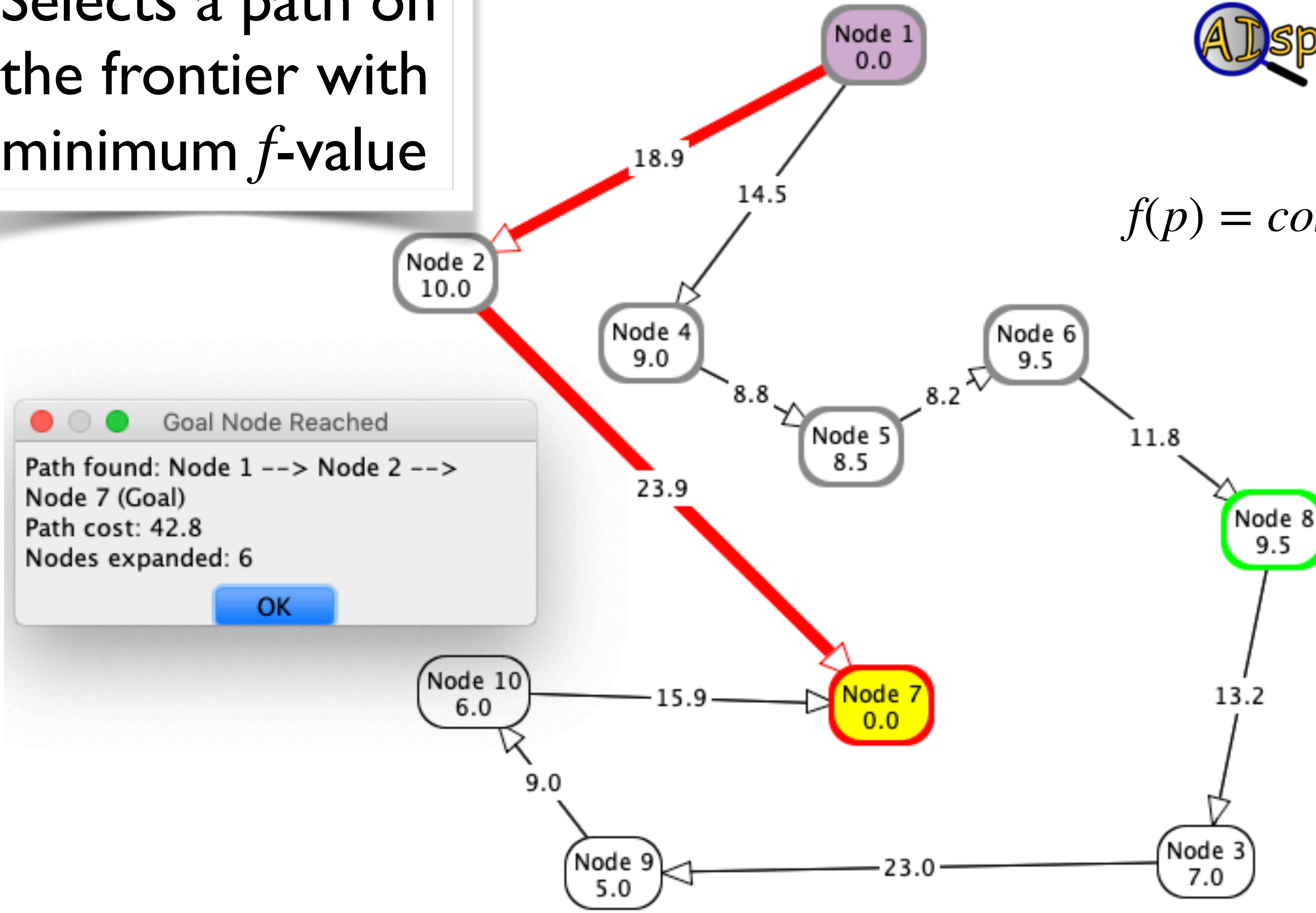
$$f(p) = cost(p) + h(p)$$



A* search

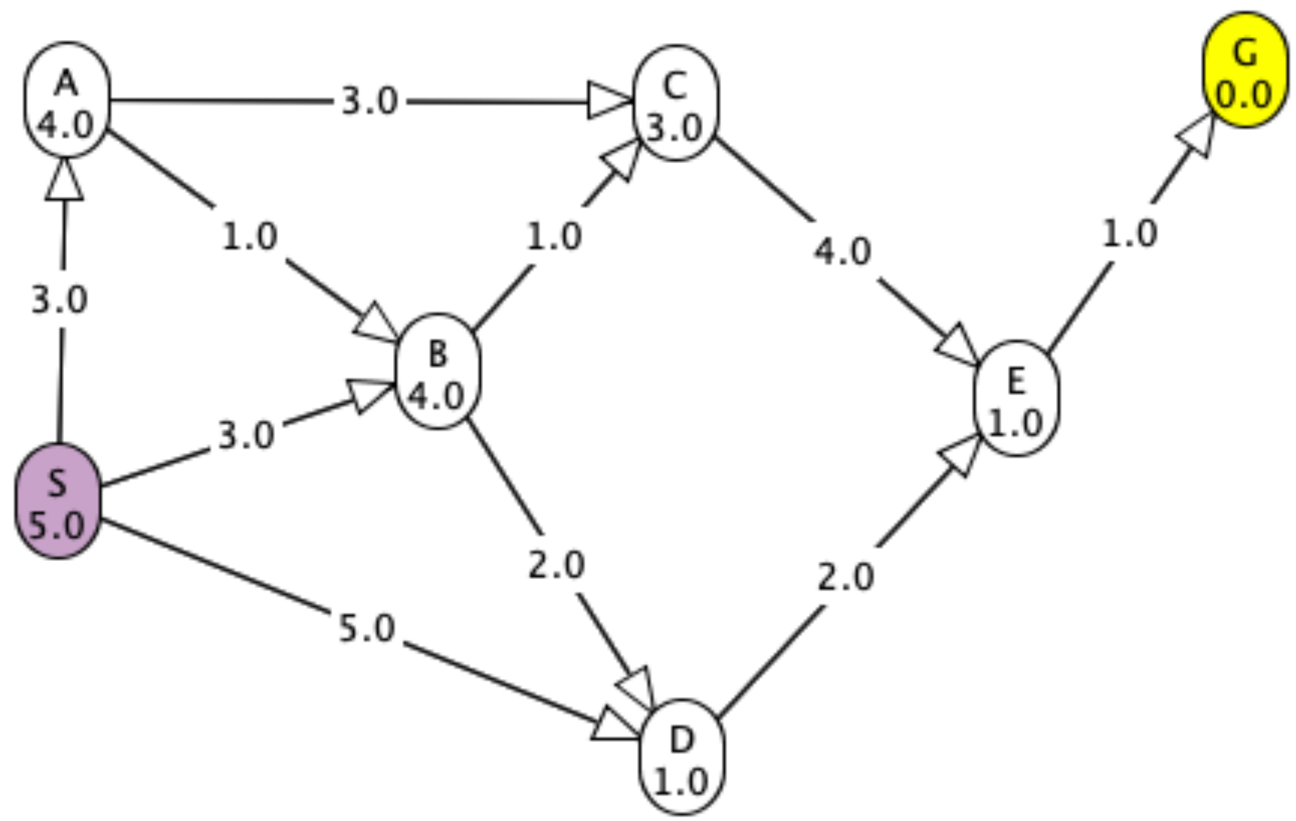
Selects a path on the frontier with minimum f-value

$$f(p) = cost(p) + h(p)$$



Computing f-values

The f-value is an estimate of the cost of getting to the goal via this node (path).



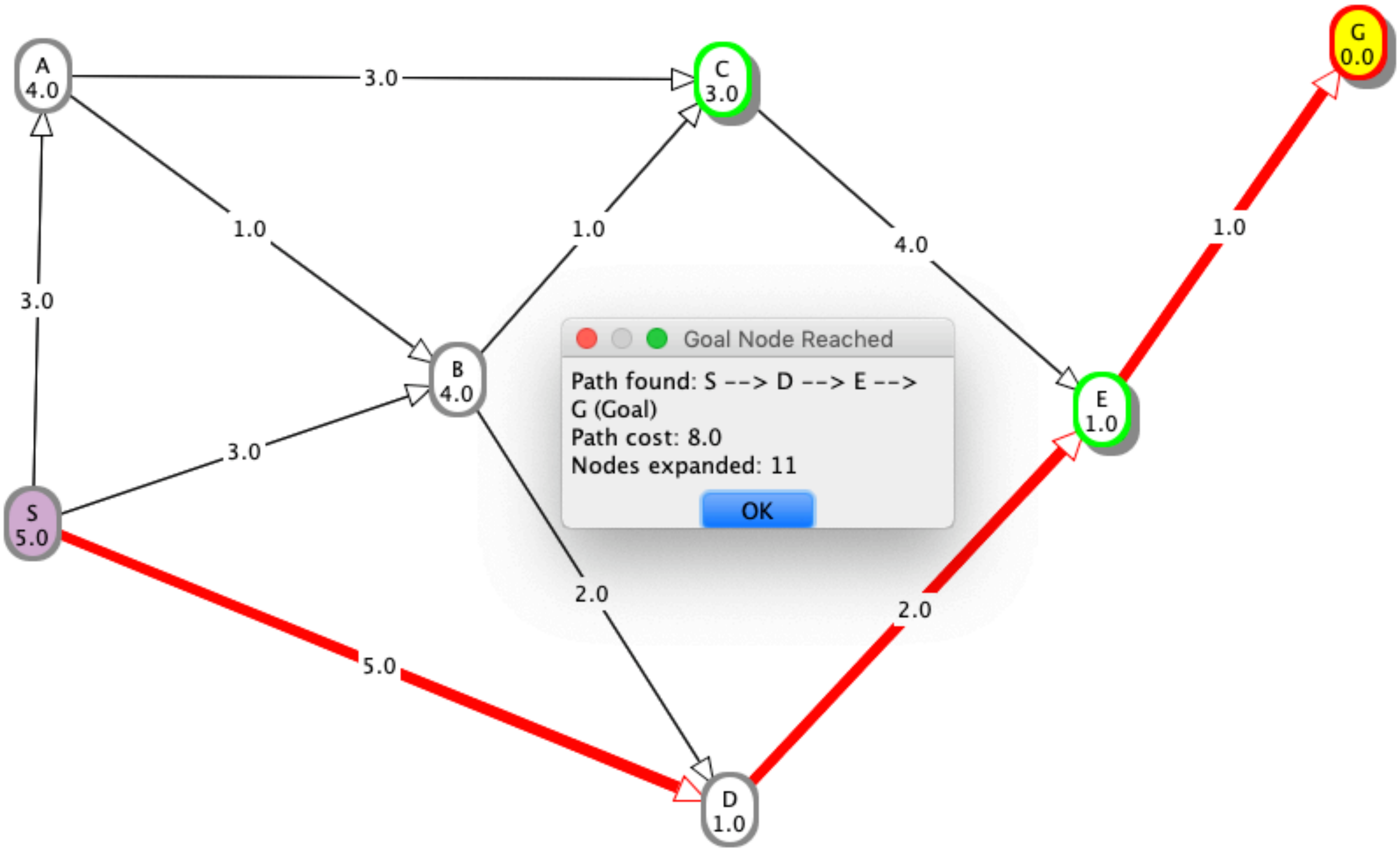
What is f-value of $s \rightarrow A \rightarrow B \rightarrow D$?

$$cost(s \rightarrow A \rightarrow B \rightarrow D) + h(D)$$

$$= (3 + 1 + 2) + 1 = 7$$

RECAP: A* search

$$f(p) = cost(p) + h(p)$$



Today: Learning outcomes

From this lecture, students are expected to be able to:

- Analyze A*
  - Formally prove A* optimality
- Define optimally efficient
- Define/read/write/trace/debug branch and bound search algorithm and other enhancements of A*
- Explain pruning

Lecture outline

- Recap from last lecture (~10 mins)
- **A* analysis (~15 mins)** 👈
- Branch and bound (~10 mins)
- A* enhancements (~5 mins)
- Class activity (~10 mins)
- Pruning (~15 mins)
- Summary and wrap-up (~5 mins)

A* search

If the heuristic is completely uninformative (e.g., $h = 0$ for all nodes) and the edge costs are all the same, A* is equivalent to

A. LCFS

B. BFS ✅

C. DFS

D. A and B

Analysis of A*

When the arc costs are strictly positive. The heuristic could be completely uninformative, and the edge costs could all be the same, meaning that A* would do the same thing as BFS.

- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$
- Completeness: Yes
- Optimality: ??

Optimality of A*

If A* returns a solution, that solution is guaranteed to be optimal, as long as

- the branching factor is finite
- arc costs are $> \epsilon > 0$
- $h(n)$ is admissible (an underestimate of the length of the shortest path from n to a goal node and non-negative)

This property of A* is called admissibility of A*

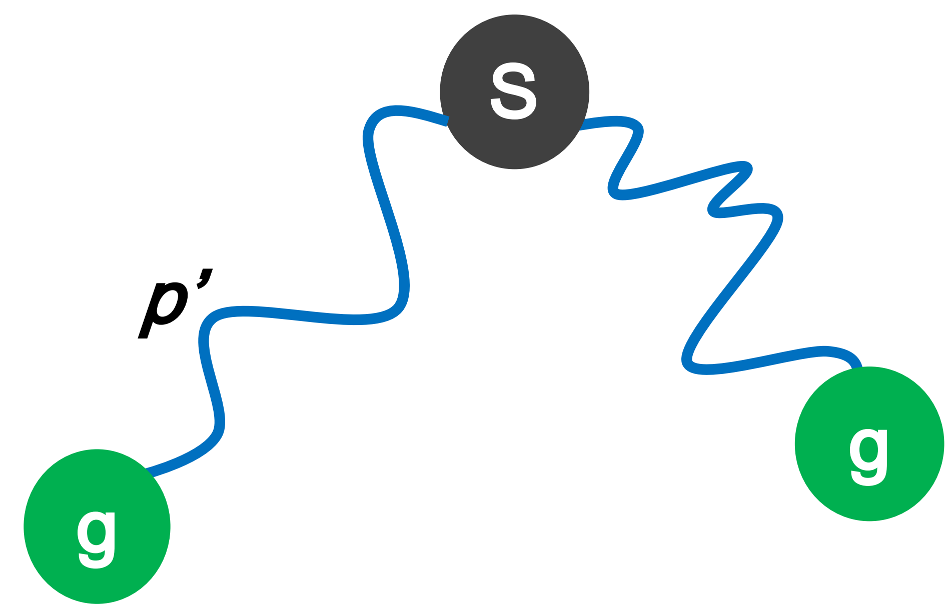
A* optimality proof

Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

Suppose A* returns path p.

Assume that there exists some other path p' that is a better path to a goal.



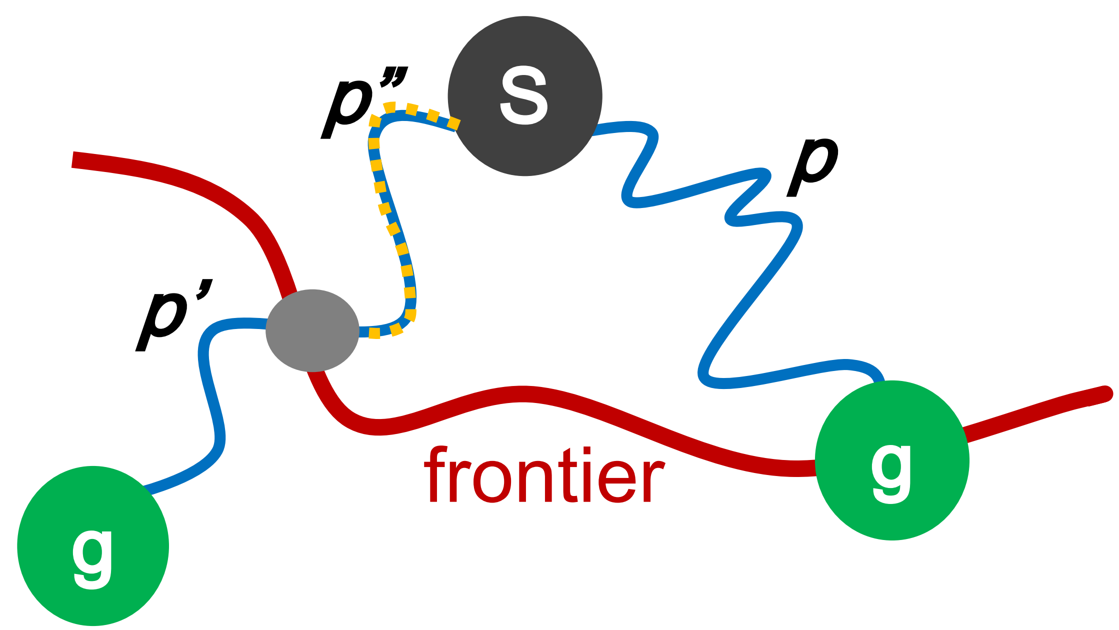
A* optimality proof

Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

Consider the moment when p is selected from the frontier.

Some part of p' will also be on the frontier. Let's call this part p''



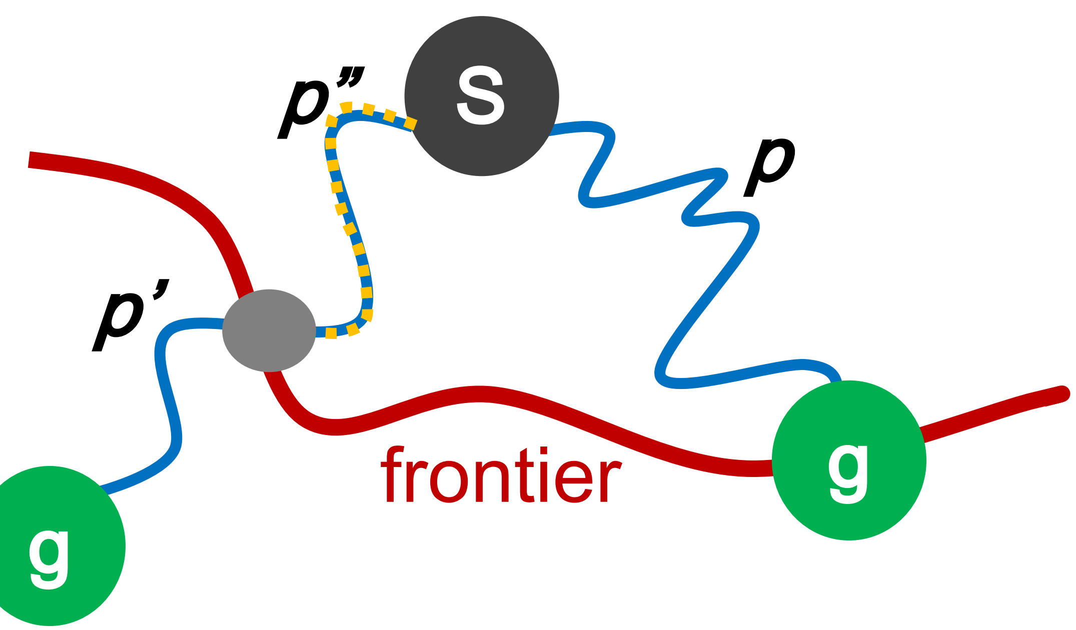
A* optimality proof

Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

Because p was expanded before p'', $f(p) \leq f(p'')$ and so

$$cost(p) + h(p) \leq cost(p'') + h(p'')$$



A* optimality proof

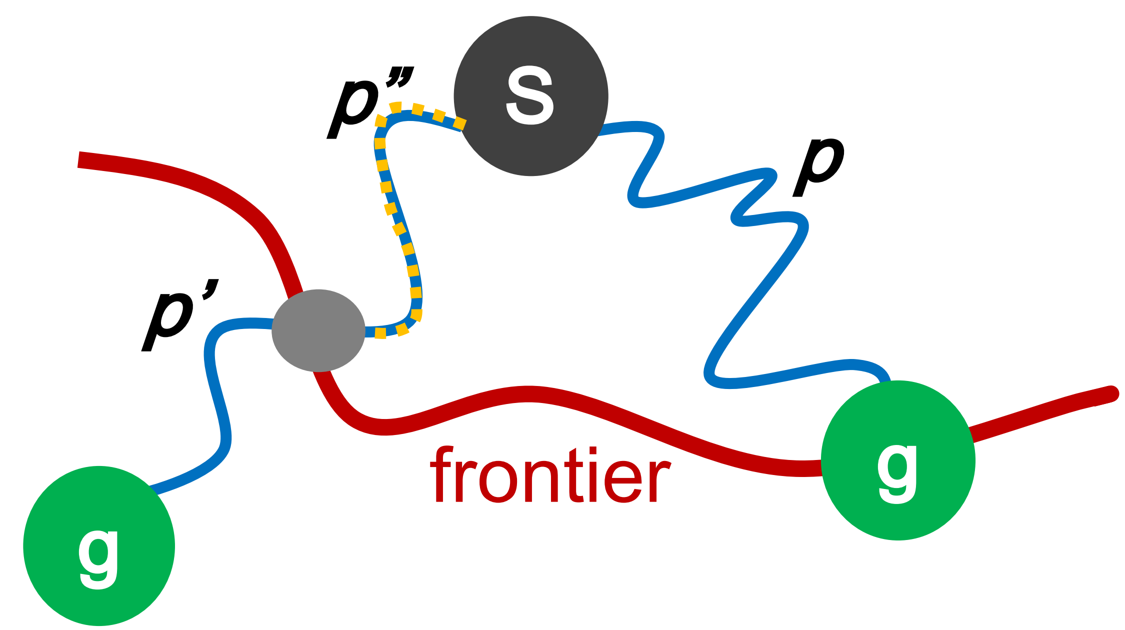
Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

Because p was ends at goal,

$$h(p) = 0$$

$$cost(p) + \cancel{h(p)} \leq cost(p'') + h(p'')$$



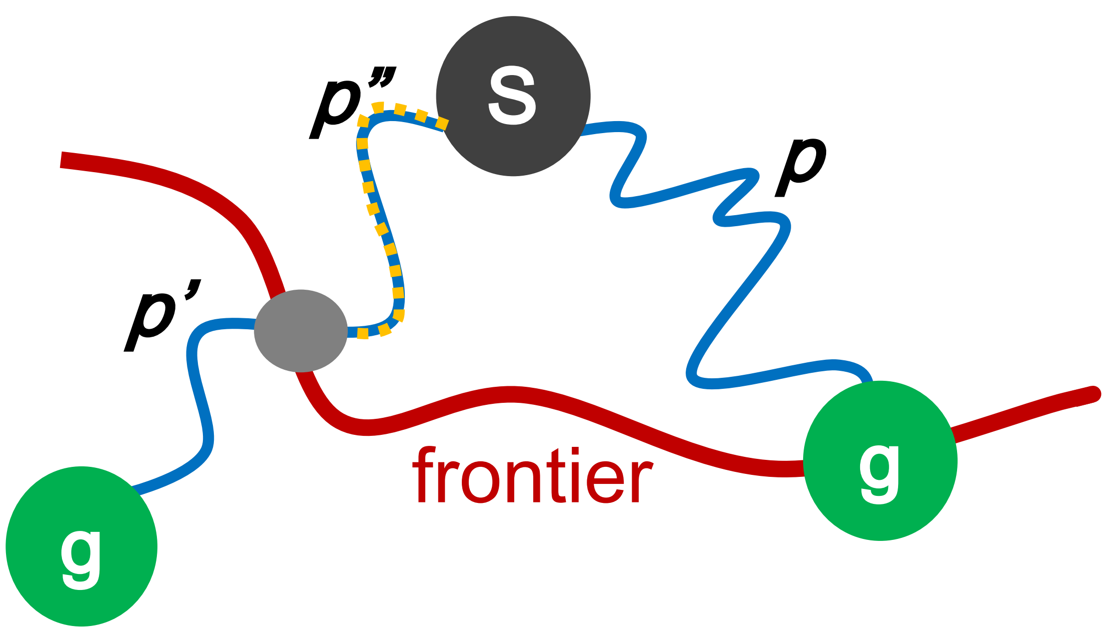
A* optimality proof

Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

Because p was ends at goal, $h(p) = 0$

$$\boxed{cost(p) \leq cost(p'') + h(p'')}$$



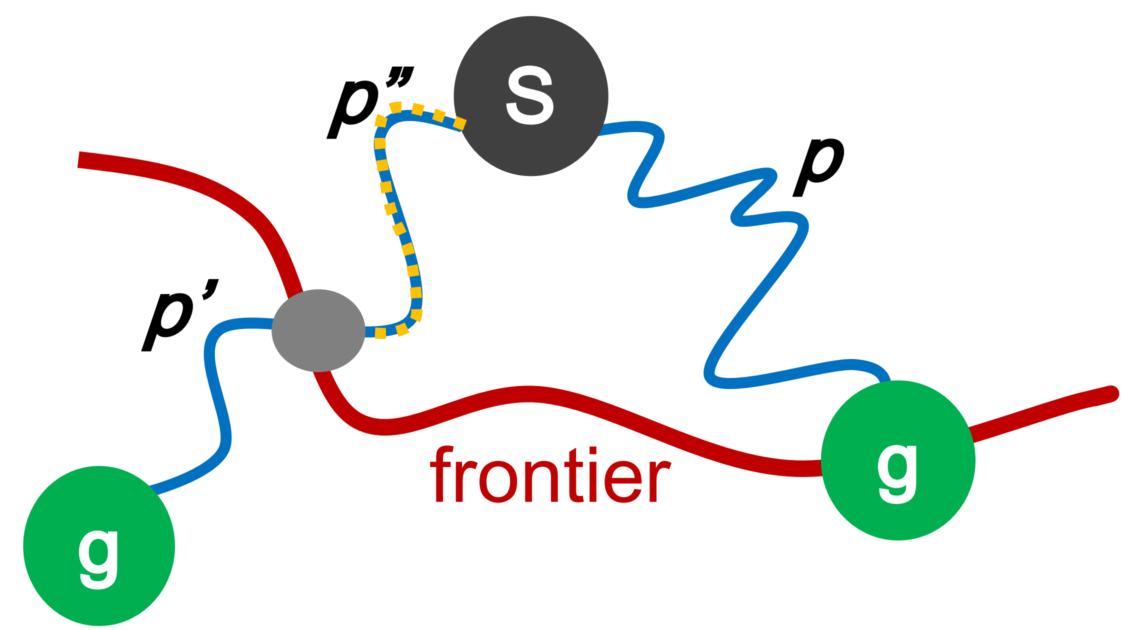
A* optimality proof

Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

Because h is admissible,

$$\boxed{cost(p'') + h(p'') \leq cost(p')}$$

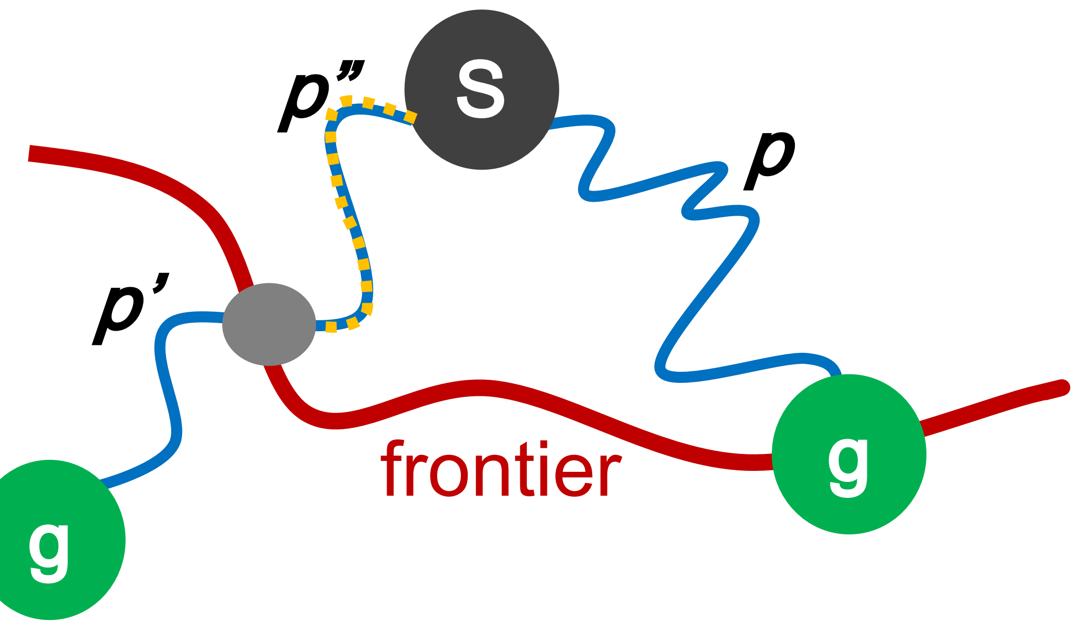


A* optimality proof

Theorem: If A* selects a path p as the solution, then p is an optimal (i.e., lowest-cost) path.

Proof by contradiction

$cost(p) \leq cost(p'') + h(p'')$ and
$cost(p'') + h(p'') \leq cost(p')$
implies that
$cost(p) \leq cost(p')$



This contradicts our assumption that p' is a better path.

Optimal efficiency of A*

In fact, we can prove something even stronger about A*: Given the particular heuristic that is available, **no search algorithm could do better!**

Optimal efficiency: Among all optimal algorithms, that start from the same start node and use the same heuristic h, A* expands the minimum number of paths.

Search: summary table

Uninformed Informed

	complete?	optimal?	time O( )	space O( )
DFS	No	No	b^m	mb
BFS	Yes	Yes*	b^m	b^m
IDS	Yes	Yes*	b^m	mb
LCFS	Yes^	Yes^	b^m	b^m
BestFS	No	No	b^m	b^m
A*	Yes	Yes^+	b^m	b^m

*Assuming arc costs are equal
^ Assuming arc costs are positive
+ Assuming $h(n)$ is admissible

ASIDE: History of A*

- The algorithm was first published in 1968 at Stanford Research Institute as part of the Shakey project for Shakey's path planning.
- They started with Dijkstra's algorithm.
- Then created extensions A1, A2, and then collectively named them as A*.

Lecture outline

- Recap from last lecture (~10 mins)
- A* analysis (~15 mins)
- **Branch and bound (~10 mins)** 👈🏻
- A* enhancements (~5 mins)
- Class activity (~10 mins)
- Pruning (~15 mins)
- Summary and wrap-up (~5 mins)

Is A* the answer to all searching needs?

- Informed ✅
- Complete ✅
- Optimal ✅
- Optimally efficient ✅

For most problems, the number of states is still a problem; it is exponential in the length of the solution 😞. So **space** is still an issue.

Branch and bound search (B&B)

- Follow exactly the same search path as depth-first search
  - Treat the frontier as a stack
  - Expand the most-recently added path first
  - The order in which neighbours are added to the frontier can be governed by some arbitrary node-ordering approach
    - Assignment 1 specified alphabetical ordering
    - We could also use some ordering based on f-score

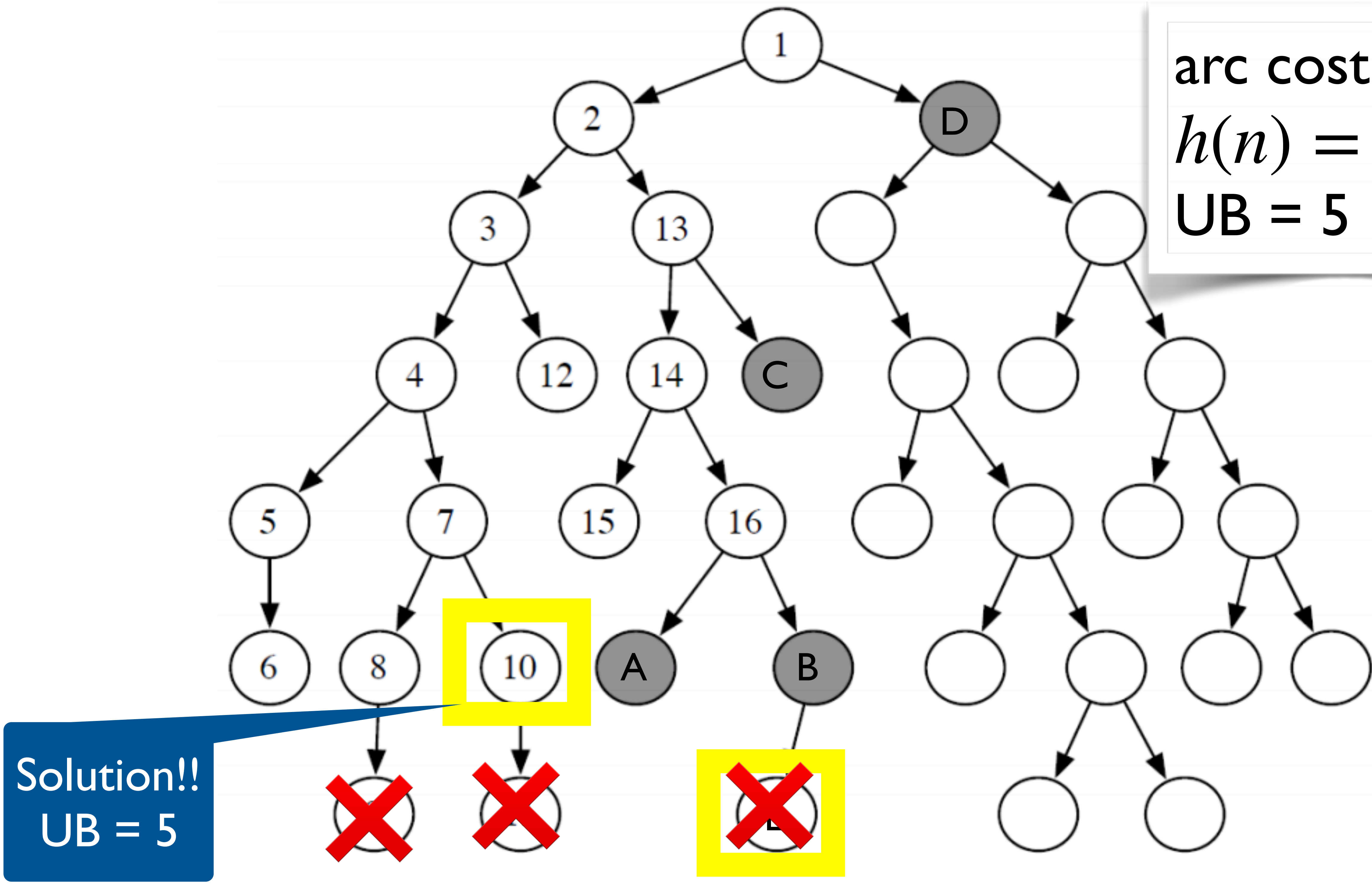
B&B

- A way to combine DFS with heuristic guidance
- Follows exactly the same search path as DFS but to ensure optimality, it does not stop at the first solution found
- Then prune all paths encountered that have cost $\geq$ the cost of the best solution so far

B&B

- Keep track of a lower bound and upper bound on solution cost at each path.
  - Lower bound: $LB(p) = f(p) = cost(p) + h(p)$
  - Upper bound: $UB(p) =$ cost of the best solution found so far
  - Initialize $UB = \infty$ or some overestimate of the solution cost
- When path p is selected for expansion, if $LB(p) \geq UB$, remove p from frontier without expanding it (pruning) else expand p adding all of its neighbours to the frontier.

Example: B&B

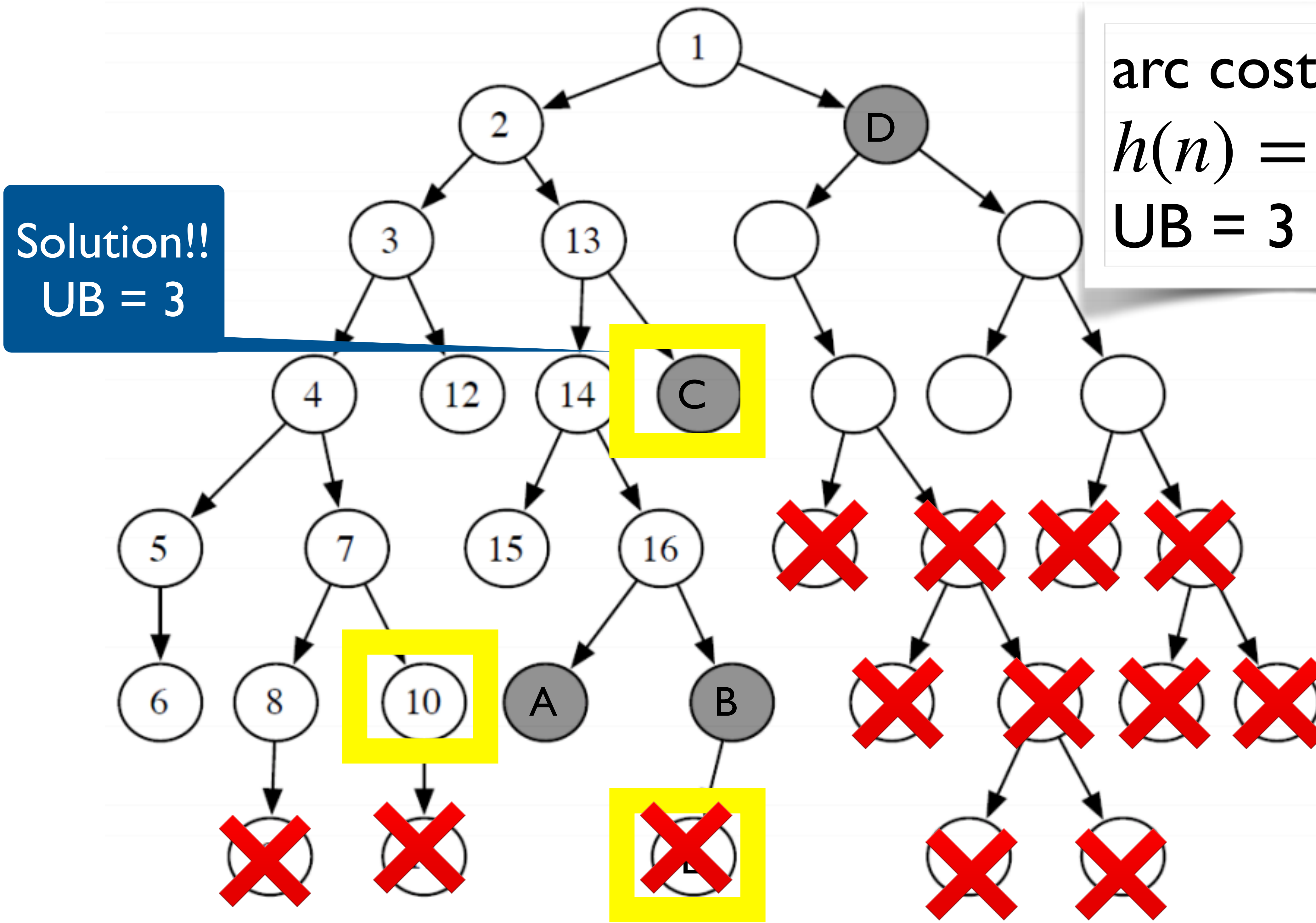


arc cost = 1

$h(n) = 1 \, \forall n$

UB = 5

Example: B&B



arc cost = 1

$h(n) = 1 \; \forall n$

UB = 3

B&B

Once B&B has found a solution, what does it do next?

A. Stop and return that solution

B. Keep searching looking for deeper solutions

C. Keep searching but only for shorter solutions ✅

D. None of the above

E. Create a start up with this novel algorithm

B&B completeness

Is branch and bound complete?

A. Yes

B. No ✅

B&B completeness

- Not in general, for the same reasons that DFS isn't complete.
    - But complete if initialized with some finite upper bound (an overestimate of the solution cost).
- For many problems of interest there are no infinite paths and no cycles.
- Hence, for many problems B&B is complete.

B&B optimality

Is branch and bound optimal?

A. Yes ✅

B. No

B&B time and space complexity

- Time complexity: $O(b^m)$
- Space complexity: $O(mb)$ like DFS!
  - Big improvement over A*

Lecture outline

- Recap from last lecture (~10 mins)
- A* analysis (~15 mins)
- Branch and bound (~10 mins)
- **A* enhancements (~5 mins)** 👈
- Class activity (~10 mins)
- Pruning (~15 mins)
- Summary and wrap-up (~5 mins)

Other A* tricks

The primary problem with A* is that in the worst case, it uses exponential space. Branch and bound is a way around this problem. Are there other ways?

- Iterative Deepening A* (IDA*)
- Memory-bounded A*

Heuristic iterative deepening (IDA*)

- Branch and bound improves over A* but it can still get stuck in infinite (extremely long) paths
- Search depth-first but to a fixed depth/bound
  - Depth is measured in f-values
  - If you do not find a solution, update the bound with the lowest f that passed the previous bound and try again

Analysis of IDA*

- Complete and optimal? Same conditions as A*
  - the branching factor is finite
  - arc costs are $> \epsilon > 0$
  - $h(n)$ is admissible
- Space complexity? $O(mb)$ 😊
- Time complexity? $O(b^m)$

Memory-bounded A*

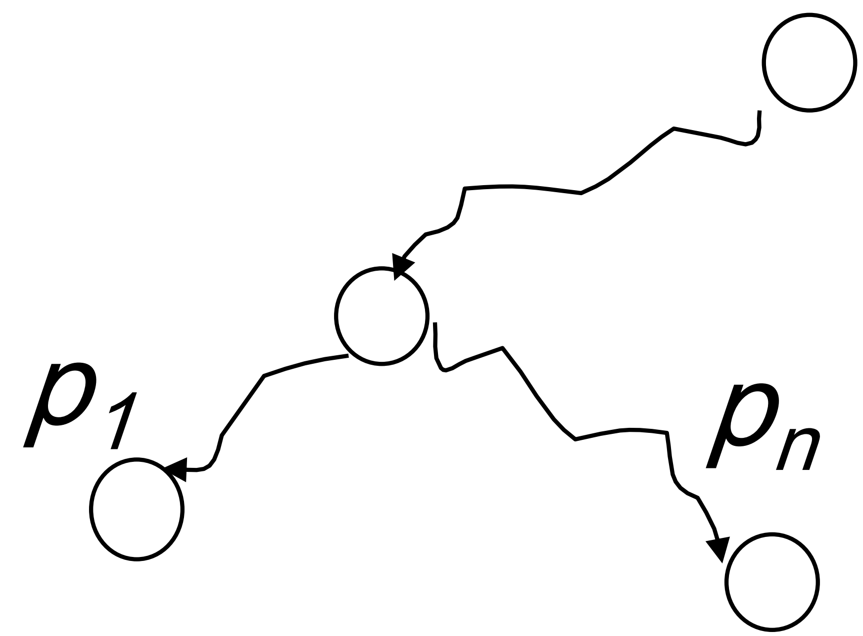
- IDA* and B&B use a tiny amount of memory
  What if we have more memory available?
  Keep as much of the frontier in memory as we can

- If we have to delete something:

  - delete the "worst" paths (with highest f-values.)

  - "back them up" to a common ancestor

- Update the heuristic value of the ancestor if possible



Analysis of MBA*

- Complete?
  Yes, as long as there is enough memory to store the solution

- Optimal?
  Yes, if h is admissible and if there is enough memory to store the solution

- Space complexity? $O(b^m)$

- Time complexity? $O(b^m)$

Lecture outline

- Recap from last lecture (~10 mins)
- A* analysis (~15 mins)
- Branch and bound (~10 mins)
- A* enhancements (~5 mins)
- **Class activity (~10 mins)** 👈
- Pruning (~15 mins)
- Summary and wrap-up (~5 mins)

Class activity (10 mins)

Analysis of search methods and teaching feedback

	selection	complete?	optimal?	time O( )	space O( )
DFS					
BFS					
IDS					
LCFS					
BestFS					
A*					
B&B					
IDA*					
MBA*					

Search strategies: summary

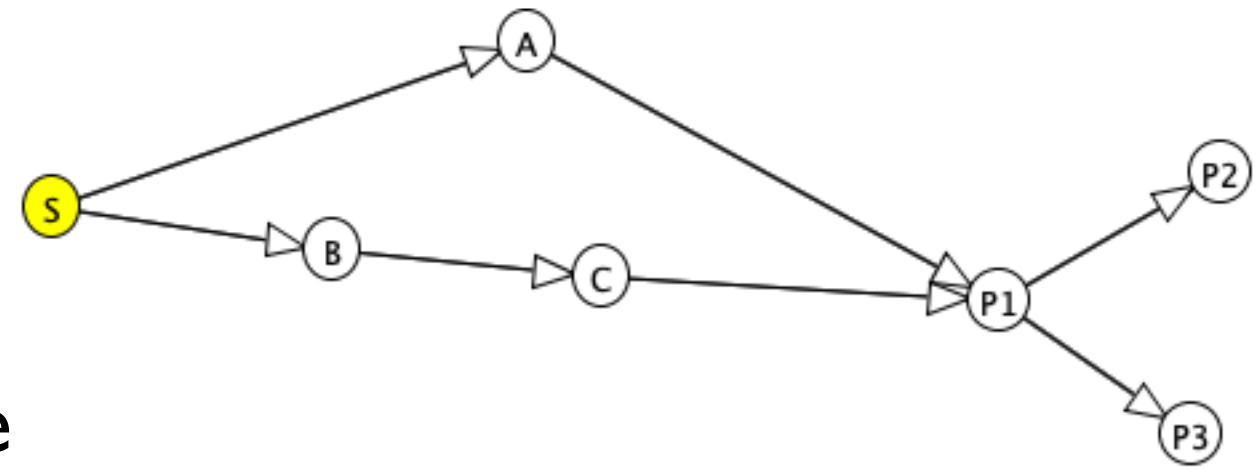
	Method	Selection	Complete	Optimal	Time O()	Space O()
Uninformed	**DFS**	LIFO	N (Y if no cycles)	N	b^m	mb
	BFS	FIFO	Y	Y	b^m	b^m
	IDS	LIFO	Y	Y	b^m	mb
	LCFS (when arc costs available)	min cost	Y (if costs > 0)	Y (if costs ≥ 0)	b^m	b^m
informed	**BestFS (When *h* available)**	min *h*	N	N	b^m	b^m
	A* (when arc costs and *h* available)	min *f*	Y if branching factor finite, *h* is admissible, and costs > 0	Y if branching factor finite, *h* isadmissible, and costs > 0	b^m	b^m
	Branch and Bound	LIFO + pruning	N (Y UB finite)	Y	b^m	mb
	IDA*	LIFO	Y (same as A*)	Y	b^m	mb
	MBA*	min *f*	Y if enough memory	Y if enough memory and h is admissible	b^m	b^m

Lecture outline

- Recap from last lecture (~10 mins)
- A* analysis (~15 mins)
- Branch and bound (~10 mins)
- A* enhancements (~5 mins)
- Class activity (~10 mins)
- **Pruning (~15 mins)**
- Summary and wrap-up (~5 mins)

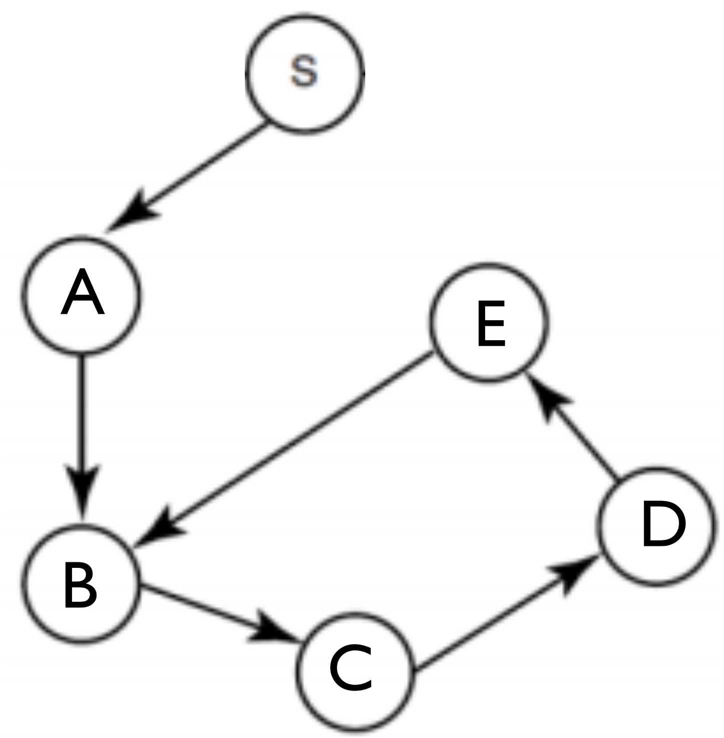
Pruning: if we only want optimal solutions

- Often there are multiple paths to a node and we only need one path.



- The search algorithms we have seen so far can be improved using pruning strategies

- Cycle pruning: Avoid using paths with cycles



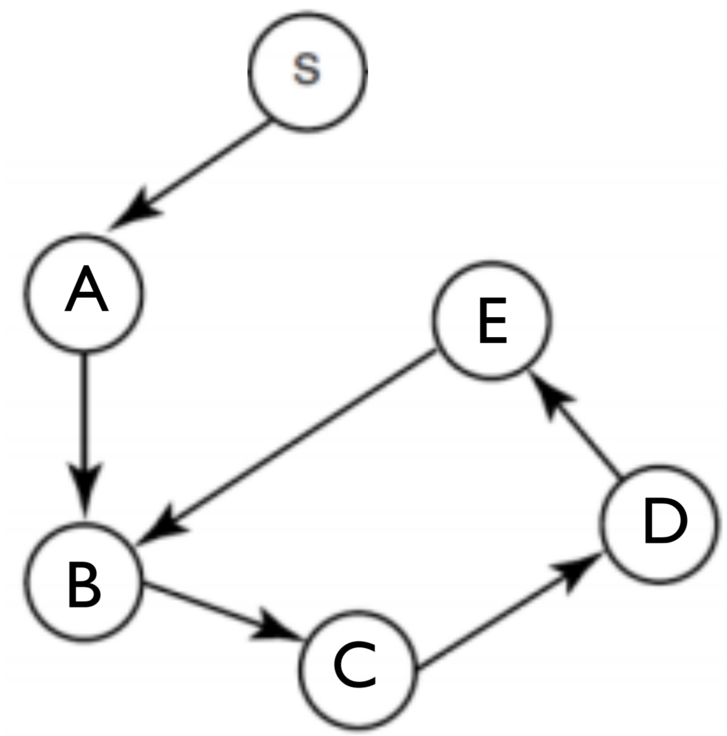
- Multiple-path pruning: Only consider one path to a node and prune all other paths

Cycle checking

- Ensure that the algorithm does not consider neighbours that are already on the path from the start.

- Check whether the last node on the path already appears earlier on the path from the start node to that node.

- What is the computational cost of cycle checking?



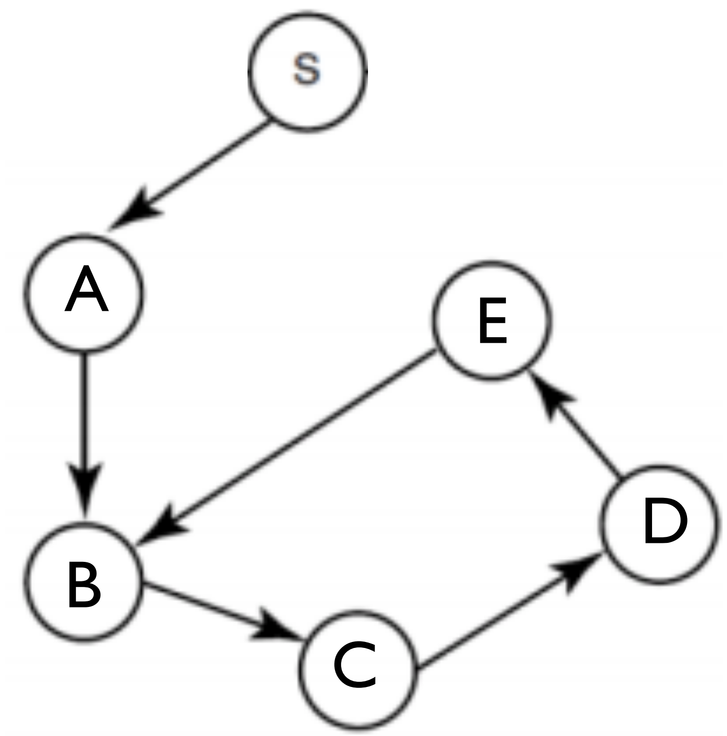
[<S,A,B,C,D,E>]

[<S,A,B,C,D,E,B>]

Do not add the path on the frontier.

Cycle checking

- What is the computational cost of cycle checking?
  - In general, linear in path length.



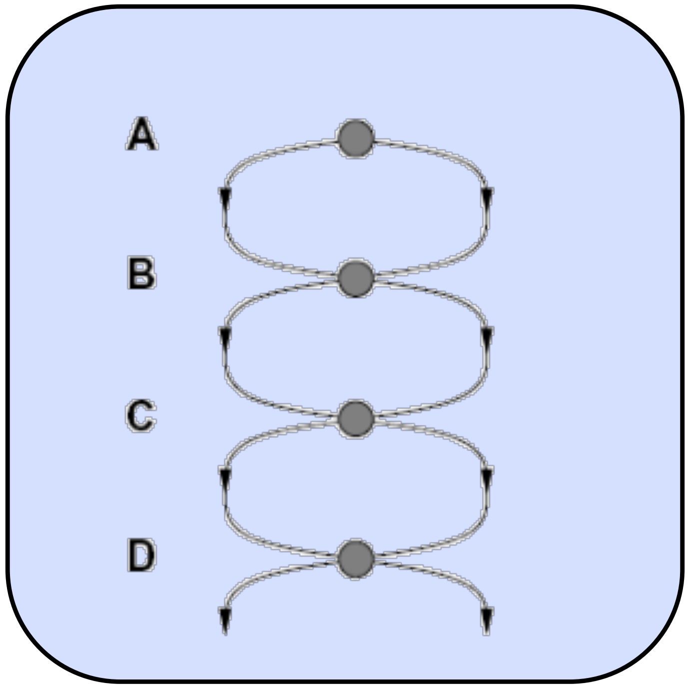
[<S,A,B,C,D,E>]

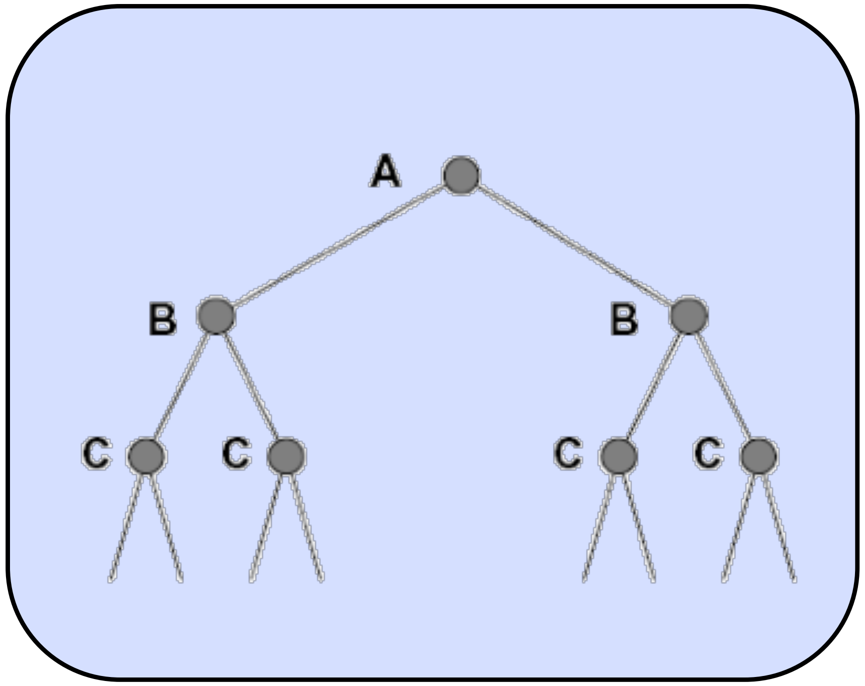
Do not add the path on the frontier.

[<S,A,B,C,D,E,B>]

Repeated states/multiple paths

Failure to detect repeated states can turn a linear problem into an exponential one!



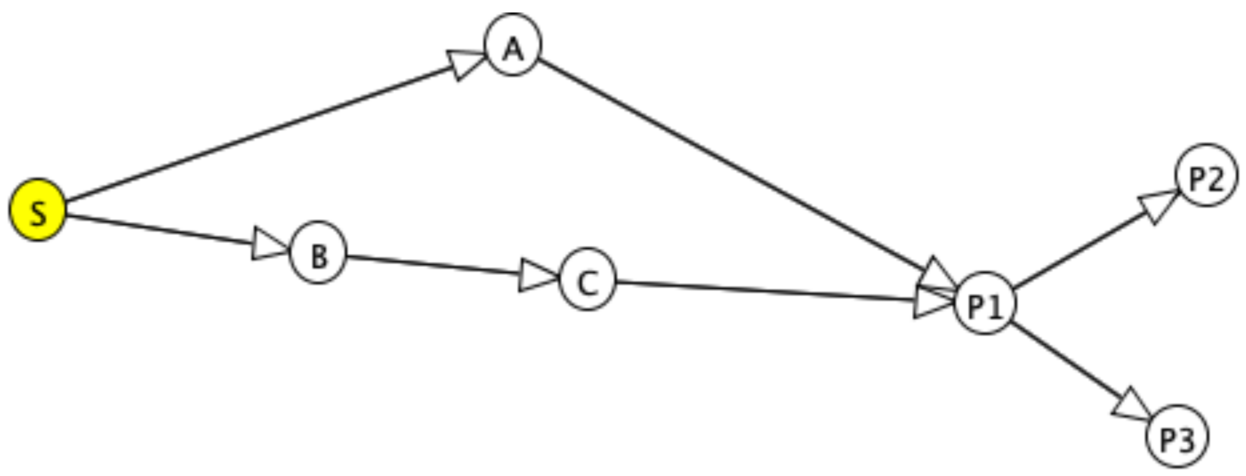


State space: 2 actions from each state to the next with d+1 states, search tree has depth d and there are 2^d possible paths.

Multiple path pruning

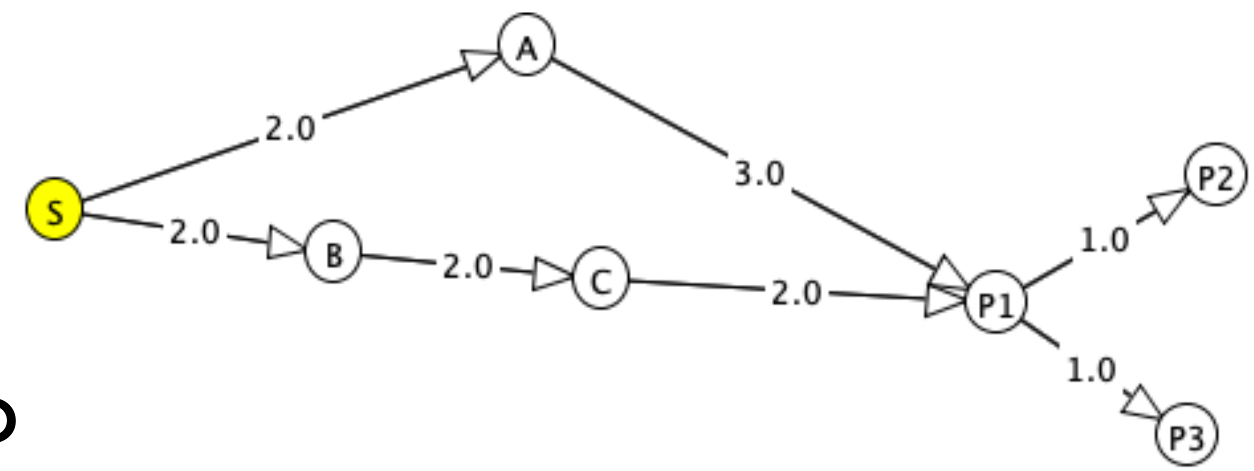
- The search algorithm can prune from the frontier any path that leads to a node to which it already has found paths.

- Implemented by **maintaining an explored set** (also called closed list), which is empty at the beginning, and is populated with the last node on the selected paths from the frontier.



Multiple path pruning

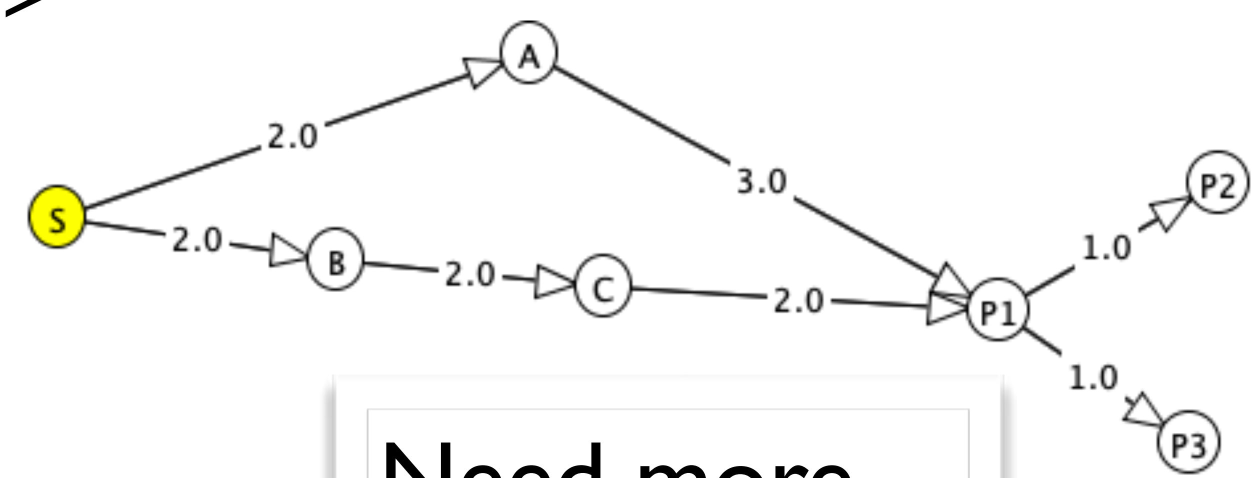
- Maintain an explored set.
- When a path $< n_0, n_1, \ldots, n_k >$ is selected from the frontier, check if n_k is already in the explored set.
- If yes, it can be discarded. If no we add n_k to the explored set.



Does this method guarantee that the least-cost path is not discarded?

Multiple path pruning

- Maintain an explored set.
- When a path $< n_0, n_1, \ldots, n_k >$ is selected from the frontier, check if n_k is already in the explored set.
- If yes, it can be discarded. If not, we add n_k to the explored set.



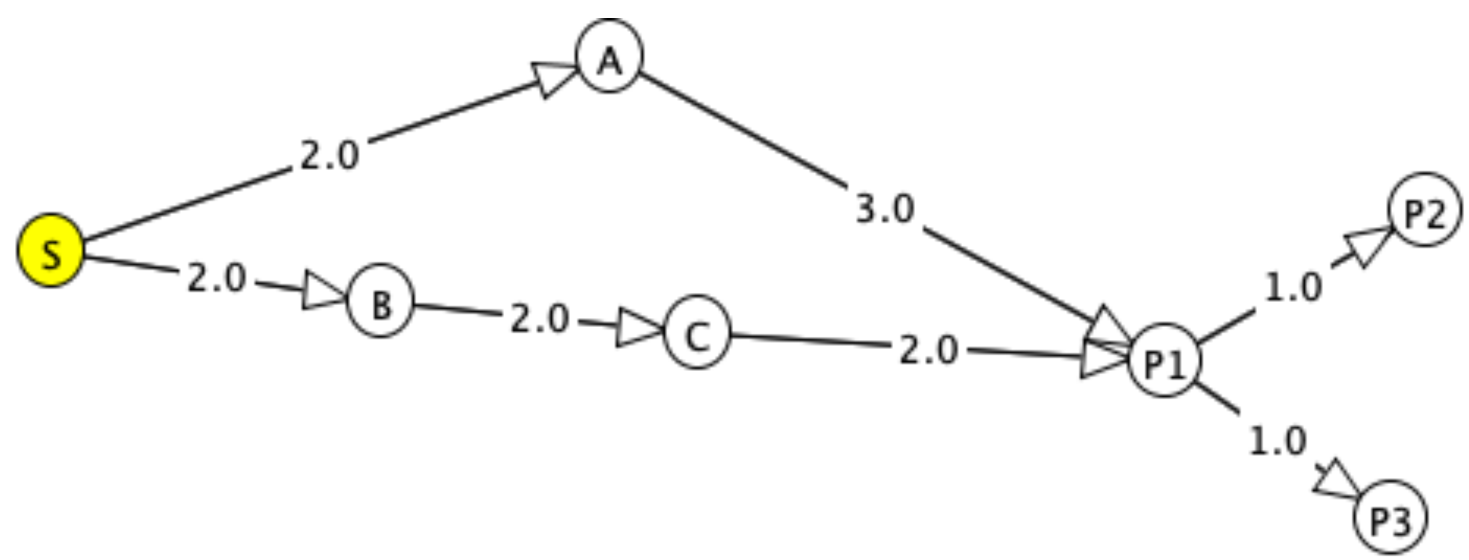
Need more sophisticated approaches

Does this method guarantee that the least-cost path is not discarded? ✗

Multiple path pruning

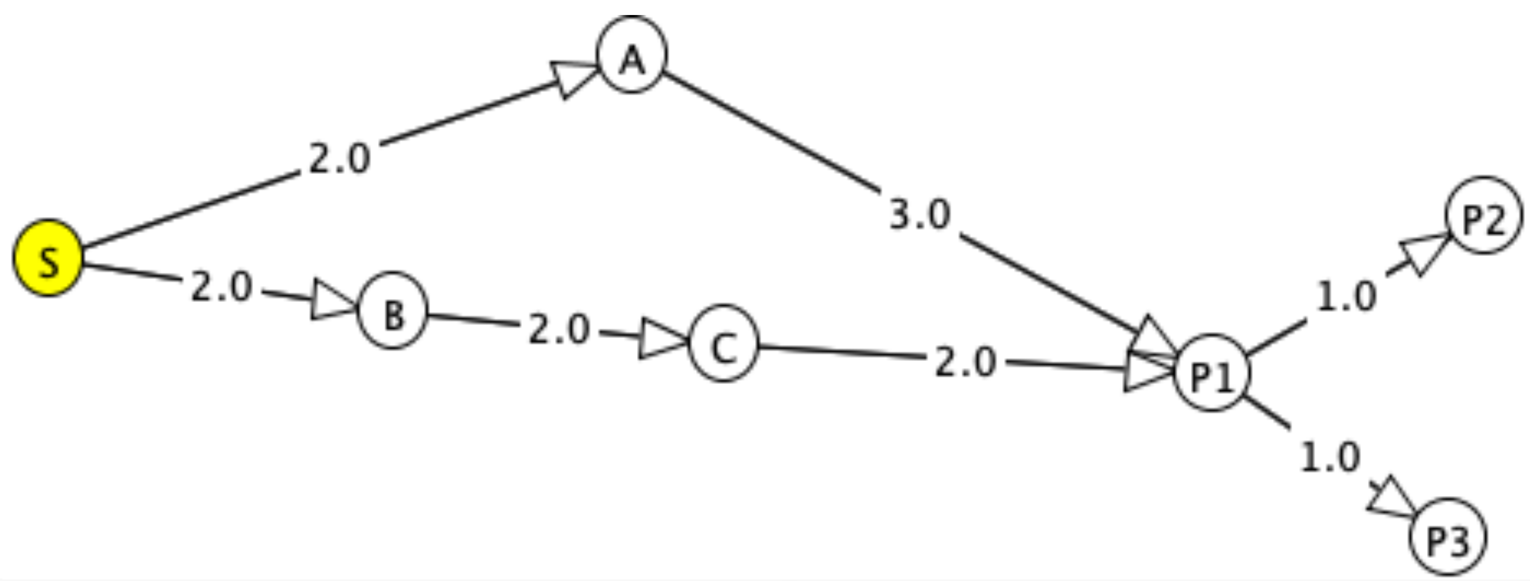
Approach 1: Make sure that the first path found to any node is a lowest-cost path to that node, then prune all subsequent paths found to that node.

Works for LCFS but not for A*



Multiple path pruning

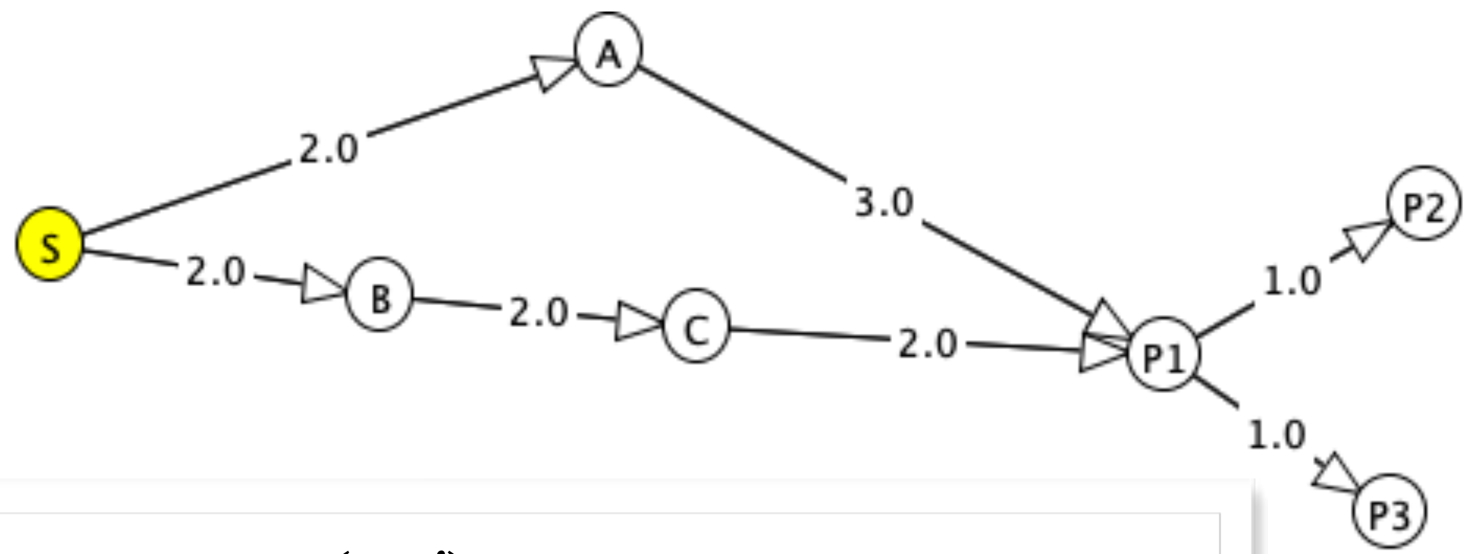
Approach 2: If the search algorithm finds a lower-cost path to a node than one already found, it could remove all paths that used the higher-cost path to the node because they cannot be on an optimal solution.



If $cost(p) < cost(p')$, remove all paths from the frontier with prefix p'

Multiple path pruning

Approach 3: Whenever the search finds a lower-cost path to a node than a path to that node already found, change the initial segment of the paths on the frontier to use the lowest-cost path.

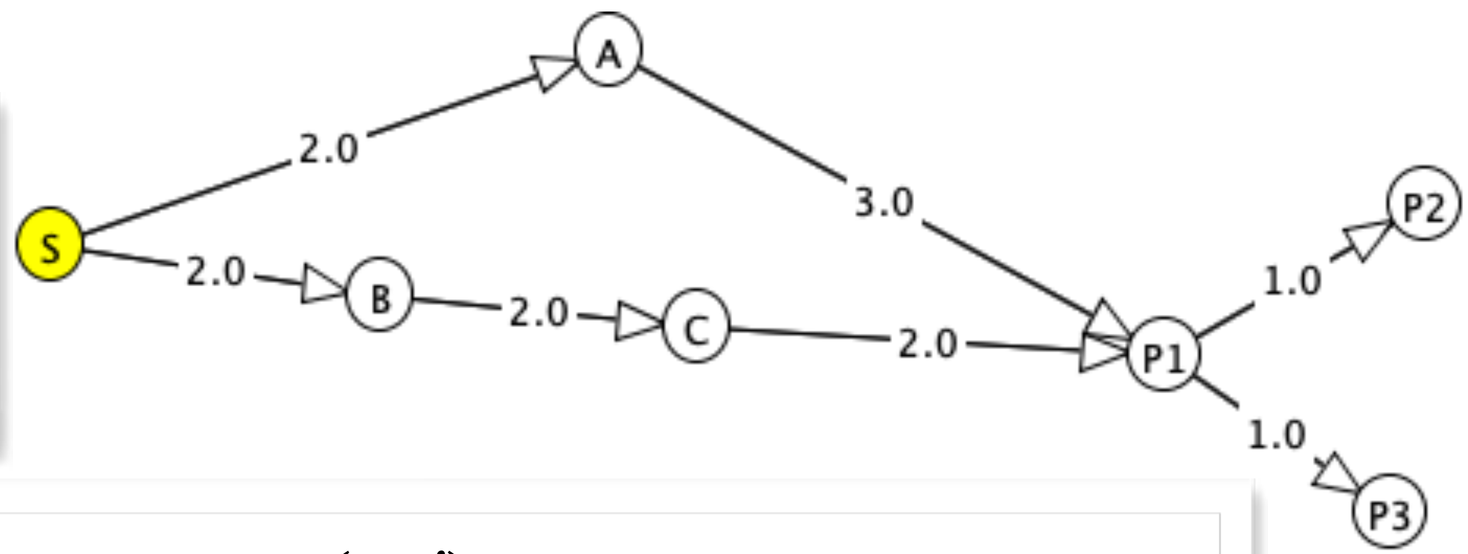


If $cost(p) < cost(p')$, replace prefixes in those paths (replace p' with p)

Multiple path pruning

Approach 3: Whenever the search finds a lower-cost path to a node than a path to that node already found, change the initial segment of the paths on the frontier to use the lowest-cost path.

Would it guarantee optimality for A*?

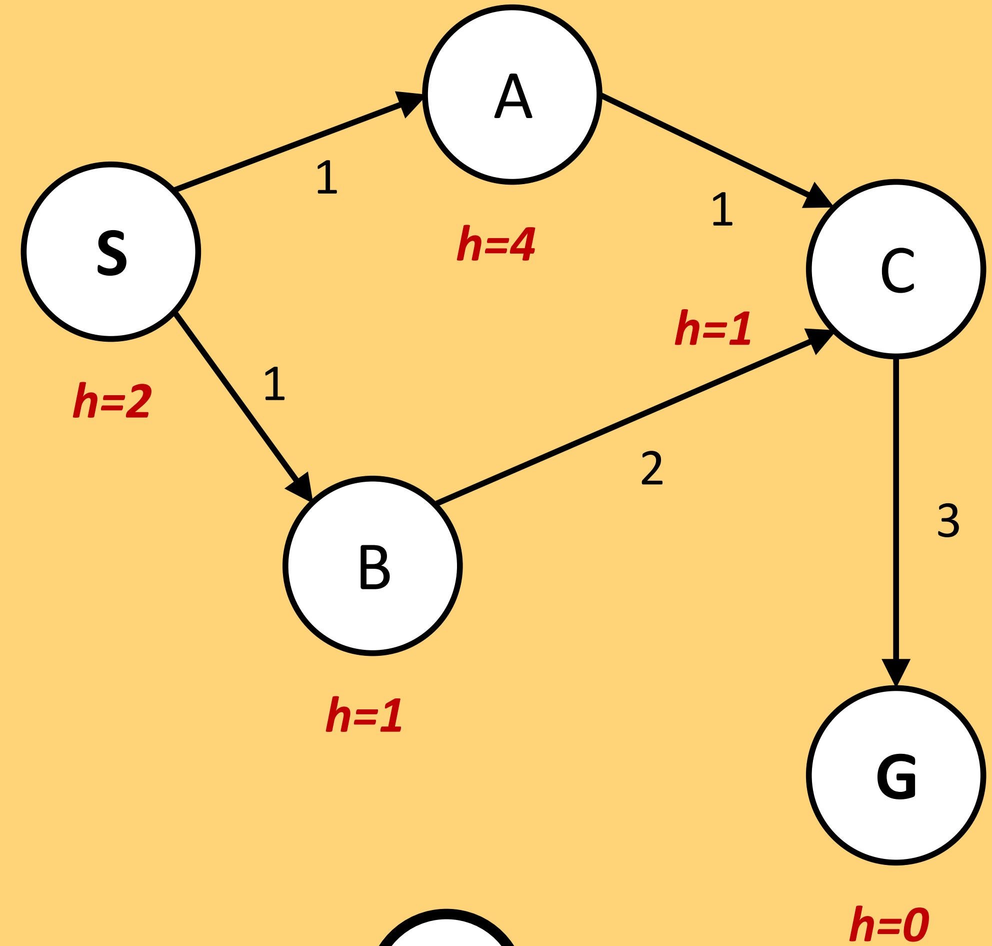


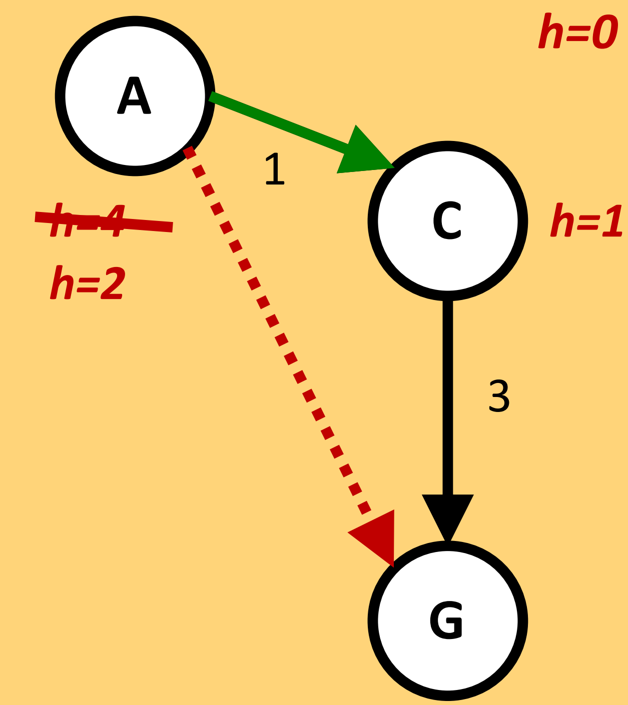
If $cost(p) < cost(p')$, replace prefixes in those paths (replace p' with p)

ASIDE: Consistency of heuristics

Main idea: estimated heuristic cost $\leq$ actual cost

- Admissibility: heuristic cost $\leq$ actual cost to goal
  - h(A) $\leq$ actual cost from A to G
- Consistency: heuristic "arc" cost $\leq$ actual cost for each arc
  - $h(A) - h(C) \leq cost(A \text{ to } C)$
- The f value along a path never decreases
  - $h(A) \leq cost(A \text{ to } C) + h(C)$





Lecture outline

- Recap from last lecture (~10 mins)
- A* analysis (~15 mins)
- Branch and bound (~10 mins)
- A* enhancements (~5 mins)
- Class activity (~10 mins)
- Pruning (~15 mins)
- **Summary and wrap-up (~5 mins)** 👈

Revisiting learning outcomes for search

Important for exams

- Identify real world examples that make use of deterministic, goal-driven search agents
- Assess the size of the search space of a given search problem.
- Implement the generic solution to a search problem.
- Apply basic properties of search algorithms: completeness, optimality, time and space complexity
- Select the most appropriate search algorithms for specific problems.

Revisiting learning outcomes for search

Important for exams

- Define/read/write/trace/debug different search algorithms
- Construct heuristic functions for specific search problems
- Formally prove A* optimality.
- Define optimally efficient

A rough CPSC 322 overview

Representation and reasoning

Environment

Problem		Deterministic	Stochastic
Static	Constraint satisfaction	Variables + constraints Arc consistency Search	
Static	Query	Logics Search	Belief networks Variable elimination
Sequential	Planning	STRIPS Search	Decision networks Variable elimination Markov decision processes Value iteration

Search is everywhere!

Coming up

- Submit your assignment on Sept. 30
- Start Constraint Satisfaction Problems (Chapter 4)