Heuristics

CS3243: Introduction to Artificial Intelligence – Lecture 4

NUS | Computing
National University of Singapore

1 Administrative Matters

Midterm Assessment

- Week 7 Lecture Slot
  - Date: 2 October (Wednesday)
  - Time: 1005-1135 hrs
  - Venue: LT16 (COM2)
- Format
  - Duration: 1 hour and 30 minutes
  - Total: 30 marks
  - Closed-book + Cheat Sheet (1 × Double-sided A4 Sheet)
- Topics
  - Everything up to Lecture 5a (i.e., up to and including Local Search)

Practice Papers:
Canvas > CS3243 > Files > Past Papers > Midterm

Upcoming...

- Deadlines
  - Tutorial Assignment 2 (released last week)
    - Post-tutorial submission: Due this Friday!
  - Tutorial Assignment 3 (released today)
    - Pre-tutorial Submission: Due this Sunday!
    - Post-tutorial Submission: Due next Friday!
  - Project 1.1 (released last Monday – 26 August)
    - Due next Sunday! 15 September 2024

Project 1 Consultations:
Thursdays via Zoom
Canvas > Announcements

Remember that there are attendance marks for tutorials!

Late Penalties
- < deadline +24 hours = 80% of score
- < deadline +48 hours = 50% of score
- ≥ deadline +48 hours = 0% of score

Contents

- Reviewing Informed Search
- Heuristics & Dominance
- Relaxing the Problem
- Reviewing the Midterm

Reference: AIMA 4th Edition, Section 3.6

2 Reviewing Informed Search

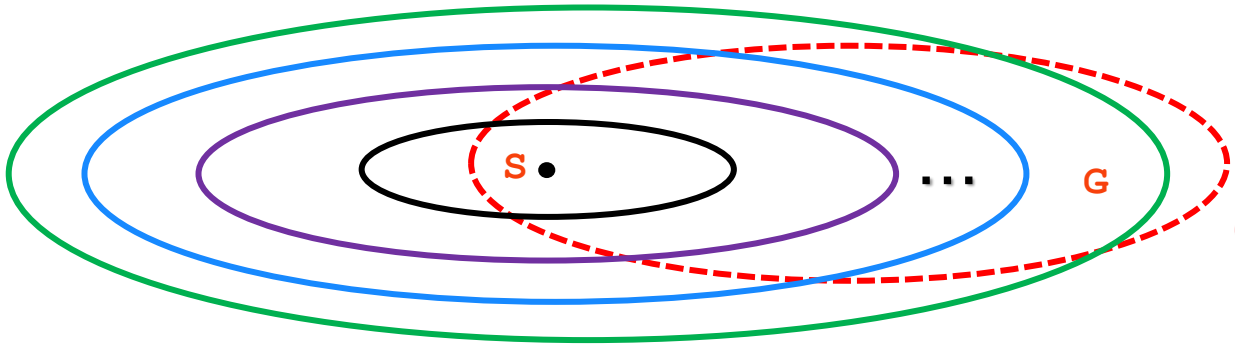
Searching Less?

- Uninformed search algorithms are systematic
  - Search outward from the initial state
  - Search in ALL directions
- Search spaces are LARGE
- Can we search more efficiently?

General Idea

Use domain knowledge about the problem environment to determine the cost required to go from a particular state to its nearest goal

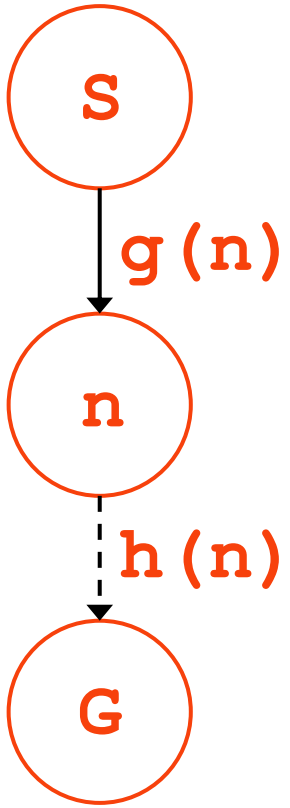
→ informed search



try to focus on searching in the direction of the goal

Approximate Path Costs to Guide Search

- Frontier → Priority Queue
  - Ordered by evaluation function `f(n)`
    - Priority for path corresponding to node `n`
  - Uniform Cost Search priority: `f(n) = g(n)`
  - Greedy Best-First Search priority: `f(n) = h(n)`
  - A* Search priority: `f(n) = g(n) + h(n)`
- General idea
  - use domain knowledge design heuristic function `h` to estimate the cost from `n.state` to `G`

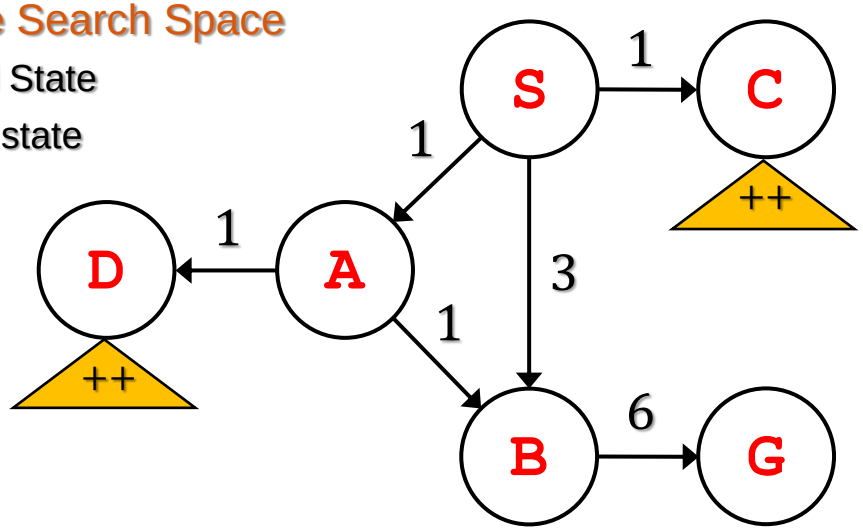


`g(n)` : Actual path cost from `S` to `n` based on the specific path taken by node `n`

`h(n)` : Approximate path cost from `n` to its nearest goal given by heuristic function `h`

Translating a Search Graph to Search Tree

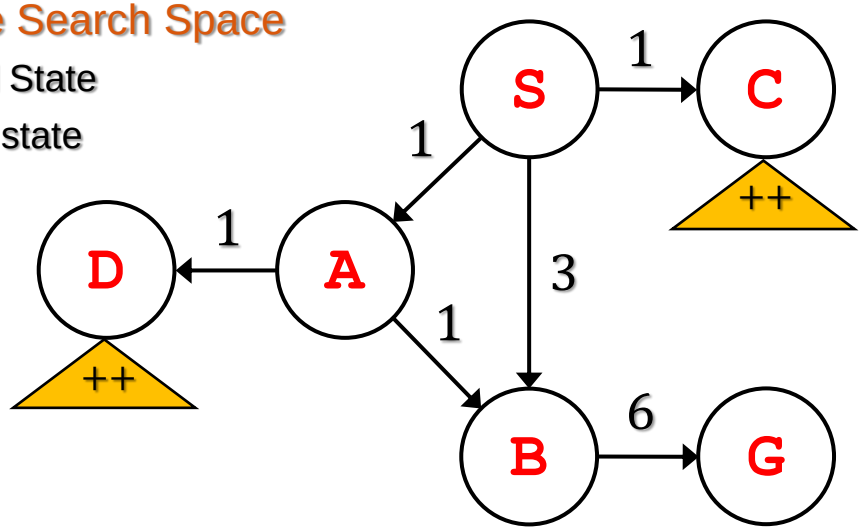
- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

Translating a Search Graph to Search Tree

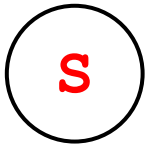
- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

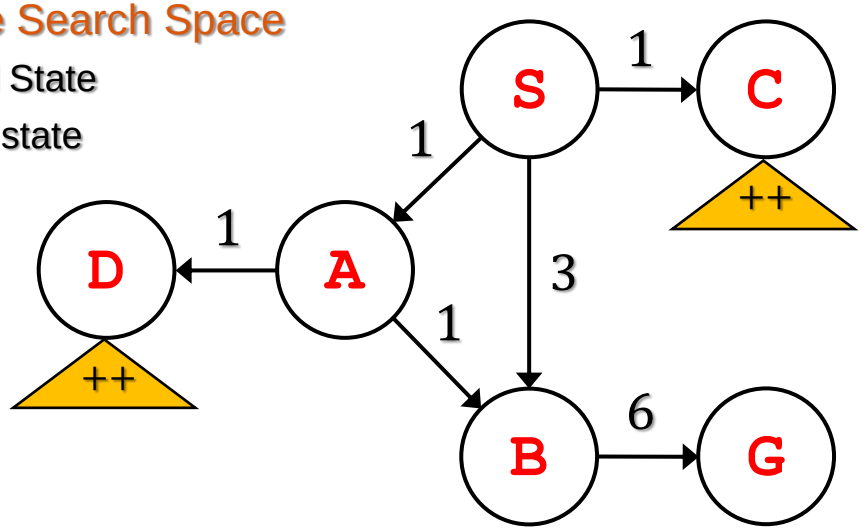
- Paths (and sub-path) costs

- Corresponding Search Tree



Translating a Search Graph to Search Tree

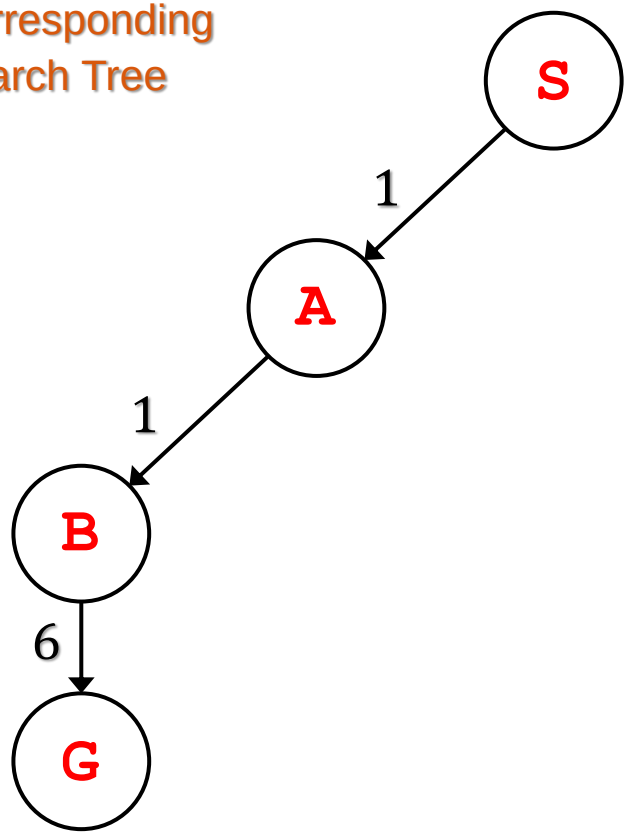
- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

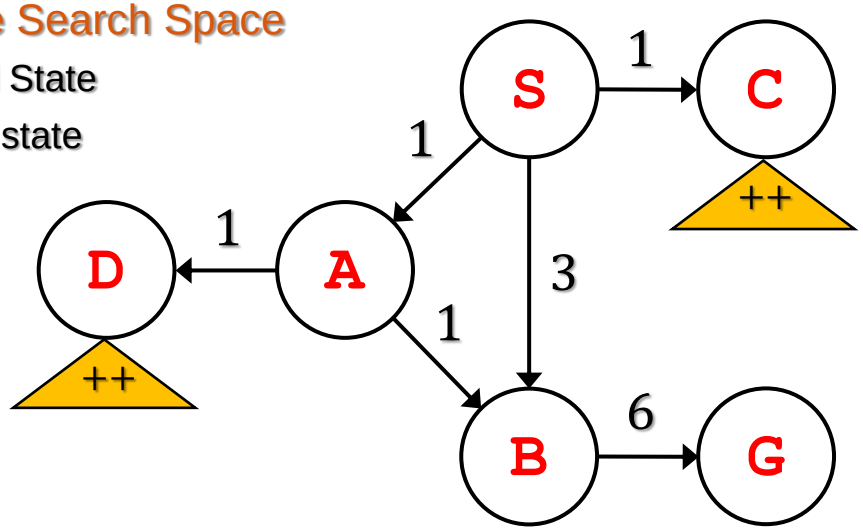
- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal

- Corresponding Search Tree



Translating a Search Graph to Search Tree

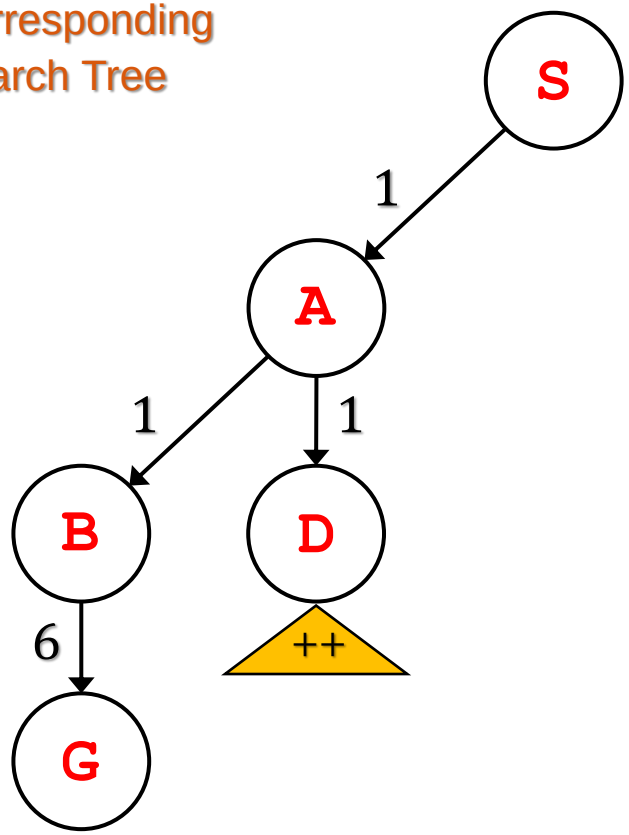
- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

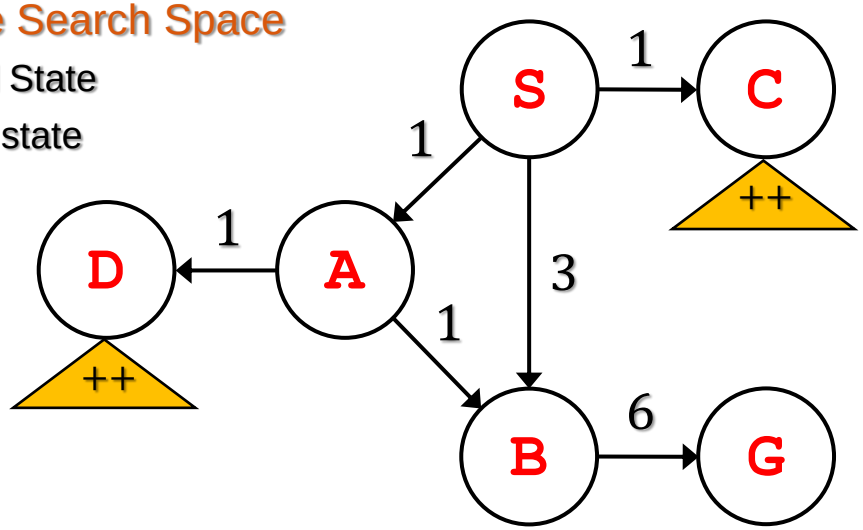
- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal
  - **S-A-D++** : ∞ units to goal

- Corresponding Search Tree



Translating a Search Graph to Search Tree

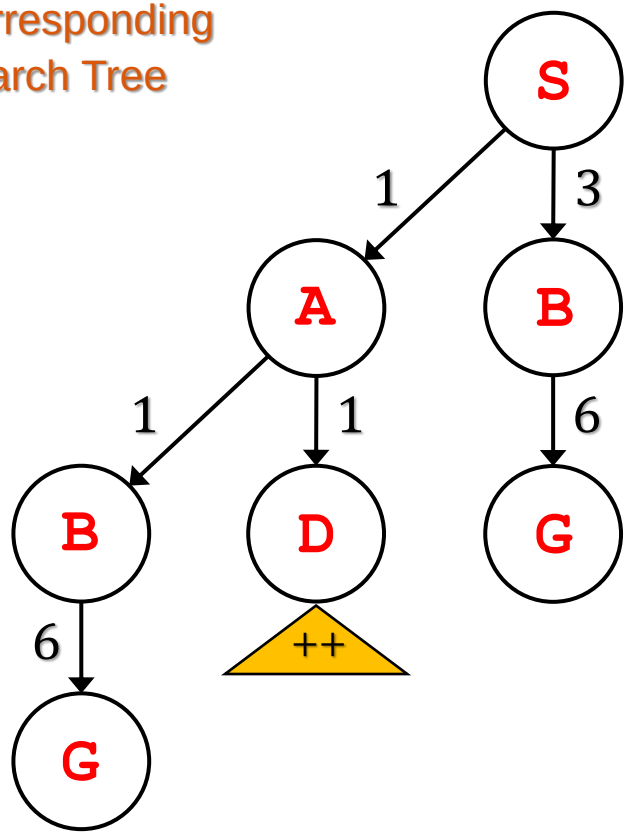
- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

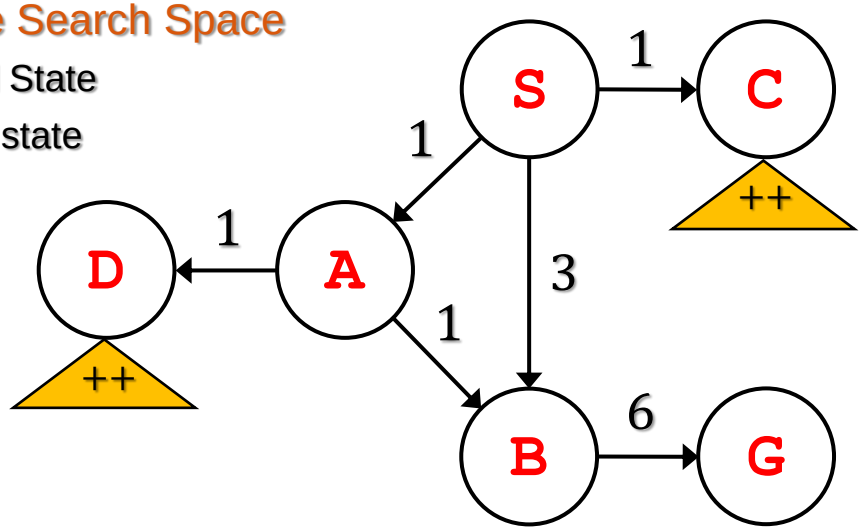
- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal
  - **S-A-D++** : ∞ units to goal
  - **S-B-G** : 9 units to goal

- Corresponding Search Tree



Translating a Search Graph to Search Tree

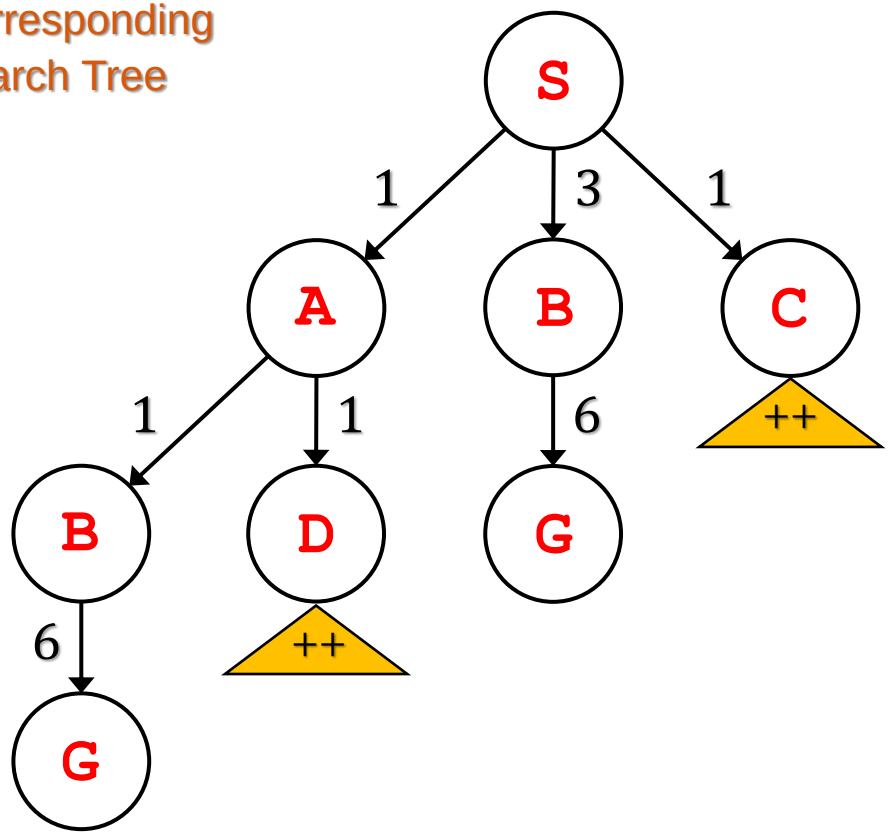
- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

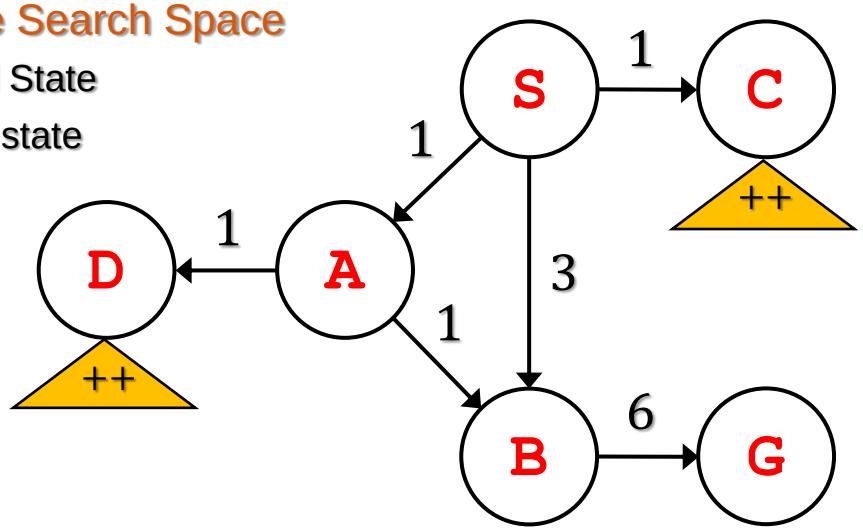
- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal
  - **S-A-D++** : ∞ units to goal
  - **S-B-G** : 9 units to goal
  - **S-C++** : ∞ units to goal

- Corresponding Search Tree



What are the Paths Discovered by UCS?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state

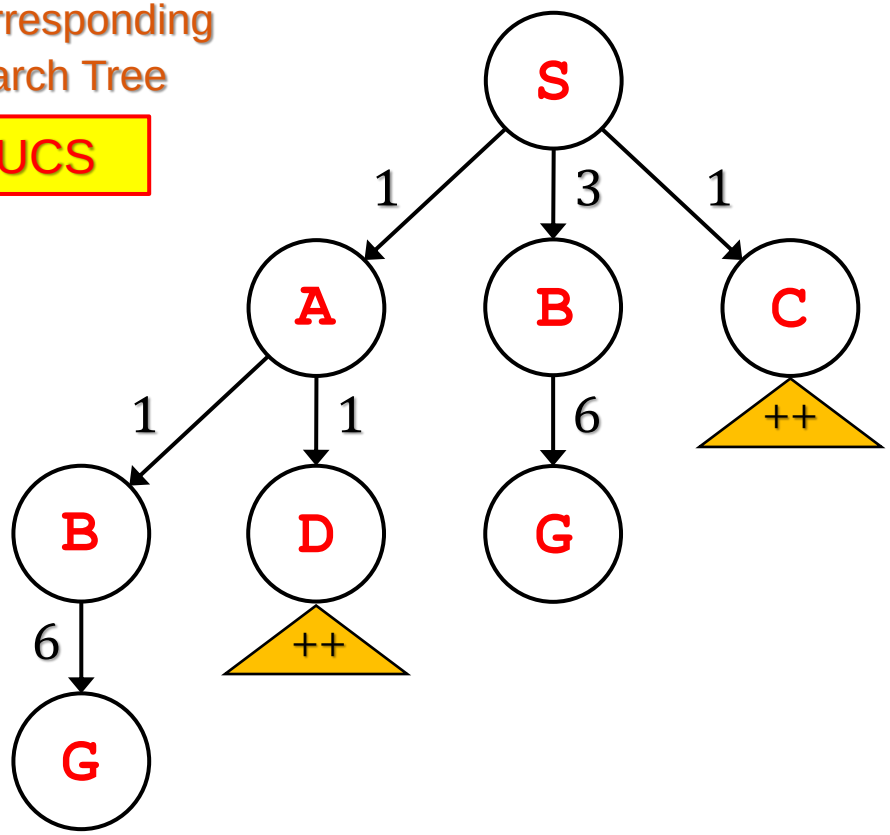


++ Subtrees with no goal

- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal
  - **S-A-D++** : ∞ units to goal
  - **S-B-G** : 9 units to goal
  - **S-C++** : ∞ units to goal

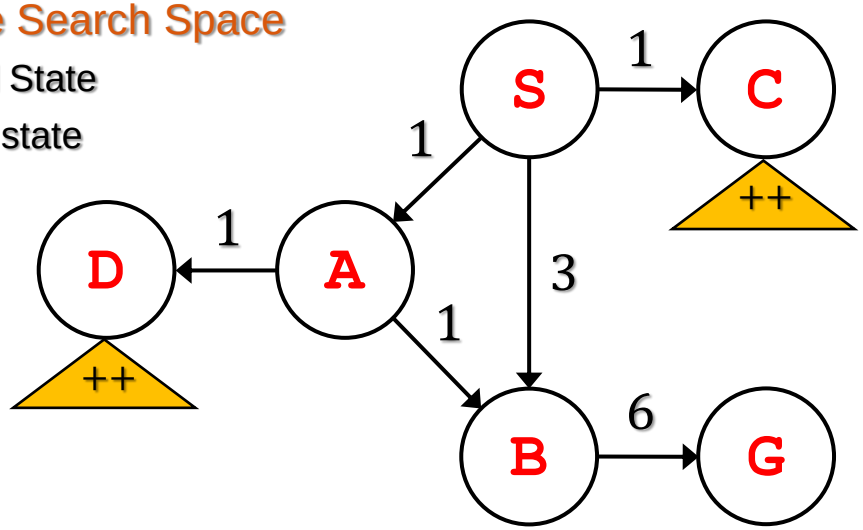
- Corresponding Search Tree

UCS



What are the Paths Discovered by UCS?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state

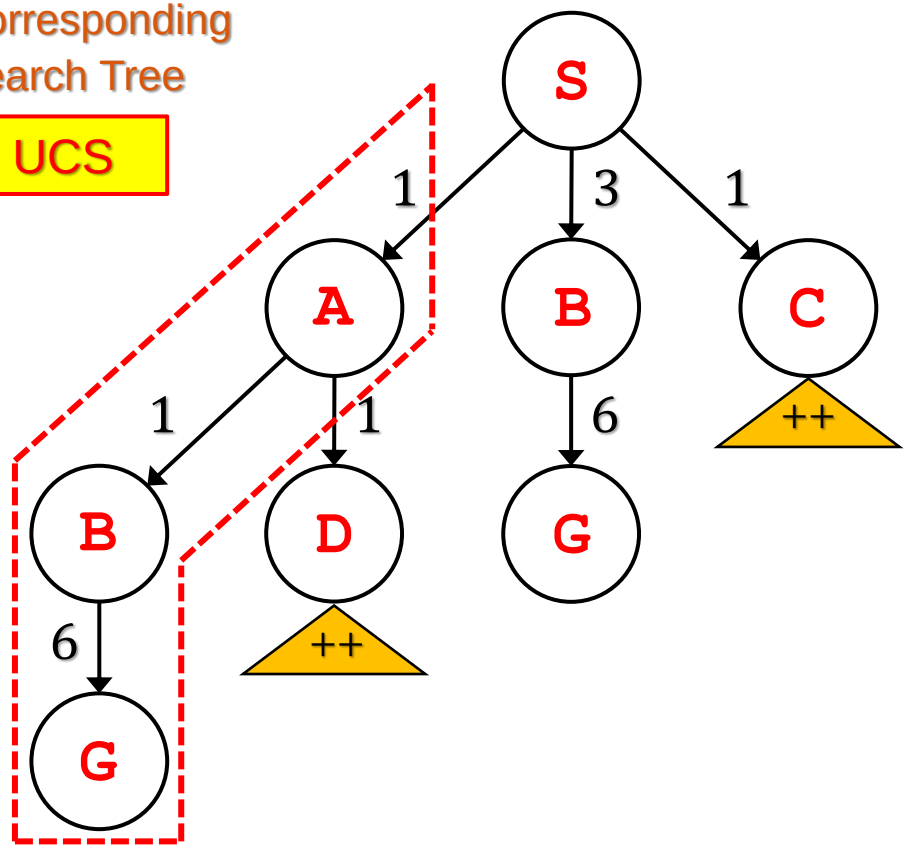


++ Subtrees with no goal

- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal (all explored)
  - **S-A-D++** : ∞ units to goal
  - **S-B-G** : 9 units to goal
  - **S-C++** : ∞ units to goal

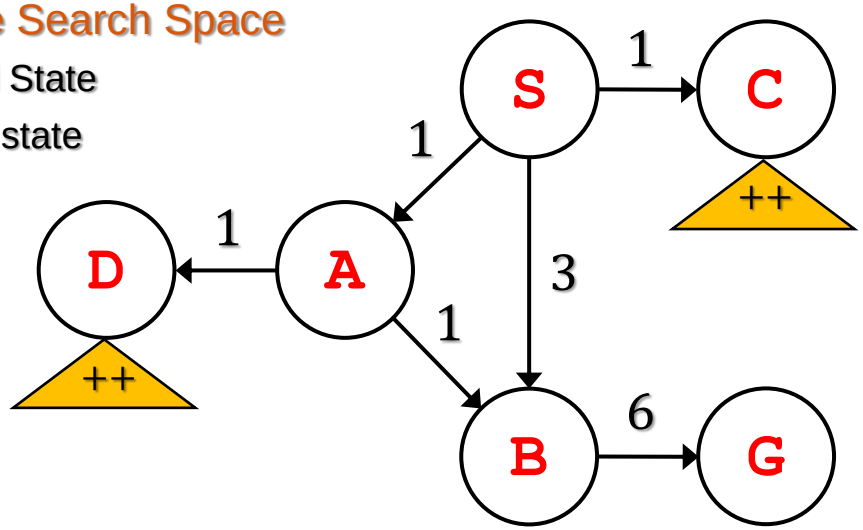
- Corresponding Search Tree

UCS



What are the Paths Discovered by UCS?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state

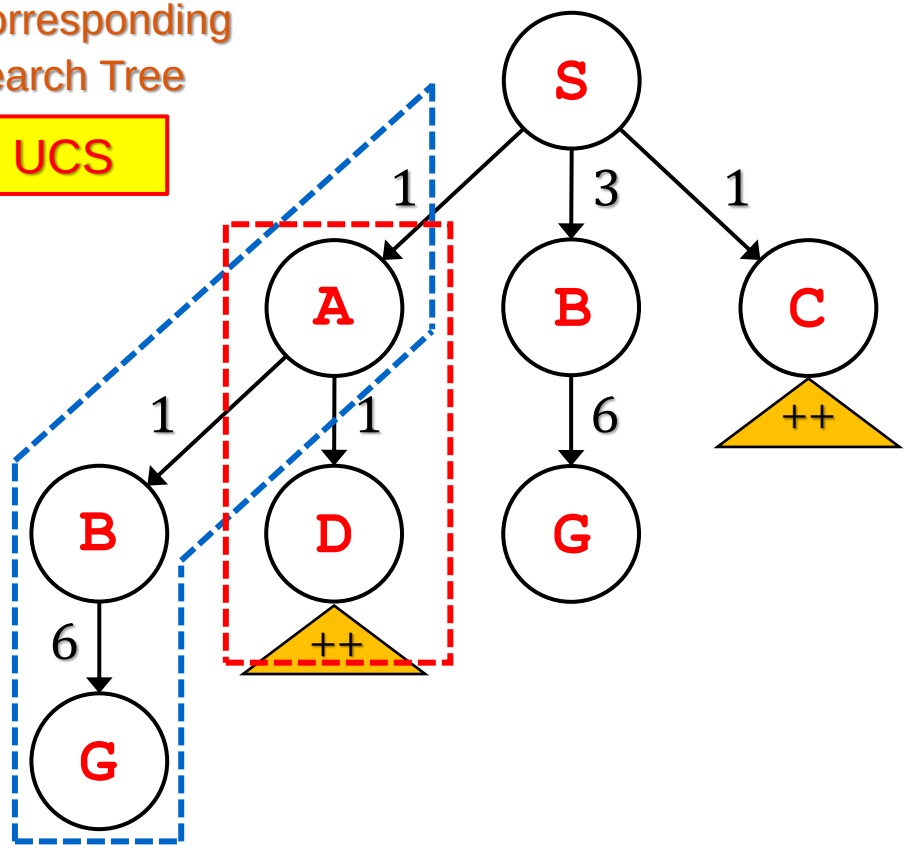


++ Subtrees with no goal

- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal (all explored)
  - **S-A-D++** : ∞ units to goal (some explored)
  - **S-B-G** : 9 units to goal
  - **S-C++** : ∞ units to goal

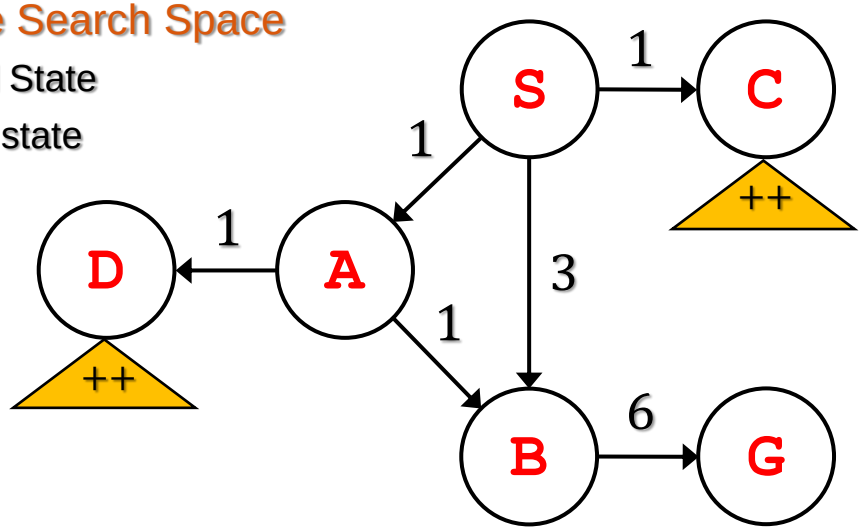
- Corresponding Search Tree

UCS



What are the Paths Discovered by UCS?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state

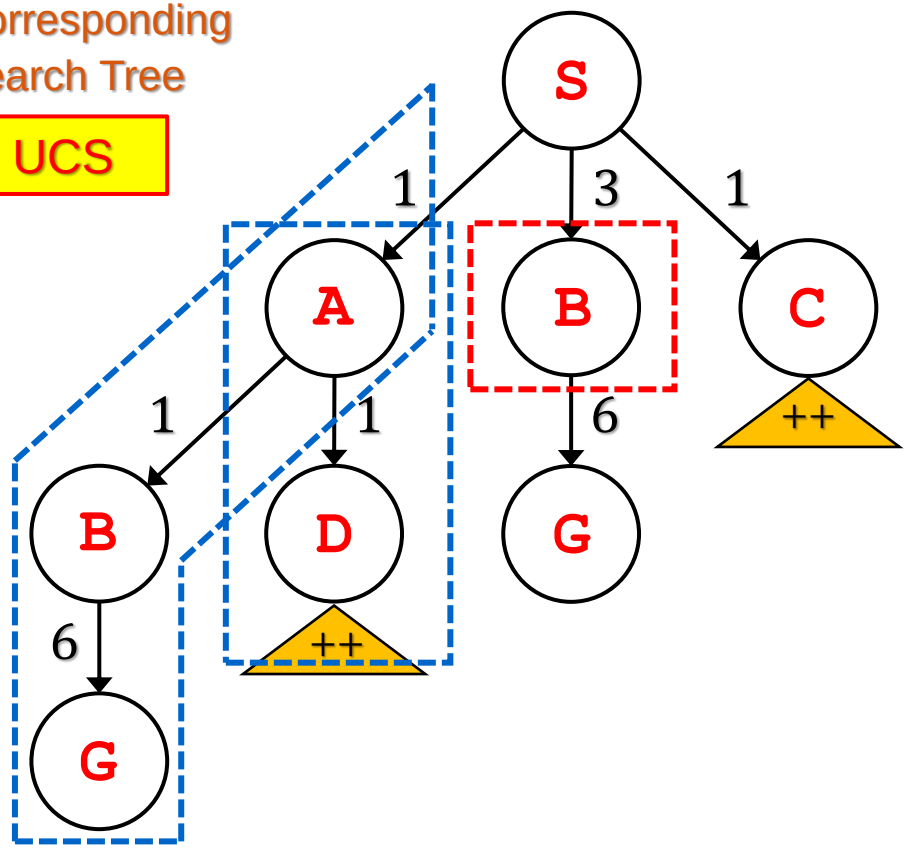


++ Subtrees with no goal

- Paths (and sub-path) costs
  - `S-A-B-G` : 8 units to goal (all explored)
  - `S-A-D++` : ∞ units to goal (some explored)
  - `S-B-G` : 9 units to goal (only `S-B` explored)
  - `S-C++` : ∞ units to goal

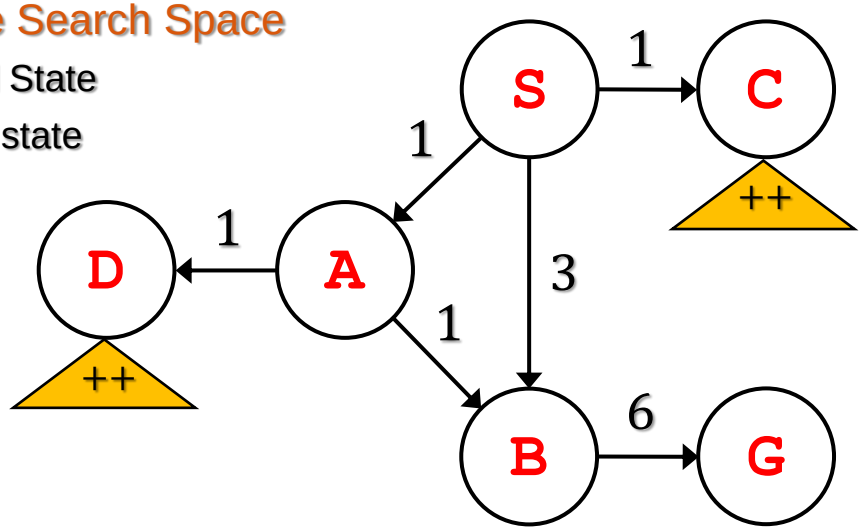
- Corresponding Search Tree

UCS



What are the Paths Discovered by UCS?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state

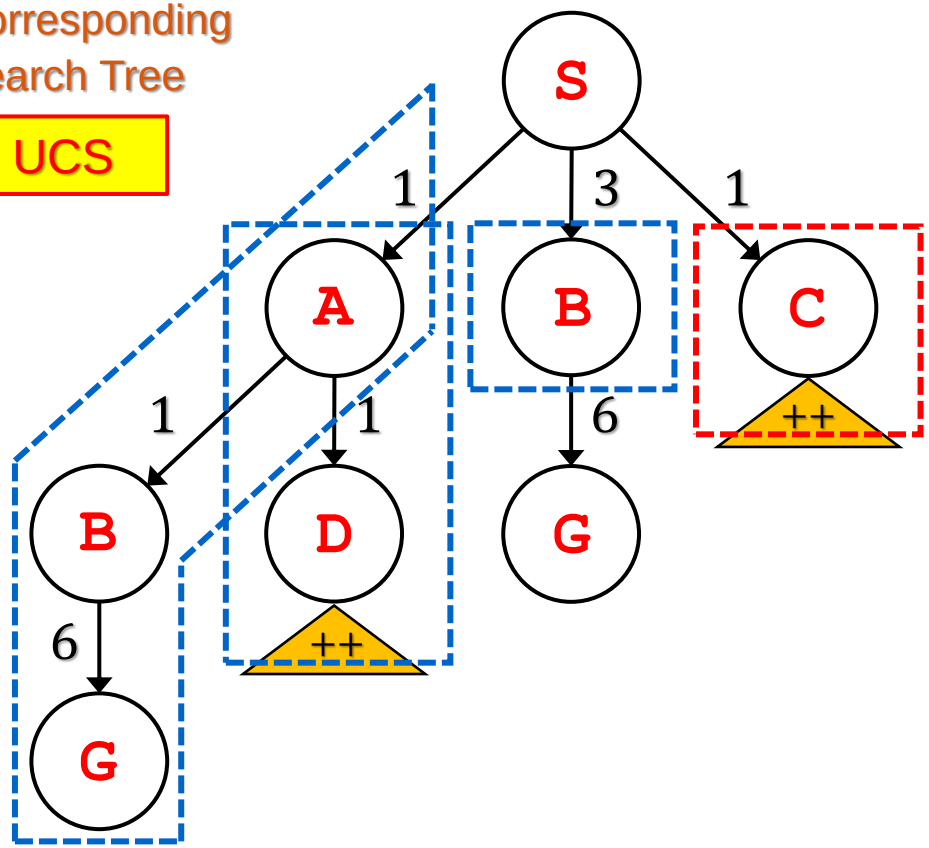


++ Subtrees with no goal

- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal (all explored)
  - **S-A-D++** : ∞ units to goal (some explored)
  - **S-B-G** : 9 units to goal (only **S-B** explored)
  - **S-C++** : ∞ units to goal (some explored)

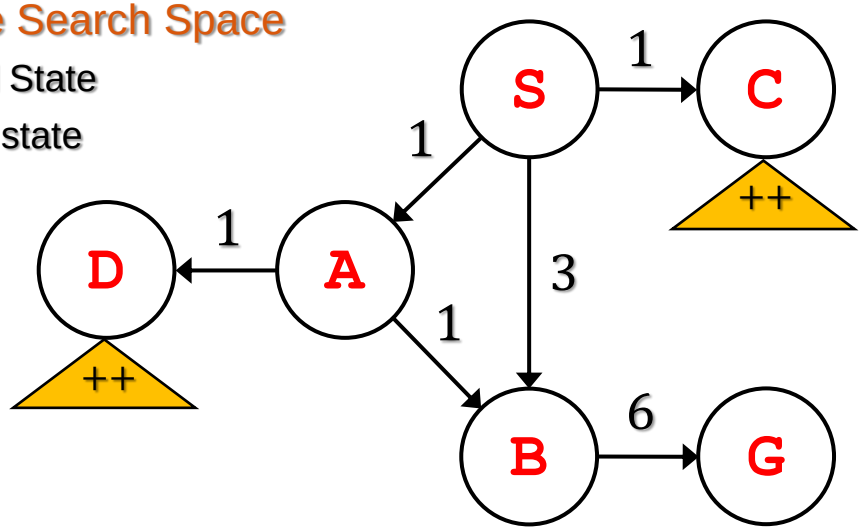
- Corresponding Search Tree

UCS



What are the Paths Discovered by UCS?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state

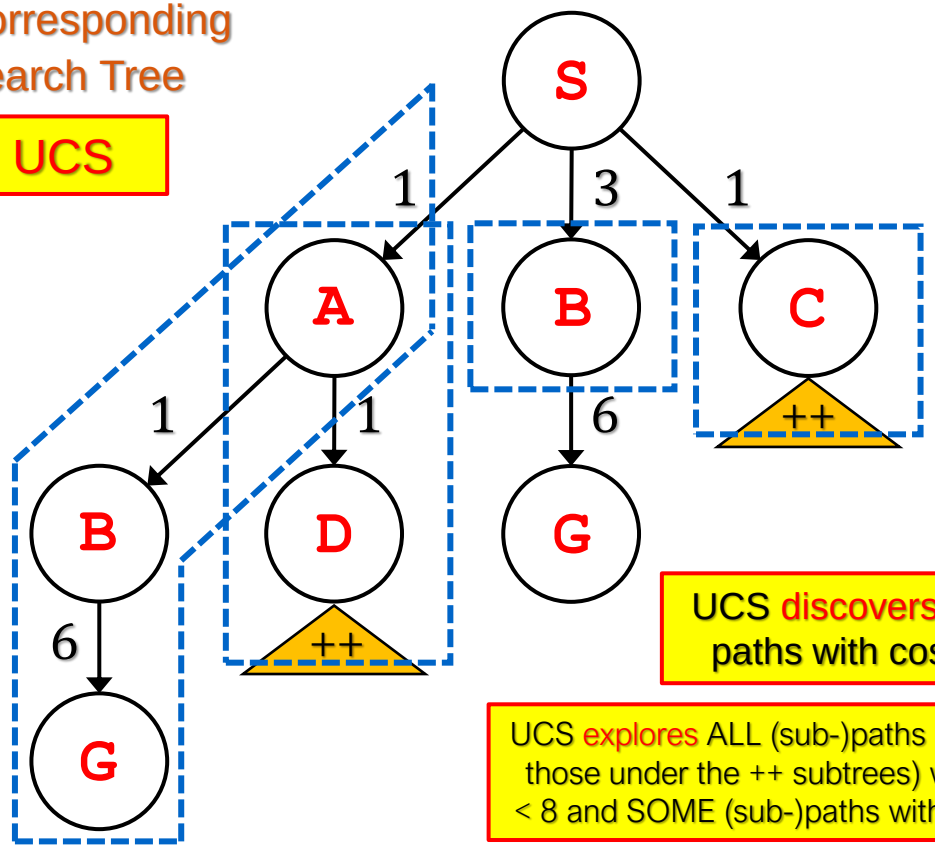


++ Subtrees with no goal

- Paths (and sub-path) costs
  - **S-A-B-G** : 8 units to goal (all explored)
  - **S-A-D++** : ∞ units to goal (some explored)
  - **S-B-G** : 9 units to goal (only **S-B** explored)
  - **S-C++** : ∞ units to goal (some explored)

- Corresponding Search Tree

UCS

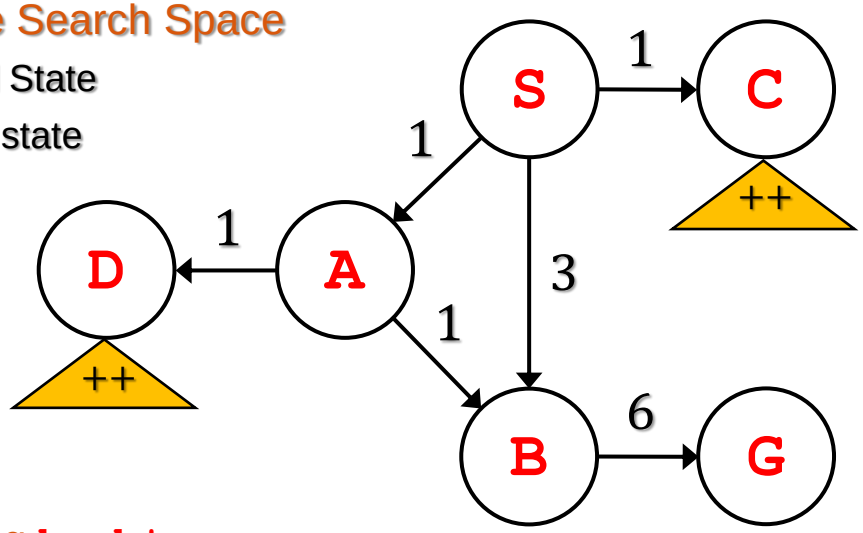


UCS discovers some paths with cost > 8

UCS explores ALL (sub-)paths (including those under the ++ subtrees) with cost < 8 and SOME (sub-)paths with cost = 8

What are the Paths Discovered by Greedy Best-First Search?

- Example Search Space
  - S: Initial State
  - G: Goal state



- Assuming h = h*
- Paths discovered:
  - S (8)

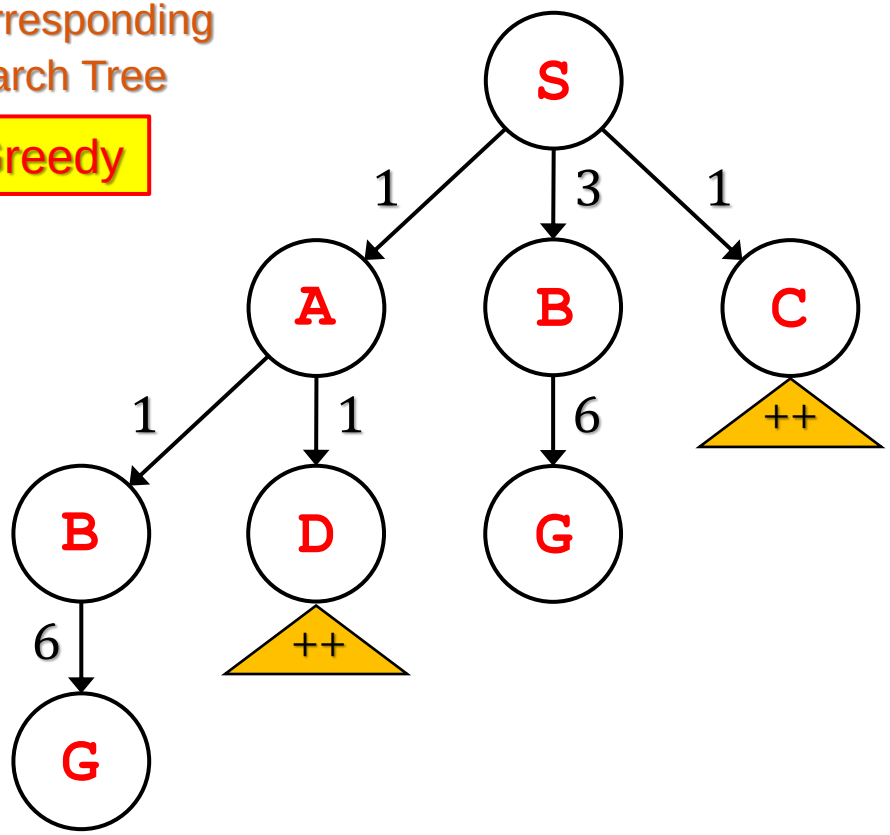
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

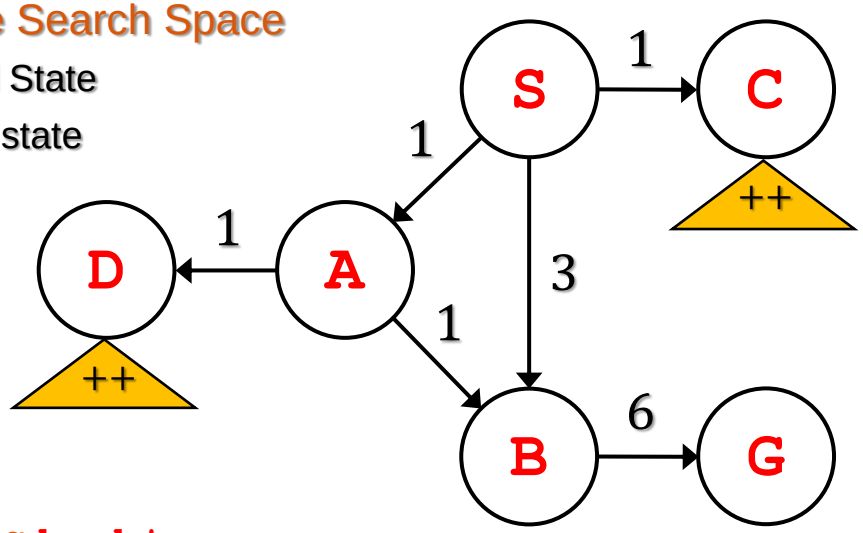
- Corresponding Search Tree

Greedy



What are the Paths Discovered by Greedy Best-First Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



- Assuming **h = h***
- Paths discovered:
  - **S** (8)
  - **S-B** (6), **S-A** (7), **S-C** (∞)

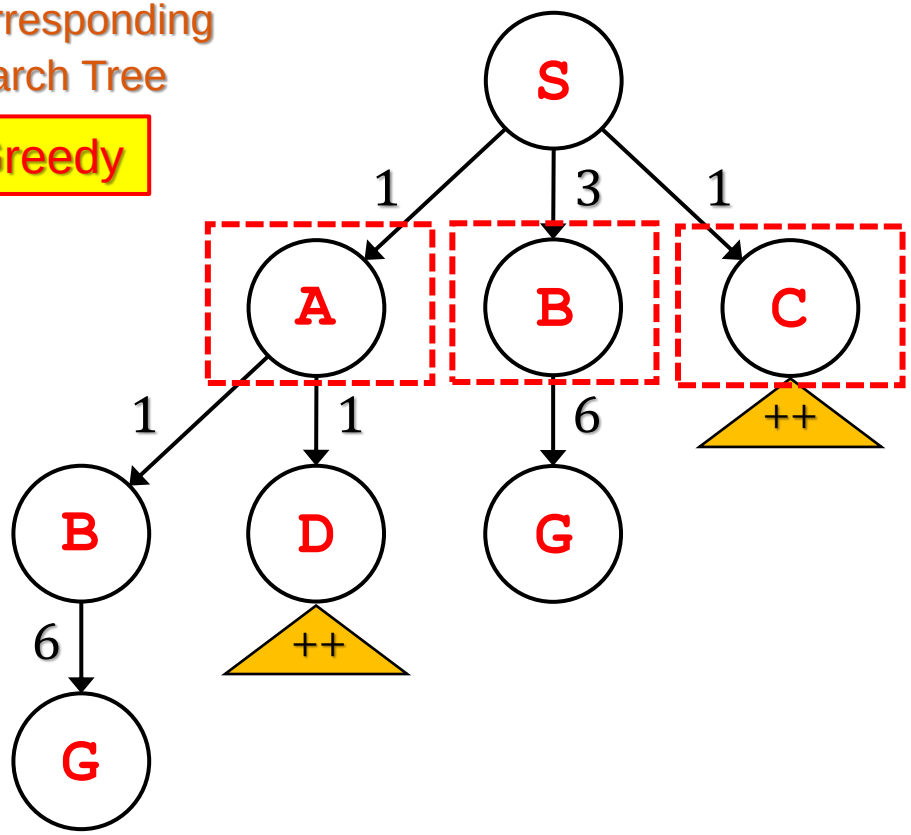
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

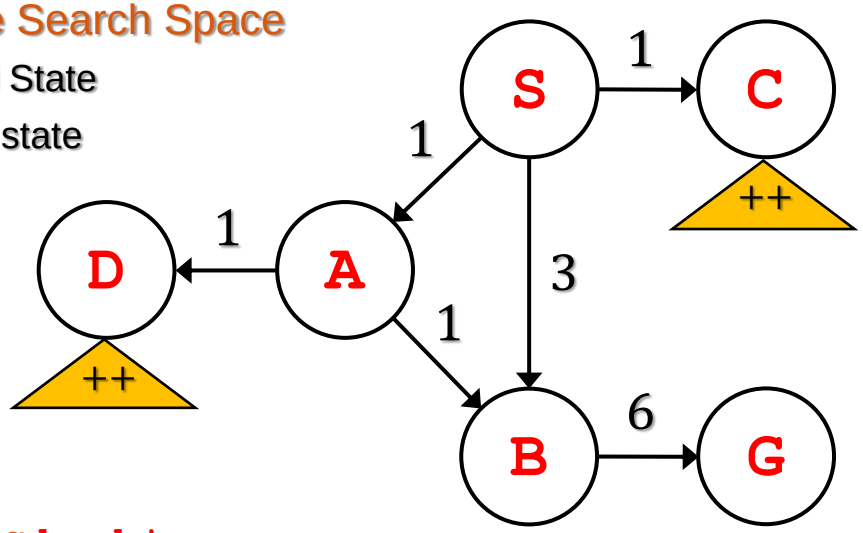
- Corresponding Search Tree

Greedy



What are the Paths Discovered by Greedy Best-First Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

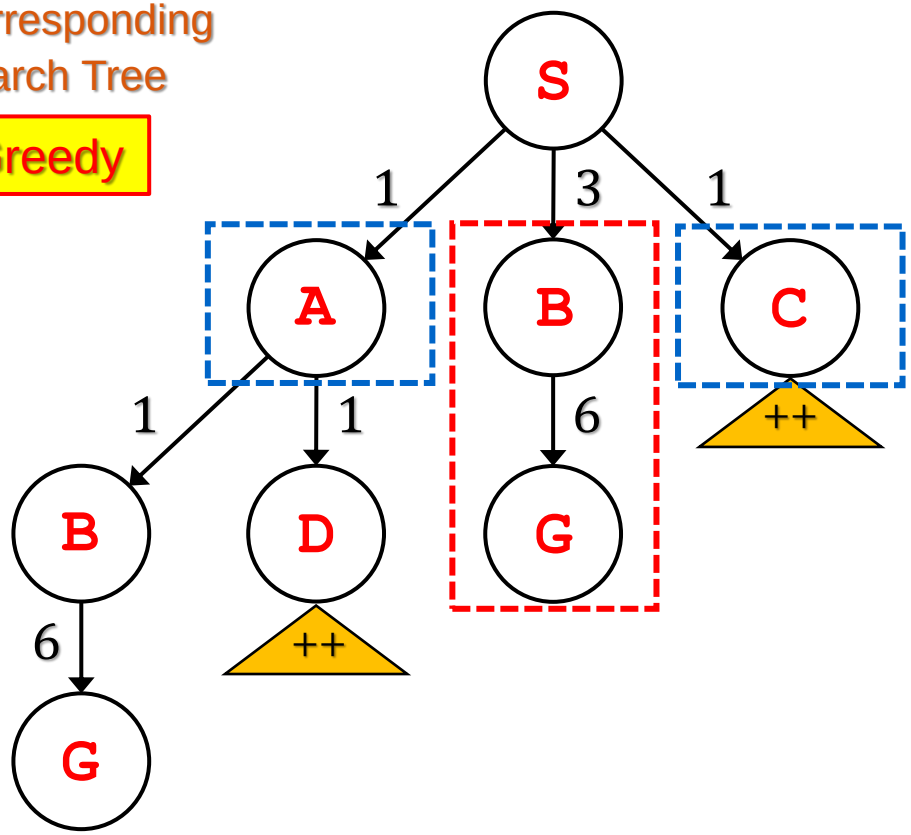
n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

- Assuming **h = h***
- Paths discovered:
  - **S** (8)
  - **S-B** (6), **S-A** (7), **S-C** (∞)
  - **S-B-G** (0) // return

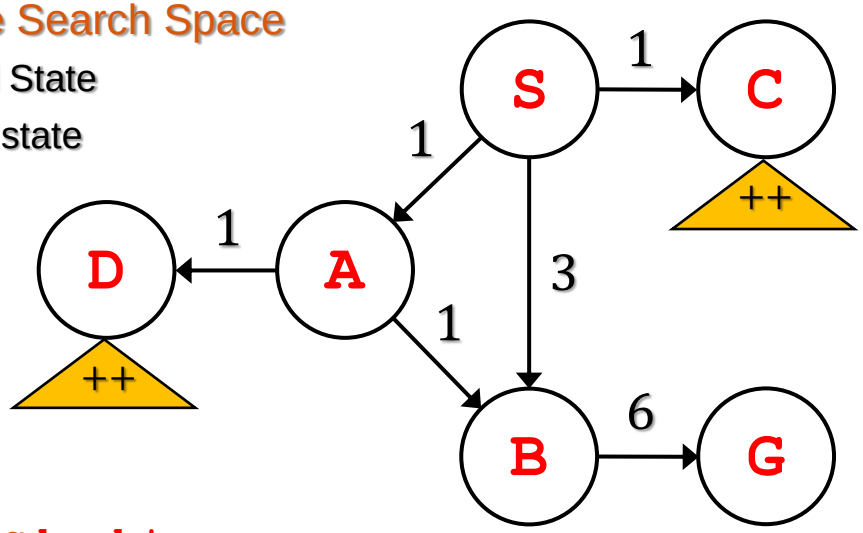
- Corresponding Search Tree

Greedy



What are the Paths Discovered by Greedy Best-First Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



++ Subtrees with no goal

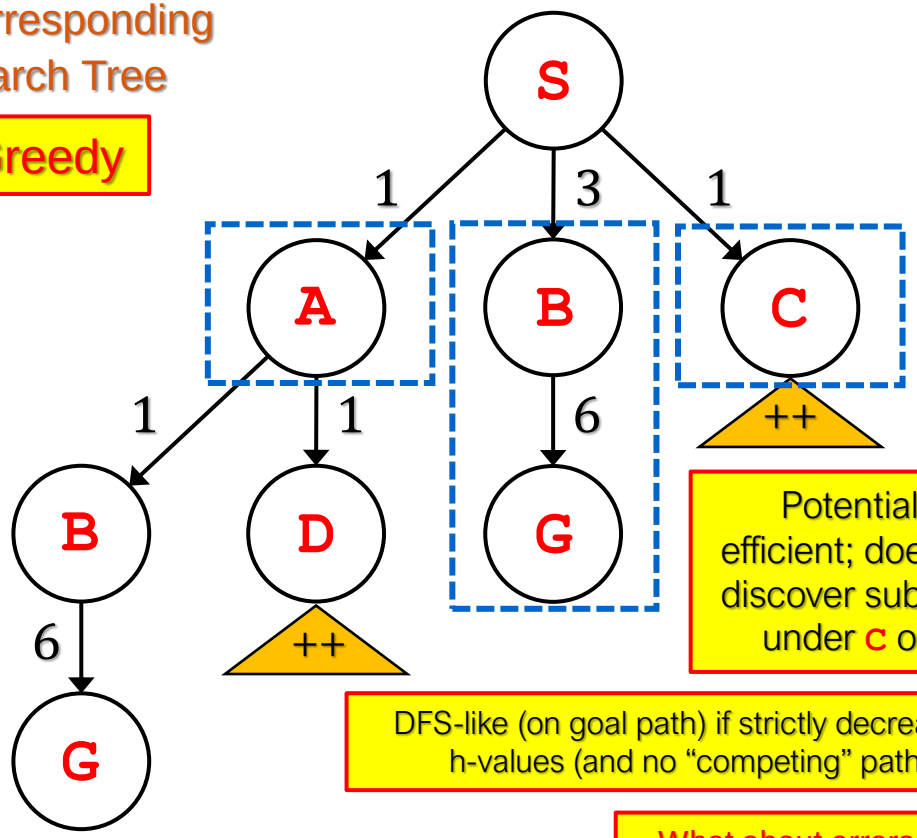
n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

- Assuming **h = h***
- Paths discovered:
  - **S** (8)
  - **S-B** (6), **S-A** (7), **S-C** (∞)
  - **S-B-G** (0) // return

- Corresponding Search Tree

Greedy



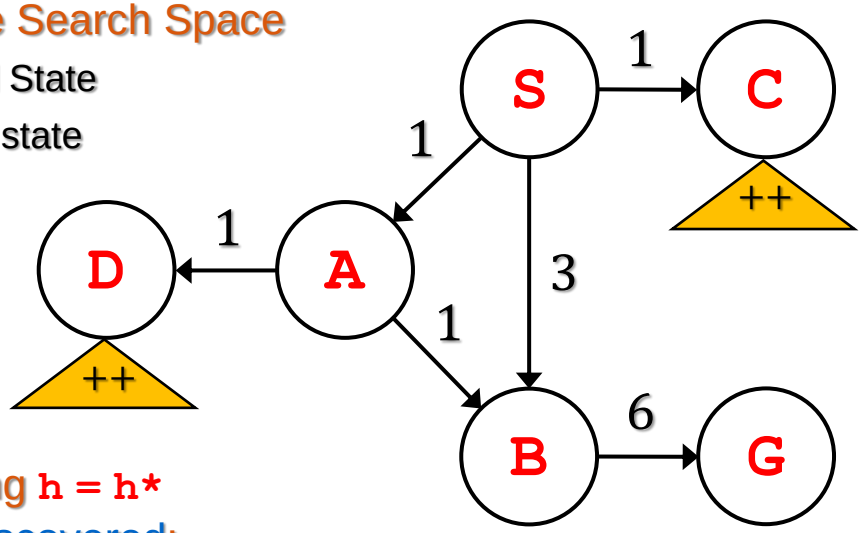
Potentially efficient; does not discover subtrees under **C** or **D**

DFS-like (on goal path) if strictly decreasing h-values (and no "competing" path)

What about errors in h?

What are the Paths Discovered by A* Search?

- Example Search Space
  - S: Initial State
  - G: Goal state



- Assuming h = h*
- Paths discovered:
  - S (0+8)

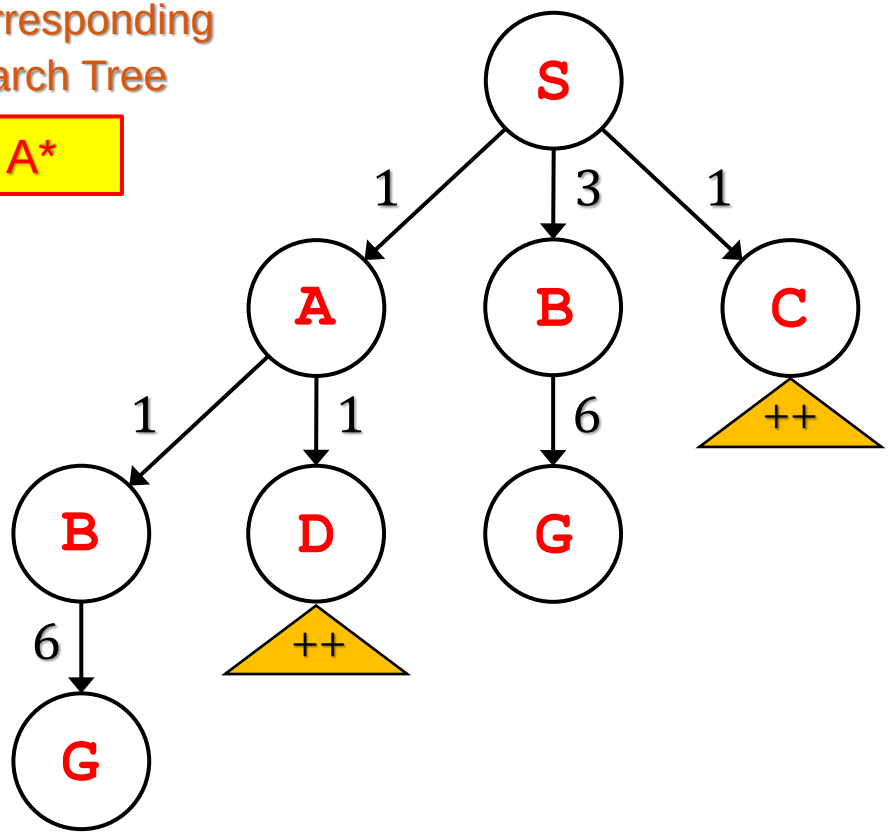
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

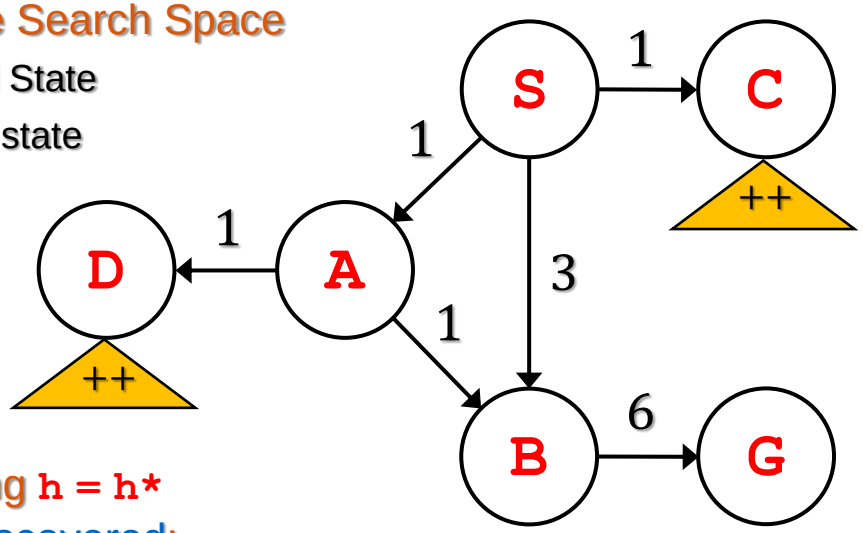
- Corresponding Search Tree

A*



What are the Paths Discovered by A* Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



- Assuming **h = h***
- Paths discovered:
  - **S** (0+8)
  - **S-A** (1+7), **S-B** (3+6), **S-C** (1+∞)

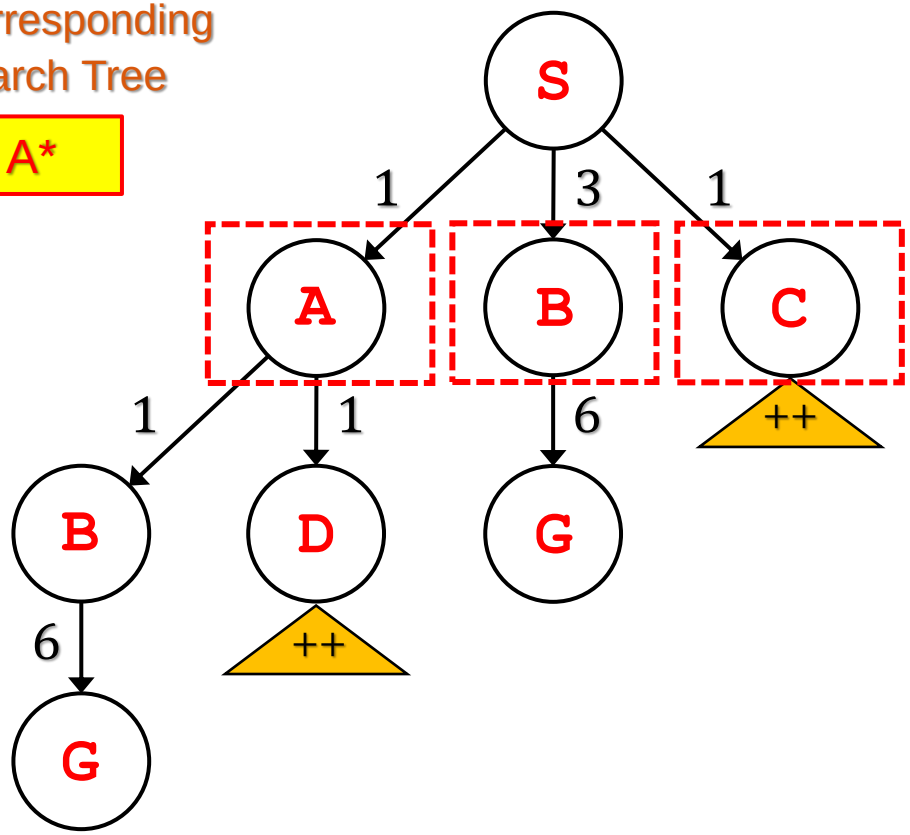
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

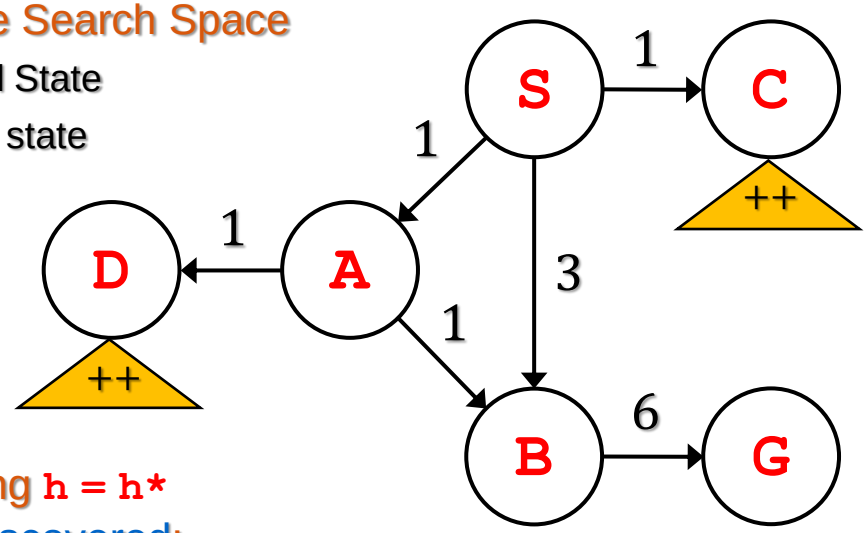
- Corresponding Search Tree

A*



What are the Paths Discovered by A* Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



- Assuming **h = h***
- Paths discovered:
  - **S** (0+8)
  - **S-A** (1+7), **S-B** (3+6), **S-C** (1+∞)
  - **S-A-B** (2+6), **S-A-D** (2+∞)

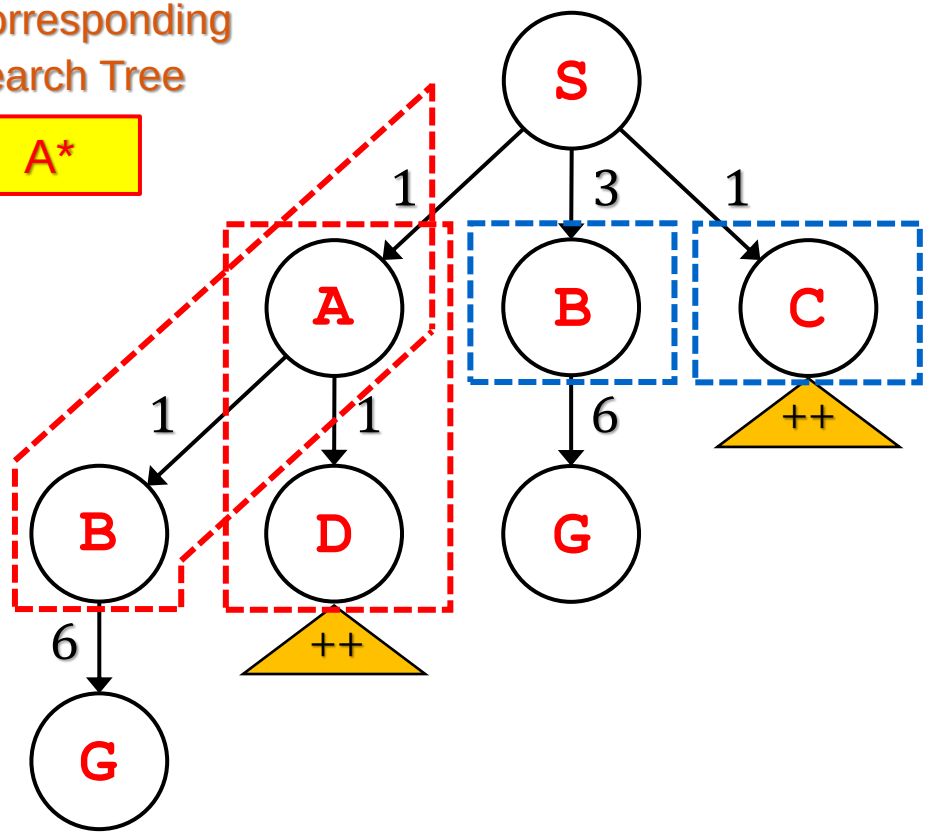
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

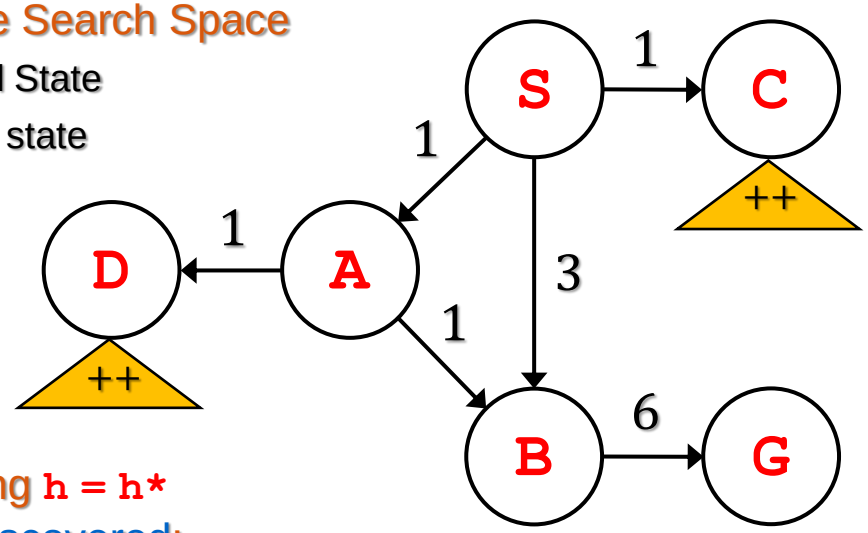
- Corresponding Search Tree

A*



What are the Paths Discovered by A* Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



- Assuming **h = h***
- Paths discovered:
  - **S** (0+8)
  - **S-A** (1+7), **S-B** (3+6), **S-C** (1+∞)
  - **S-A-B** (2+6), **S-A-D** (2+∞)
  - **S-A-B-G** (8+0) // return

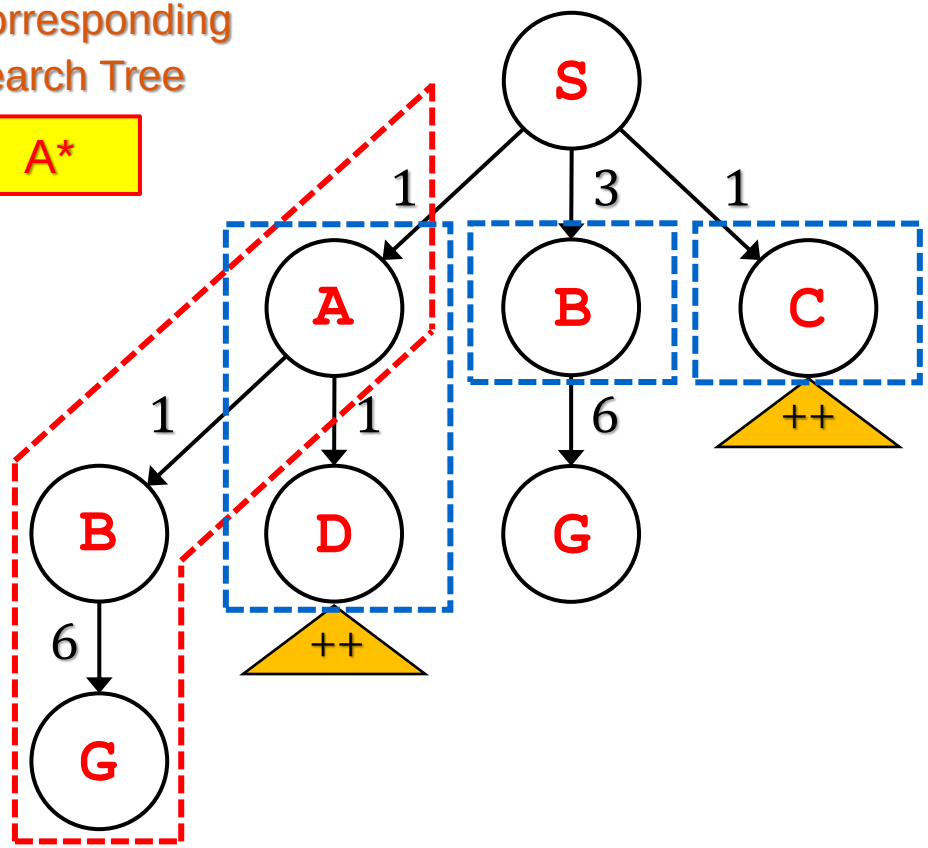
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

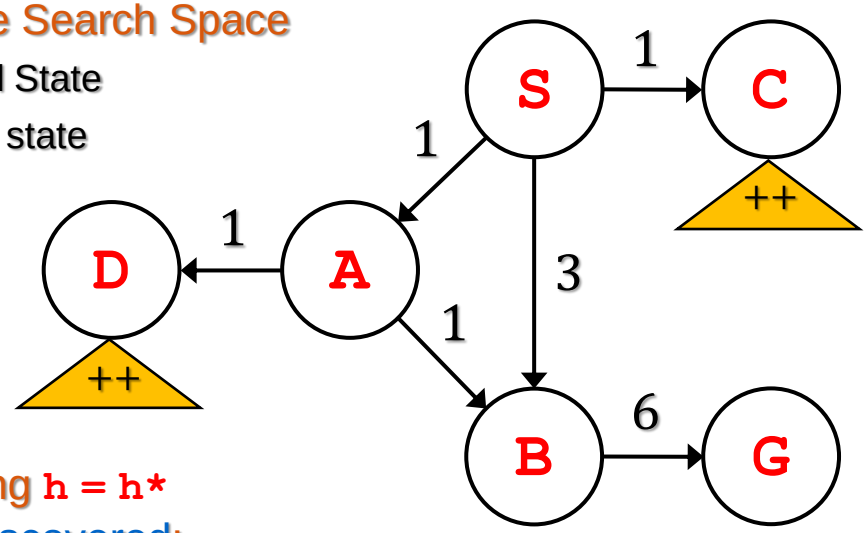
- Corresponding Search Tree

A*



What are the Paths Discovered by A* Search?

- Example Search Space
  - **S**: Initial State
  - **G**: Goal state



- Assuming **h = h***
- Paths discovered:
  - **S** (0+8)
  - **S-A** (1+7), **S-B** (3+6), **S-C** (1+∞)
  - **S-A-B** (2+6), **S-A-D** (2+∞)
  - **S-A-B-G** (8+0) // return

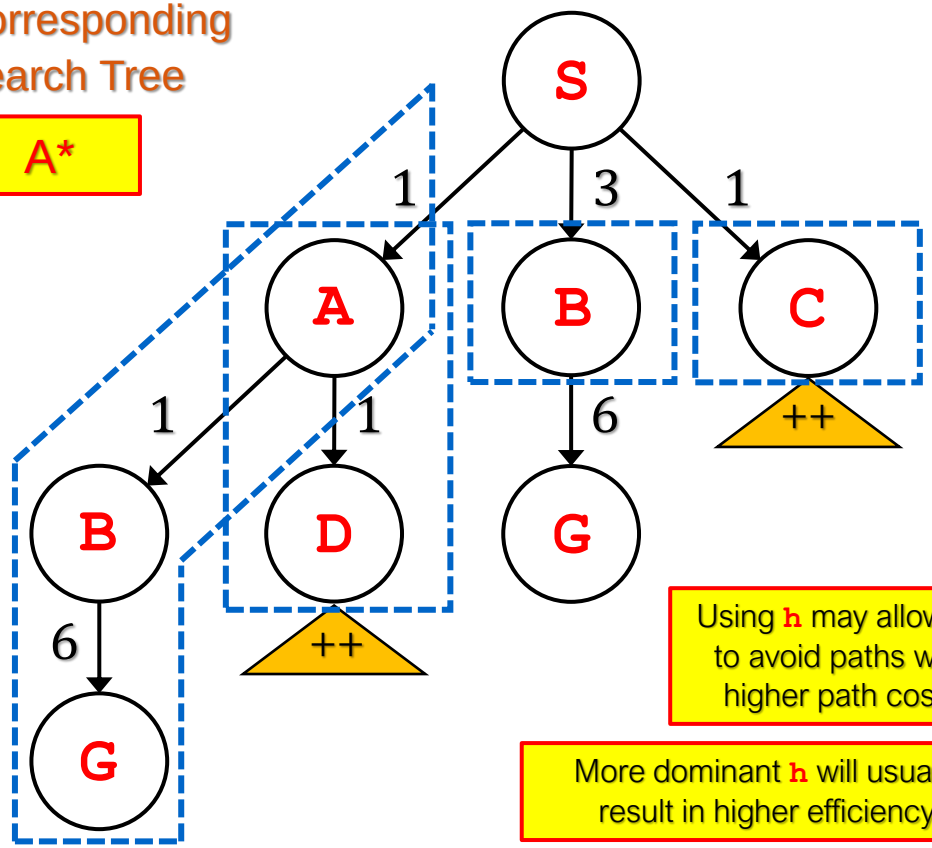
++ Subtrees with no goal

n	h*(n)
S	8
A	7
B	6

n	h*(n)
C	∞
D	∞
G	0

- Corresponding Search Tree

A*



Using **h** may allow A* to avoid paths with higher path costs

More dominant **h** will usually result in higher efficiency

Admissible & Consistent Heuristics

- Admissible `h`
  - `h(n)` is admissible if `∀n: h(n) ≤ h*(n)`
  - `h(n)` never overestimates the cost

Main idea: by the time we explore a path to a goal, `P`, all paths with actual costs less than `P` must be explored

- Theorem: If `h(n)` is admissible, then A* using tree search is optimal

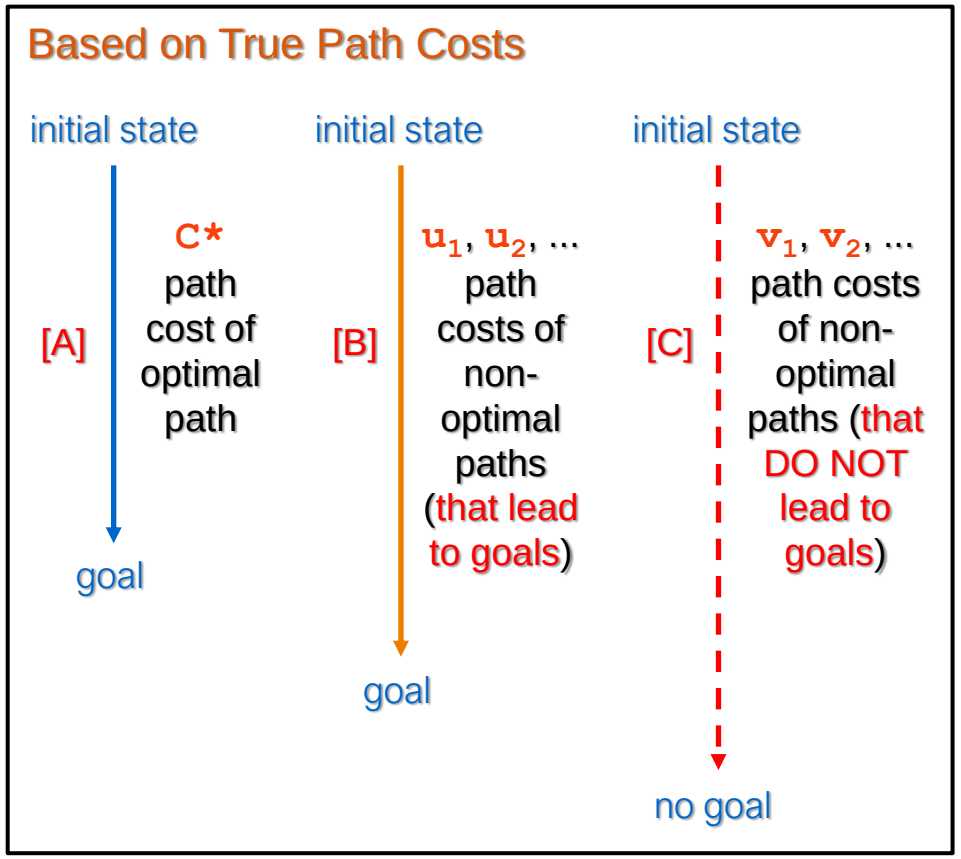
- Consistent `h`
  - `h(n)` is consistent if `∀n` and each successor of `n`, `n'`: `h(n) ≤ cost(n,a,n') + h(n')`

Main idea: ensures that `f` costs are monotonically increasing along a path

- Theorem: If `h(n)` is consistent, then A* using graph search is optimal

Under Version 2 & 3 (but not Version 1)!

Taxonomy of Paths Under A* Search



- Using admissible heuristics (never overestimates)
  - explores all paths on [A]
  - explores some paths under [B] and [C]
    - Note that severe underestimations are possible (e.g., `h(n)= 0`, which is equivalent to UCS priority)
    - All paths estimated to be less than `C*` will be explored
    - Paths estimated to be equal to `C*` may be explored
- Using consistent heuristics (never overestimates; monotonically increasing)
  - only makes a difference under graph search
- Using neither admissible nor consistent heuristics
  - depends on the types of errors
  - sequence of paths explored may vary wildly

Summary

Completeness assumptions:
- `b` finite AND (`m` finite OR has a solution)
- All action costs are > `ε` > 0

- UCS
  - Optimal under
    - Tree search
    - Graph search (version 2 and 3)
  - Popping node `n` ⇒ optimal path to `n` found
- Greedy Best-First Search
  - Not optimal
  - Incomplete under tree search
- A* Search
  - Assuming admissible `h`
    - May not have monotonically increasing path cost
    - Optimal under
      - Tree search
      - Graph search (version 2)
  - Assuming consistent `h`
    - On popping node `n`, optimal path to n found
    - Optimal under
      - Tree search
      - Graph search (version 2 and 3)

Assume graph search version 2 and late goal testing (unless otherwise stated)

3 Heuristics & Dominance

Efficiency & Dominance

- Efficiency of A* depends on the accuracy of its heuristics
  - Higher heuristic accuracy means we need to try fewer paths
  - Specifics not covered in CS3243

- Which heuristics are better?

- If , $h_1(n) \geq h_2(n)$, then h_1 dominates h_2
  - If h_1 is also admissible
    - h_1 must be closer to h^* than h_2
    - h_1 is usually more efficient than h_2 when used with A*

Recall that in CS3243, we assume that a dominant heuristic is an admissible one

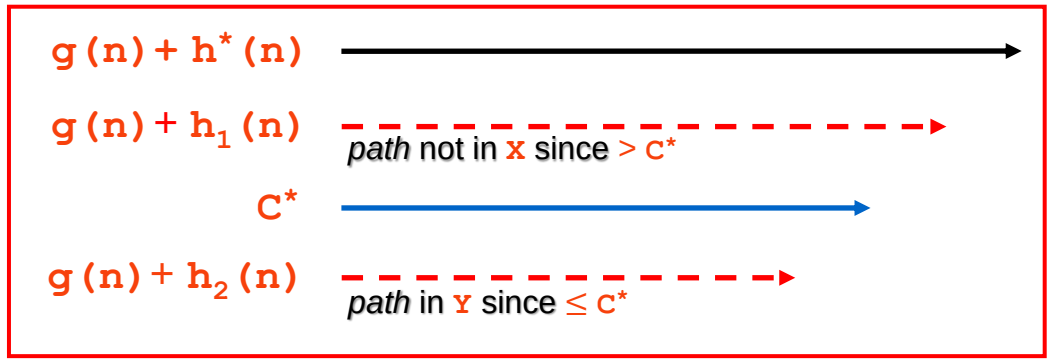
Efficiency & Dominance

- Proving a dominant heuristic explores fewer nodes
  - All paths with approximated path cost ≤ optimal path cost are explored
  - A* explores fewer nodes under h_1 since $|X| \leq |Y|$, where
    - X denotes the set of paths with costs below C^* based on h_1
    - Y denotes the set of paths with costs below C^* based on h_2
  - Consider example path, node n; assume**
    - n not on optimal path and $g(n) + h^*(n) > C^*$
    - possible: (1) $h_2(n) \leq C^* - g(n) < h^*(n) \Rightarrow n \in Y$
    - and that: (2) $C^* - g(n) < h_1(n) \leq h^*(n) \Rightarrow n \notin X$
  - Consider the reverse of the above:
    - $h_1(n) \leq C^* - g(n) \leq h^*(n)$ & $C^* < h_2(n) \leq h^*(n)$
  - The reverse case is impossible!
    - As we have: $h_1(n) \leq C^*$ & $C^* < h_2(n)$
    - Or: $h_1(n) < h_2(n)$ which contradicts our dominance assumption

Assumes tree search with tie-breaking unnecessary

Assume h_1 dominates h_2 (i.e., both admissible)

Note: *path* refers to node n



** For completeness, also consider the other possible assumptions over node n as an exercise

Gauging Improvement with Effective Branching Factor

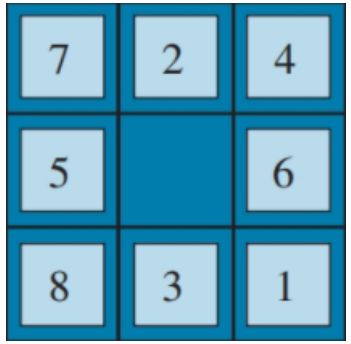
- How do we measure the improvement when using dominant h?
- Measure effective branching factor, b^*
  - Given empirical results
    - N nodes expanded
    - Solution path at depth d
  - Solve for b^* using
    - $N + 1 = (b^*)^0 + (b^*)^1 + \ldots + (b^*)^d$
    - i.e., given d and N, solve $N + 1 = ((b^*)^{d+1} - 1) / ((b^*) - 1)$ to determine effective branching factor b^*

Note that you will NOT be required to calculate b^* in this manner for CS3243 exams!

How To Craft Heuristics?

- Goal is to identify a function that approximates $h^*(n)$
  - Cost from n to the nearest goal
- Can we implement another search to give us this?
  - E.g., use UCS as h since it will give us $h^*(n)$
    - $h(n)$ called on each node → need UCS to find the optimal path from the start state to ALL n
    - Defeats the purpose of using h, which is to improve A*
    - Better to just use $h(n) = 0$ and just run UCS once
- We want efficient h
  - Ideally, h is O(1), or something reasonably cheap
  - Set our objective to admissible heuristics (since consistent is much more difficult)

Example Problem: 8-Puzzle



Example Initial State

Goal State

- Search Problem Specification
  - State Representation (Initial State):
    - Matrix representing the grid, with each `(r, c) ∈ [0, 8]`
    - `0` is the blank cell
  - Actions:
    - Move a chosen cell adjacent `(r, c)` to the blank cell `(r', c')`
    - Same as "moving" blank cell `(r', c')` to a chosen cell adjacent `(r, c)`
      - i.e., legal `(r'-1, c')`, `(r'+1, c')`, `(r',c-1)`, `(r', c+1)` cells → possible actions
  - Goal Test:
    - Current state matrix = goal state matrix
  - Transition Model:
    - Swap the contents of `(r, c)` and `(r', c')`
  - Cost Function:
    - Each action cost is 1 unit

This Puzzle requires the player to shift the numbered squares into the empty cell until the final pattern is obtained

How do we get an admissible heuristic out of this puzzle?

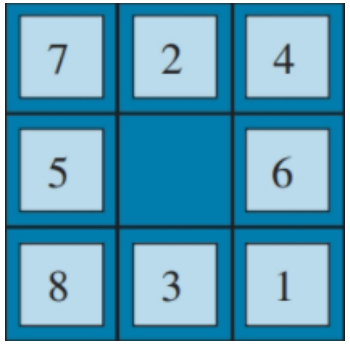
Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/43723553202

4 Relaxing the Problem

8-Puzzle Example: Defining Admissible Heuristics by Relaxing the Problem (Version A)



Example Initial State

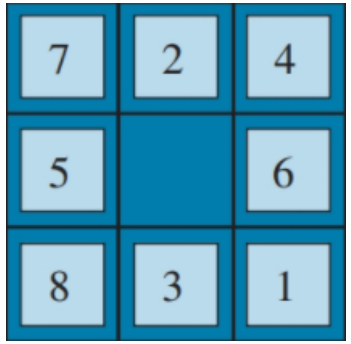
Goal State

- Define an easier problem based on the same context
  - Let heuristic **h** be a function that counts actions required to solve the problem
  - The puzzle is constrained by this rule
    - A tile can move from square **X** to square **Y** if **X** is adjacent to **Y** and **Y** is blank
  - Relaxed 8-Puzzle: Version A (h_0)
    - Remove all tiles (1 move)
    - Place them in the correct positions (8 moves)
    - $h_0 = 9$, for any problem
    - h_0 overestimates the moves required on some cases

Test admissibility by trying to work backwards from the goal state

We did not properly relax the rules and instead just defined new ones

8-Puzzle Example: Defining Admissible Heuristics by Relaxing the Problem (Version B)



Example Initial State

Goal State

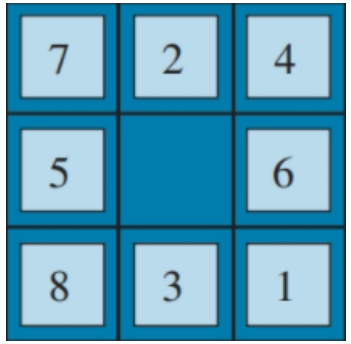
- Define an easier problem based on the same context
  - Let heuristic `h` be a function that counts actions required to solve the problem
  - The puzzle is constrained by this rule
    - A tile can move from square `X` to square `Y` if `X` is adjacent to `Y` and `Y` is blank
  - Relaxed 8-Puzzle: Version B (h_1)
    - A tile can move from square `X` to square `Y` ~~if `X` is adjacent to `Y` and `Y` is blank~~
    - h_1 = number of cells in the wrong position
    - Complexity: O(`n`), where `n` is the size of the grid
    - h_1 is admissible!

Note: also removes implied constraint that each square can only house one tile – i.e., now we can more than 1 tile on each square

By properly relaxing the rule, we got an admissible heuristic: h_1 – misplaced tiles

Can we do better? Find an admissible `h` that dominates this one

8-Puzzle Example: Defining Admissible Heuristics by Relaxing the Problem (Version C)



Example Initial State

Goal State

- Define an easier problem based on the same context
  - Let heuristic **h** be a function that counts actions required to solve the problem
  - Relaxed 8-Puzzle: Version B (h_1)
    - A tile can move from square **X** to square **Y** ~~if **X** is adjacent to **Y** and **Y** is blank~~
  - Relaxed 8-Puzzle: Version C (h_2)
    - A tile can move from square **X** to square **Y** if **X** is adjacent to **Y** ~~and **Y** is blank~~
    - h_2 = sum over each Manhattan distance between a square and its goal location
    - Complexity: O(**n**), where n is the size of the grid (given that grid dimensionality here is fixed at 2)
    - h_2 is admissible and dominates Version B

Note: this STILL removes the implied constraint that each square can only house one tile – i.e., now we can more than 1 tile on each square

New admissible heuristic: h_2 – Manhattan distance
- h_2 dominates h_1 (h_1 is a relaxation of h_2)
- h_2 is admissible (h_2 is a relaxation of the original rule)

Effects of Dominance Under 8-Puzzle

	Search Cost (nodes generated)			Effective Branching Factor		
d	BFS	$A^*(h_1)$	$A^*(h_2)$	BFS	$A^*(h_1)$	$A^*(h_2)$
6	128	24	19	2.01	1.42	1.34
8	368	48	31	1.91	1.40	1.30
10	1033	116	48	1.85	1.43	1.27
12	2672	279	84	1.80	1.45	1.28
14	6783	678	174	1.77	1.47	1.31
16	17270	1683	364	1.74	1.48	1.32
18	41558	4102	751	1.72	1.49	1.34
20	91493	9905	1318	1.69	1.50	1.34
22	175921	22955	2548	1.66	1.50	1.34
24	290082	53039	5733	1.62	1.50	1.36
26	395355	110372	10080	1.58	1.50	1.35
28	463234	202565	22055	1.53	1.49	1.36

Recall: Effective Branching Factor computed by substituting d and total nodes generated into geometric series sum and solving to find b^*

Data are averaged over 100 puzzles for each solution length d from 6 to 28

Table Reference: AIMA 4th Edition, Figure 3.26 (pp 117)

Rules to Functions

- Will we always be able to define functions h_1 and h_2 to match the relaxed rules (or even define the original rules)?

- Models and approximations
  - Finding functions that model or approximate the quantity you want (efficiently)
  - Constructing models
    - Bottom-up
      - What variables can you efficiently calculate?
      - What can these variables model?
    - Top-down
      - What (dependent) variables do I want to model / approximate?
      - What are the (independent) variables that help to calculate these?

Questions about the Lecture?

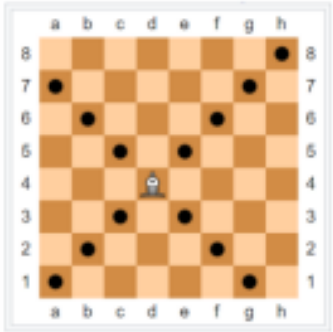
- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/43723553202

Midterm Question 3

- Heuristic Design
  - Example: AY22/23 S3 Q3:

3. Consider the Project 1 problem description – i.e., given a Chess board, some obstacles and opposing pieces – find a path from a starting position to a goal position. However, here, instead of the agent corresponding to a Knight piece, the agent will correspond to a Bishop piece. The movement range of the Bishop piece is depicted on the right. As with Project 1, assume that board corresponds to a rectangular grid. For this question, you may further assume that there is only one goal position.

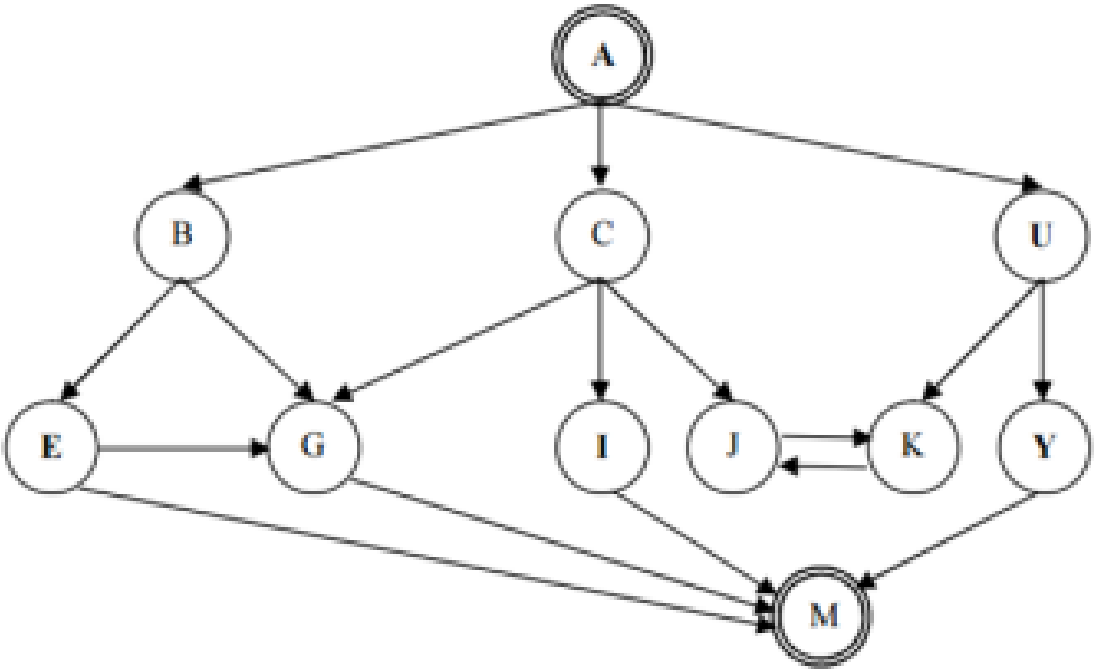


Let a heuristic, h_1, be defined as follows for any given state n.

- $h_1(n) = 0$ if $n = goal$, where n and $goal$ are coordinates on the Chess board
- $h_1(n) = \infty$ if $(n \neq goal) \wedge (\text{cellColour}(n) \neq \text{cellColour}(goal))$
- $h_1(n) = 0$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 1)$
- $h_1(n) = 1$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0)$

Midterm Question 2

- Search Algorithm Trace (& Proof)
  - Example: AY22/23 S3 Q2b:

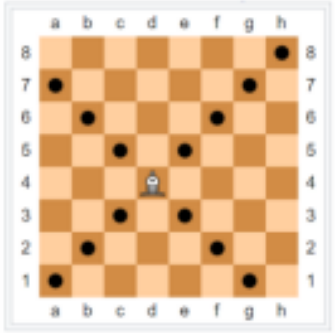


h(A) = 7
h(B) = 5
h(C) = 3
h(U) = 4
h(E) = 2
h(G) = 3
h(I) = 6
h(J) = 2
h(K) = 1
h(Y) = 2
h(M)= 0

Midterm Question 1

- Search Problem Modelling
  - Example: AY21/22 S2 Q1: Cleaning Robot

3. Consider the Project 1 problem description – i.e., given a Chess board, some obstacles and opposing pieces – find a path from a starting position to a goal position. However, here, instead of the agent corresponding to a Knight piece, the agent will correspond to a Bishop piece. The movement range of the Bishop piece is depicted on the right. As with Project 1, assume that board corresponds to a rectangular grid. For this question, you may further assume that there is only one goal position.



Let a heuristic, h_1, be defined as follows for any given state n.

- $h_1(n) = 0$ if $n = goal$, where n and $goal$ are coordinates on the Chess board
- $h_1(n) = \infty$ if $(n \neq goal) \wedge (\text{cellColour}(n) \neq \text{cellColour}(goal))$
- $h_1(n) = 0$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 1)$
- $h_1(n) = 1$ if $(n \neq goal) \wedge (\text{cellColour}(n) = \text{cellColour}(goal)) \wedge (\text{ManhattanDist}(n, goal) \% 2 = 0)$

Midterm Question 3

- Search Problem Modelling
  - Example: AY22/23 S1 Q1: Dining Philosophers

1. There are n philosophers dining together at a circular table (where $n \geq 2$). Each philosopher has their own place at the table. Their philosophical problem in this instance is that the dish served is a kind of spaghetti that must be eaten with two forks.

Each of the n philosophers has their own plate of spaghetti. However, there is only one fork between each plate (notice therefore that there are also n forks).

Each philosopher can only alternately think or eat. Moreover, a philosopher can only eat their spaghetti when they have both a left and right fork. Thus, two forks will only be available when their two nearest neighbours are thinking, not eating. After an individual philosopher finishes eating, they may put down both forks.

Image from Wikipedia

Questions about the Lecture?

- Was anything unclear?
- Do you need to clarify anything?
- Ask on Archipelago
  - Specify a question
  - Upvote someone else's question

Invitation Link (Use NUS Email --- starts with E)
https://archipelago.rocks/app/resend-invite/43723553202