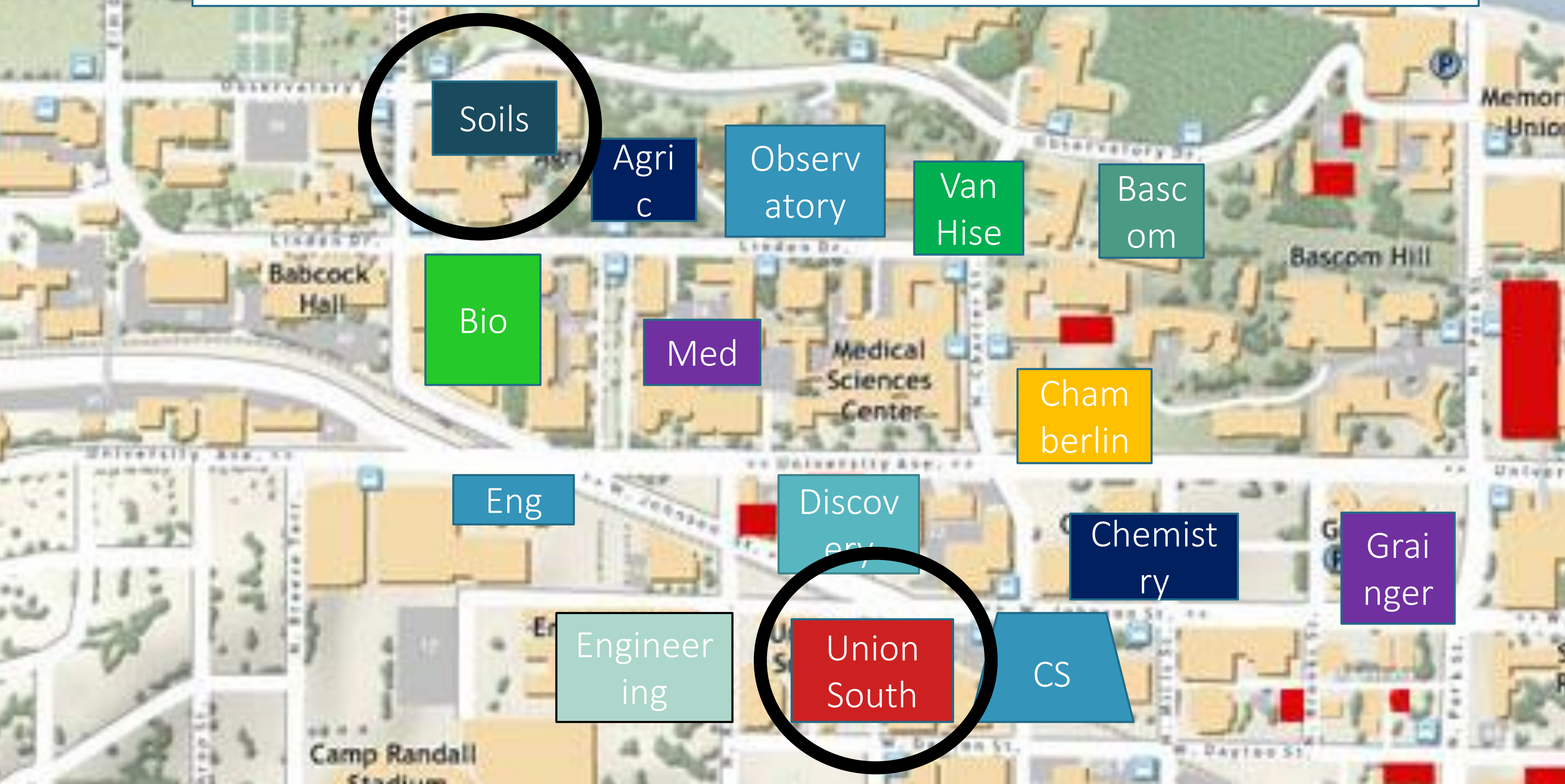
CS-540: Intro to Artificial Intelligence

SECTION 1

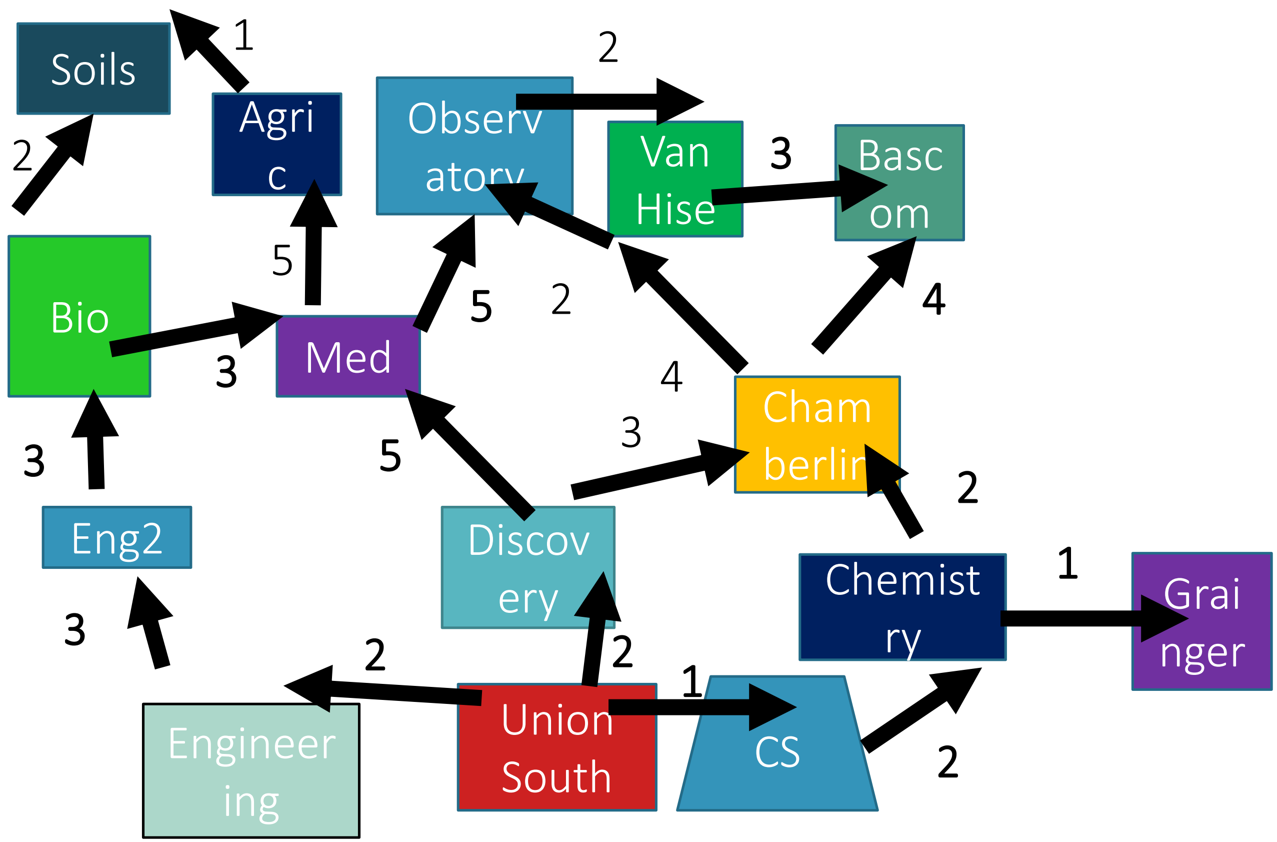
LECTURER: ERIN WINTER

Informed Search

Given the search techniques you currently know, how would you find the fastest route from Union South to Soil Sciences?

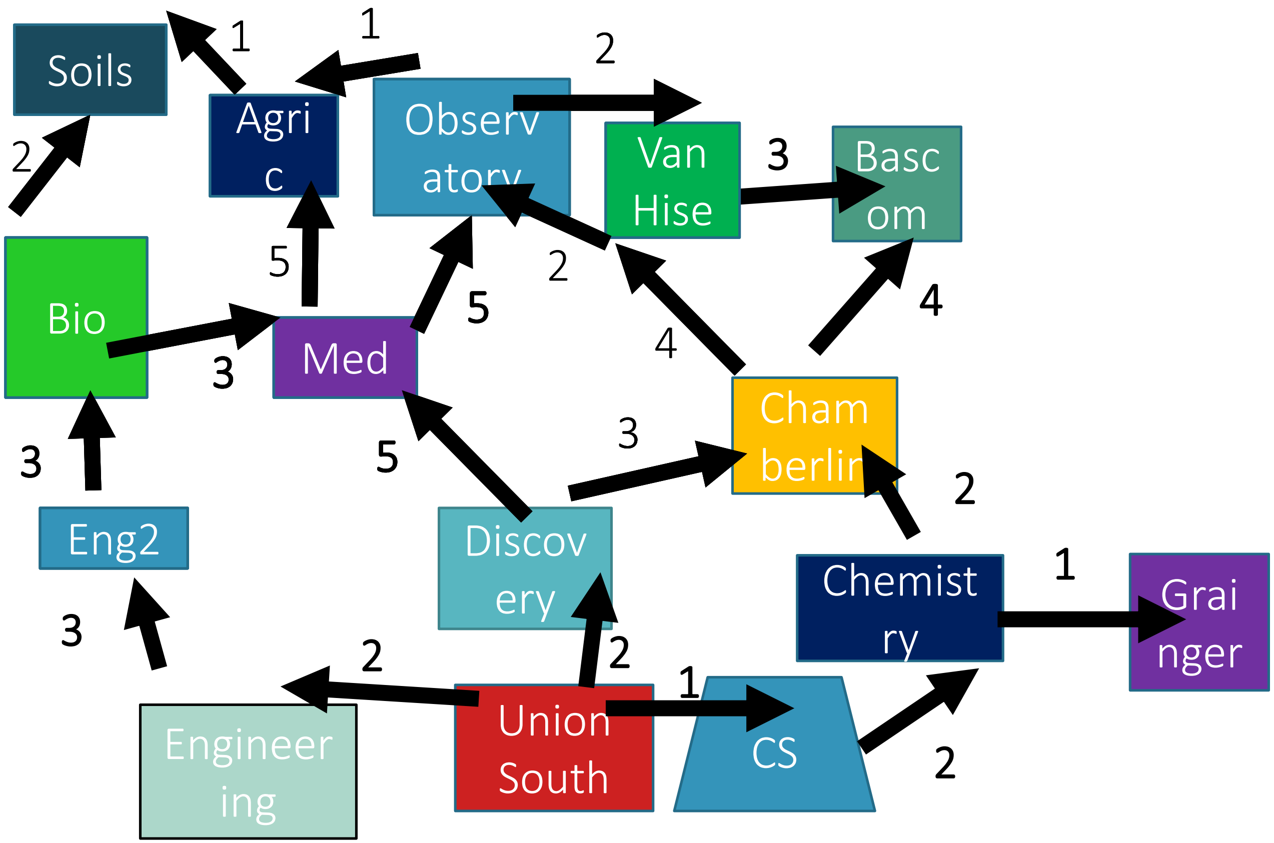


Because you have a **weighted, directed** graph Dijkstra's Search is your only suitable choice. It is also **optimal** and **complete**. Observe the first few states that Dijkstra's chooses to expand.



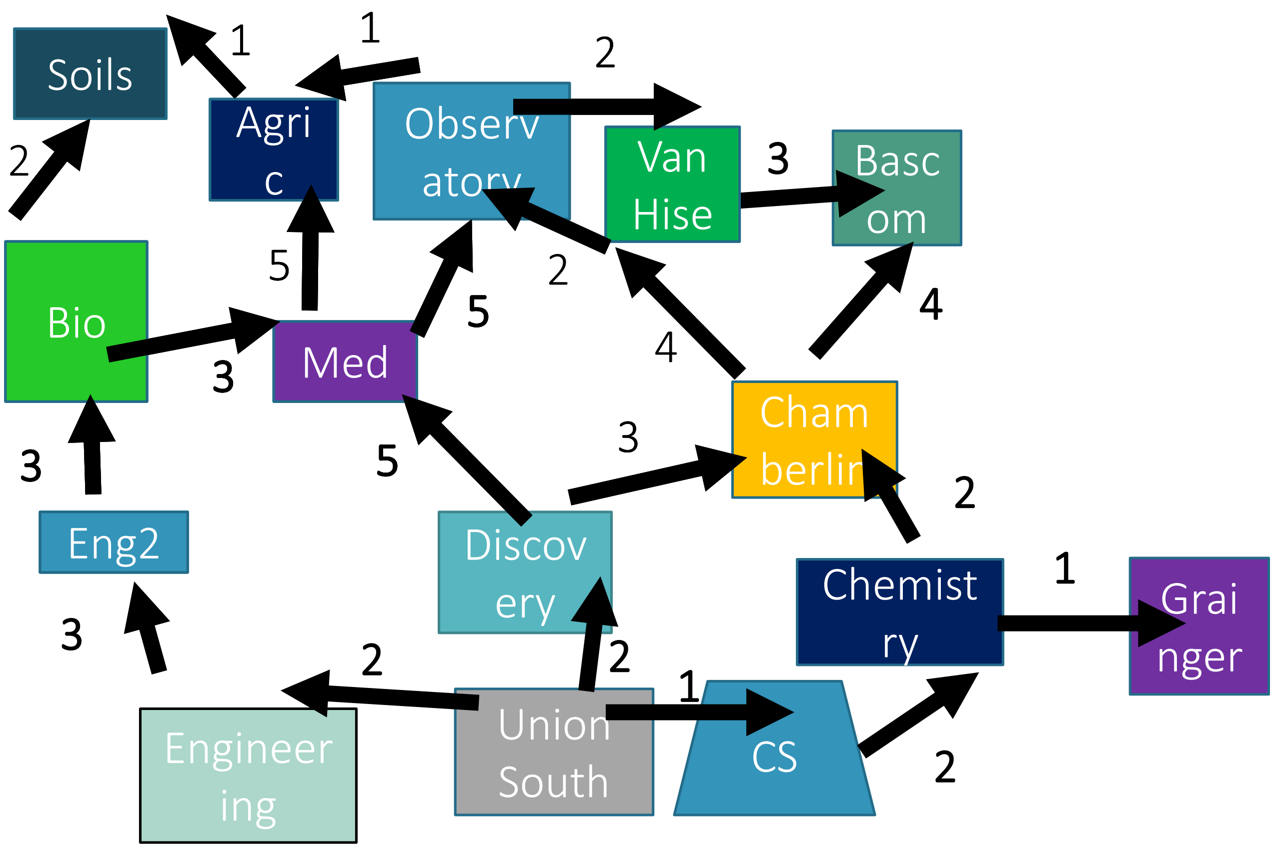
*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue

Frontier	Expanded
{US:0}	{}



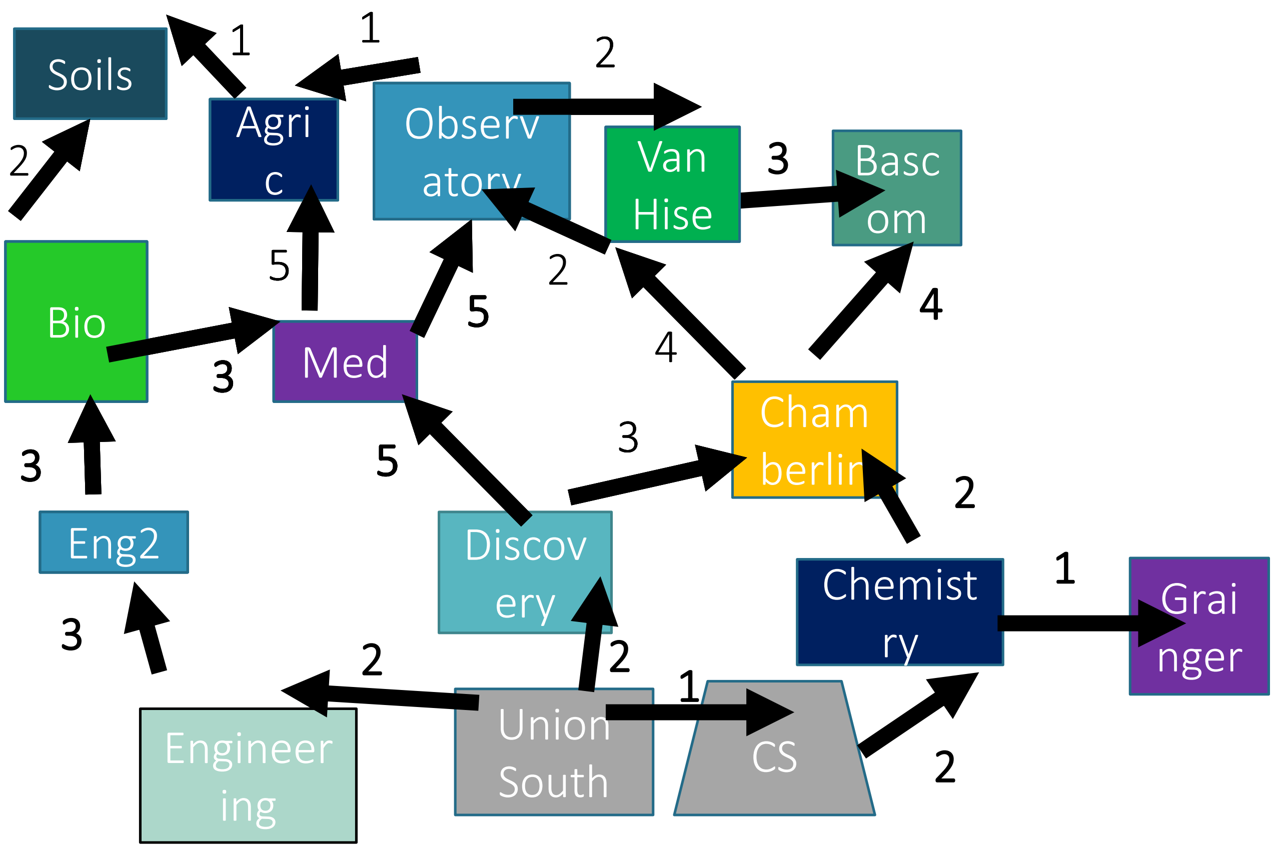
*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue

Frontier	Expanded
{US:0}	{}
{CS:1,DIS:2,ENG:2}	{US}



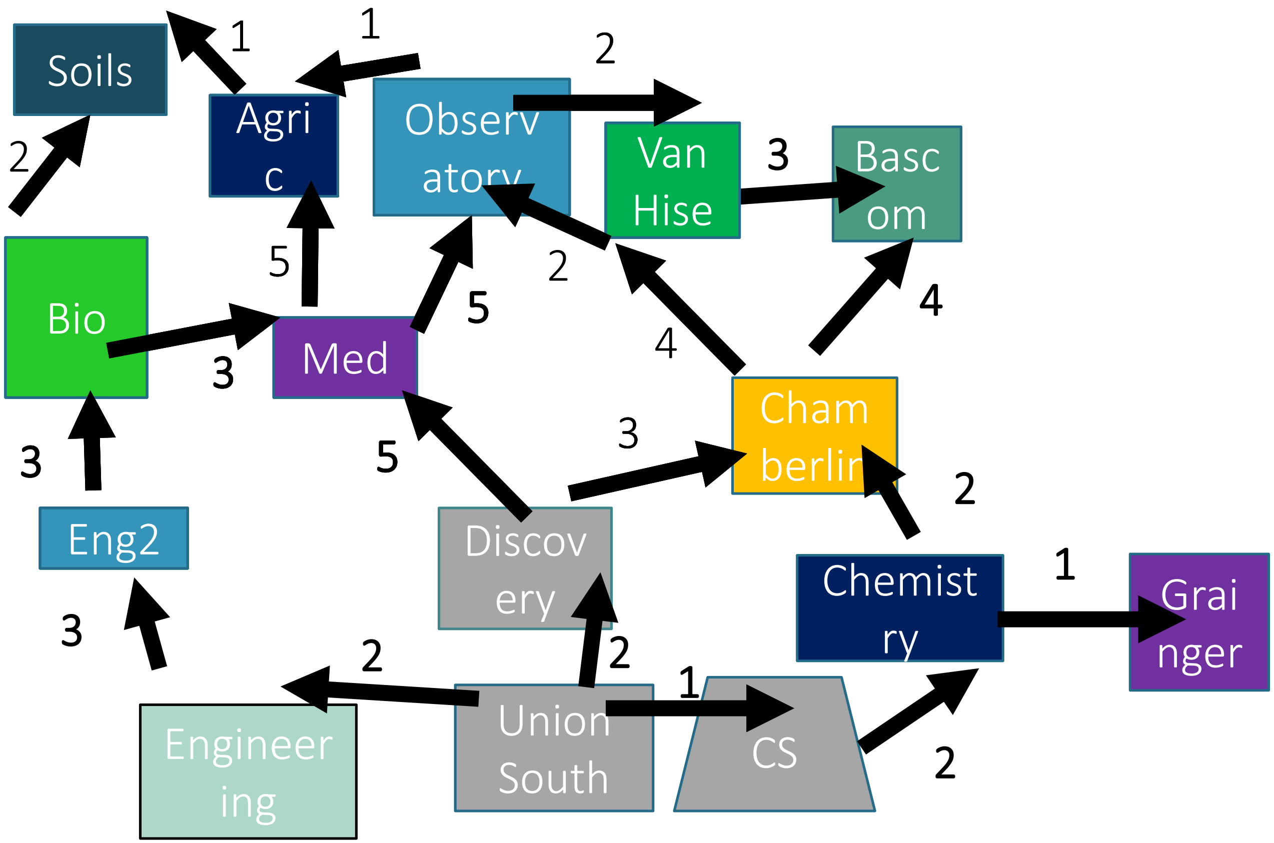
*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue

Frontier	Expanded
{US:0}	{}
{CS:1,DIS:2,ENG:2}	{US}
{DIS:2, ENG:2,CHEM:3}	{US,CS}



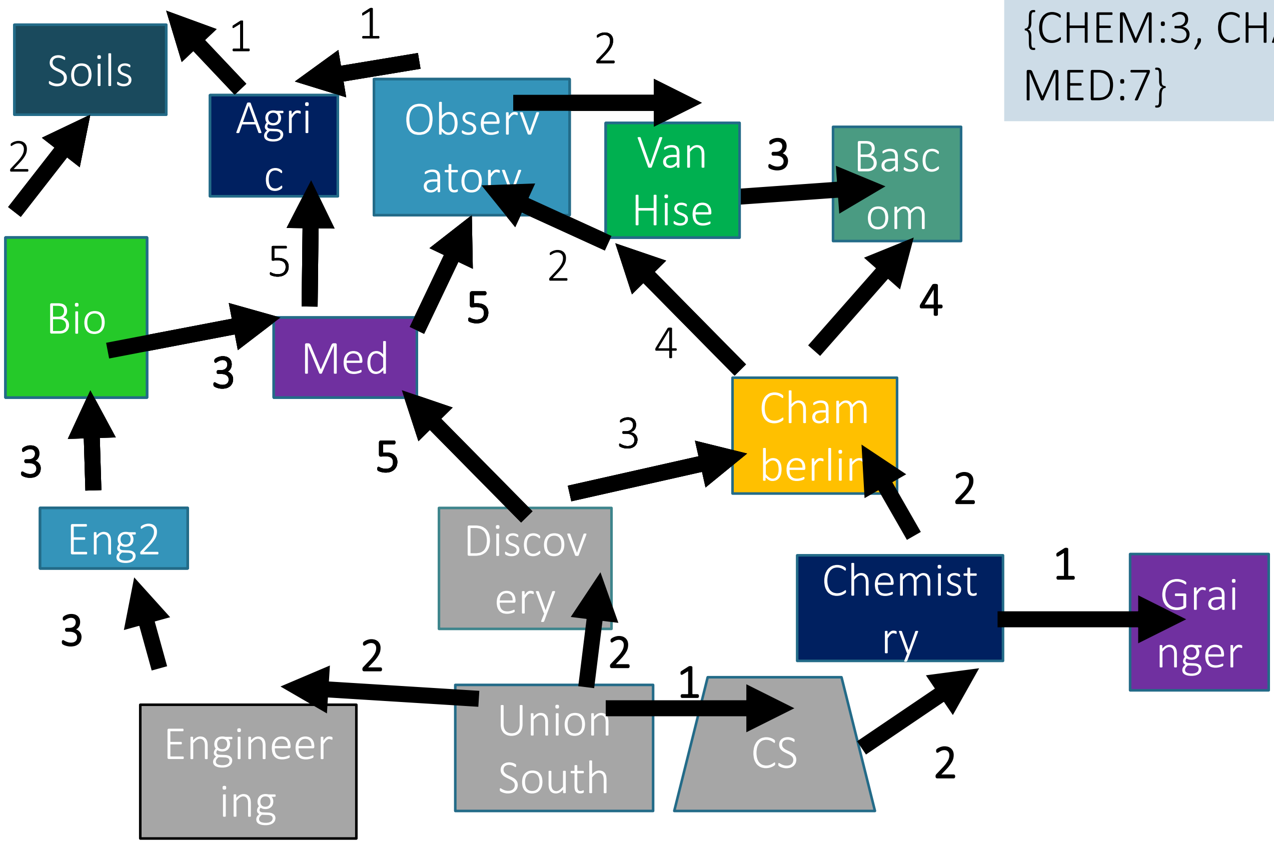
*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue

Frontier	Expanded
{US:0}	{}
{CS:1,DIS:2,ENG:2}	{US}
{DIS:2, ENG:2,CHEM:3}	{US,CS}
{ENG:2,CHEM:3, CHA:5,MED:7}	{US,CS,DIS}



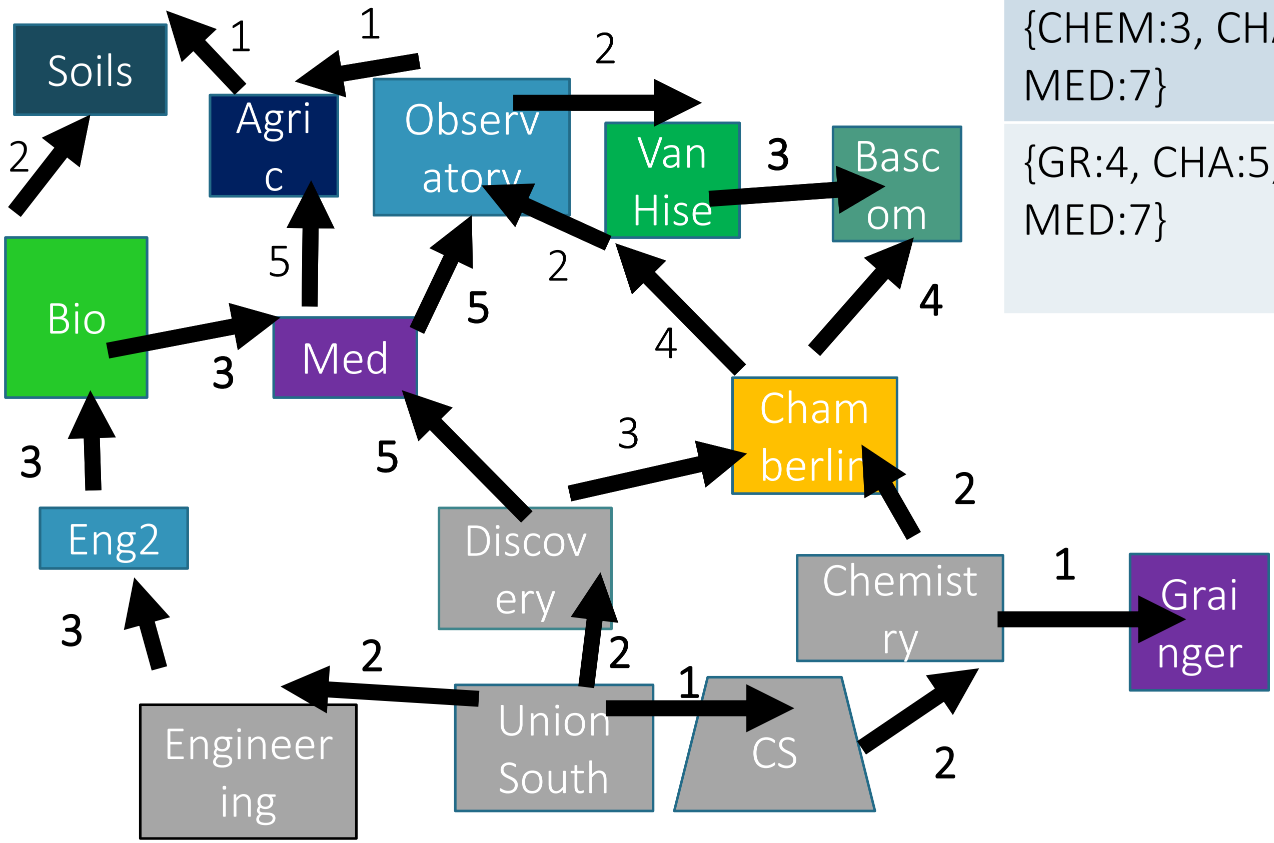
*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue

Frontier	Expanded
{US:0}	{}
{CS:1,DIS:2,ENG:2}	{US}
{DIS:2, ENG:2,CHEM:3}	{US,CS}
{ENG:2,CHEM:3, CHA:5,MED:7}	{US,CS,DIS}
{CHEM:3, CHA:5, EN2:5, MED:7}	{US, CS, DIS, ENG}

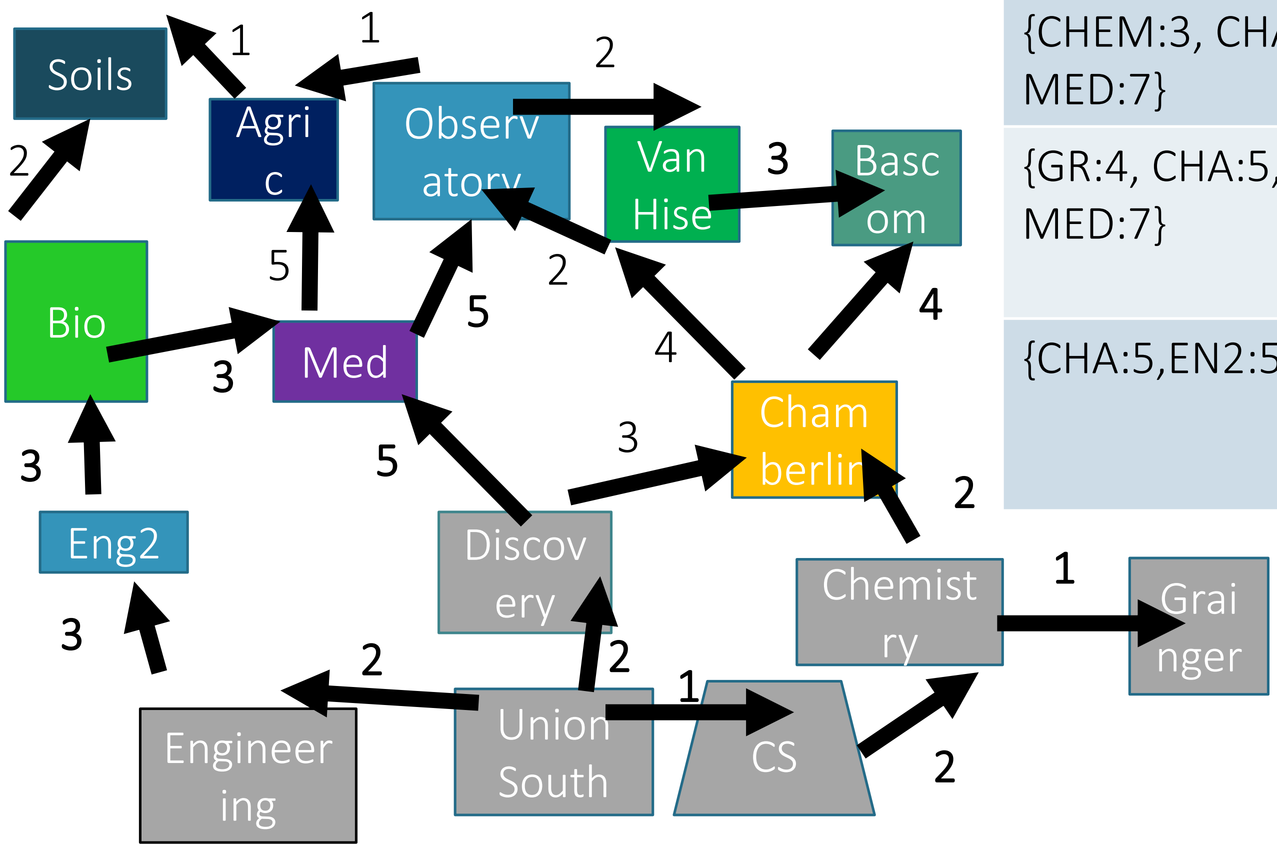


*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue

Frontier	Expanded
{US:0}	{}
{CS:1,DIS:2,ENG:2}	{US}
{DIS:2, ENG:2,CHEM:3}	{US,CS}
{ENG:2,CHEM:3, CHA:5,MED:7}	{US,CS,DIS}
{CHEM:3, CHA:5, EN2:5, MED:7}	{US, CS, DIS, ENG}
{GR:4, CHA:5, EN2:5, MED:7}	{US, CS,DIS, ENG, CHEM}



*Assume ties broken alphabetically
*Allow state to appear multiple times in priority queue



Frontier	Expanded
{US:0}	{}
{CS:1,DIS:2,ENG:2}	{US}
{DIS:2, ENG:2,CHEM:3}	{US,CS}
{ENG:2,CHEM:3, CHA:5,MED:7}	{US,CS,DIS}
{CHEM:3, CHA:5, EN2:5, MED:7}	{US, CS, DIS, ENG}
{GR:4, CHA:5, EN2:5, MED:7}	{US, CS,DIS, ENG, CHEM}
{CHA:5,EN2:5,MED:7}	{US,CS,DIS, ENG,CHEM ,GR}

What you know that Dijkstra's doesn't…

- The distance of each state relative to Soil Sciences!
- It doesn't make sense *in context* to investigate Grainger Hall or the Chemistry building as potential stops on the shortest route to Soil Sciences.
- How to reduce wasted effort? **A heuristic function that utilizes relative distance from the goal state.**

Big Idea of Informed Search

- Search problems are a pursuit of some goal state through a succession of intermediary states
- Instead of ordering states within the frontier by path cost, try to quantify the distance of a given state to a goal state using a **heuristic function**
- Heuristic functions are educated **guesses**, even with domain knowledge we can't expect them to be perfect

Heuristic Functions

Define a heuristic function, ***h*(*n*)**

- uses domain-specific information in some way
- is computable from the current state description
- it estimates
  - *the "goodness" of node **n***
  - *how close node **n** is to a goal*
  - *the cost of minimal cost path from node **n** to a goal state*

Heuristic Function Properties

- $h(n) \geq 0$, for all nodes n*
- *h(n)* close to 0 means we think *n* is close to a goal state
- *h(n)* much greater than 0 means we think *n* is far from a goal state

*this property is especially important for ensuring correctness in algorithms that utilize heuristic functions

(Greedy) Best-First Search

- Implement a graph search with a **priority queue** frontier
- Prioritize nodes by lowest *h(n)* value, that is, expand nodes estimated closest to the goal first
- Called a “greedy” algorithm because it makes shortsighted decisions based on what seems best at the given moment – selecting the lowest *h(n)* value node from the frontier

Greedy Best-First Search

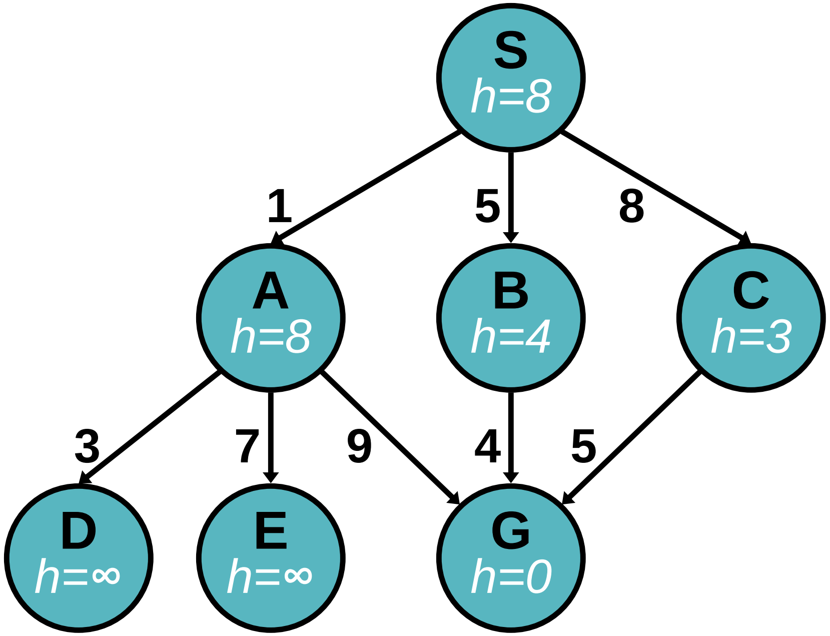
- Is it optimal?
- Is it complete?
- If it doesn’t have both of these properties, it would be wiser to simply use Dijkstra’s algorithm.

Greedy Best-First Search

$f(n) = h(n)$

\# of nodes tested: 0, expanded: 0

expnd. node	Frontier
	{S:8}

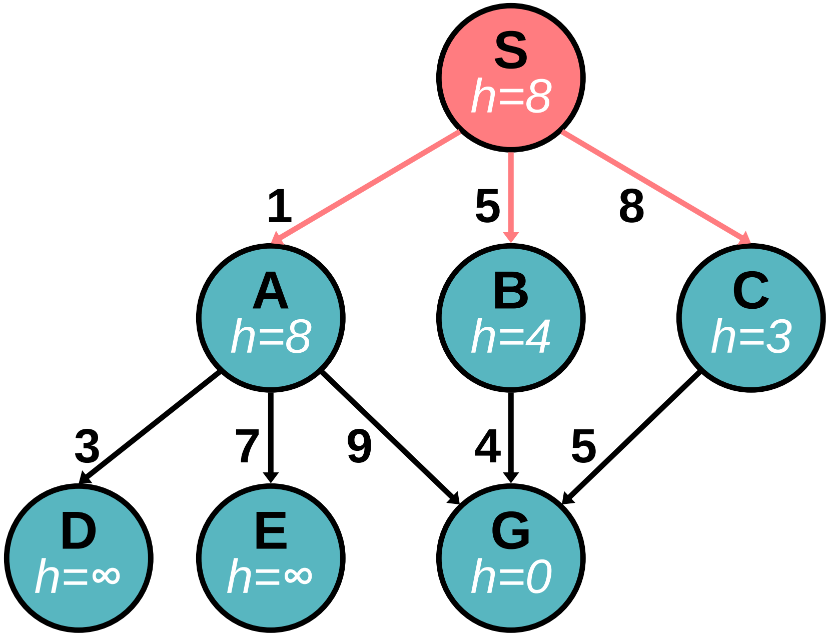


Greedy Best-First Search

$$f(n) = h(n)$$

# of nodes tested: 1, expanded: 1

expnd. node	Frontier
	{S:8}
S not goal	{C:3,B:4,A:8}

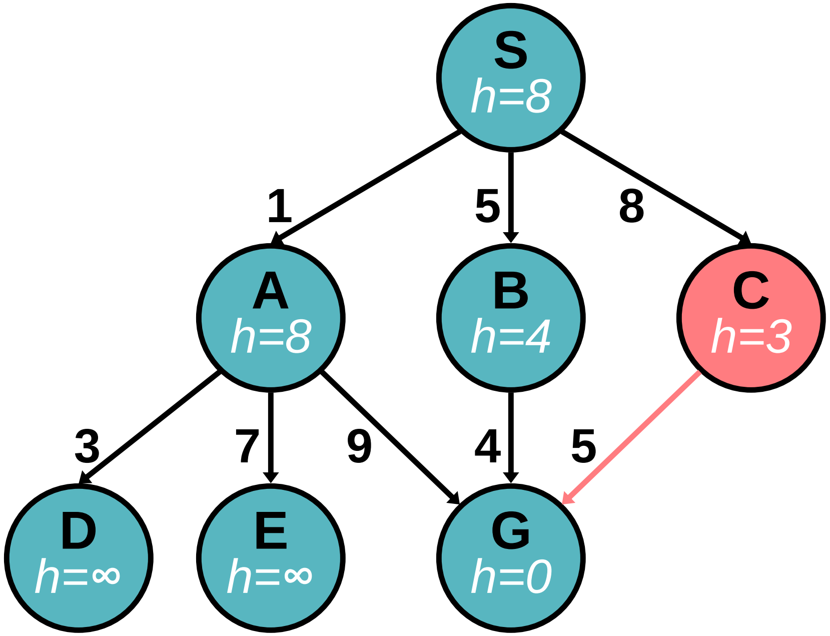


Greedy Best-First Search

$$f(n) = h(n)$$

# of nodes tested: 2, expanded: 2

expnd. node	Frontier
	{S:8}
S	{C:3,B:4,A:8}
C not goal	{G:0,B:4,A:8}

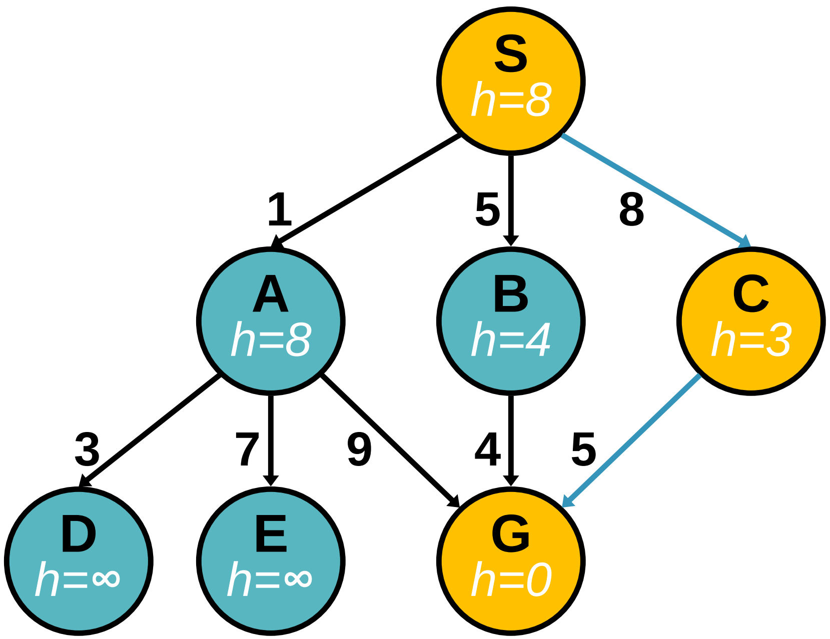


Greedy Best-First Search

$f(n) = h(n)$

\# of nodes tested: 3, expanded: 2

expnd. node	Frontier
	{S:8}
S	{C:3,B:4,A:8}
C	{G:0,B:4, A:8}
G	{B:4, A:8}



Optimal Path: S, B, G

Optimal Path Cost: 9

BFS Path: S,C,G

BFS Path Cost: 13

Learning from GBFS's Mistakes

- Greedy best-first search makes the unsound assumption that the heuristic function always estimates distance to a goal state correctly
- With an especially bad heuristic function, greedy best-first search can traverse the entire graph **$O(b^m)$** or worse, get on an infinite incorrect path like depth-first search
- Time and space complexity: **$O(b^m)$**

Attempt 2: A-Search

same computation as Dijkstra's

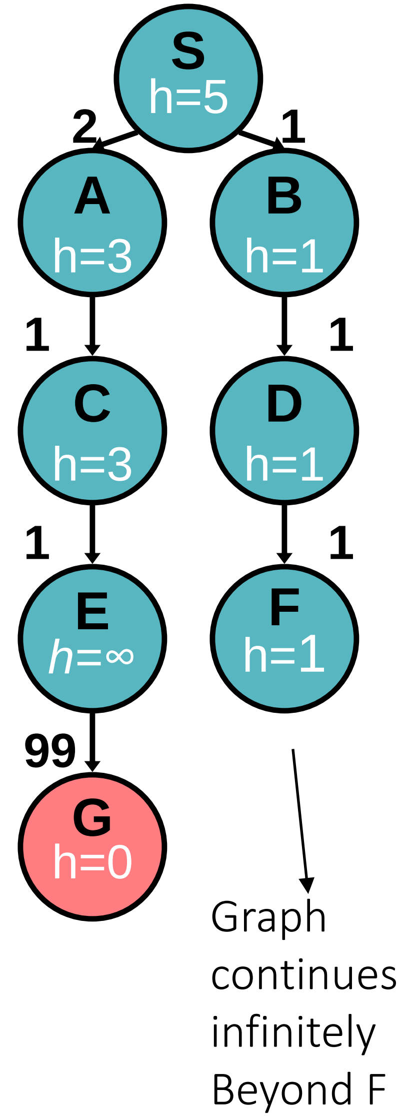
- Use as an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is minimum cost path from start to current node n (as defined in UCS)
- Nodes in *the frontier* are ranked by the estimated cost of a solution, where $g(n)$ is the cost from the start node to node n, and $h(n)$ is the estimated cost from node n to a goal
- Think of it as a "first half" (from the start to node n using known edges) + "second half" (estimated cost from n to goal) computation

Evaluating A-Search

- Smarter, but still falls into the some of the same traps as Greedy Best-First Search with a bad heuristic
- Neither admissible nor complete.
- $O(b^m)$ time and space complexity.

Counterexample:

Because the heuristic function returns infinity on node E, E looks so bad that it will never be expanded



How to improve A-Search…

- Constrain the heuristic function!
- A-Search incorporates desirable information, cost to current node and the estimated cost from current node to the goal
- However, any heuristic function that **prevents expansion of nodes along the goal path** can never be optimal or complete.

Admissible Heuristics

- An admissible heuristic *h(n)* never **overestimates** the cost from n to the goal
- So, an admissible heuristic applied to A-search $f(n) = g(n) + h(n)$ **can never overestimate the true solution path cost**

Admissible Heuristic Functions

Which of the following are admissible heuristics? Assume h* is the true cost from n to the goal.

$h(n) = h^*(n)$ YES

$h(n) = \max(2, h^*(n))$ NO, what if h*(n) = 1

$h(n) = \min(2, h^*(n))$ YES, this never overestimates

$h(n) = h^*(n) - 2$ NO, negative if h*(n) < 2

$h(n) = \sqrt{h^*(n)}$ NO, square root of h*(n) is > h*(n) if 0 < h*(n) < 1

Comparing Heuristic Functions

- A heuristic function h2 **dominates** h1 if *for all s*

  $h1(n) \leq h2(n) \leq h^*(n)$

- Want heuristic functions as close to h* as possible, **but not over h***
- Good heuristic functions can be slow, so sometimes it's better to use a weaker heuristic that expands more nodes in a given interval

A-Search + Admissible (or Consistent) Heuristic =

A* Search

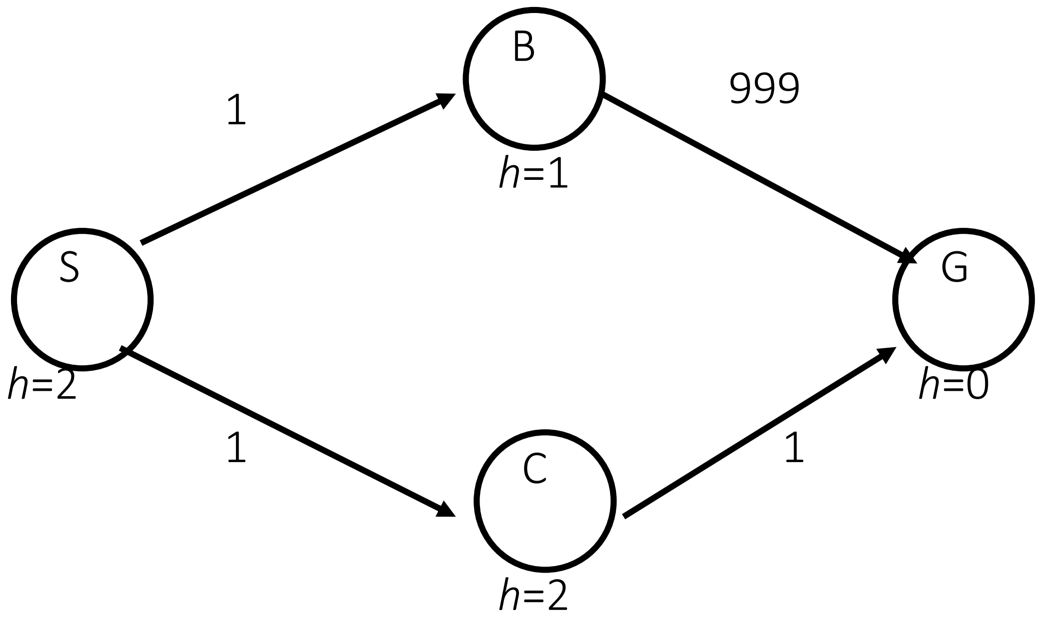
effective and used in practice

A* Algorithm

Frontier = {S} where S is the start node
Explored ={}
Loop do
  if Frontier is empty **then return** failure
  pick node, n, with min **f(n) = h(n) + g(n)** value from Frontier
  if n is a goal state **then return** solution
  for each each child, n’, of n **do**
    if n’ is not in Explored or Frontier
      then add n’ to Frontier
    else if n’ in Frontier as m
      if g(m) <= g(n’) throw n’ away
      else add n’ to Frontier and remove m
  Remove n from Frontier and add n to Explored

When Should A* Stop?

To ensure **optimality**, A* should terminate only when a goal state is ***removed*** from the priority queue



Same rule as for Dijkstra's

A* with $h(n) = 0$ ***is Dijkstra's***

Consistent Heuristic Functions

A heuristic, h, is called ***consistent*** (aka ***monotonic***) if, for every node n and every successor n' of n, the estimated cost of reaching the goal from n is *no greater than* the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$c(n, n') \geq h(n) - h(n')$$

or, equivalently: $h(n) \leq c(n, n') + h(n')$

- Values of $f(n)$ along any path are *nondecreasing*
- Consistency is a stronger condition than admissibility

Admissible vs. Consistent Heuristics

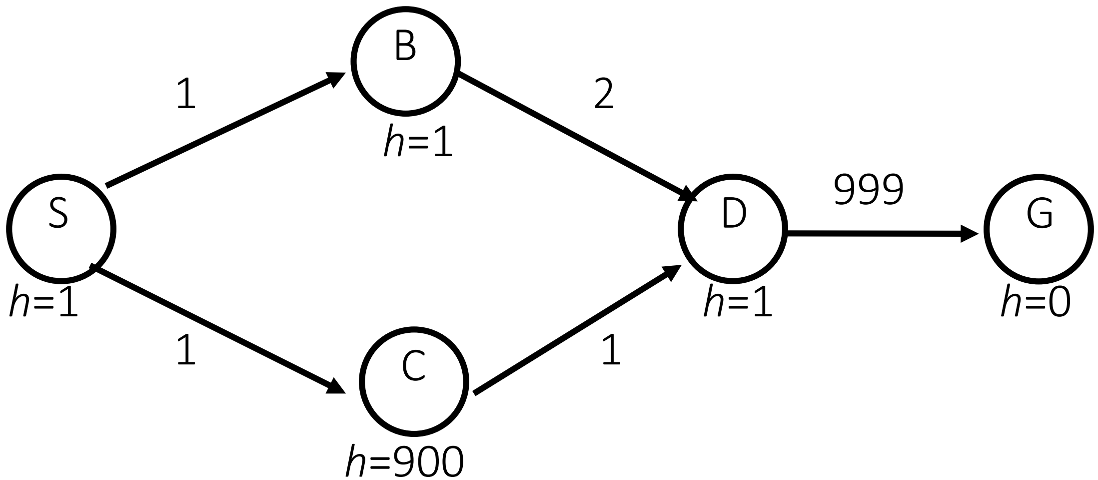
A* using an **admissible** heuristic will give an optimal solution on an acyclic graphs (trees)

However, a **consistent** heuristic must be used with A* to guarantee optimality on graphs with cycles

Remember, all *consistent heuristics are admissible, but not all admissible heuristics are consistent*

Consistent Heuristics

Is this *h* consistent?



h(C)=900, h(D)=1, c(C, D) = 1 < 900 – 1, so h is *NOT* consistent

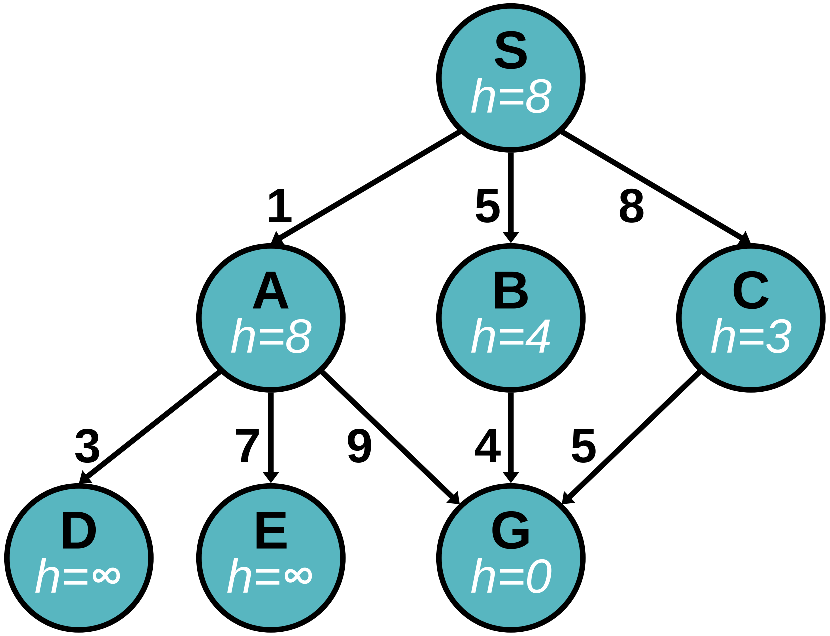
A* Search Practice

Is *h* is admissible and/or consistent? Consistent

$f(n) = g(n) + h(n)$

# of nodes tested: 0, expanded: 0

expnd. node	**Frontier**
	{S:0+8}



To test for admissibility, check that h(n) never overestimates the true cost to the goal.

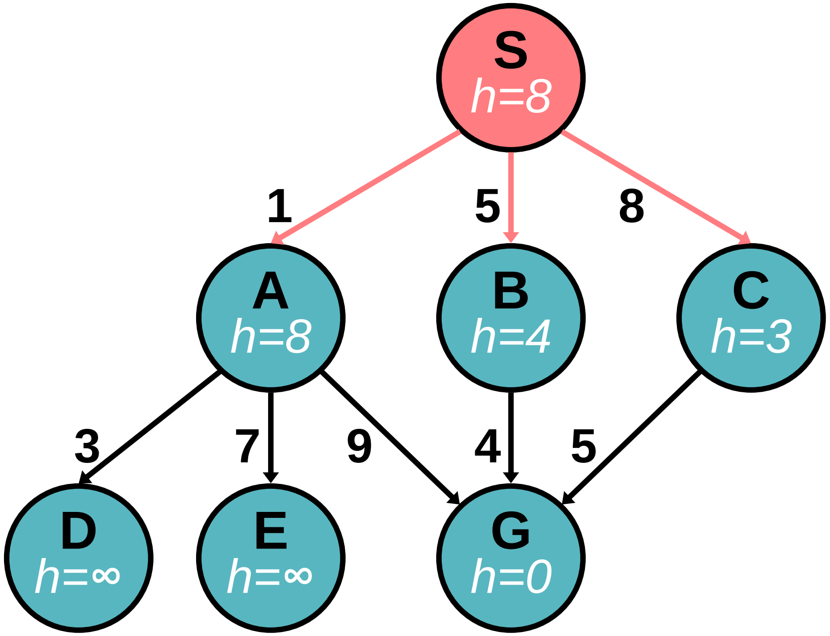
To check for consistency, check h(n) is consistent AND $h(n) \le c(n, n') + h(n')$ for all successor nodes n'.

A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 1, expanded: 1

expnd. node	Frontier
	{S:8}
S not goal	{A:1+8,B:5+4,C:8+3}

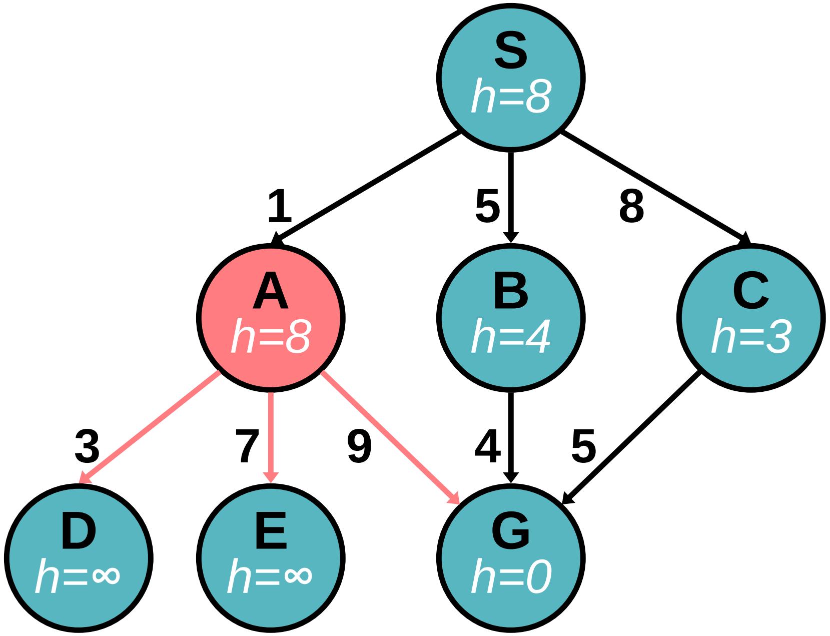


A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 2, expanded: 2

expnd. node	**Frontier**
	{S:8}
S	{A:9,B:9,C:11}
A not goal	{B:9,G:1+9+0,C:11, D:1+3+∞,E:1+7+∞}

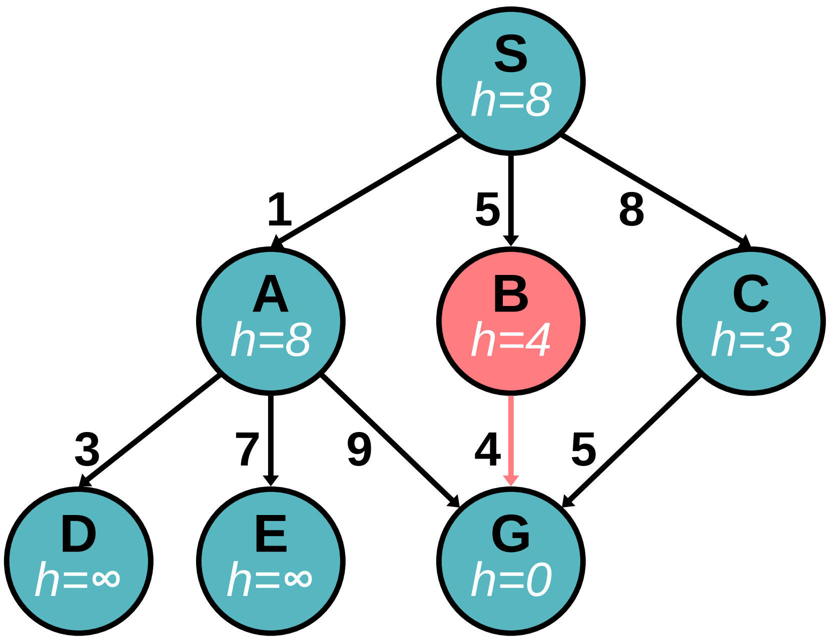


A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 3, expanded: 3

expnd. node	Frontier
	{S:8}
S	{A:9,B:9,C:11}
A	{B:9,G:10,C:11,D:∞,E:∞}
B not goal	{G:5+4+0,~~G:10~~,C:11, D:∞,E:∞} replace

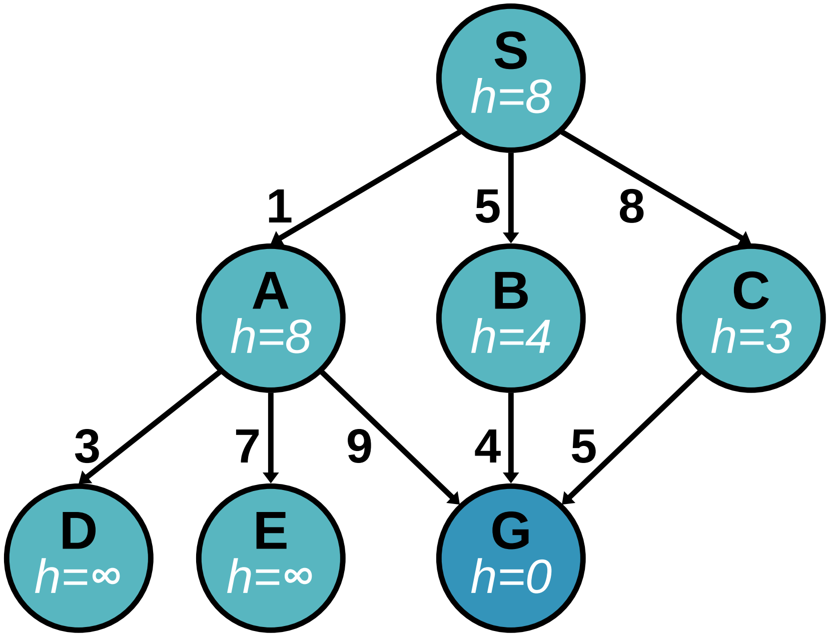


A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 4, expanded: 3

expnd. node	**Frontier**
	{S:8}
S	{A:9,B:9,C:11}
A	{B:9,G:10,C:11,D:∞,E:∞}
B	{G:9,C:11,D:∞,E:∞}
G goal	{C:11,D:∞,E:∞} not expanded

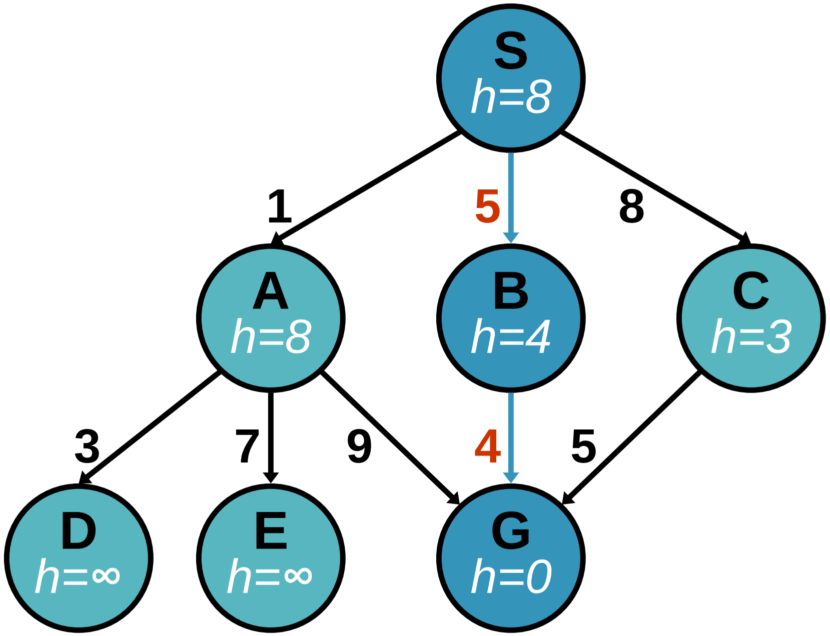


A* Search

$f(n) = g(n) + h(n)$

# of nodes tested: 4, expanded: 3

expnd. node	**Frontier**
	{S:8}
S	{A:9,B:9,C:11}
A	{B:9,G:10,C:11,D:∞,E:∞}
B	{G:9,C:11,D:∞,E:∞}
G	{C:11,D:∞,E:∞}



Fast, optimal, complete under same graph conditions as Dijkstra's

path: S,B,G
cost: 9

The Bad News About A*

- A* uses a lot of memory, ie. O(number of states)
- For some search problems, A* will fill the available system memory before finding a solution

How do we use less memory?

1. Sacrifice optimality – beam search
2. Sacrifice some time – iterative deepening A*

Beam Search

Use an evaluation function ***f*(*n*) = *h*(*n*)** as in Greedy Best-First search, ***and*** restrict the maximum size of the Frontier to a constant, ***k***

- Only keep *k* best nodes as candidates for expansion, and throw away the rest
- More space efficient than Greedy Best-First Search, but may throw away a node on a solution path. Space complexity **O(km)**.
- **Not complete**
- **Not optimal/admissible**

Iterating Deepening A*

- Iterative-deepening A*
- Cutoff based on $f = g + h$ value rather than depth
- At each iteration do loop-avoiding DFS, not expanding any node with f-value that exceeds current threshold
- At each iteration increase the f-value threshold by setting it to the smallest f-value of any node that exceeded the cutoff in the previous iteration
- Complete
- Optimal / Admissible
- Linear space required

How to come up with admissible/consistent heuristics?

Heuristics are often defined by relaxing the problem, i.e., computing the exact cost of a solution to a simplified version of problem

- remove constraints
  - **8-puzzle:** Each tile moves independently
- simplify problem
  - **8-puzzle:** A tile can move to any adjacent position ➔ Number of moves to get a tile to its goal position = City-Block distance (aka Manhattan distance or L_1 distance)
  - **Map navigation:** Use distance formula from n to goal state as h(n), which will always be optimistic because straight line distance is best case, reality has obstacles