CS 540 Introduction to Artificial Intelligence
Informed Search

Yingyu Liang
University of Wisconsin-Madison

Nov 16, 2021

Based on slides by Fred Sala

Outline

- Uninformed continued
- A* Search
  - Heuristic properties, stopping rules, analysis

General State-Space Search Algorithm

```
function general-search(problem, QUEUEING-FUNCTION)
  ;; problem describes the start state, operators, goal test, and
  ;;   operator costs
  ;; queueing-function is a comparator function that ranks two states
  ;; general-search returns either a goal node or "failure"

  nodes = MAKE-QUEUE(MAKE-NODE(problem.INITIAL-STATE))
  loop
    if EMPTY(nodes) then return "failure"
    node = REMOVE-FRONT(nodes)
    if problem.GOAL-TEST(node.STATE) succeeds then return node
    nodes = QUEUEING-FUNCTION(nodes, EXPAND(node,
                                   problem.OPERATORS))
    ;; succ(s)=EXPAND(s, OPERATORS)
    ;; Note: The goal test is NOT done when nodes are generated
    ;; Note: This algorithm does not detect loops
  end
```

Recall the bad space complexity of BFS

Four measures of search algorithms:

Solution: Uniform-cost search

- Completeness (not finding all goals): … find a goal.
- Optimality: yes if edges cost 1 (more generally positive non-decreasing with depth), no otherwise.
- Time comple…): goal is the last node at radius d.

Solution: Depth-first search

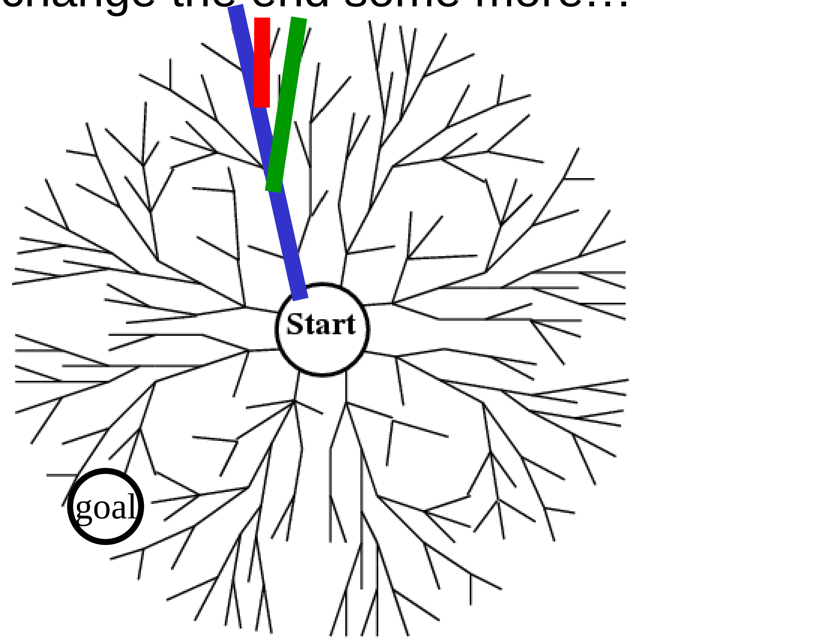
  - Have to … nodes at radius d.
  - $b + b^2 + \ldots + b^d \sim O(b^d)$
- Space complexity (bad, see the Figure)
  - Back points for all generated nodes $O(b^d)$
  - The queue (smaller, but still $O(b^d)$)

Depth-first search

Expand the deepest node first

1. Select a direction, go deep to the end
2. Slightly change the end
3. Slightly change the end some more…

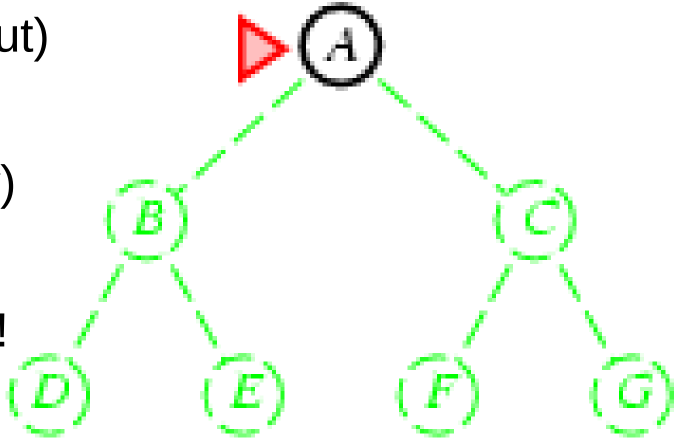
fan



Depth-first search (DFS)

Use a stack (First-in Last-out)

1. push(Initial states)
2. While (stack not empty)
3.   s = pop()
4.   if (s==goal) success!
5.   T = succs(s)
6.   push(T)
7. endWhile



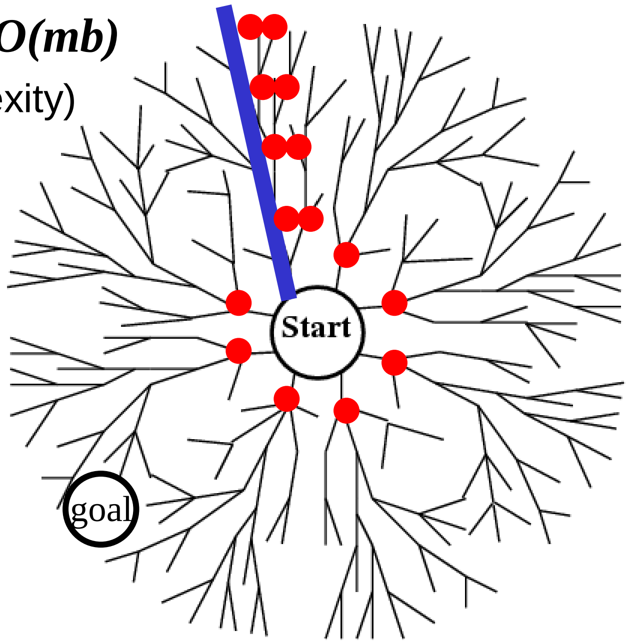
stack (fringe)

[] ⇔

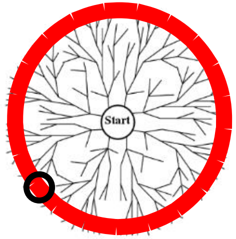
What's in the fringe for DFS?

- m = maximum depth of graph from start
- ***m(b-1) ~ O(mb)***

(Space complexity)



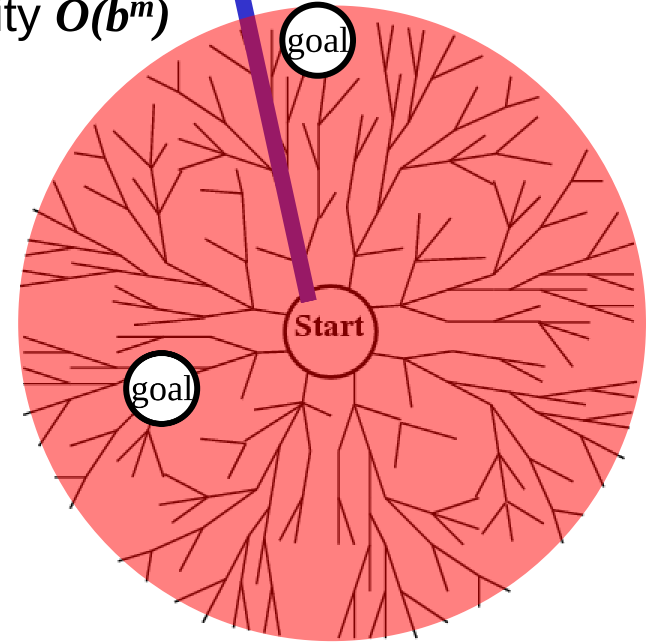
c.f. BFS $O(b^d)$



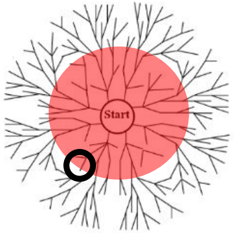
- "backtracking search" even less space
  - generate siblings (if applicable)

What's wrong with DFS?

- Infinite tree: may not find goal (incomplete)
- May not be optimal
- Finite tree: may visit almost all nodes, time complexity $O(b^m)$



c.f. BFS $O(b^d)$



Performance of search algorithms on trees

b: branching factor (assume finite) d: goal depth m: graph depth

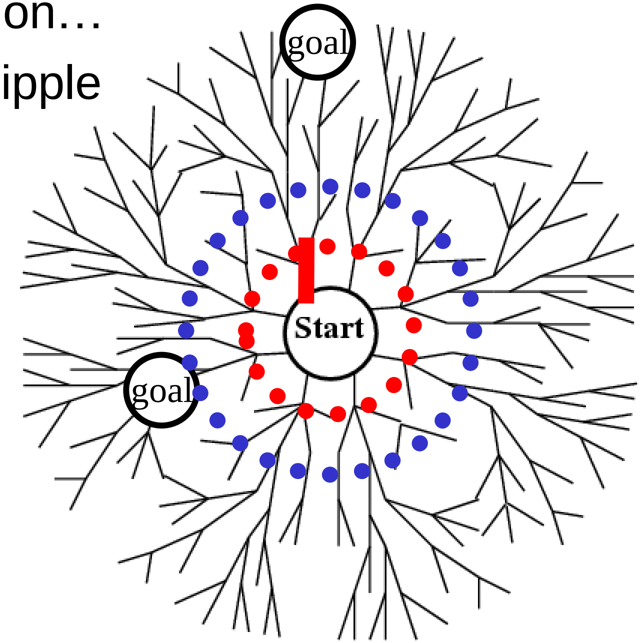
	Complete	optimal	time	space
Breadth-first search	Y	Y, if [1]	$O(b^d)$	$O(b^d)$
Uniform-cost search[2]	Y	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Depth-first search	N	N	$O(b^m)$	$O(bm)$

1. edge cost constant, or positive non-decreasing in depth
2. edge costs $\geq \varepsilon > 0$. C^* is the best goal path cost.

How about this?

1. DFS, but stop if path length > 1.
2. If goal not found, repeat DFS, stop if path length > 2.
3. And so on…

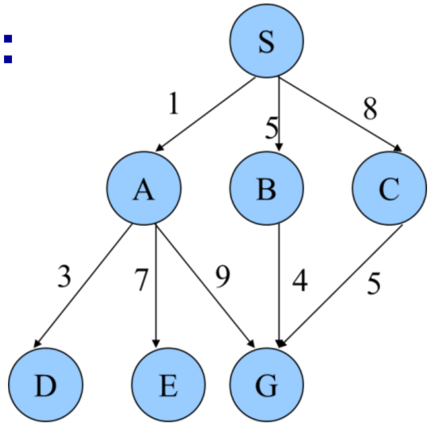
fan within ripple



Iterative deepening

- Search proceeds like BFS, but fringe is like DFS
  - Complete, optimal like BFS
  - Small space complexity like DFS
  - Time complexity like BFS
- Preferred uninformed search method

Nodes expanded by:



- Breadth-First Search: S A B C D E G

  Solution found: S A G

- Uniform-Cost Search: S A D B C E G

  Solution found: S B G (This is the only uninformed search that worries about costs.)

- Depth-First Search: S A D E G

  Solution found: S A G

- Iterative-Deepening Search: S A B C S A D E G

  Solution found: S A G

Performance of search algorithms on trees

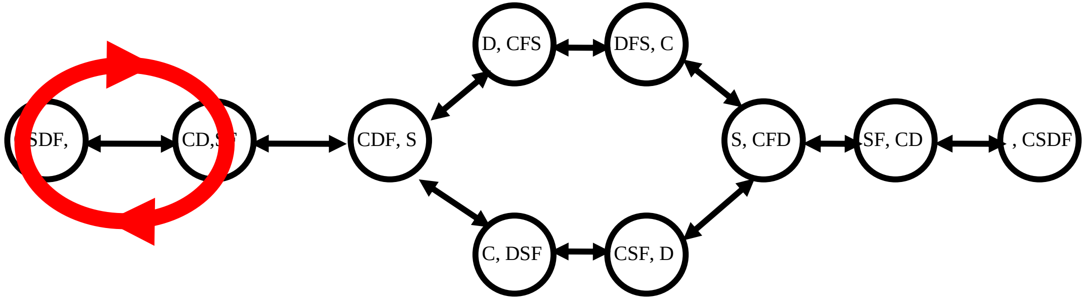
b: branching factor (assume finite) d: goal depth m: graph depth

	Complete	optimal	time	space
Breadth-first search	Y	Y, if [1]	$O(b^d)$	$O(b^d)$
Uniform-cost search[2]	Y	Y	$O(b^{C^*/\varepsilon})$	$O(b^{C^*/\varepsilon})$
Depth-first search	N	N	$O(b^m)$	O(bm)
Iterative deepening	Y	Y, if [1]	$O(b^d)$	O(bd)

1. edge cost constant, or positive non-decreasing in depth
2. edge costs $\geq \varepsilon > 0$. C* is the best goal path cost.

If state space graph is not a tree

- The problem: repeated states



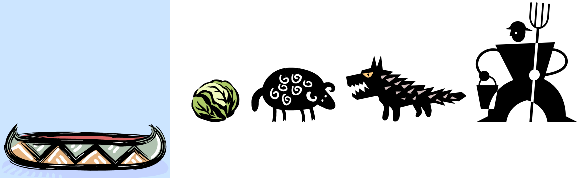
- Ignore the danger of repeated states: wasteful (BFS) or impossible (DFS). Can you see why?
- How to prevent it?

If state space graph is not a tree

- We have to remember already-expanded states (CLOSED).
- When we take out a state from the fringe (OPEN), check whether it is in CLOSED (already expanded).
  - If yes, throw it away.
  - If no, expand it (add successors to OPEN), and move it to CLOSED.

What you should know

- Problem solving as search: state, successors, goal test
- Uninformed search
  - Breadth-first search
    - Uniform-cost search
  - Depth-first search
  - Iterative deepening



- Can you unify them using the same algorithm, with different priority functions?
- Performance measures
  - Completeness, optimality, time complexity, space complexity

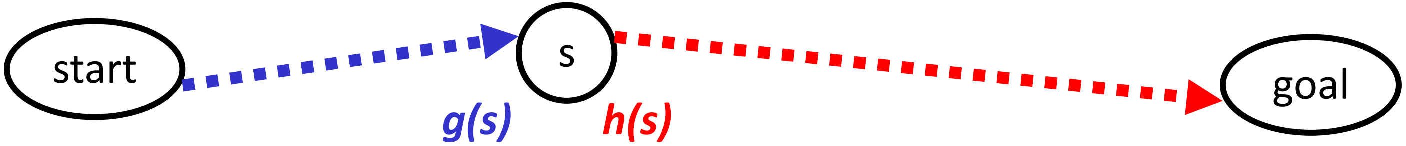
Uninformed vs Informed Search

Uninformed search (all of what we saw). Know:

- Path cost ***g***(*s*) from start to node *s*
- Successors.

Informed search. Know:

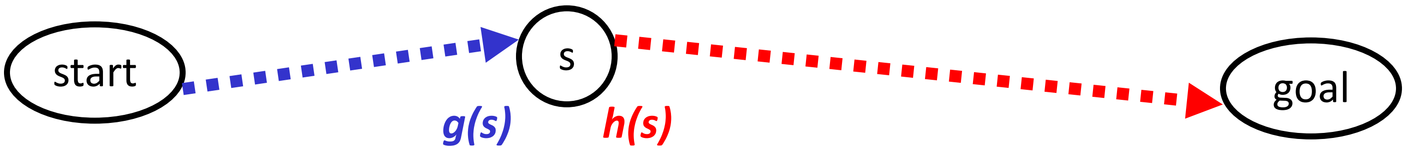
- All uninformed search properties, plus
- Heuristic **h(s)** from s to goal



Informed Search

Informed search. Know:

- All uninformed search properties, plus
- Heuristic ***h*(*s*)** from *s* to goal

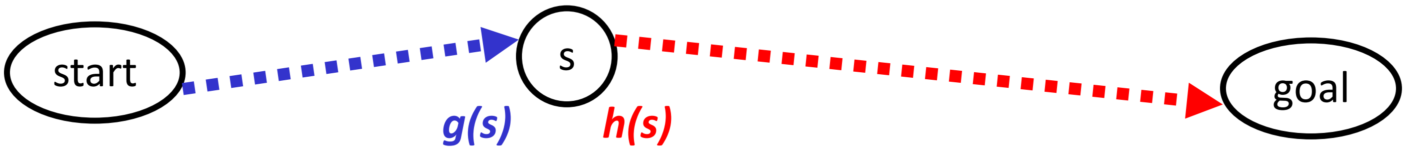


- Use information to **speed up search.**

Using the Heuristic

Back to uniform-cost search

- We had the priority queue
- Expand the node with the smallest ***g*(*s*)**
  - ***g*(*s*)** "first-half-cost"

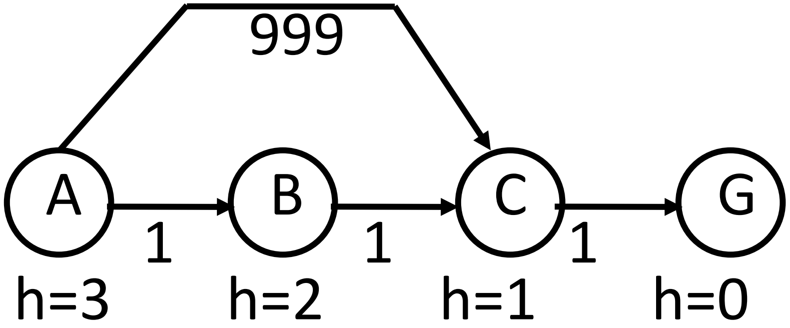


- Now let's use the heuristic ("second-half-cost")
  - Several possible approaches: let's see what works

Attempt 1: Best-First Greedy

One approach: just use ***h*(*s*)** alone

- Specifically, expand node with smallest ***h*(*s*)**
- This isn't a good idea. Why?

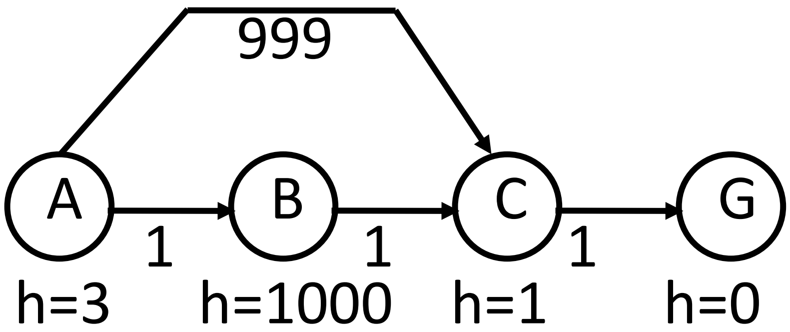


- Not optimal! **Get** A → C → G. **Want**: A →B → C → G

Attempt 2: **A Search**

Next approach: use both ***g*(*s*)** + ***h*(*s*)**

- Specifically, expand node with smallest ***g*(*s*)** + ***h*(*s*)**
- Again, use a priority queue
- Called **“A” search**

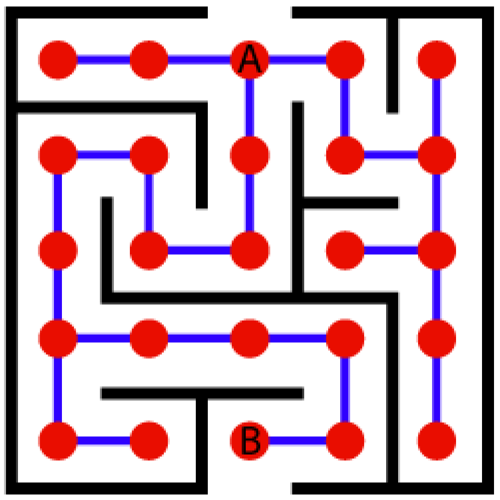


- **Still not optimal!** (Does work for former example).

Attempt 3: **A* Search**

Same idea, use ***g*(*s*)** + ***h*(*s*)**, with one **requirement**

- Demand that $0 \le h(s) \le h^*(s)$, the actual cost
- If heuristic has this property, "admissible"
  - Optimistic! Never over-estimates
- Still need $h(s) \ge 0$
  - Negative heuristics can lead to strange behavior
- This is **A* search**



V. Batoćanin

Admissible Heuristic Functions

Have to be careful to ensure admissibility (**optimism!**)

- Example: **8-puzzle**

Example State

1		5
2	6	3
7	4	8

Goal State

1	2	3
4	5	6
7	8	

- One useful approach: **relax constraints**
  - ***h*(*s*)** = number of tiles in wrong position
    - allows tiles to fly to destination in a single step

Heuristic Function Tradeoffs

Dominance: h_2 dominates h_1 if for all states *s*,

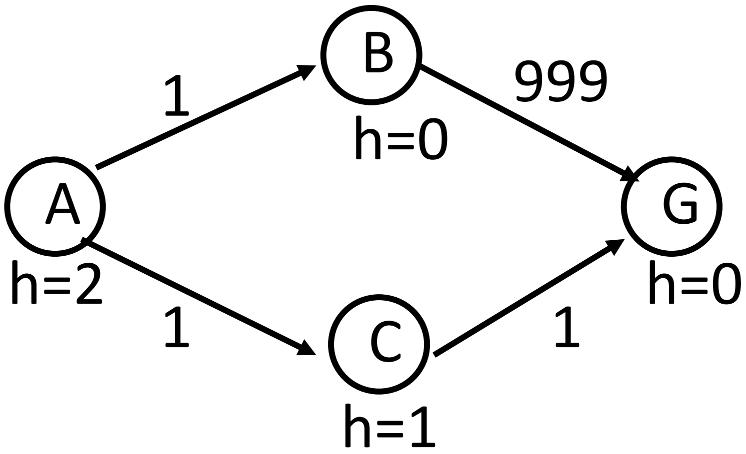
$$h_1(s) \leq h_2(s) \leq h^*(s)$$

- **Idea**: we want to be as close to h^* as possible
  - But not over!
- **Tradeoff**: being very close might require a very complex heuristic, expensive computation
  - Might be better off with cheaper heuristic & expand more nodes.

A* Termination

When should A* **stop**?

- One idea: as soon as we reach goal state?

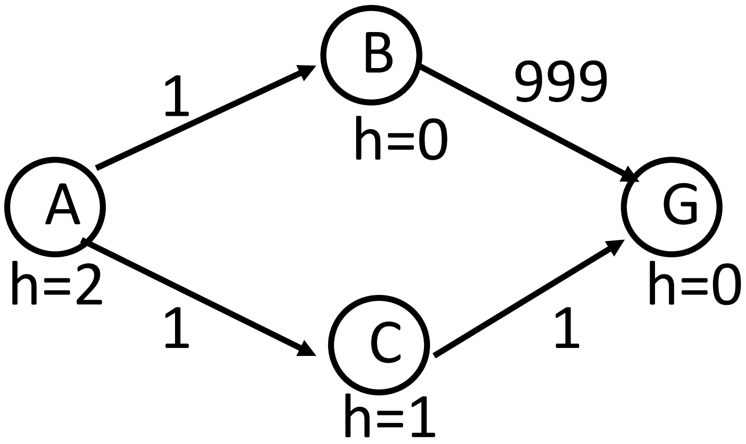


- ***h*** admissible, but note that we get A →B → G (**cost 1000**)!

A* Termination

When should A* stop?

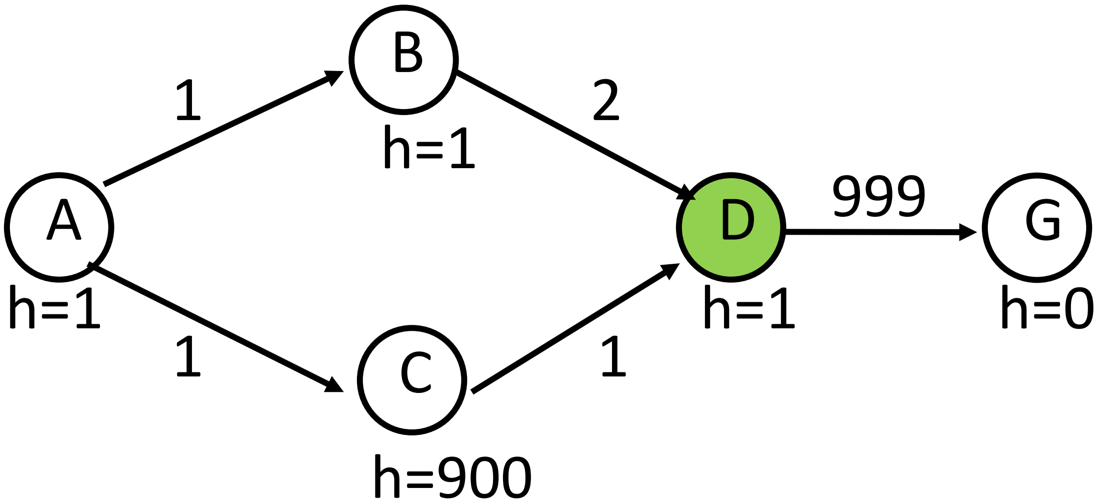
- **Rule**: terminate **when a goal is popped** from queue.



- Note: taking ***h*** =0 reduces to uniform cost search rule.

A* Revisiting Expanded States

Possible to revisit an expanded state, get a shorter path:



- Put D back into priority queue, smaller **g**+**h**

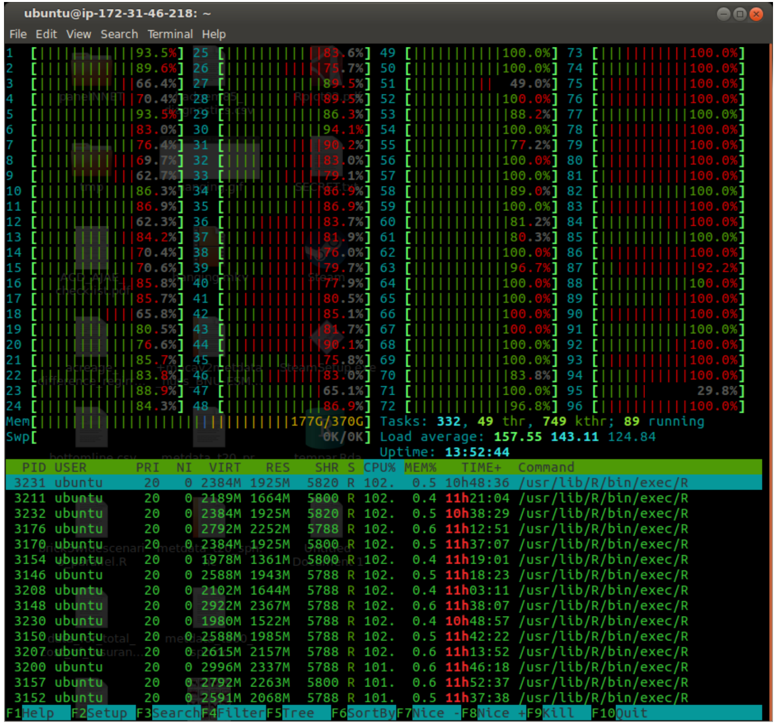
A* Full Algorithm

1. Put the start node S on the priority queue, called OPEN
2. If OPEN is empty, exit with failure
3. Remove from OPEN and place on CLOSED a node n for which f(n) is minimum (note that f(n)=g(n)+h(n))
4. If n is a goal node, exit (trace back pointers from n to S)
5. Expand n, generating all successors and attach to pointers back to n. For each successor n' of n
    1. If n' is not already on OPEN or CLOSED estimate h(n'), g(n')=g(n)+ c(n,n'), f(n')=g(n')+h(n’), and place it on OPEN.
    2. If n' is already on OPEN or CLOSED, then check if g(n') is lower for the new version of n'. If so, then:
        1. Redirect pointers backward from n' along path yielding lower g(n').
        2. Put n' on OPEN.
    3. If g(n') is not lower for the new version, do nothing.
6. Goto 2.

A* Analysis

Some properties:

- Terminates!
- A* can use **lots of memory**: O(# states).
- Will run out on large problems.



Summary

- Informed search: introduce heuristics
  - Not all approaches work: best-first greedy is bad
- A* algorithm
  - Properties of A*, idea of admissible heuristics

Acknowledgements: Adapted from materials by Jerry Zhu, Anthony Gitter, and Fred Sala (University of Wisconsin-Madison).