CMSC 471
Artificial Intelligence

Search

KMA Solaiman – ksolaima@umbc.edu

Slide adoption courtesy: Tim Finin, Frank Ferraro (UMBC),
Charles R. Dyer (UW-Madison), Dan Klein and Pieter Abbeel (CS188, UC Berkeley)

A General Searching Algorithm

Core ideas:

1. Maintain a list of frontier (fringe) nodes
    1. Nodes coming *into* the frontier have been explored
    2. Nodes going out of the frontier have not been explored
2. Iteratively select nodes from the frontier and explore unexplored nodes from the frontier
3. Stop when you reach your **goal**

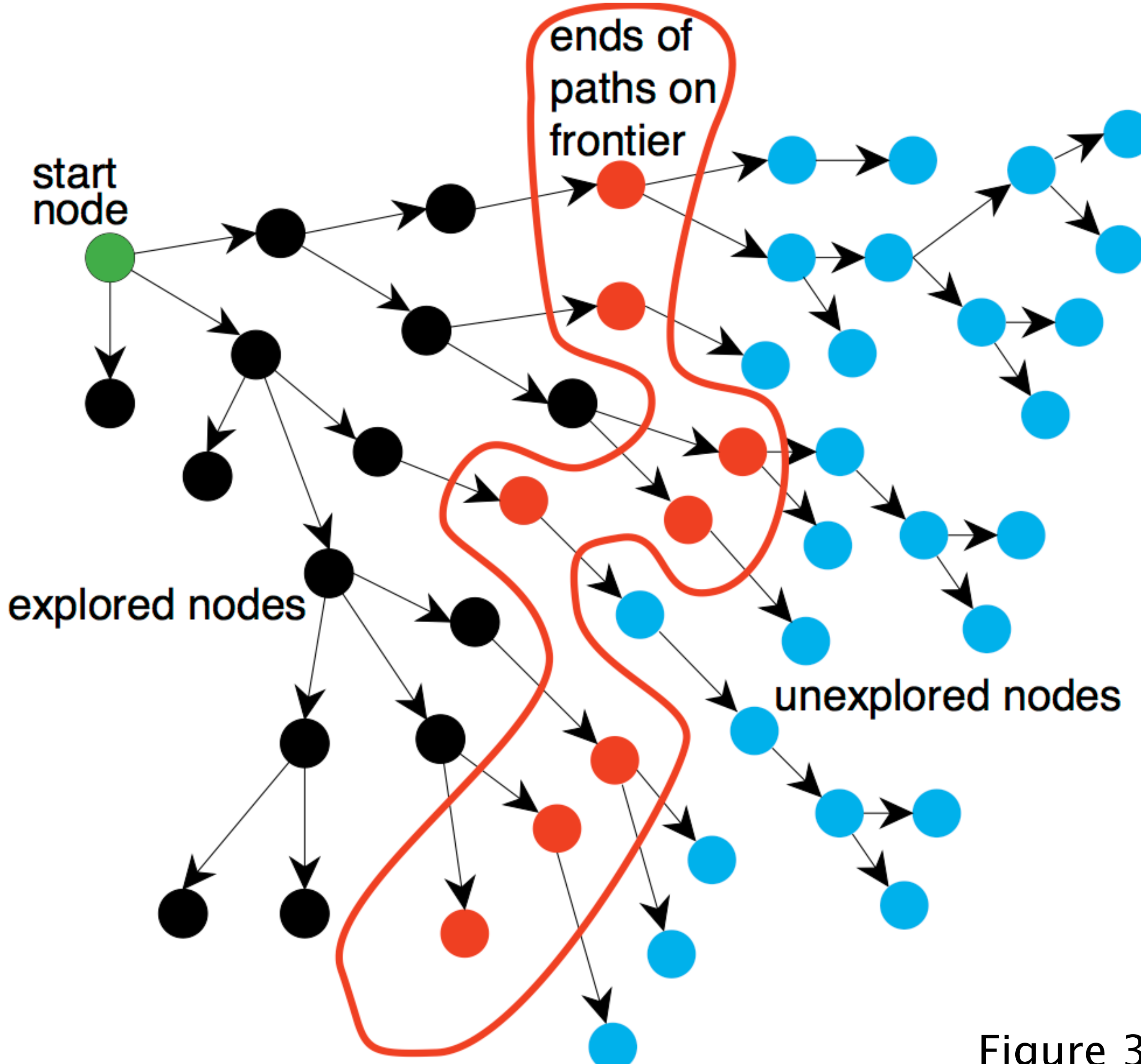


Figure 3.3

Uniform Cost Search

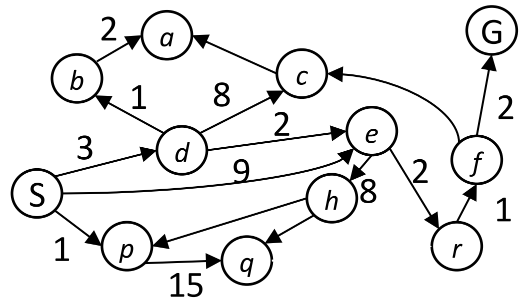
$f(n) = g(n)$

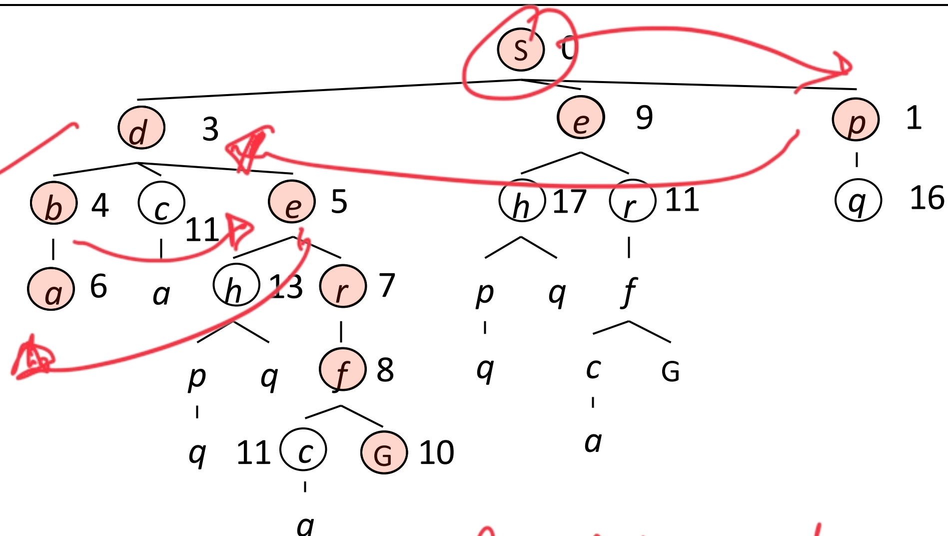
Best-first search uses $f(n)$ evaluation func.

g(n) = cost from root to n

Strategy: expand lowest g(n)

Frontier is a priority queue sorted by g(n)

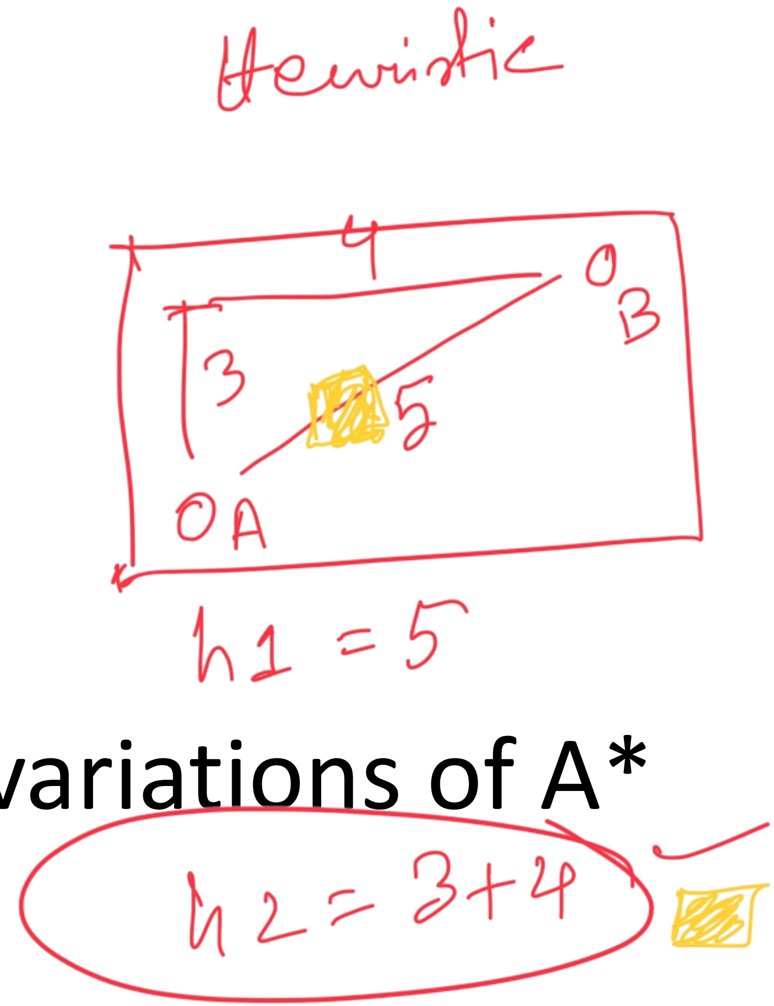




So not backtrack, moves around with least $g(n)$

Informed (Heuristic) Search

- Heuristic search
- Best-first search
  - Greedy search
  - Beam search
  - A* Search
- Memory-conserving variations of A*
- Heuristic functions



Best-first search

- Search algorithm that improves **depth-first search** by expanding most promising node chosen according to heuristic rule

- Order nodes on Fringe list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information

Best-first search

- Search algorithm that improves **depth-first search** by expanding most promising node chosen according to heuristic rule
- Order nodes on Fringe list by increasing value of an evaluation function, **f(n)**, incorporating domain-specific information
- This is a generic way of referring to the class of informed methods

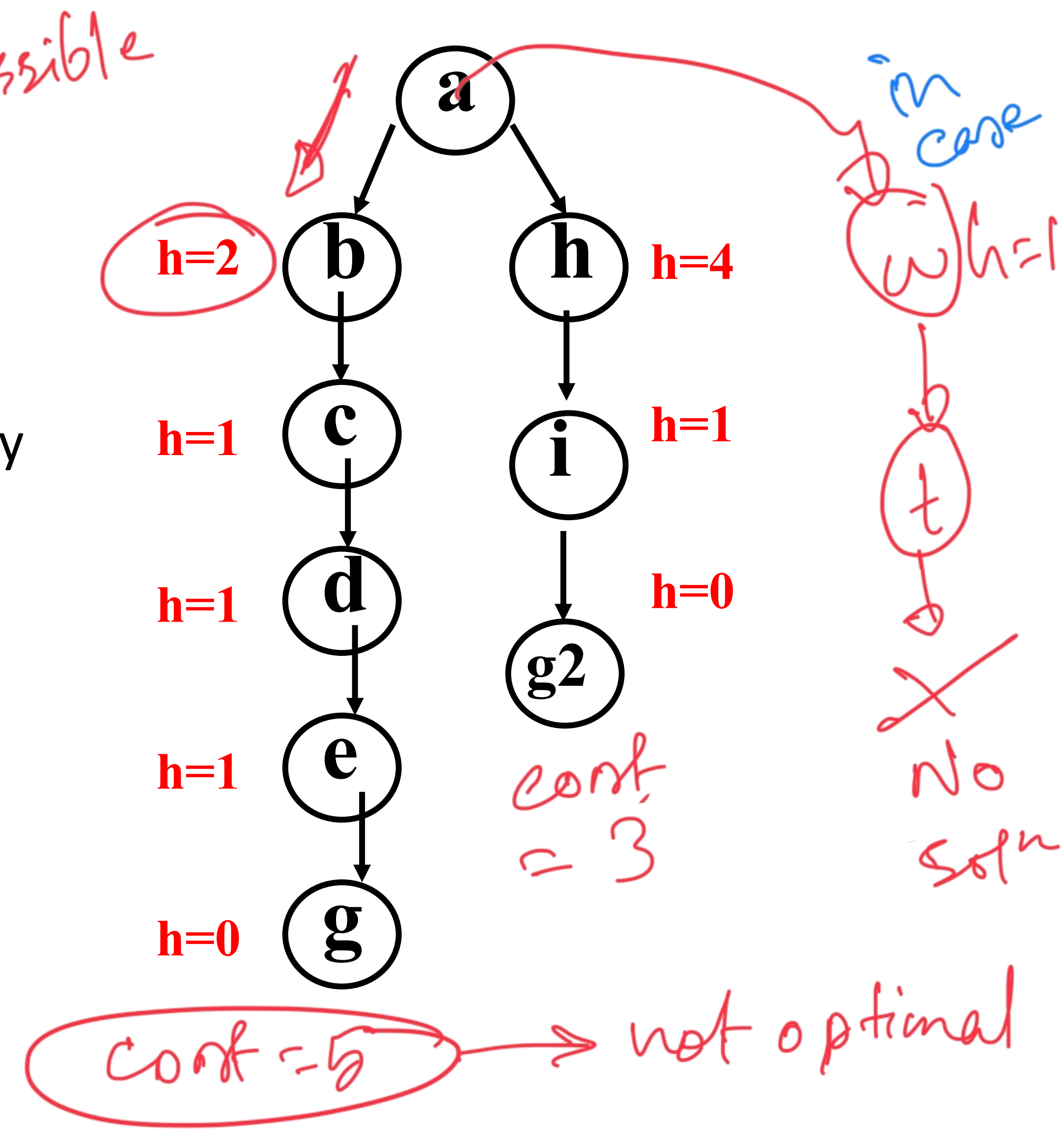
Greedy best first search

No Backtrack

- A greedy algorithm makes locally optimal choices in hope of finding a global optimum
- Uses evaluation function *f(n) = h(n)*, sorting nodes by increasing values of *f*
- Selects node to expand appearing **closest** to goal (i.e., node with smallest f value)
- Not complete → in case ends up in path w/ no soln
- Not admissible, as in example
  - Assume arc costs = 1, greedy search finds goal g, with solution cost of 5
  - Optimal solution is path to goal with cost 3

Greedy best first search example

- Proof of non-admissibility
  - Assume arc costs = 1, greedy search finds goal g, with solution cost of 5
  - Optimal solution is path to goal with cost 3



Greedy best first search example

- Makes locally optimal choices at each step based on the current information and do not reconsider past decisions.
- Once a greedy algorithm makes a choice and moves to the next step, it does not go back to reconsider or explore alternative paths. In some cases, **they can get stuck in local optima or suboptimal solutions.**
- If fails to find a path to the goal, then the chosen path based on the heuristic did not lead to a solution. In such cases, the algorithm may terminate without finding a solution or may need to be modified to explore alternative paths, possibly incorporating backtracking, to improve its search capabilities.

Beam search

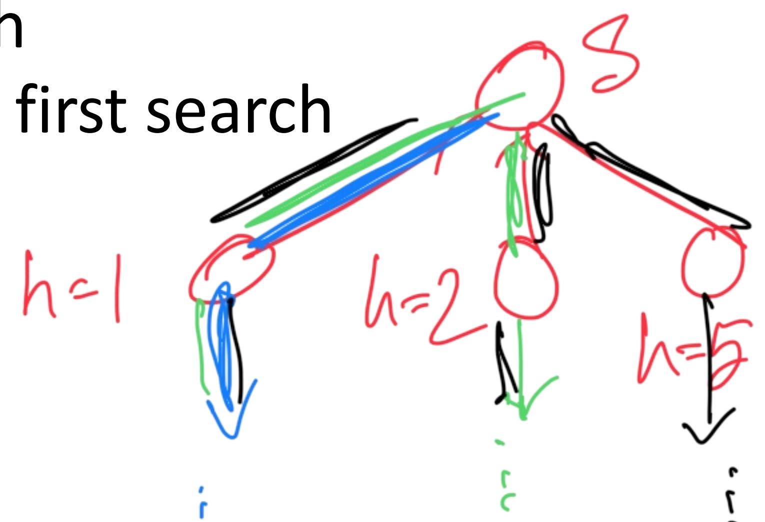
- Instead of picking one child per iteration, it expands k number of children**, in parallel.**
- Use evaluation function f(n), but maximum size of the nodes list is k, a fixed constant
- Only keep k best nodes as candidates for expansion, discard rest
- k is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As k increases, approaches best first search
- Complete?
- Admissible?

K → size of beam

K = 2

K = 1

K = 3



Beam search

- Instead of picking one child per iteration, it expands k number of children**, in parallel.**
- Use evaluation function f(n), but maximum size of the nodes list is k, a fixed constant
- Only keep k best nodes as candidates for expansion, discard rest
- k is the *beam width*
- More space efficient than greedy search, but may discard nodes on a solution path
- As k increases, approaches best first search
- Not complete
- Not admissible

We’ve *got* to be able to do better, right?

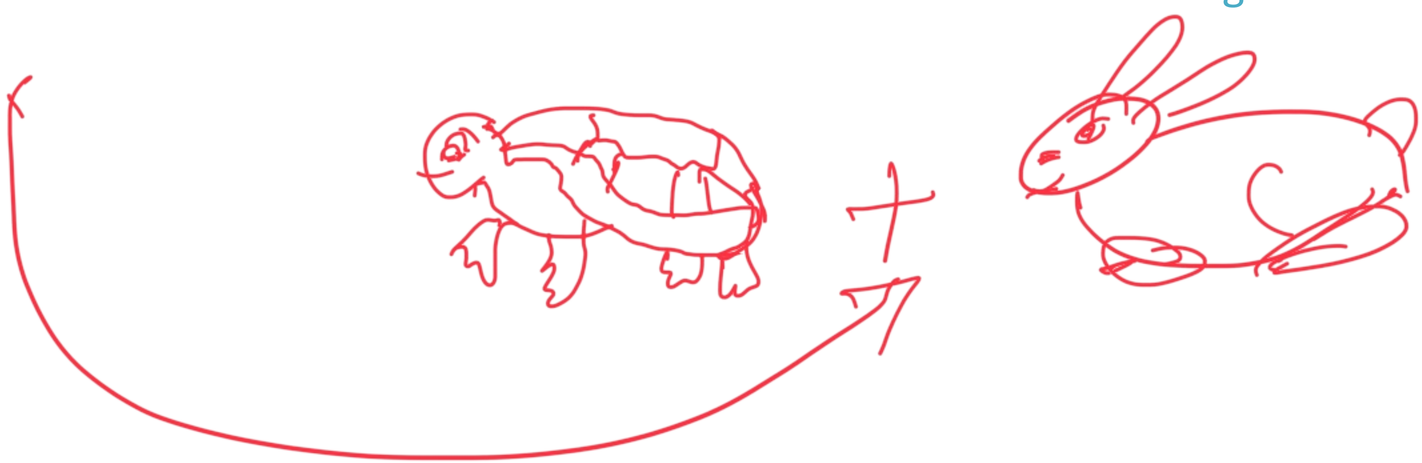
Let’s think about car trips…

A* Search

Use an evaluation function

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

estimated total cost from start to goal via state n = minimal-cost path from the start state to state n + cost estimate from state n to the goal

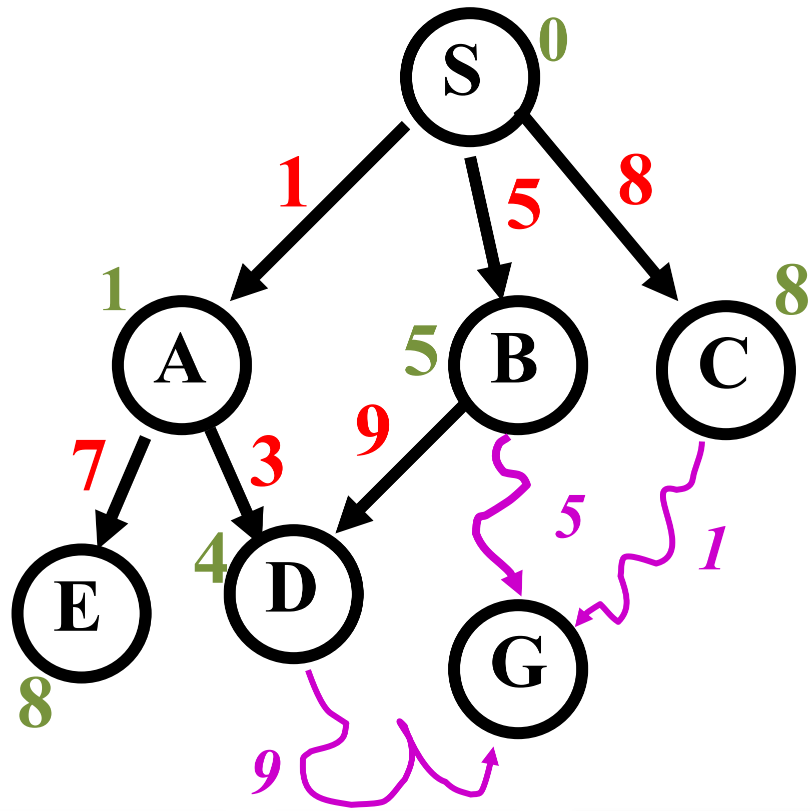


A* Search

- Use as an evaluation function

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

- g(n) = minimal-cost path from the start state to state n
- Ranks nodes on search frontier by estimated cost of solution from start node ***via given node*** to goal
- Combining UCS and Greedy



g(d)=4
h(d)=9
f(d)=13

g(b)=5
h(b)=5
f(d)=10

g(c)=8
h(c)=1
f(c)=9

C is chosen next to expand

A* Pseudo-code

1 Put the start node S on the nodes list, called OPEN

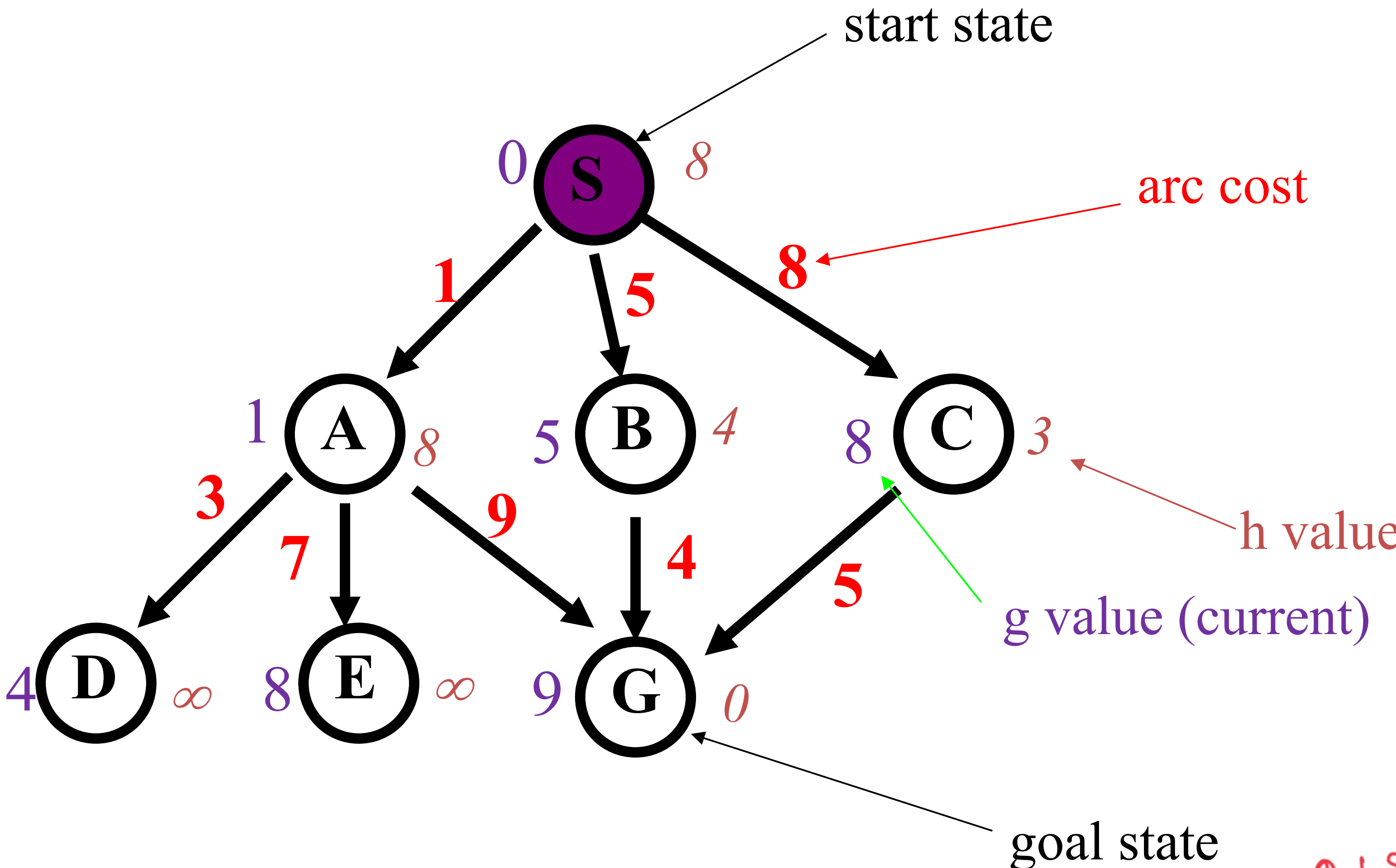
2 If OPEN is empty, exit with failure

3 Select node in OPEN with minimal f(n) and place on CLOSED

4 If n is a goal node, collect path back to start and stop

5 Expand n, generating all its successors and attach to them pointers back to n. For each successor n' of n

- **1** If n' not already on OPEN or CLOSED
  - put n' on OPEN
  - compute h(n'), g(n')=g(n)+ c(n, n'), f(n')=g(n')+h(n')
- **2** If n' already on OPEN or CLOSED and if g(n') is lower for new version of n', then:
  - Redirect pointers backward from n' on path with lower g(n')
  - Put n' on OPEN



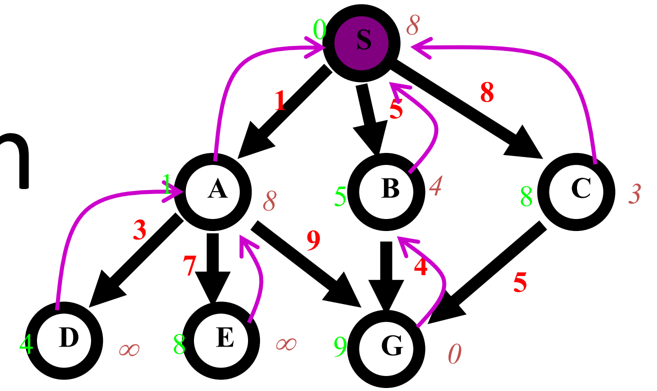
start state
0
S
8
arc cost
1
5
8
1
A
8
5
B
4
8
C
3
h value
3
7
9
4
5
g value (current)
4
D
∞
8
E
∞
9
G
0
goal state

GREEDY VS A*

8↓
S — C³ — G⁰ (8+5=13)

S⁸
A⁹
B⁹
S 0+8
A 1+8
B 5+4
C 8+3
B⁹
G (1+9)+0
C¹¹ D∞ E∞

Greedy search



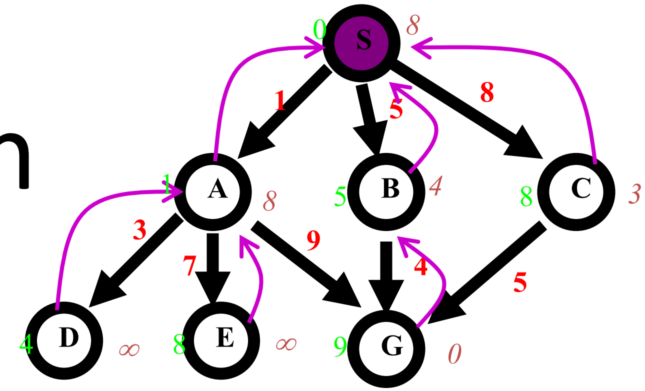
f(n) = h(n)

node expanded **nodes list**

{ S(8) }

what's next???

Greedy search

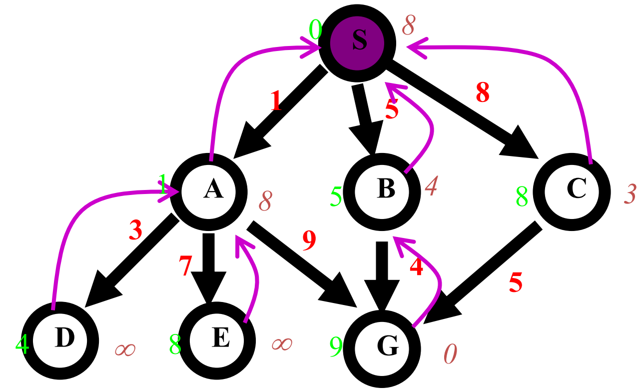


f(n) = h(n)

node expanded	**nodes list**
	{ S(8) }
S	{ C(3) B(4) A(8) }
C	{ G(0) B(4) A(8) }
G	{ B(4) A(8) }

- Solution path found is S C G, 3 nodes expanded.
- See how fast the search is!! But it is NOT optimal.

A* search



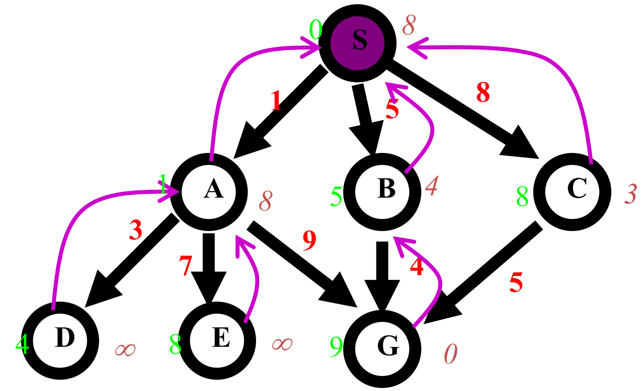
f(n) = g(n) + h(n)

node exp. nodes list

{ S(8) }

What's next?

A* search



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

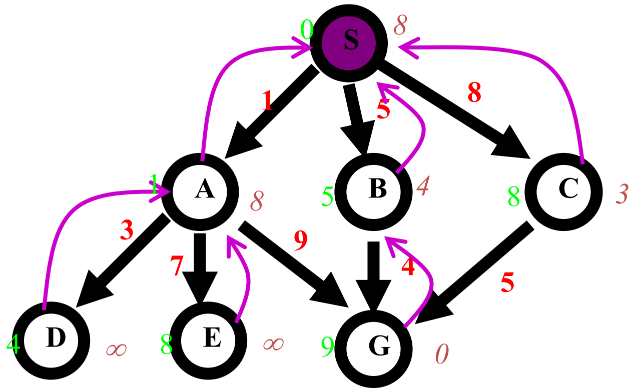
node exp.	nodes list
	{ S(8) }
S	{ A(9) B(9) C(11) }

What's next?

h(n)

h(S)=8
h(A)=8
h(B)=4
h(C)=3
h(D)=inf
h(E)=inf
h(G)=0

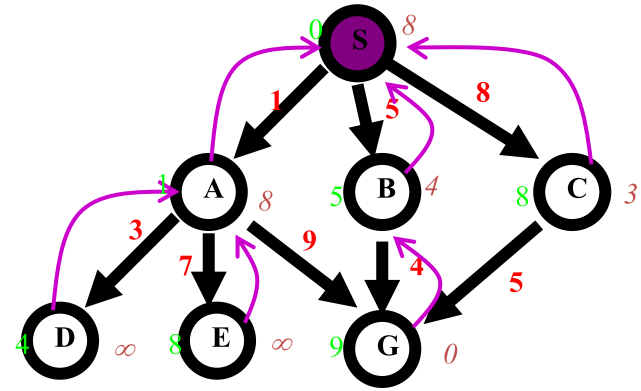
A* search



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

node exp.	nodes list
	{ S(8) }
S	{ A(9) B(9) C(11) }
A	{ B(9) G(10) C(11) D(inf) E(inf) }

What's next?

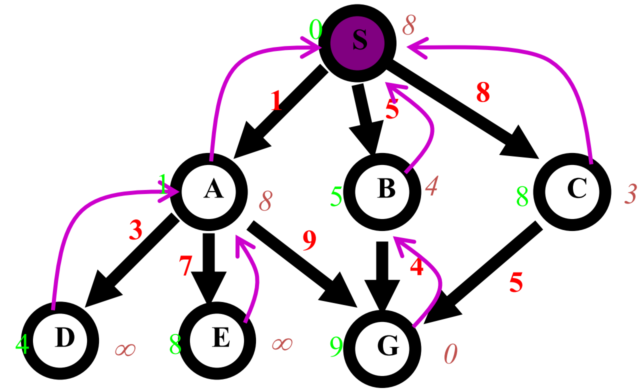


A* search

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

```
node exp.      nodes list
               { S(8) }
 S             { A(9) B(9) C(11) }
 A             { B(9) G(10) C(11) D(inf) E(inf) }
 B             { G(9) G(10) C(11) D(inf) E(inf) }
                      What's next?
```

A* search



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

node exp.	nodes list
	{ S(8) }
S	{ A(9) B(9) C(11) }
A	{ B(9) G(10) C(11) D(inf) E(inf) }
B	{ G(9) G(10) C(11) D(inf) E(inf) }
G	{ C(11) D(inf) E(inf) }

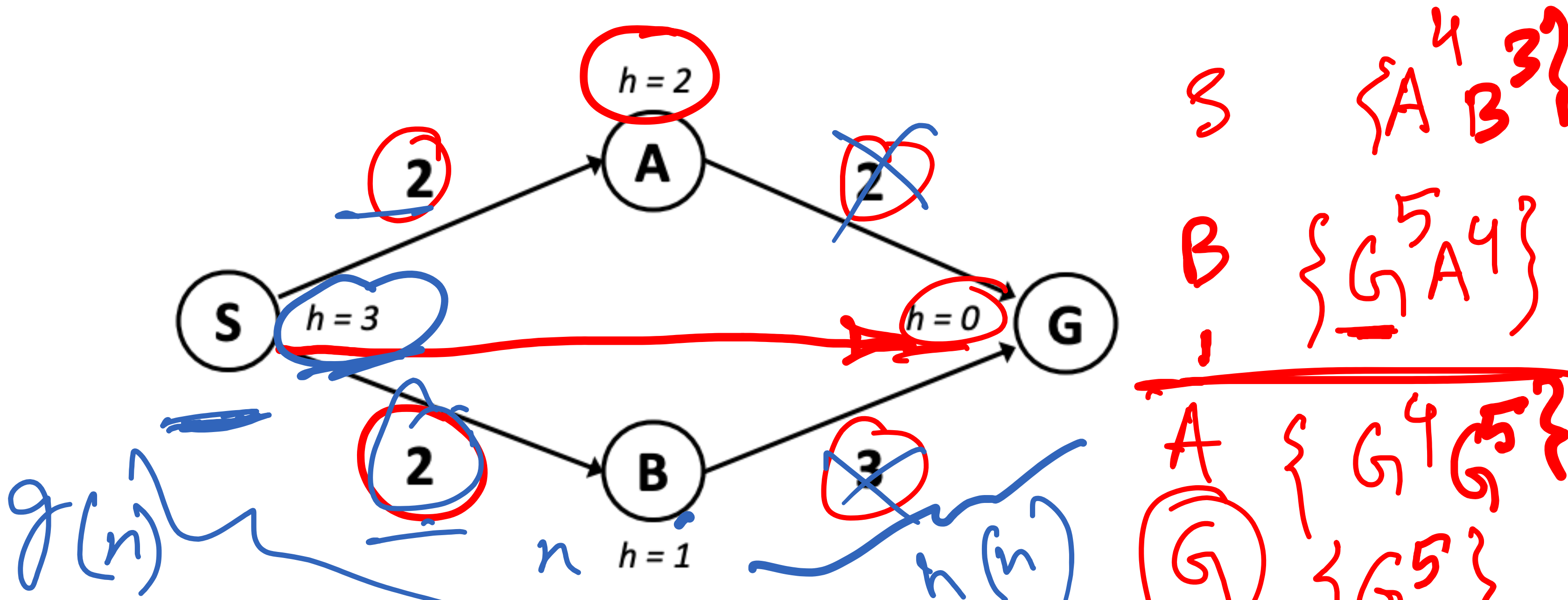
- Solution path found is S B G, 4 nodes expanded..
- Still pretty fast. And optimal, too.

S – B – G (5+4=9)

Greedy → S – C – G (13)

When should A* terminate?

- Should we stop when we enqueue a goal?



- No: only stop when we dequeue a goal

Heuristics, More Formally

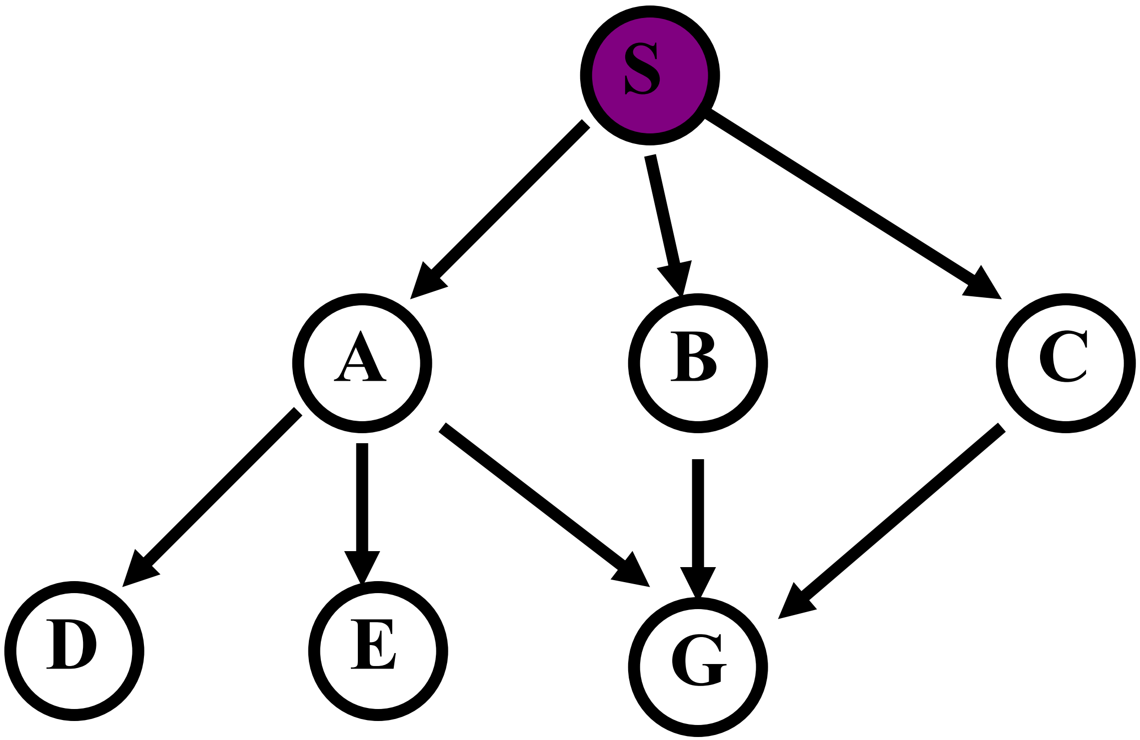
$h(n)$ is a **heuristic function**, that maps a state n to an estimated cost from n-to-goal

$h(n)$ is **admissible** iff $h(n) \leq$ the lowest actual cost from n-to-goal

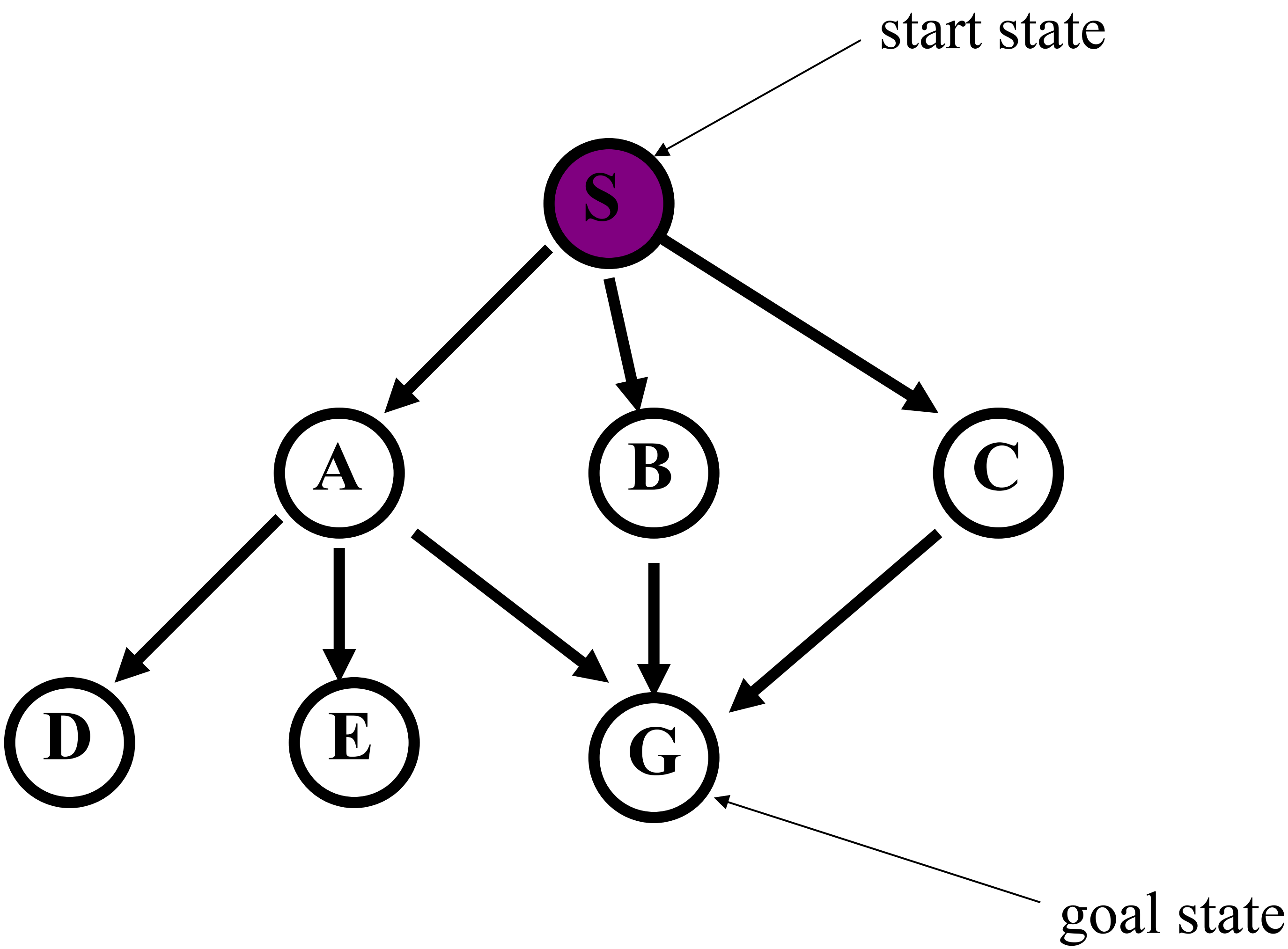
$h(n)$ is **consistent** iff
$h(n) \leq \text{lowestcost}(n, n') + h(n')$

IS A HEURISTIC ADMISSIBLE?

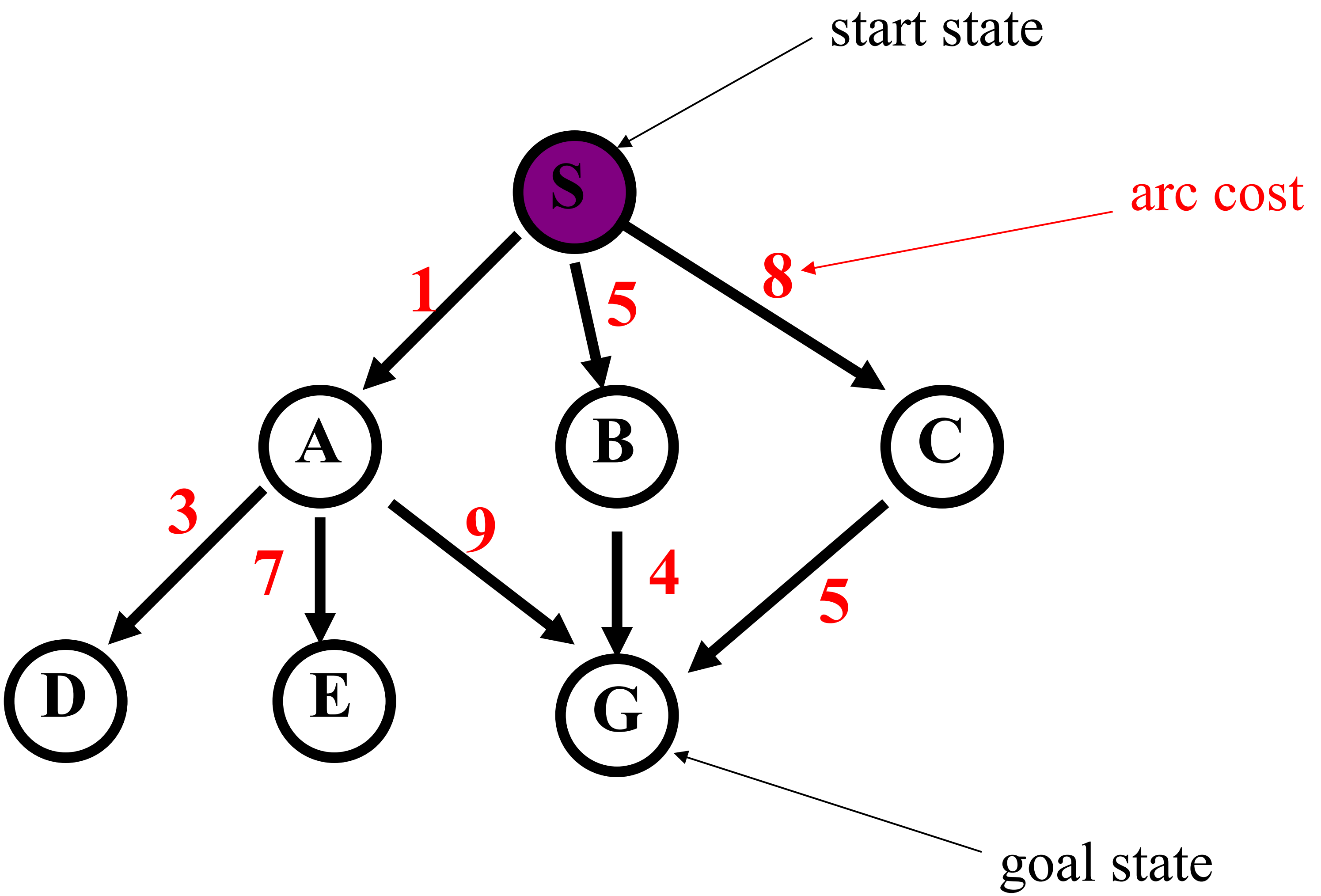
Example search space



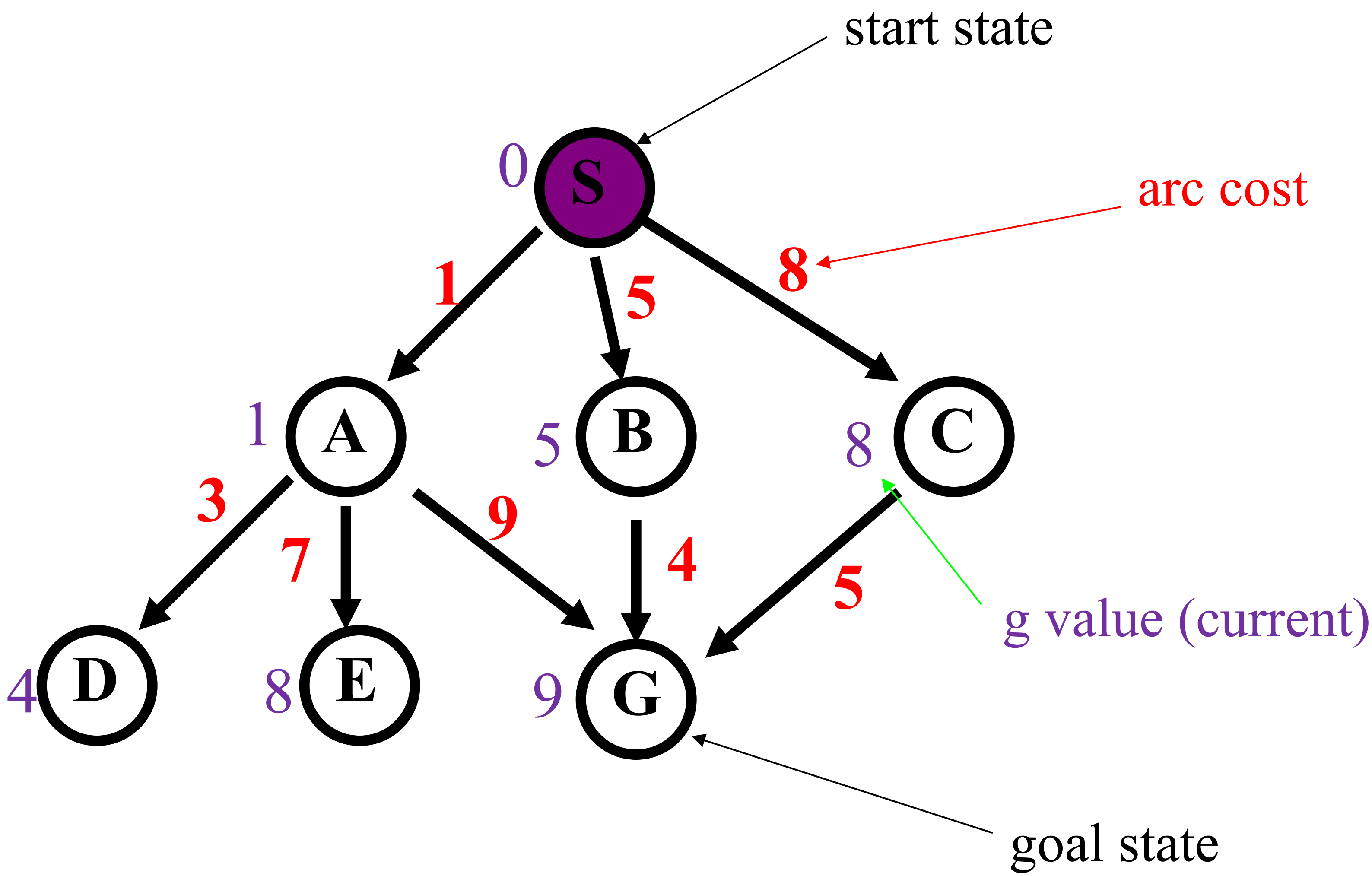
Example search space



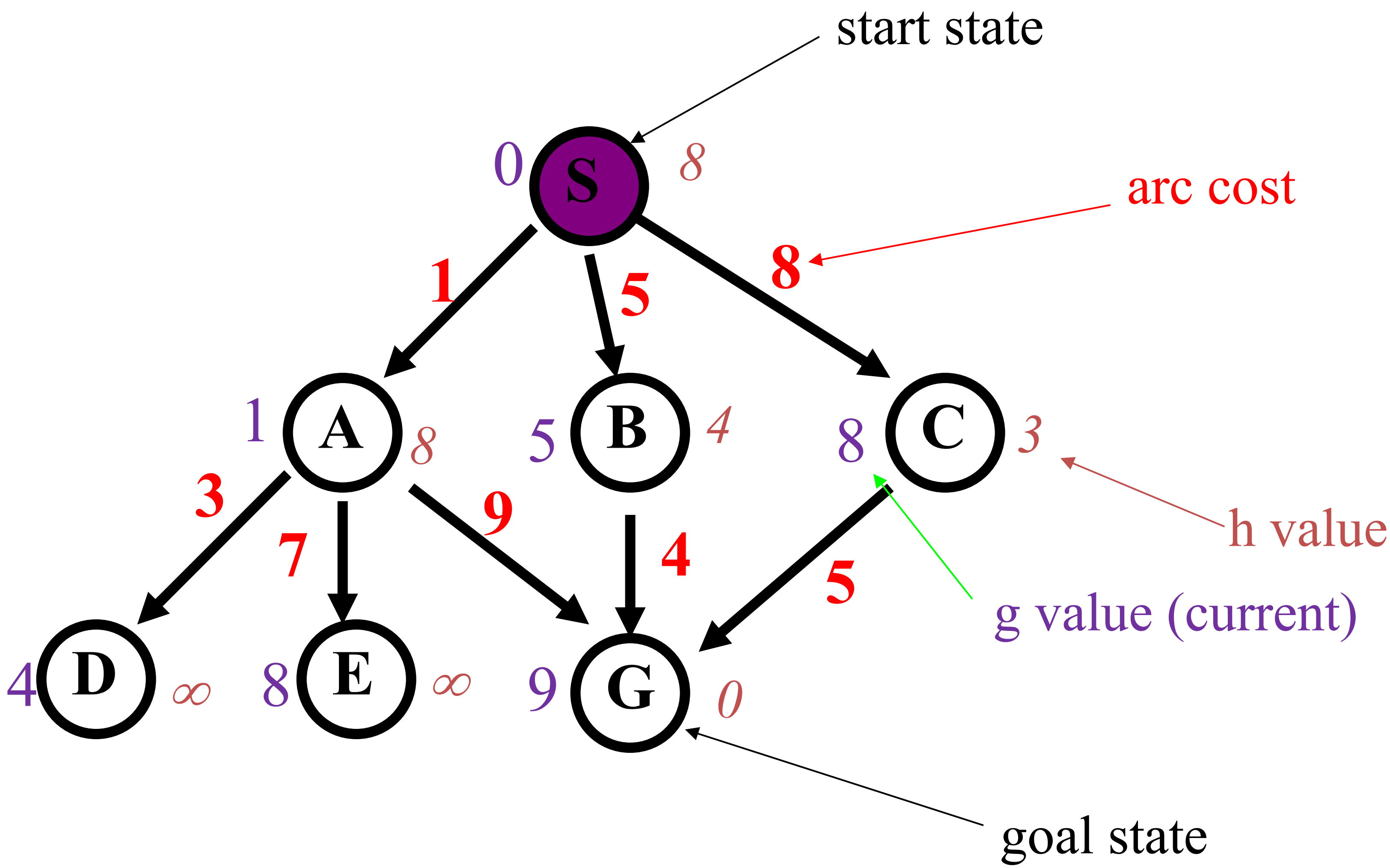
Example search space



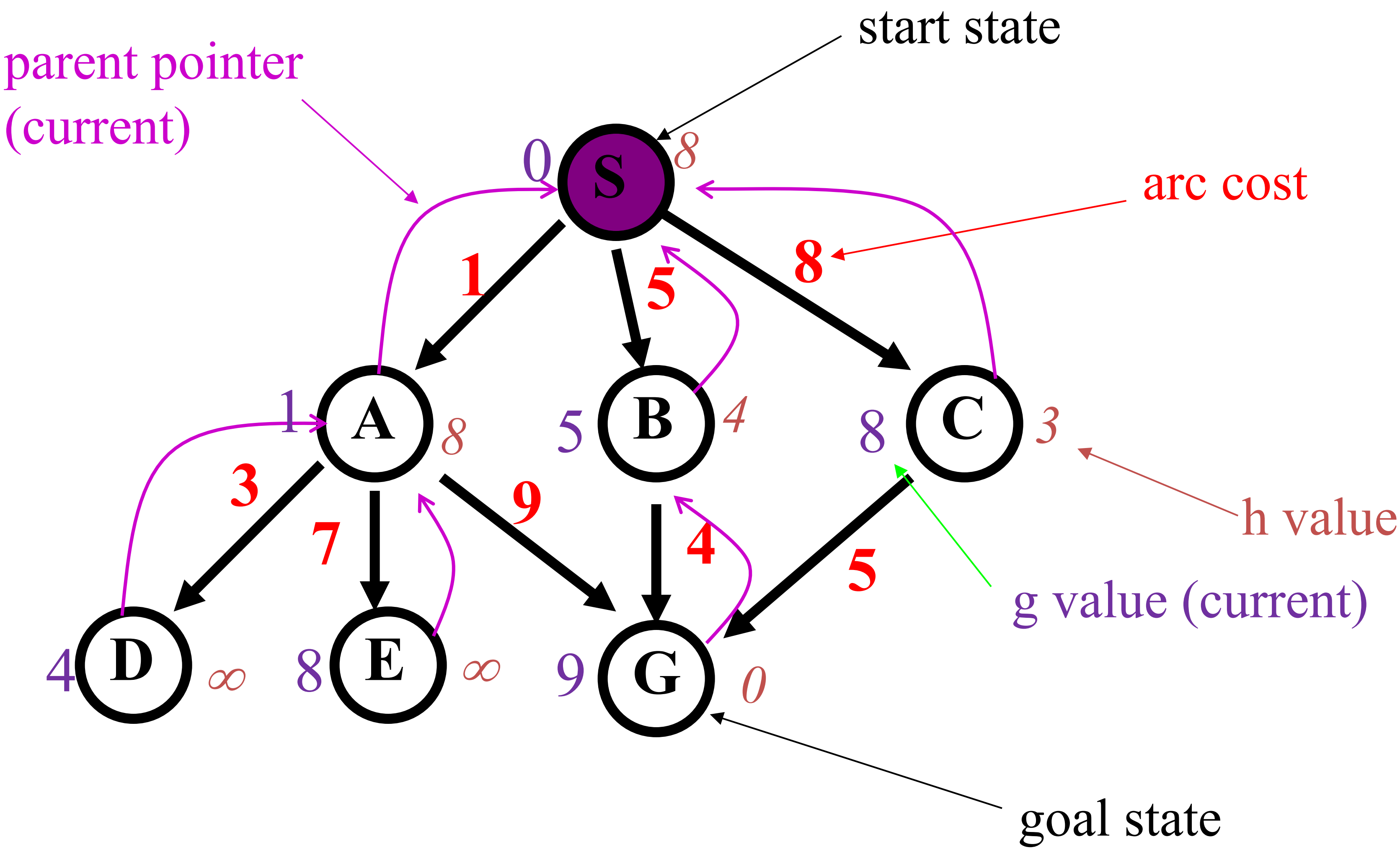
Example search space



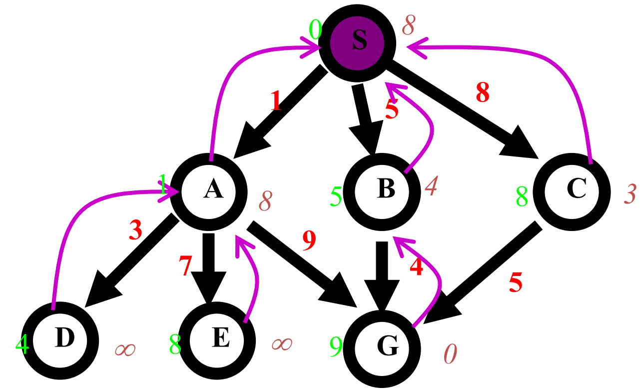
Example search space



Example search space



Example

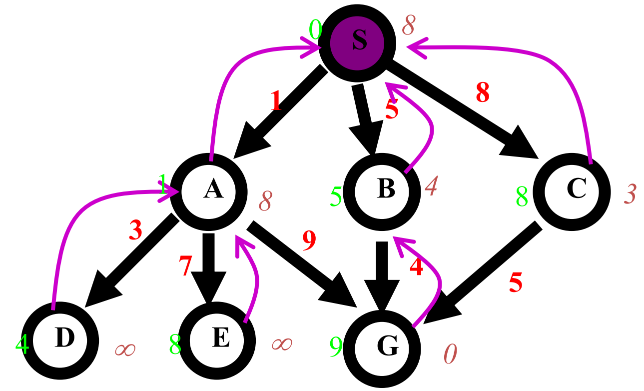


n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9

- h*(n) is (hypothetical) perfect heuristic (an oracle)
- Since h(n) <= h*(n) for all n, h is admissible (optimal)
- Optimal path = *S B G* with cost 9

The table and graph show values for the entire space, but we must discover or compute them during the search

Example



n	g(n)	h(n)	f(n)	h*(n)
S	0	8	8	9
A	1	8	9	9
B	5	4	9	4
C	8	3	11	5
D	4	inf	inf	inf
E	8	inf	inf	inf
G	9	0	9	0

- h*(n) is (hypothetical) perfect heuristic (an oracle)
- **Since h(n) <= h*(n) for all n, h is admissible (optimal)**
- Optimal path = *S B G* with cost 9

Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done

Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search

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

Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search
- **Better heuristic:** If h1(n) < h2(n) <= h*(n) for all non-goal nodes, then h2 is a *better* heuristic than h1

Observations on A*

- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search
- **Better heuristic:** If h1(n) < h2(n) <= h*(n) for all non-goal nodes, then h2 is a *better* heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*

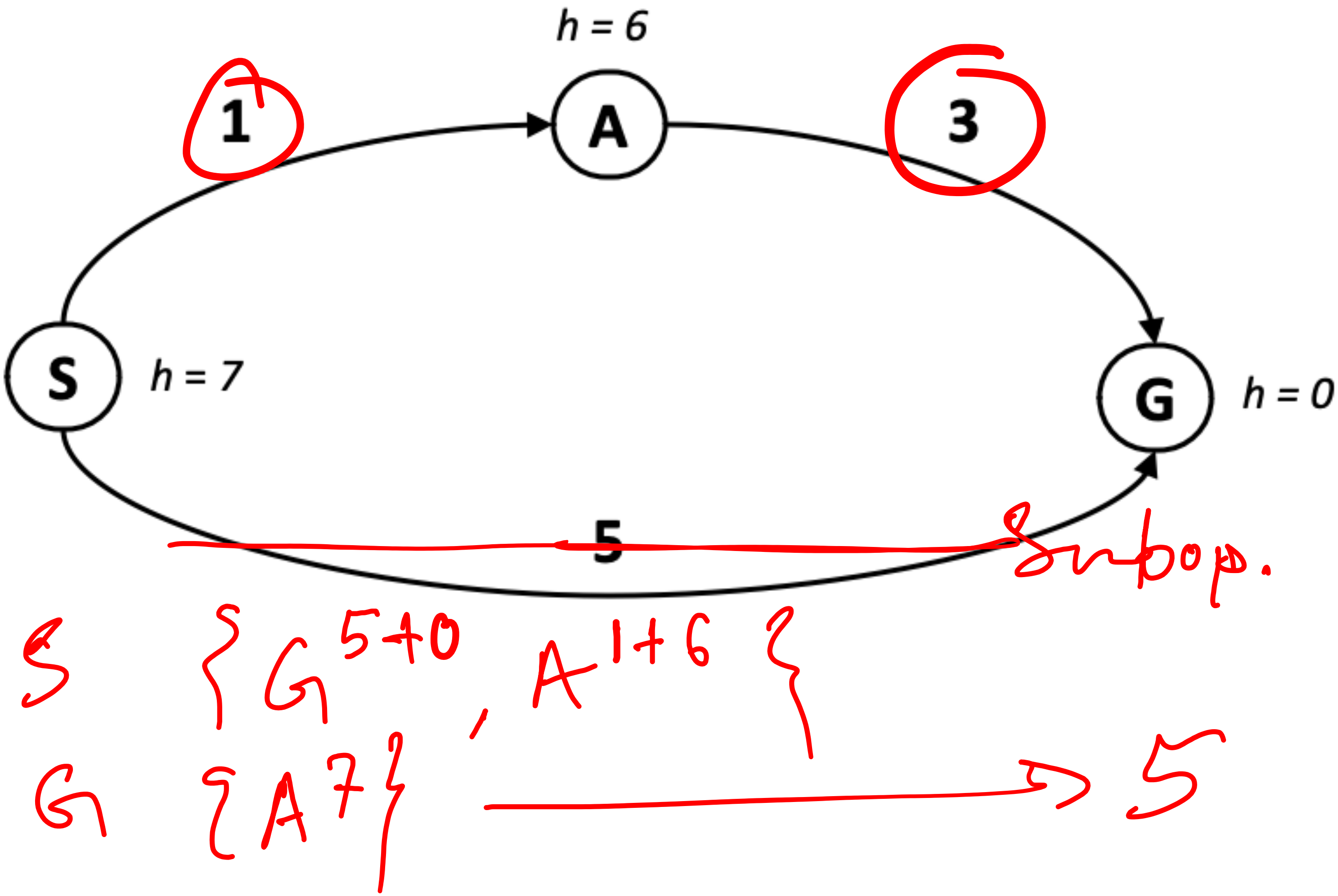
    i.e., A1 expands at least as many nodes as A2*
  - We say that A2* is *better informed* than A1*

Observations on A*

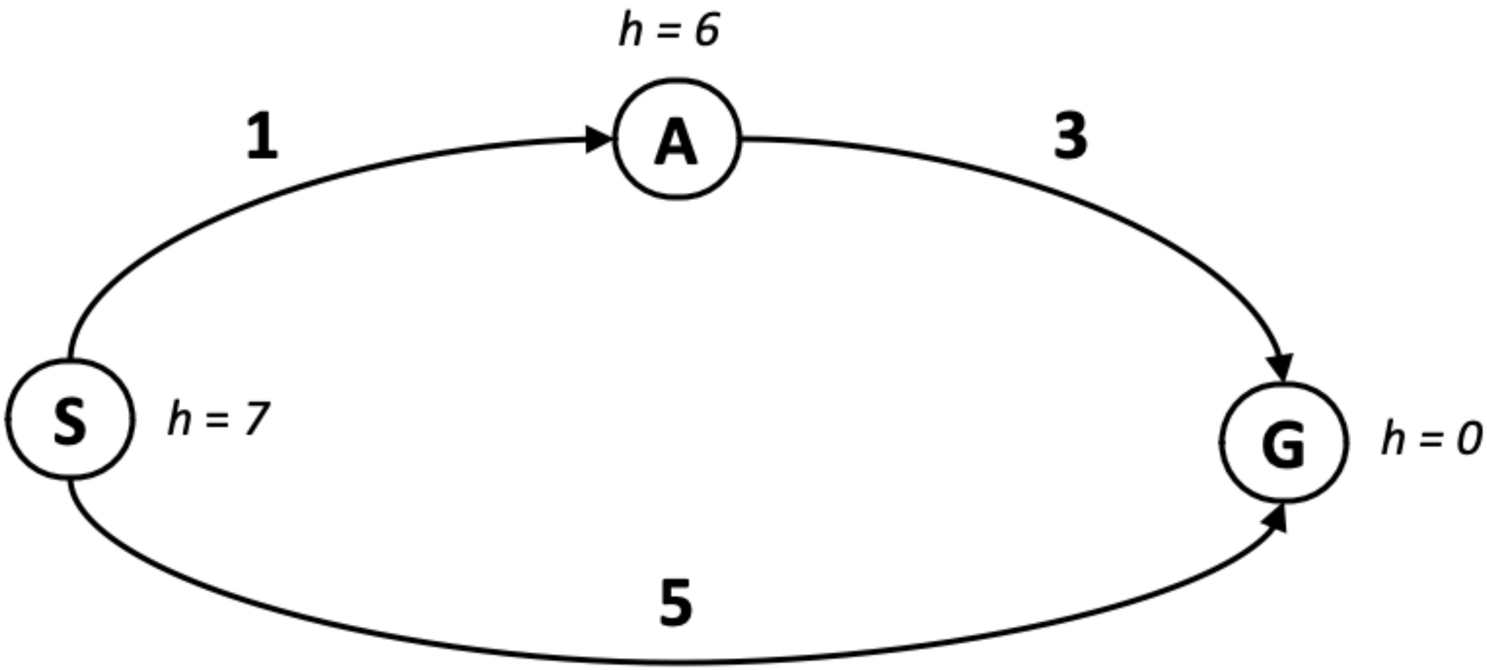
- **Perfect heuristic:** If h(n) = h*(n) for all n, only nodes on an optimal solution path expanded; no extra work is done
- **Null heuristic:** If h(n) = 0 for all n, then it is an admissible heuristic and A* acts like uniform-cost search
- **Better heuristic:** If h1(n) < h2(n) <= h*(n) for all non-goal nodes, then h2 is a *better* heuristic than h1
  - If A1* uses h1, and A2* uses h2, then every node expanded by A2* is also expanded by A1*

    i.e., A1 expands at least as many nodes as A2*
  - We say that A2* is *better informed* than A1*
- ***The closer h to h*, the fewer extra nodes expanded***

Is A* optimal?



Is A* optimal?



- What went wrong?
- Actual bad goal cost < estimated good goal cost
- We need estimates to be less than actual costs!

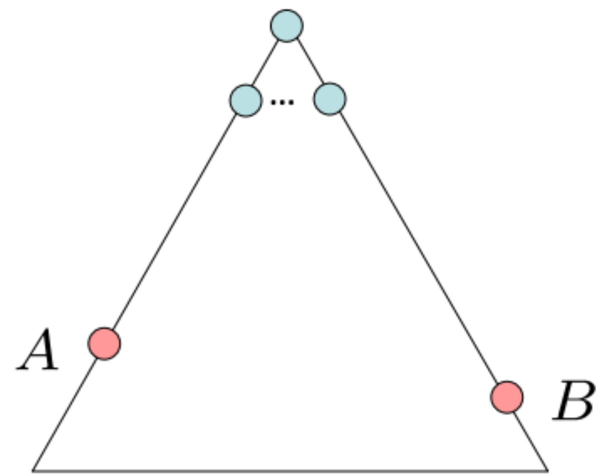
A*

- Pronounced *"a star"*
- h is **admissible** when h(n) <= h*(n) holds
  - **h*(n)** = *true cost* of *minimal cost path* from n to a goal
- Using an admissible heuristic guarantees that 1st solution found will be an **optimal** one
  - With an admissible heuristic, A* is cost-optimal
- A* is **complete** whenever branching factor is finite and every action has fixed, positive cost
- A* is **admissible**

Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4* **4** (2): 100–107.

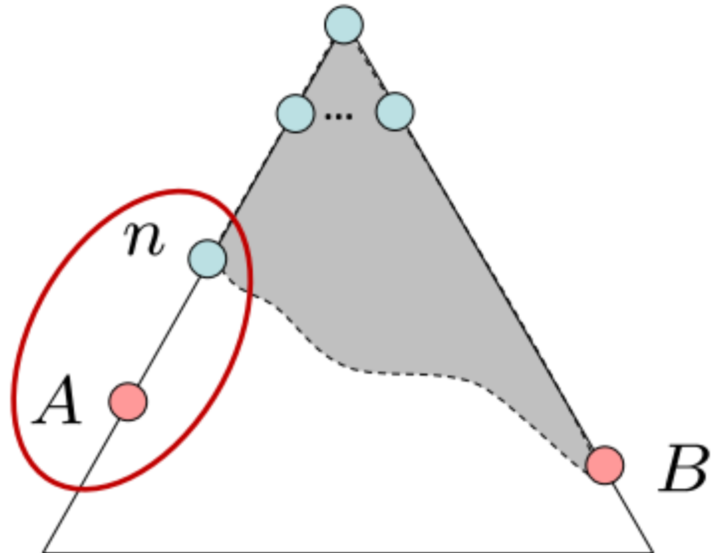
Proof of the optimality of A*

- Assume:
  - A is an optimal goal node
  - B is a suboptimal goal node
  - h is admissible
- Claim:
  - A will exit the fringe before B



Proof of the optimality of A*

- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim:
    n will be expanded before B
    1. f(n) is less or equal to f(A)

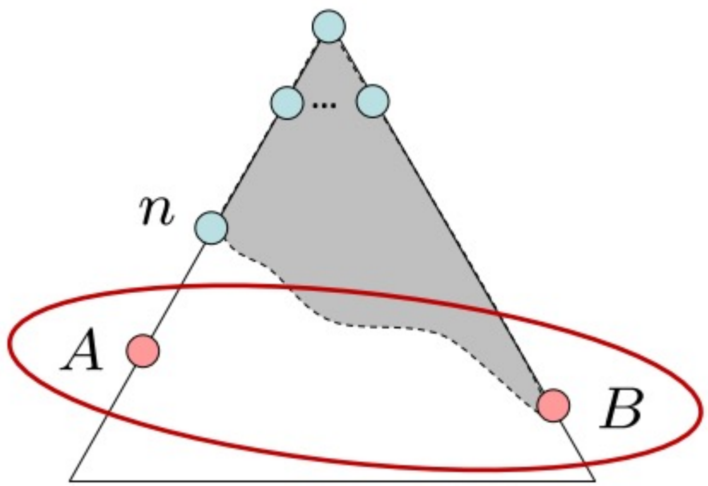


f(n) = g(n) + h(n)	Definition of f-cost
f(n) <= g(A)	Admissibility of h
f(A) = g(A)	h is 0 at goal, h(A)=0

We would not take the step if f(n) > g(A), and that is condition of admissibility

Proof of the optimality of A*

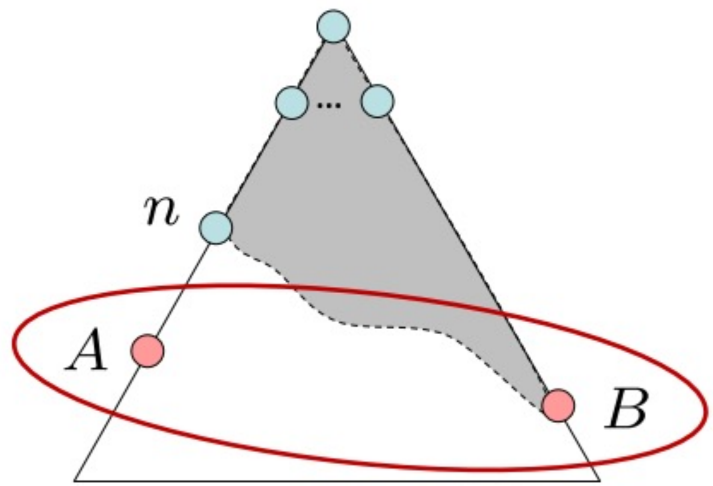
- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim:
    n will be expanded before B
    1. f(n) is less or equal to f(A)
    2. f(A) < f(B)



g(A) < g(B) as B is suboptimal
f(A) < f(B) h is 0 at goal, h(A)=h(B)=0

Proof of the optimality of A*

- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim:
    n will be expanded before B
    1. f(n) <= f(A)
    2. f(A) < f(B)

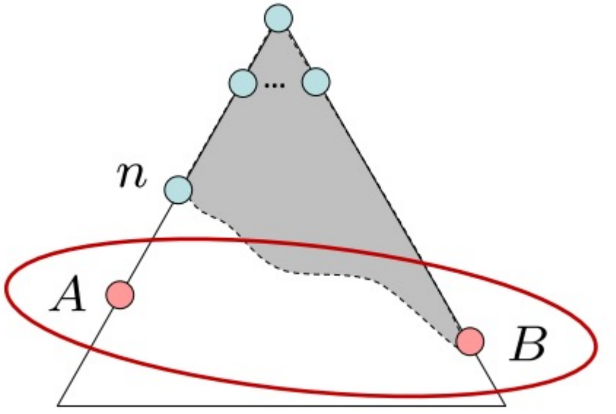


f(n) <= f(A) < f(B)

So n expands before B

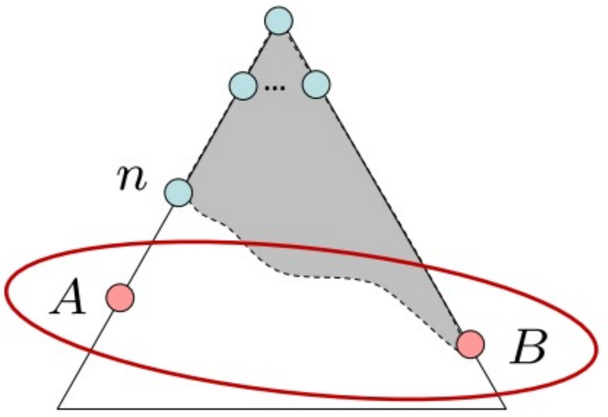
Proof of the optimality of A*

- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim: n will be expanded before B
    1. f(n) <= f(A)
    2. f(A) < f(B)
    3. n expands before B
  - All ancestors of A expand before B
  - A expands before B
  - So A* search is optimal



Proof of the optimality of A*

- Proof:
  - Imagine B is on the fringe
  - Some ancestor n of A is on the fringe, too (maybe A!)
  - Claim: n will be expanded before B
    1. f(n) <= f(A)
    2. f(A) < f(B)
    3. n expands before B
  - All ancestors of A expand before B
  - A expands before B
  - So A* search is optimal



Other ways to do it is
Proof by contradiction

How to find good heuristics

Some options (mix-and-match):

- If h1(n) < h2(n) <= h*(n) for all n, h2 is better than h1
  - h2 **dominates** h1
- **Relaxing problem:** remove constraints for easier problem; use its solution cost as heuristic function
- Max of two admissible heuristics is a **Combining heuristics**: admissible heuristic, and it's better!
- Use statistical estimates to compute h; may lose admissibility
- Identify good features, then use **machine learning** to find heuristic function; also may lose admissibility

Pruning:
Dealing with Large Search Spaces

Cycle pruning

Don’t add a node to the fringe if you’ve already expanded it (it’s already on a path you’ve considered/are considering)

Q: What type of search-space would this be approach be applicable for?

Multiple-path pruning

Pruning: Dealing with Large Search Spaces

Cycle pruning

Don't add a node to the fringe if you've already expanded it (it's already on a path you've considered/are considering)

Q: What type of search-space would this be approach be applicable for?

Multiple-path pruning

Core idea: there may be multiple possible solutions, but you only need one

Maintain an "explored" (sometimes called "closed") set of nodes at the ends of paths; discard a path if a path node appears in this set

Q: Does this return an optimal solution?

Optimality with Multiple-Path Pruning

Some options to find the optimal solution (pulled from PM 3.7.2)

- 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. OR

Optimality with Multiple-Path Pruning

Some options to find the optimal solution (pulled from Ch 3.7.2)

- 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. OR

- 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. OR

Optimality with Multiple-Path Pruning

Some options to find the optimal solution (pulled from Ch 3.7.2)

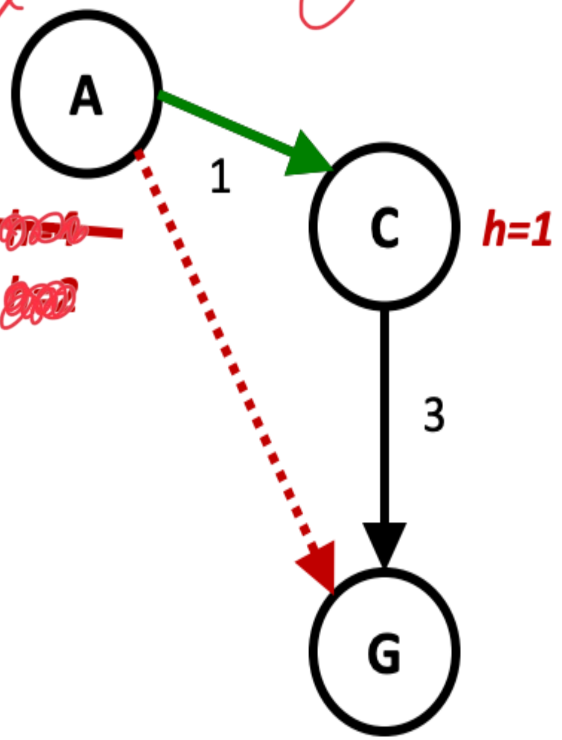
- Make sure that the first path found to any node is a lowest-cost path to that node, then prune all subsequent paths found to that node. OR
- If the search algorithm finds a lower-cost path to a node than one already found, it could remove all paths that used the higher-cost path to the node. OR
- Whenever the search finds a lower-cost path to a node than a path to that node already found, it could incorporate a new initial section on the paths that have extended the initial path.

A* and Multiple-Path Pruning

If $h(n)$ is consistent, A* with multiple-path pruning will find an optimal solution

Core Idea: Why?

Consistency of Heuristics



- Main idea: estimated heuristic costs ≤ actual costs
  - Admissibility: heuristic cost ≤ actual cost to goal

    h(A) ≤ actual cost from A to G

  - Consistency: heuristic "arc" cost ≤ actual cost for each arc

    h(A) – h(C) ≤ cost(A to C)

- Consequences of consistency:
  - The f value along a path never decreases

    h(A) ≤ cost(A to C) + h(C)

  - A* graph search is optimal

Dealing with hard problems

- For large problems, A* may require too much space
- Variations conserving memory: IDA* and SMA*
- IDA*, iterative deepening A*, uses successive iteration with growing limits on f, e.g.
  - A* but don't consider a node n where f(n) >10
  - A* but don't consider a node n where f(n) >20
  - A* but don't consider a node n where f(n) >30, ...
- SMA* -- Simplified Memory-Bounded A*
  - Uses queue of restricted size to limit memory use

A* Applications

- Video games
- Pathing / routing problems
- Resource planning problems
- Robot motion planning
- Language analysis
- Machine translation
- Speech recognition
- …

Summary: Informed search

- **Best-first search** is general search where minimum-cost nodes (w.r.t. some measure) are expanded first
- **Greedy search** uses minimal estimated cost h(n) to goal state as measure; reduces search time, but is neither complete nor optimal
- **A* search** combines uniform-cost search & greedy search: $f(n) = g(n) + h(n)$. Handles state repetitions & h(n) never overestimates
  - A* is complete & optimal, but space complexity high
  - Time complexity depends on quality of heuristic function
  - IDA* and SMA* reduce the memory requirements of A*

Summary (Fig 3.11)

Strategy	Selection from Frontier	Path found	Space
Breadth-first	First node added	Fewest arcs	Exponential
Depth-first	Last node added	No	Linear
Iterative deepening	—	Fewest arcs	Linear
Greedy best-first	Minimal $h(p)$	No	Exponential
Lowest-cost-first	Minimal cost (p)	Least cost	Exponential
A^*	Minimal cost $(p) + h(p)$	Least cost	Exponential
IDA^*	—	Least cost	Linear