Search, Part 4

Lecture 7

Chapter 3, Sections 3.5-3.6

Jim Rehg

College of Computing

Georgia Tech

January 29, 2016

Where Was Prof. Rehg last Wed?

NSF I-Corps Kick-Off Meeting (Houston, TX)

Program to explore commercialization opportunities for research technology

Participating with Agata Rozga (Res. Scientist), Yin Li (PhD student), and Ernesto Escobar (Mentor)

$50K to conduct interviews and purse customer discovery

Where Was Prof. Rehg on Mon and Wed?

Dagstuhl Seminar 16042

Eyewear Computing – Augmenting the Human with Head-Mounted Wearable Assistants

Well-known international forum for exchange of research ideas

I was co-organizer with researchers from Germany, Japan, and Google (CA)

Focus on “wearable cameras meet HCI”

A* Search – Additional Details

Identify the relationship between

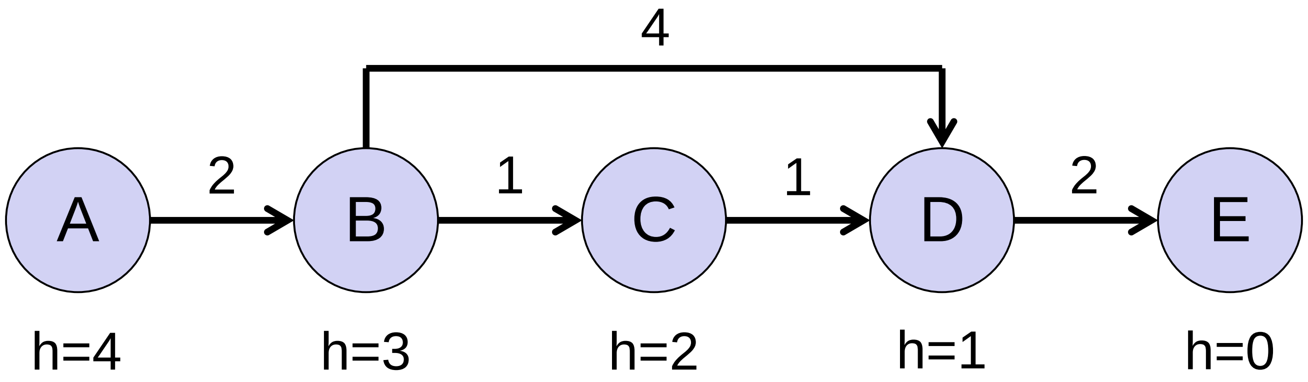
- A* Search
- Greedy Best-First Search
- Uniform Cost Search

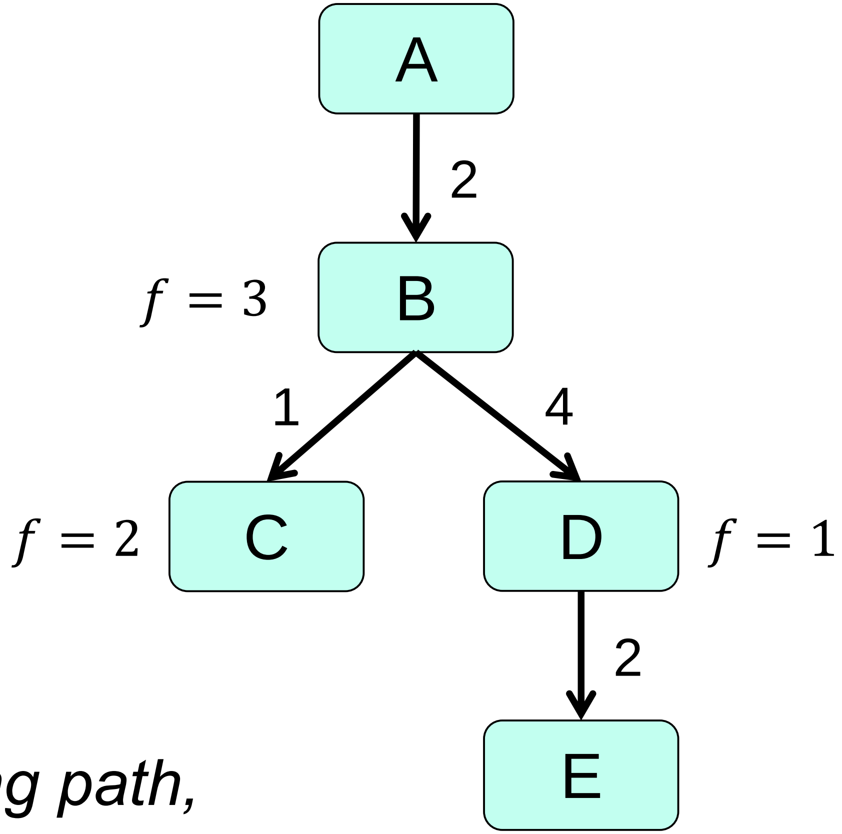
A* Search – Additional Details

Identify the relationship between

- A* Search – Optimal heuristic search
- Greedy Best-First Search – Greedy search with “wrong” cost
- Uniform Cost Search – Basic cost-based search method, basis for A* when using optimal heuristic

Problem with Best-First Greedy Search

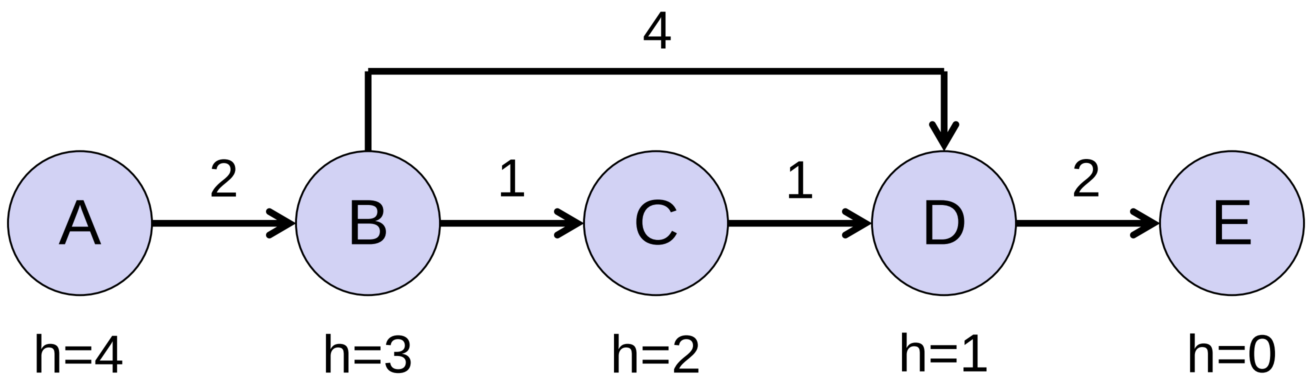


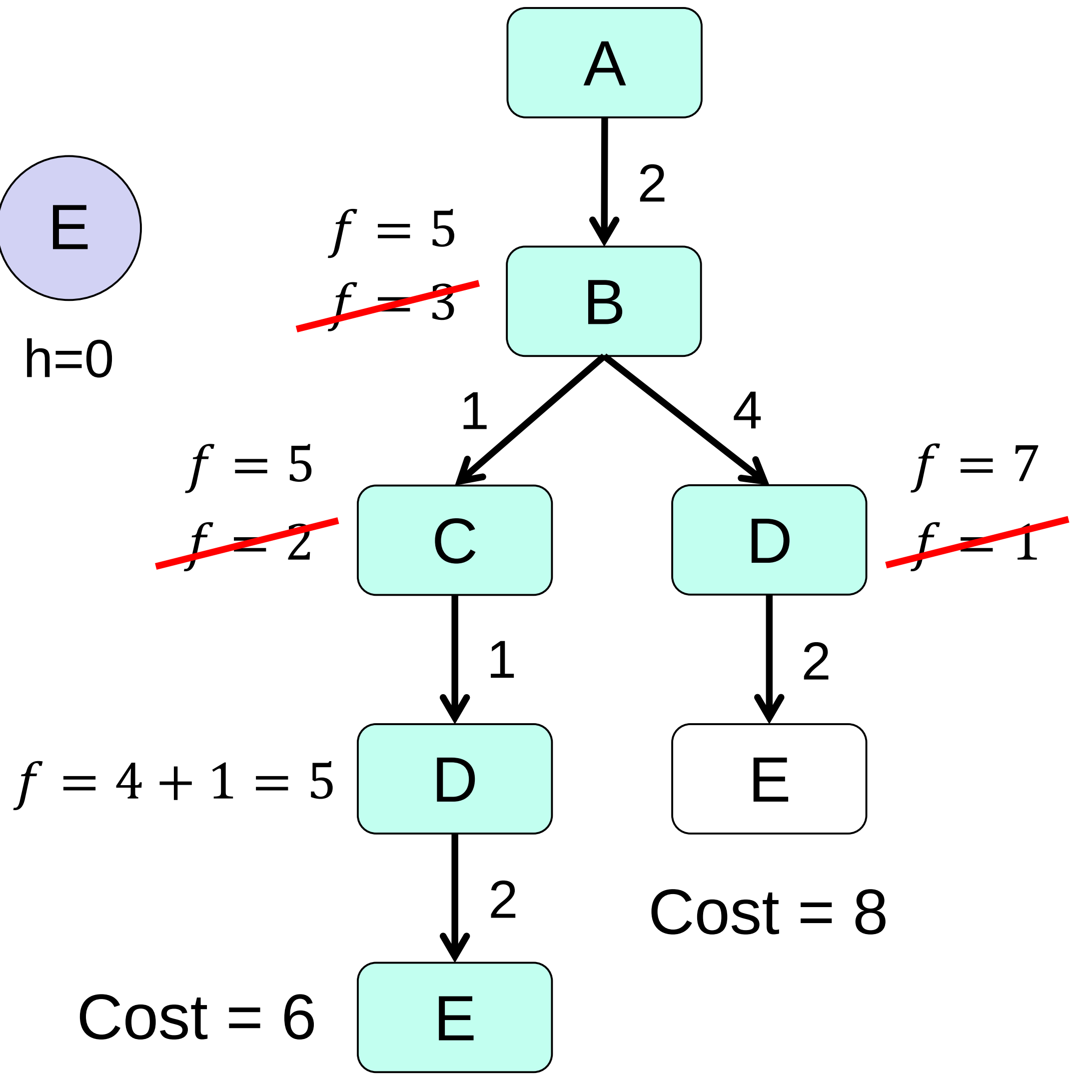


Cost = 8

Best-First doesn't keep track of cost along path, and takes a high cost arc to a low cost state

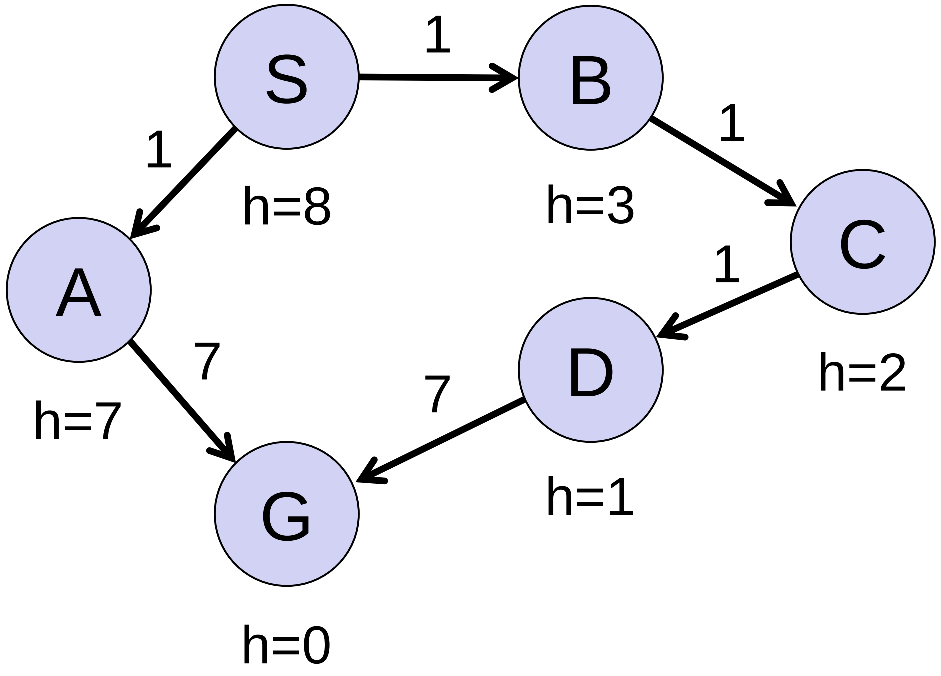
Problem with Best-First Greedy Search

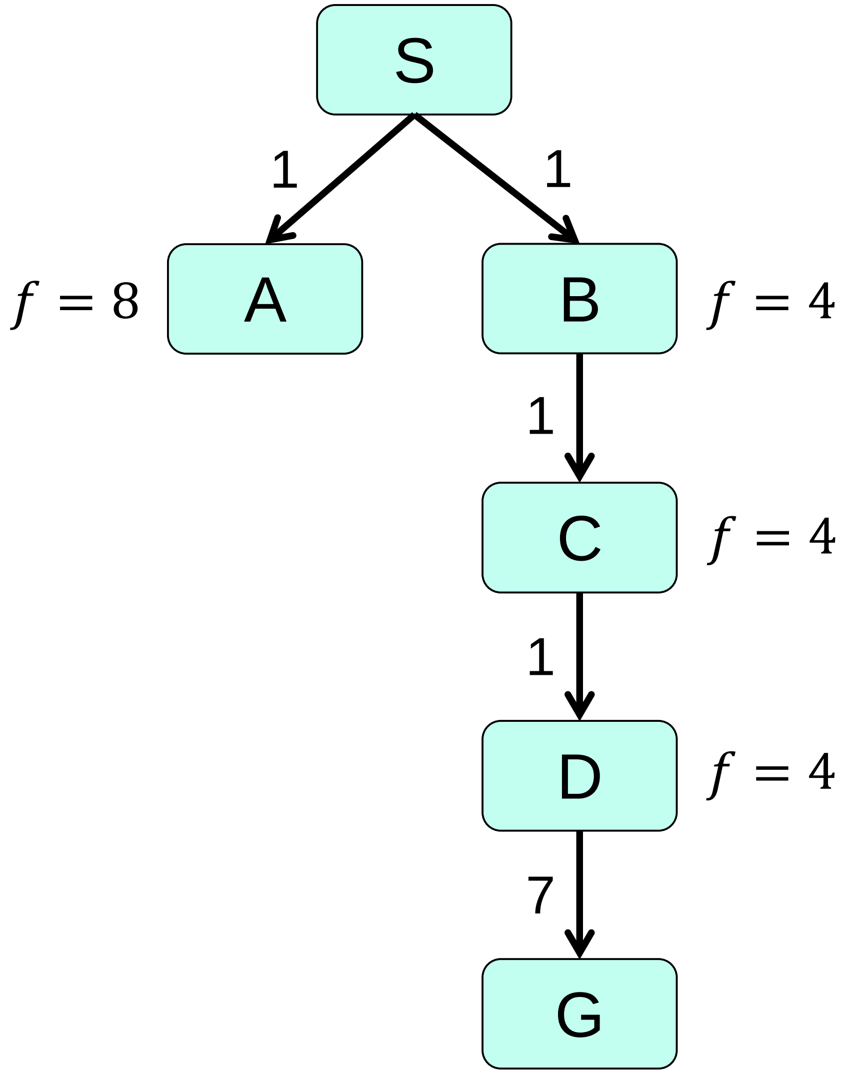




A uses path cost + heuristic, correctly chooses the lower cost path*

When Should A* Terminate?



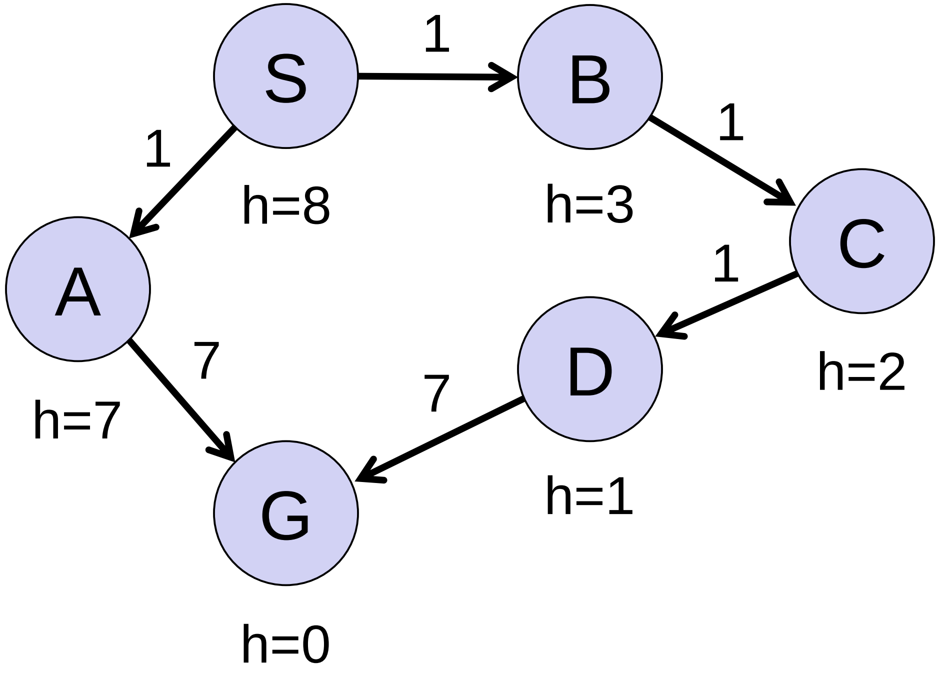


Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)

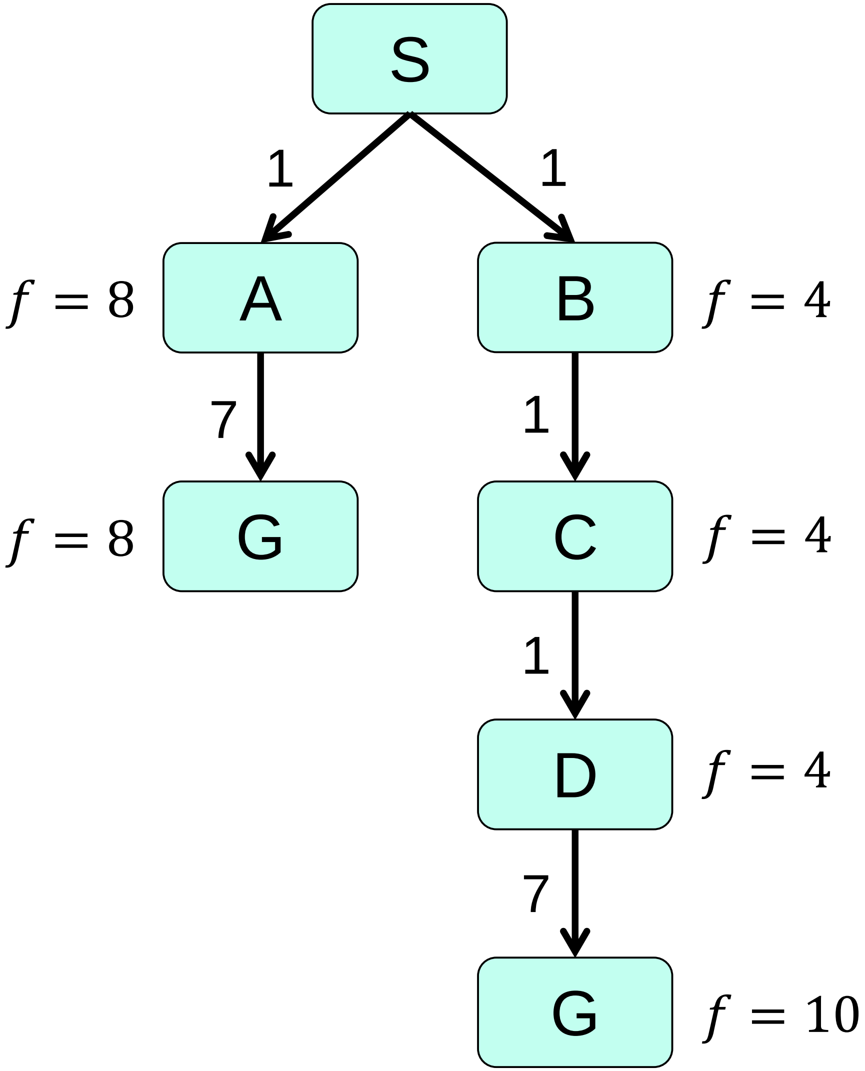
Expanding D gives goal node, but terminating at this point gives suboptimal solution

When Should A* Terminate?



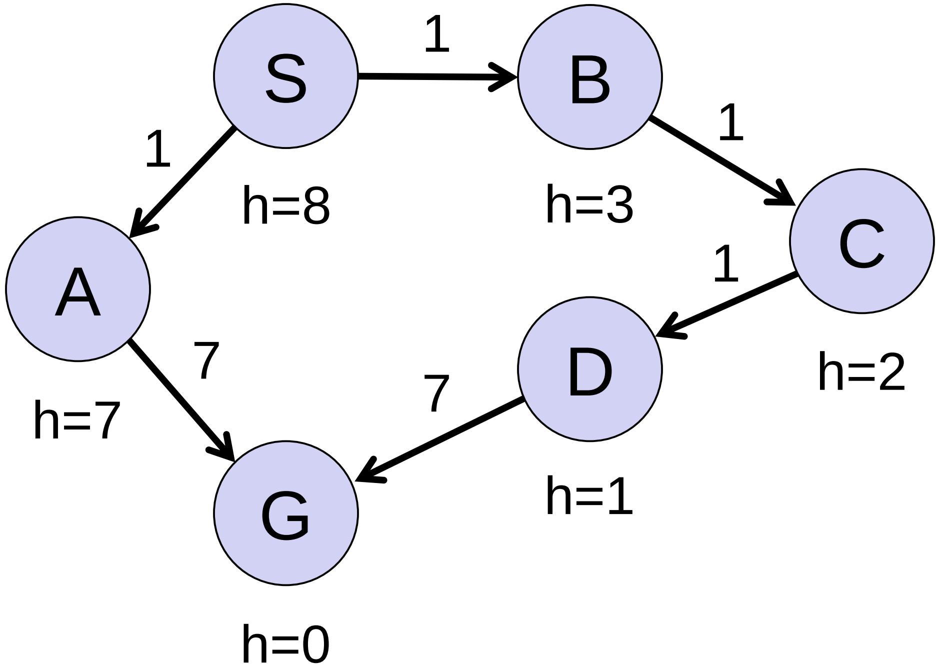
Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) G(8) ~~G(10)~~



Push goal node on queue, expanding A leads to optimal path to goal

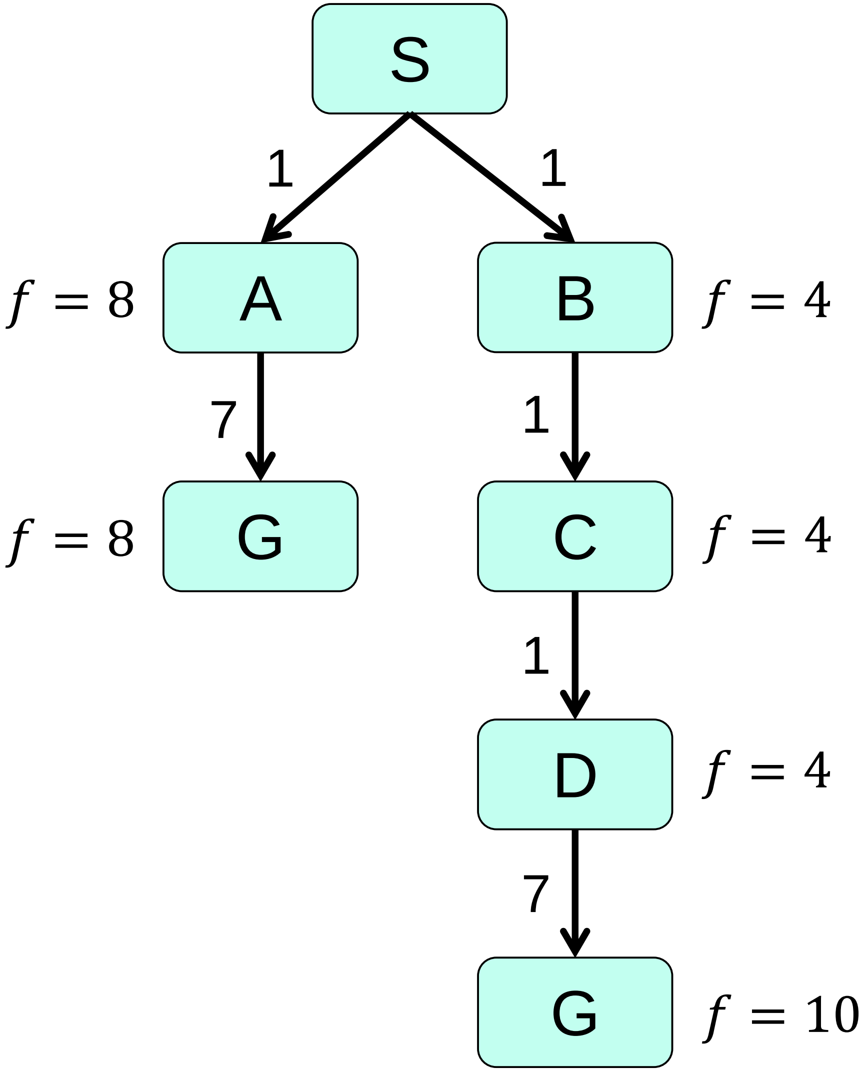
When Should A* Terminate?



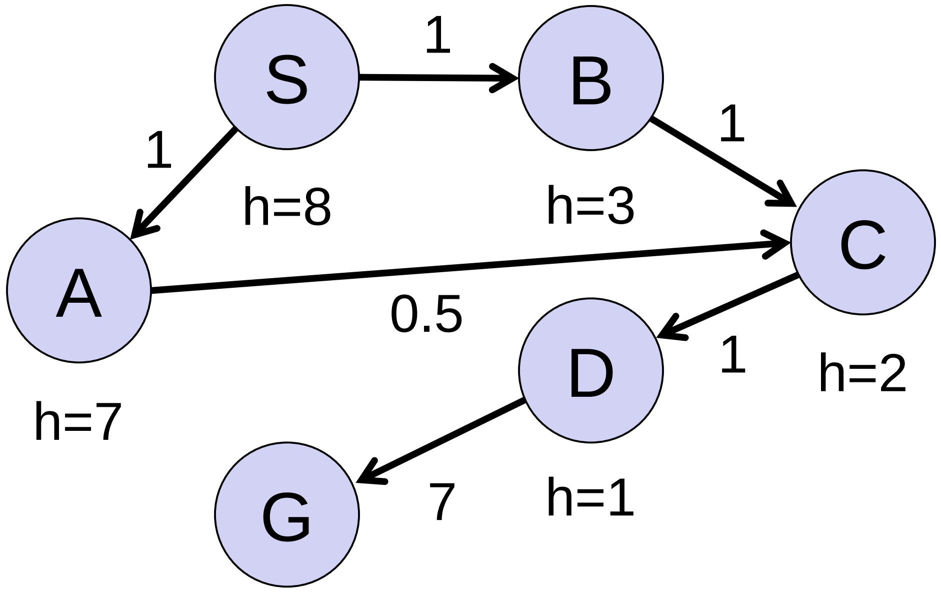
Answer: Terminate when goal node is **popped** from queue

<u>*Frontier (Queue)*</u>

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) G(8)

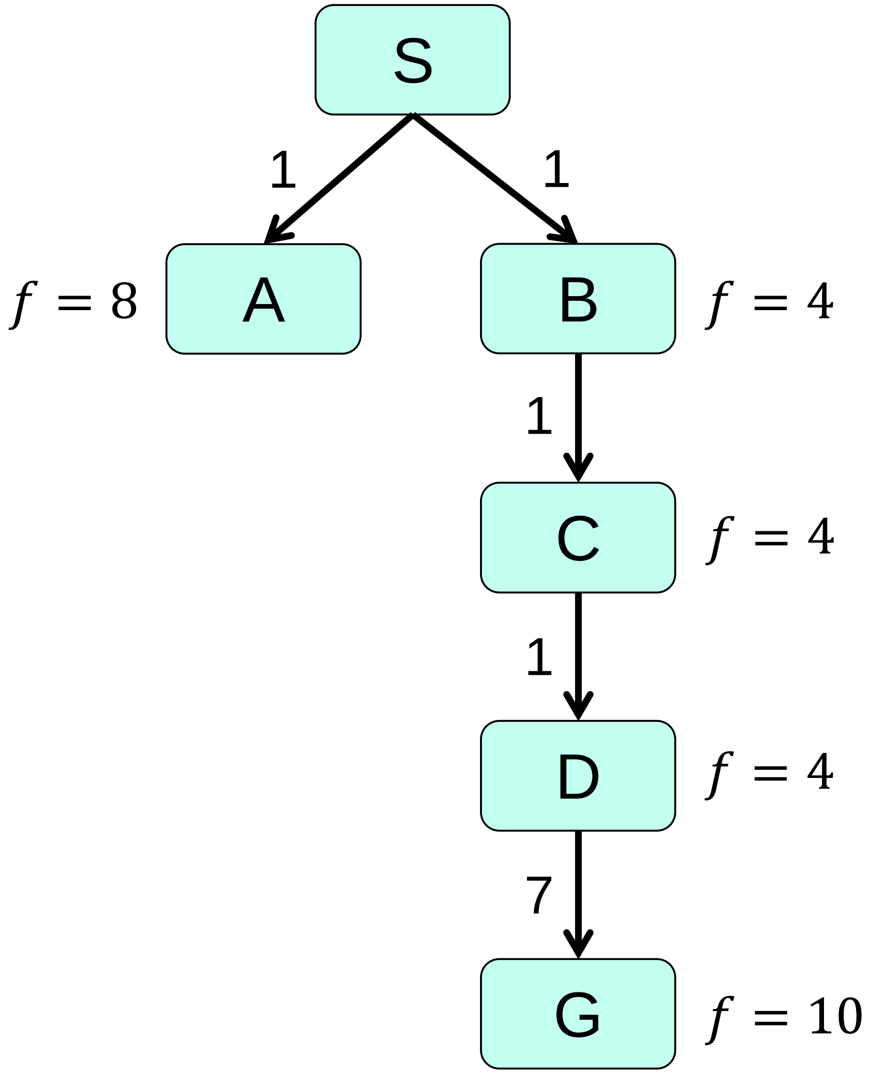


When Does A* Revisit States?

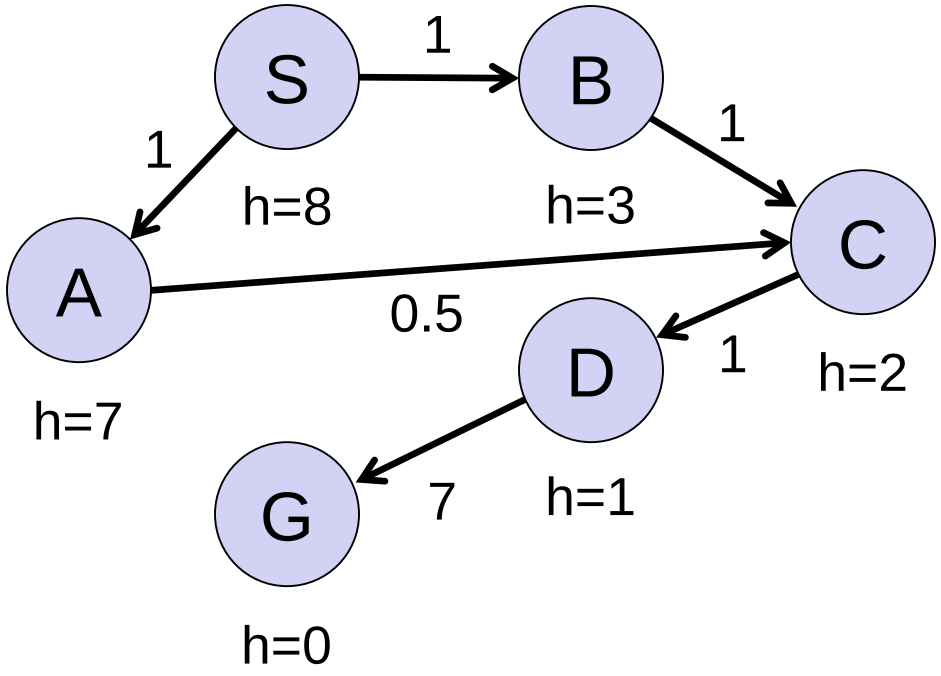


Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)



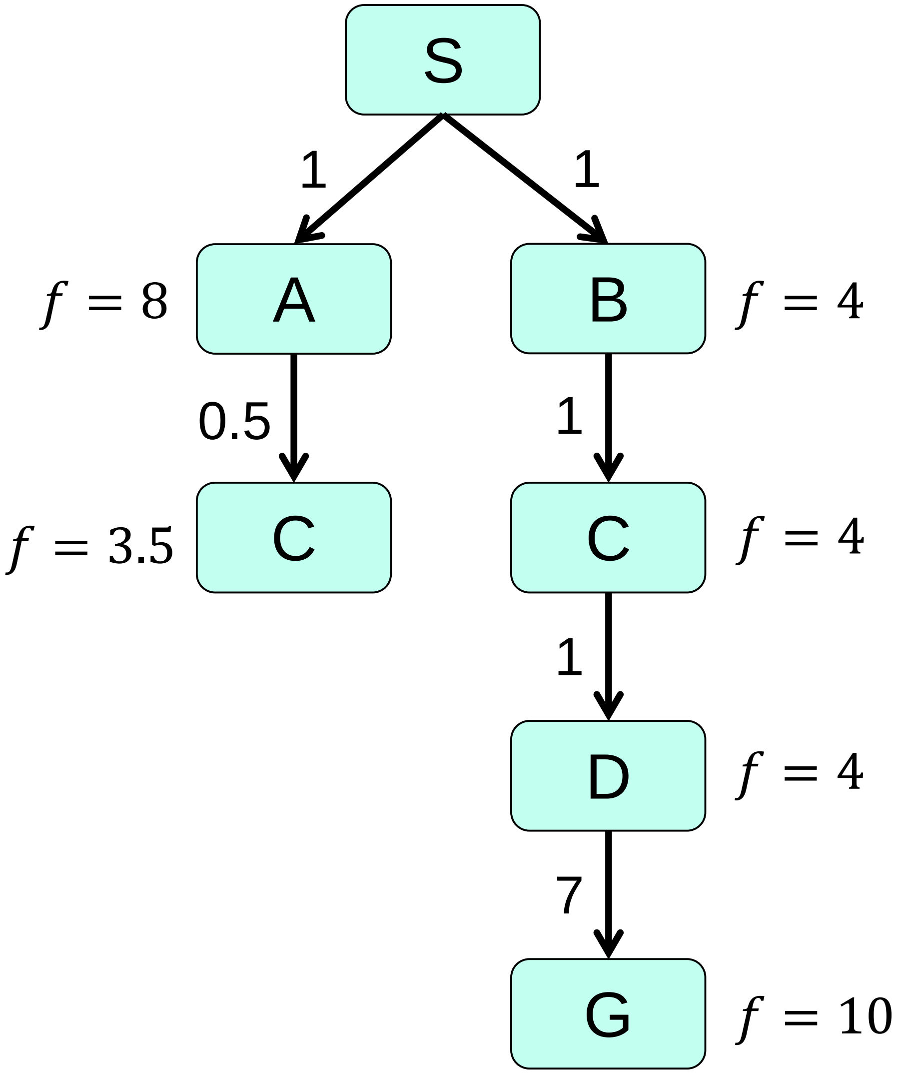
When Does A* Revisit States?



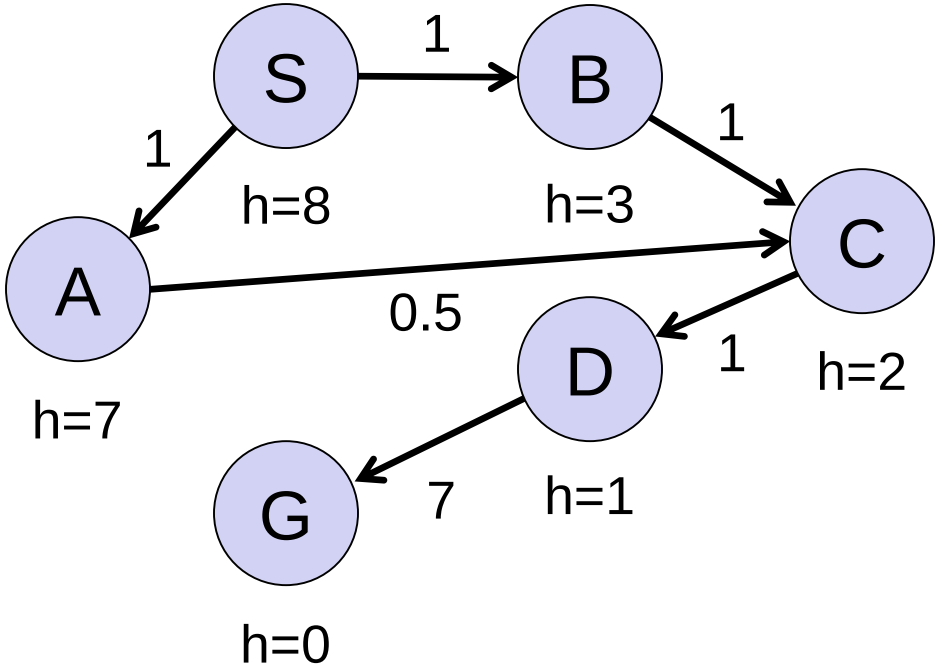
C visited on path S-B-C, added to visited list, and then re-visited on S-A-C at lower cost

Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) C(3.5) G(10)



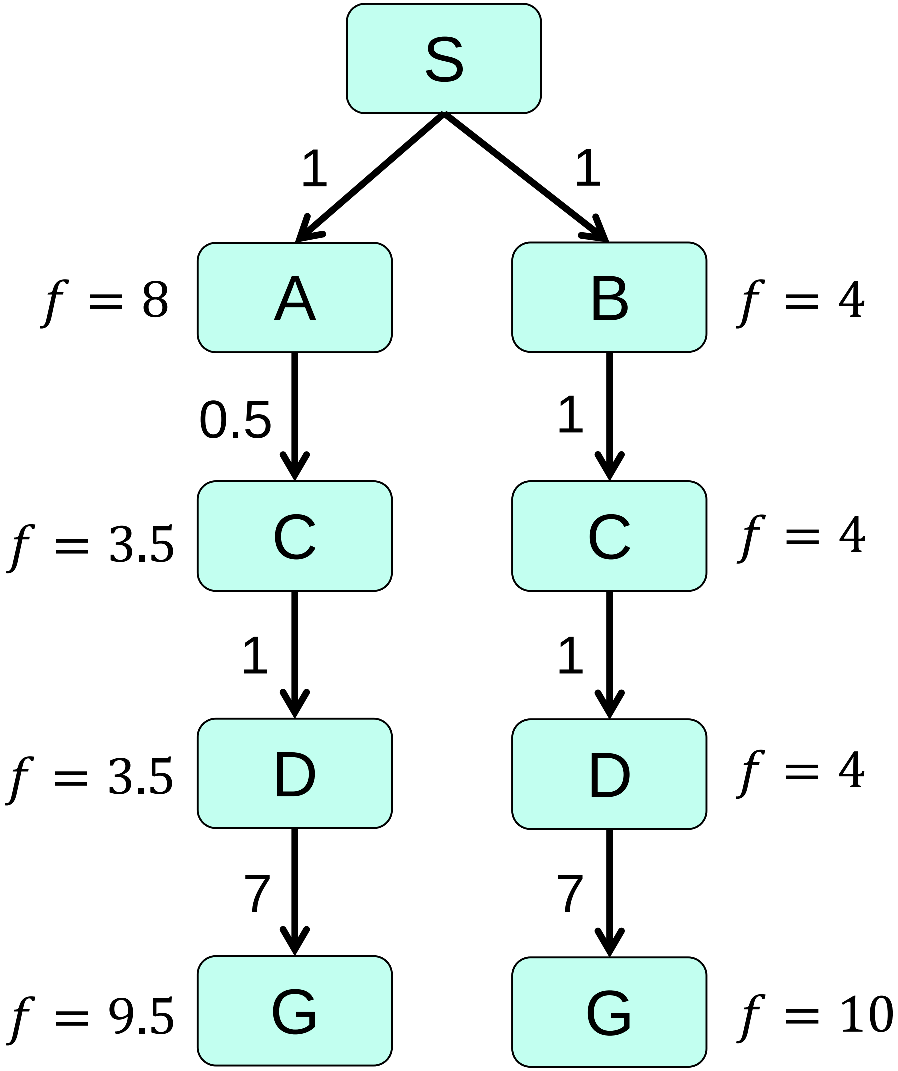
When Does A* Revisit States?



States on closed list can be revisited at a lower cost

Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) C(3.5) G(10)
7) D(3.5) G(10)
8) G(9.5)



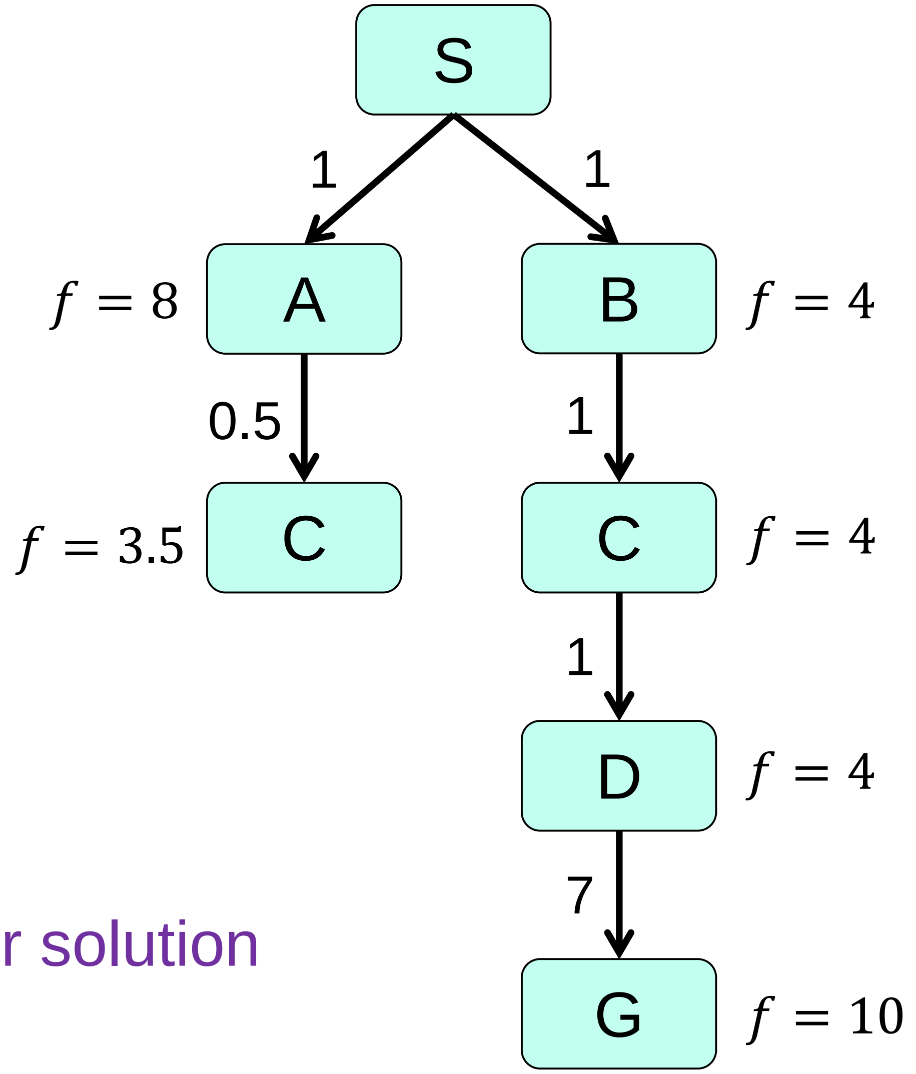
Visited List

Visited (Closed List)

1) (S, 8, null)
2) (B, 4, S)
3) (C, 4, B)
4) (D, 4, C)
5) (A, 8, S)

Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) C(3.5) G(10)



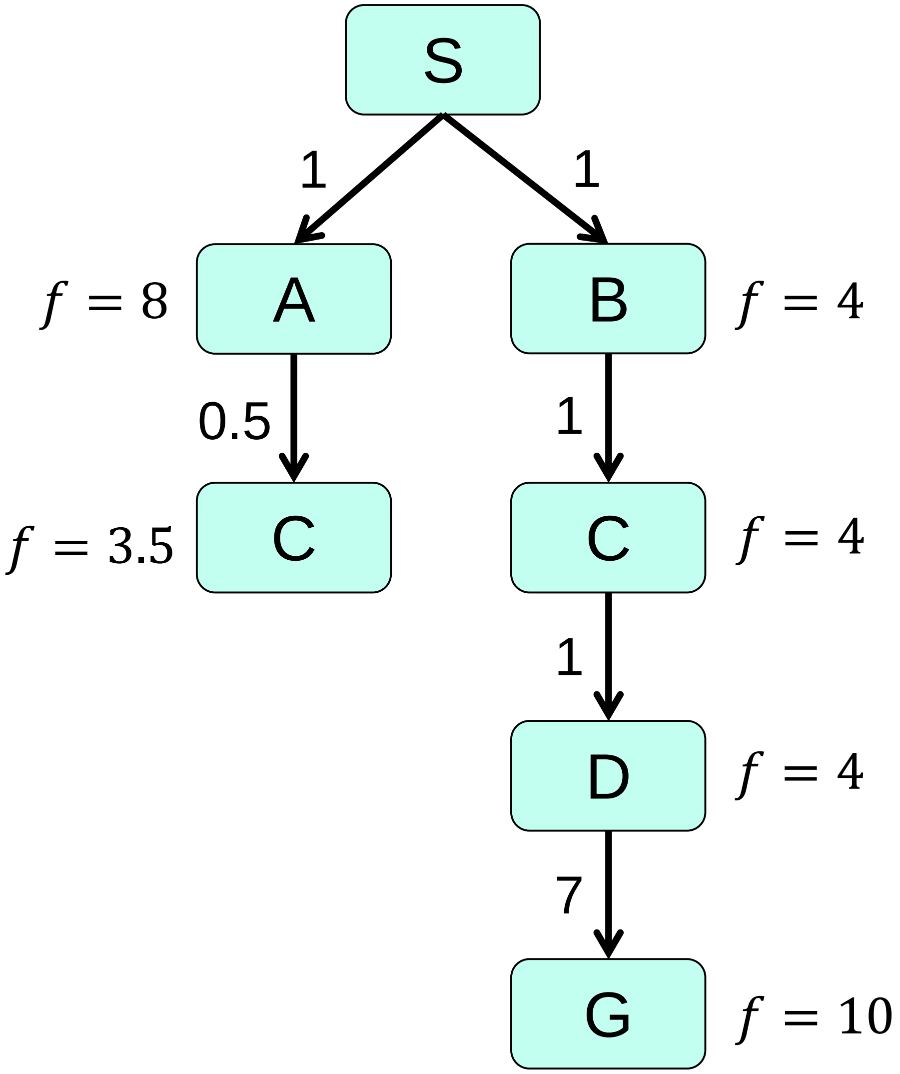
Maintain visited list for expanded nodes:
(state, cost, *backpointer*)

Will allow us to recover solution once we reach goal

Visited List

Visited (Closed List)
1) (S, 8, null)
2) (B, 4, S)
3) ~~(C, 4, B)~~
4) (D, 4, C)
5) (A, 8, S)
6) (C, 3.5, A)

Frontier (Queue)
1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) C(3.5) G(10)



States that are revisited with lower cost replace previously-added nodes

Maintain visited list, add expanded nodes:
(state, cost, *backpointer*)

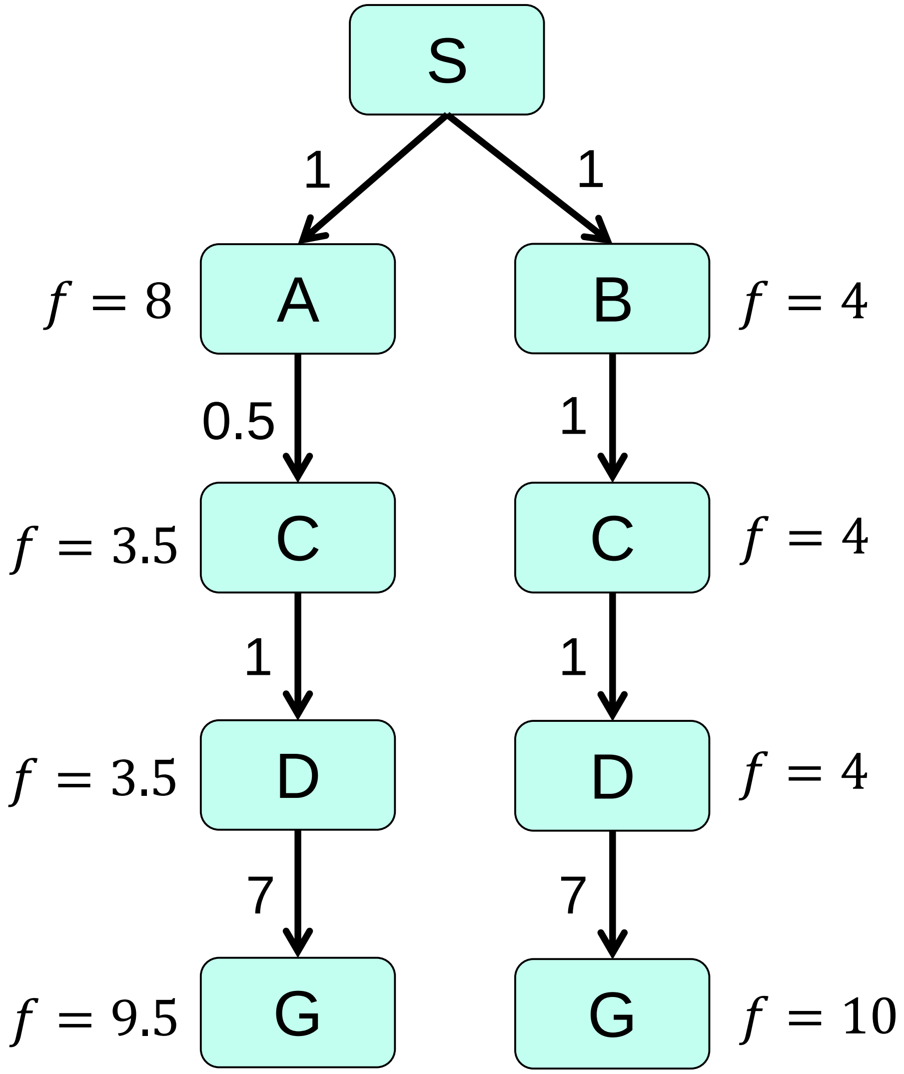
Visited List

Visited (Closed List)

1) (S, 8, null)
2) (B, 4, S)
3) ~~(C, 4, B)~~
4) ~~(D, 4, C)~~
5) (A, 8, S)
6) (C, 3.5, A)
7) (D, 3.5, C)
8) (G, 9.5, D)

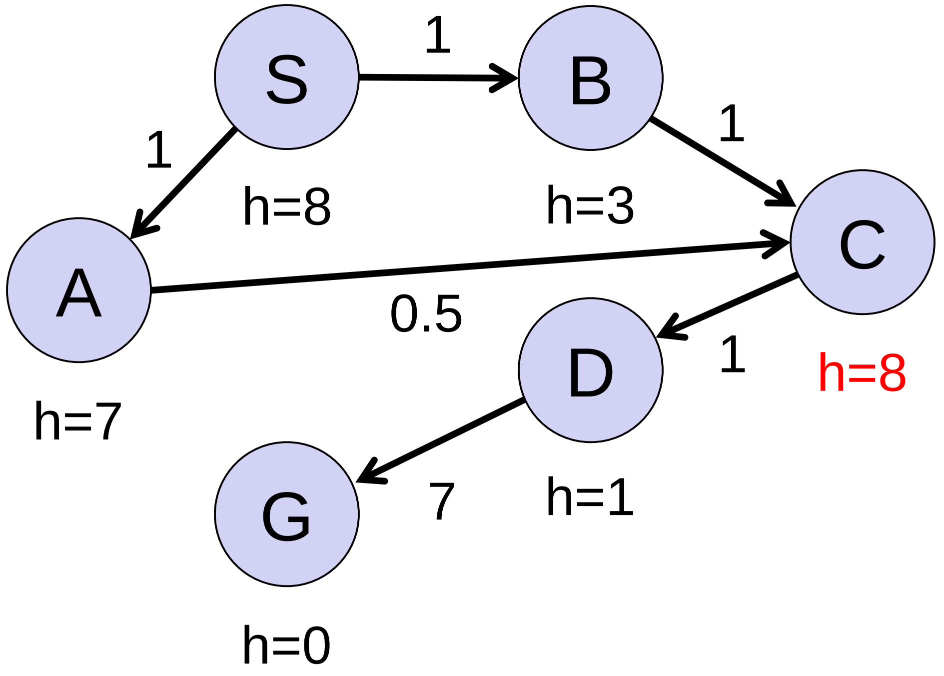
Frontier (Queue)

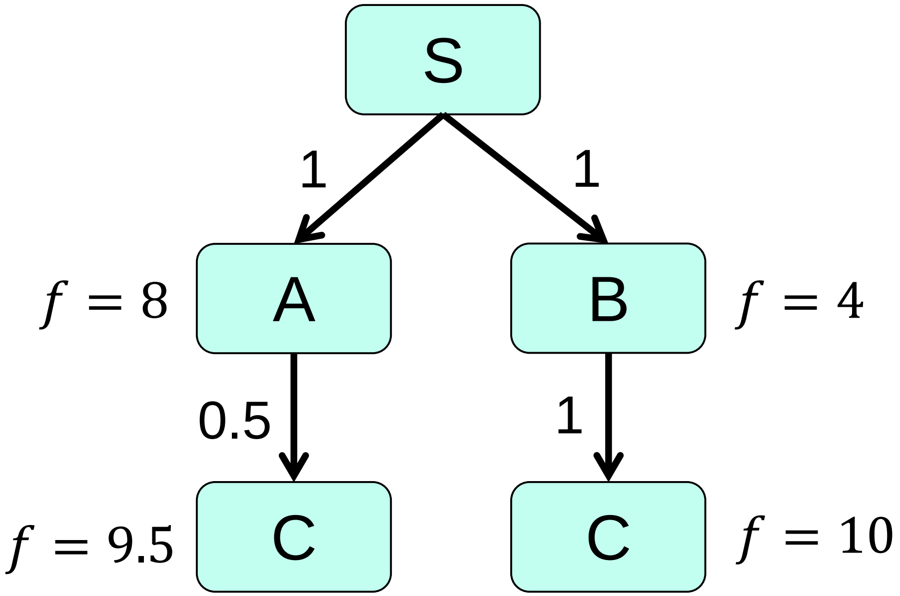
1) S(8)
2) B(4) A(8)
3) C(4) A(8)
4) D(4) A(8)
5) A(8) G(10)
6) C(3.5) G(10)
7) D(3.5) G(10)
8) G(9.5)



Backchain to retrieve state sequence:
G, D, C, A, S
and reverse it to obtain action plan

When Does A* Revisit States?





Frontier (Queue)

1) S(8)
2) B(4) A(8)
3) A(8) C(10)
4) C(9.5) ~~C(10)~~

S-A-C (9.5) vs. S-B-C(10)

With change in heuristic, state C is promoted with a lower cost after being added to queue initially

To avoid confusion, you can always write the path explicitly when adding node to queue

Pseudocode for A* Search

function ASTAR-SEARCH(*problem*) **returns** *solution*

  Initialize priority queue (*Q*) and visited list (*V*) with starting state

  while not empty *Q* **do**

    pop node n with lowest $f(n)$ from *Q*

    if n is goal node **then return** BACKCHAIN(*V*)

      foreach node s **in** *problem.*SUCCESSORS(n) **do**

        $f' = g(n) + \text{cost}(n, s) + h(s)$

        if (s not already in *V*) OR $f(s) > f'$ **then**

          Add s to *Q* with cost f'

          Update *V* with (s, f', n)

  return *failure*

8-Puzzle Example

Which of the following are admissible heuristics?

1) h(n) = Number of tiles in wrong position in state n

2) h(n) = 0

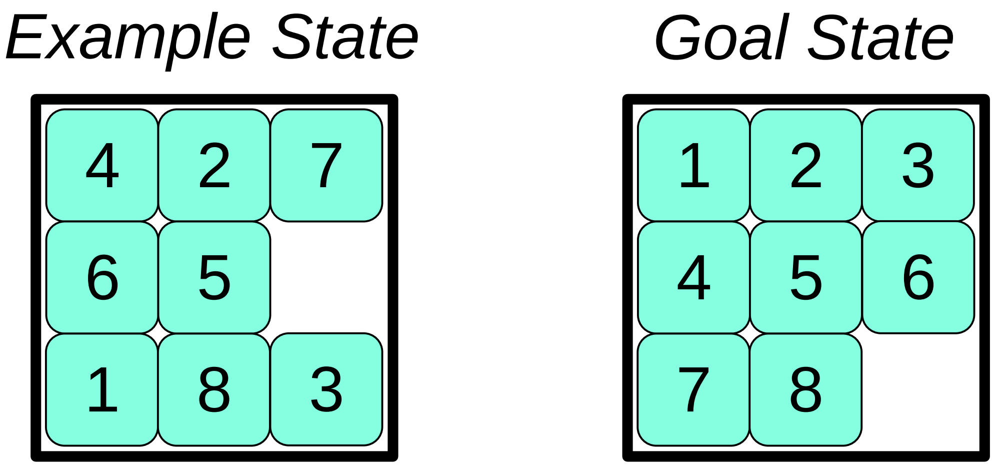
3) h(n) = Sum of Manhattan distances between each tile and its goal location

4) h(n) = 1

5) h(n) = min(2, h*(n))

6) h(n) = h*(n)

7) h(n) = max(2, h*(n))



8-Puzzle Example

Which of the following are admissible heuristics?

1) h(n) = Number of tiles in wrong position in state n YES

2) h(n) = 0 YES

3) h(n) = Sum of Manhattan distances between each tile and its goal location YES

4) h(n) = 1 NO

5) h(n) = min(2, h*(n)) YES

6) h(n) = h*(n) YES

7) h(n) = max(2, h*(n)) NO

Example State

4	2	7
6	5	
1	8	3

Goal State

1	2	3
4	5	6
7	8	

Dominance

Reminder: Heuristic $h(n)$ is *admissible* if $h(n) <= h^*(n)$

If $h1$ and $h2$ *admissible* heuristics, and

$h2(n) >= h1(n)$ for all n,

Then $h2$ *dominates* $h1$, and

$h2$ is better for search

Why?

Note: $h'(n) = \max\{ h1(n), h2(n) \}$ is admissible and dominates $h1$ and $h2$

Dominant Heuristic is Better

In A* every node n with

$f(n) < C^*$ will be expanded

$h(n) < C^* - g(n)$ will be expanded

$h2(n) > h1(n)$ for all n, then

Set of n for which $h1(n) < C^* - g(n)$

Will be *larger* than set of n for which $h2(n) < C^* - g(n)$

Thus $h1$ will expand more nodes

Questions?