Sistemi Intelligenti Avanzati
Corso di Laurea in Informatica, A.A. 2020-2021
Università degli Studi di Milan

Search algorithms for planning
L2

Matteo Luperto
Dipartimento di Informatica
matteo.luperto@unimi.it

Summing up

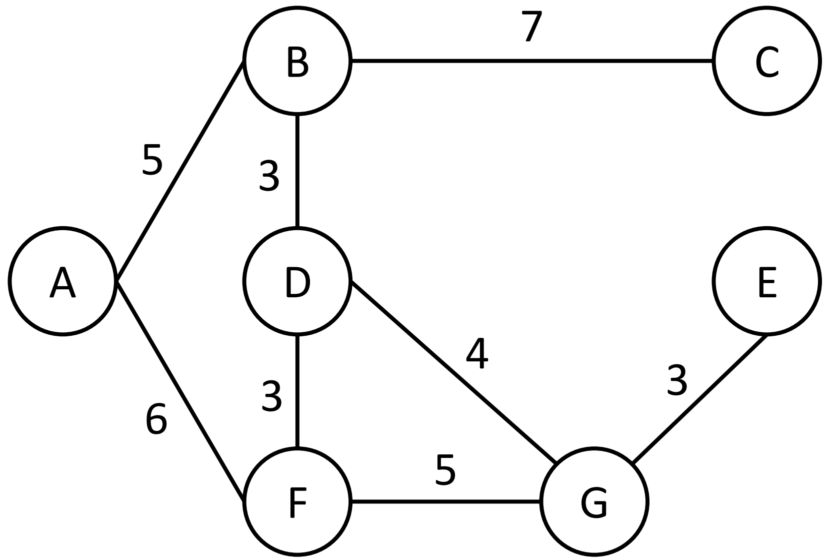
b branching factor,
q depth of the shallowest solution,
m maximum depth of search tree,
l depth limit

Criterion	BFS	UCS	DFS	Limited DFS	Iterative DFS
Complete?	Yes (if b finite)	Yes (if b finite and cost positive)	No (only for finite spaces)	No ($l > q$)	Yes (if b finite)
Time com.	$O(b^q)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^l)$	$O(b^q)$
Space com.	$O(b^q)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(bl)$	$O(bq)$
Optimal?	Yes (identical costs)	Yes	No	No	Yes (identical costs)

Running example

- To present the various search algorithms, we will use this *problem instance* as our running example

State-transition graph:



Initial state: A

Desired solution: any path to goal state E

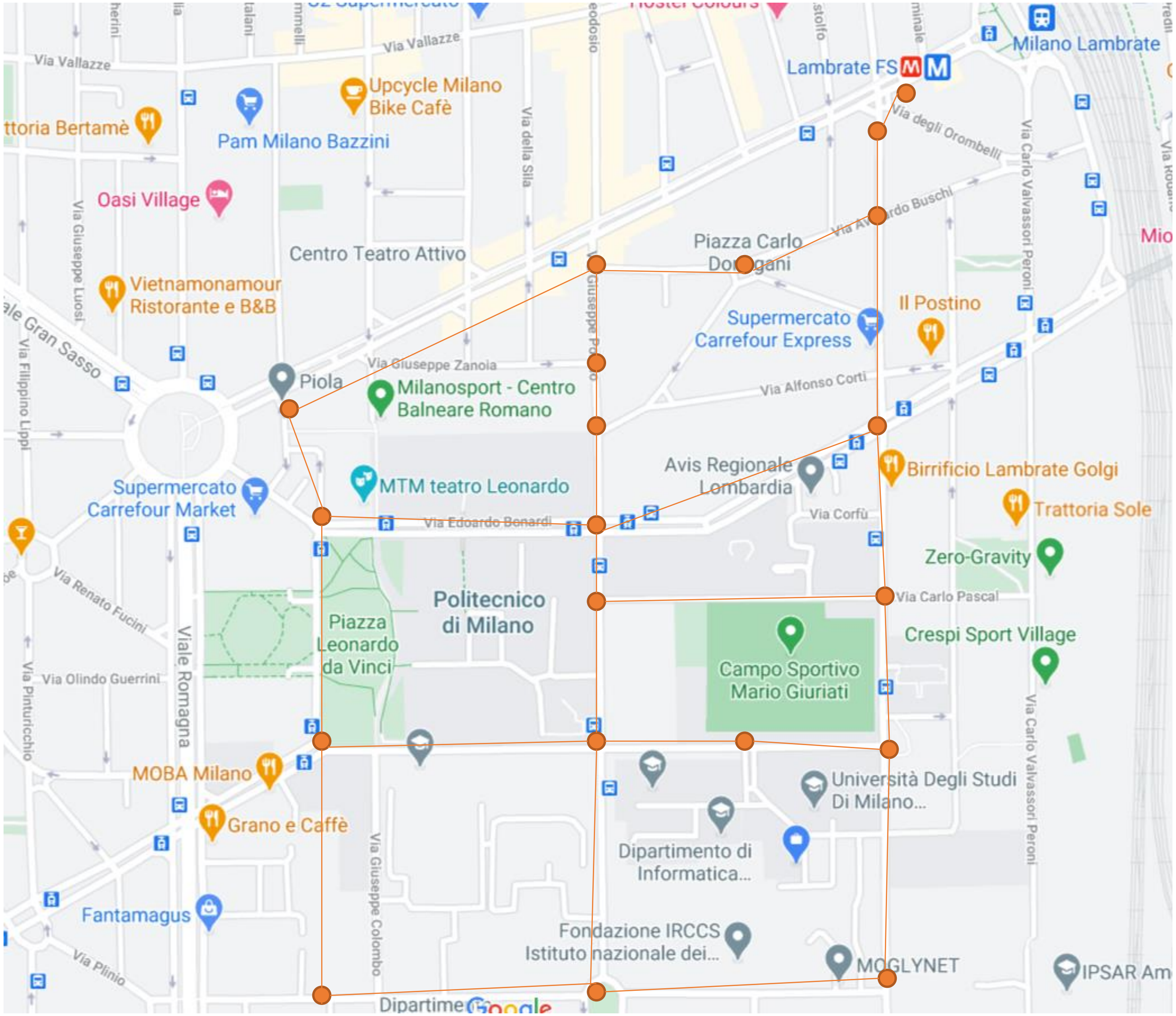
- It might be useful to think it as a map, but keep in mind that this interpretation does not hold for every instance

Informed vs non-informed search

- Besides its own rules, any search algorithm decides where to search next by leveraging some knowledge
- **Non-informed** search uses only knowledge specified at problem-definition time (e.g., goal and start nodes, edge costs), just like we saw in the previous examples
- An **informed** search might go beyond such knowledge
- Idea: using an estimate of how far a given node is from the goal
- Such an estimate is often called a **heuristic**

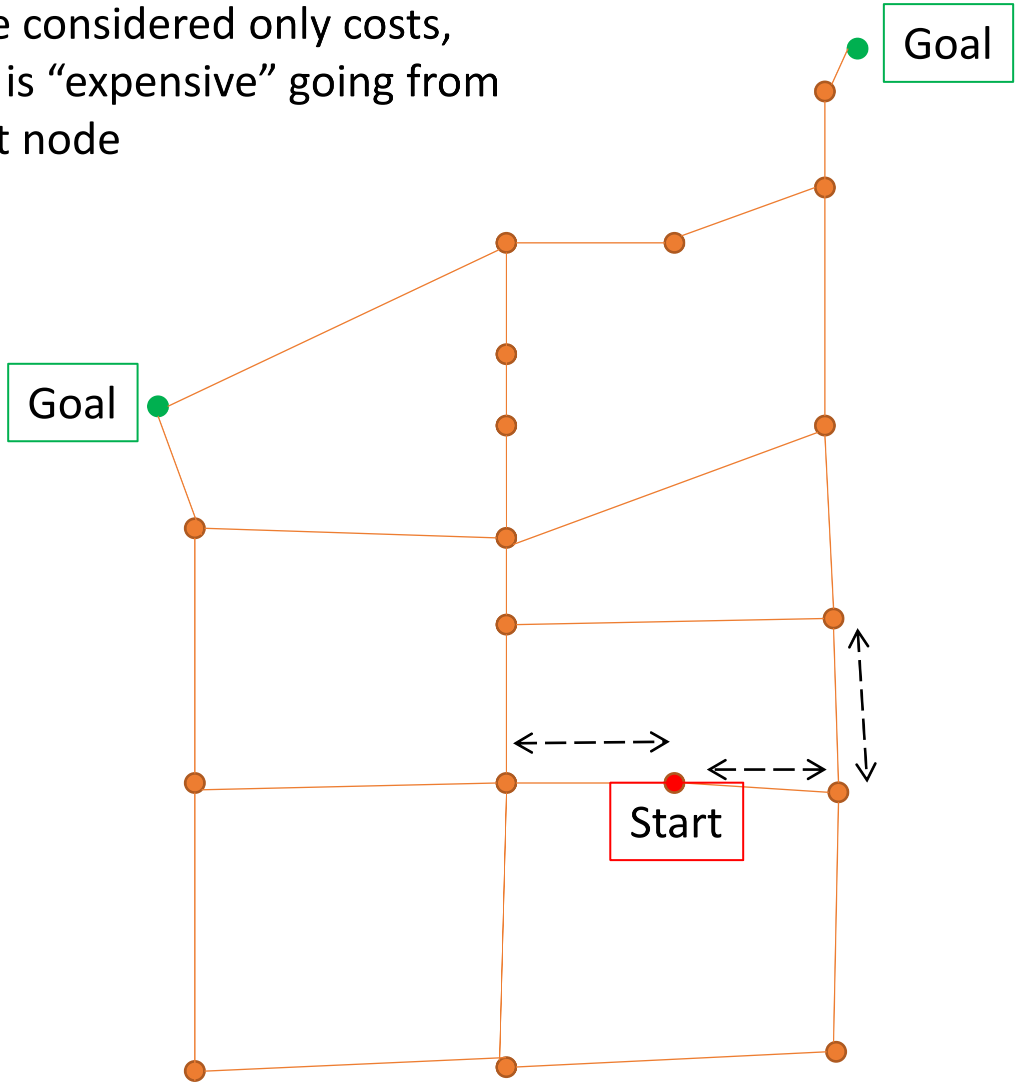
Estimate of the cost of the optimal path from node v to the goal: $h(v)$

Example: going home from the CS department with METRO



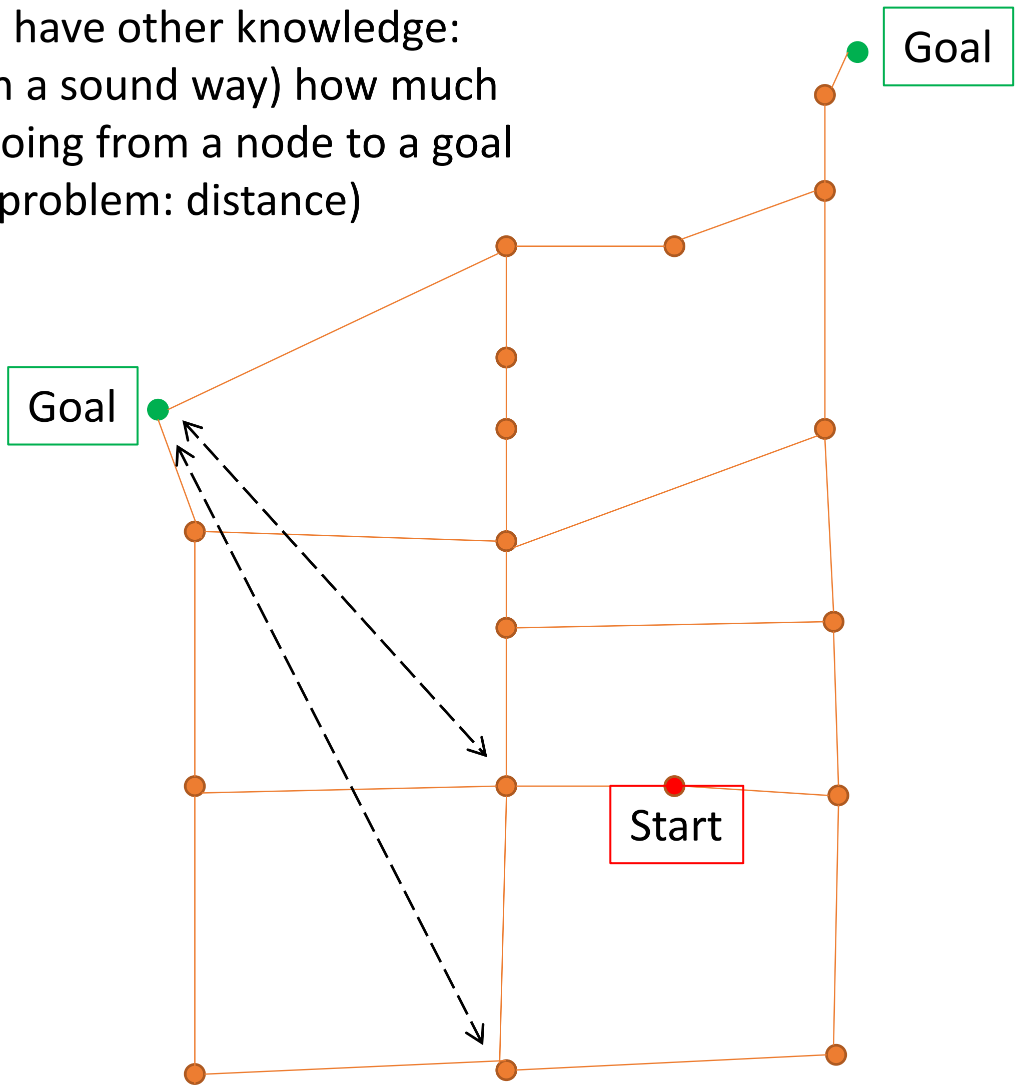
Informed Search method

Up to now we have considered only costs, namely how much is “expensive” going from Start to the current node



Informed Search method

However, we often have other knowledge:
we can estimate (in a sound way) how much
will be expensive going from a node to a goal
(for path planning problem: distance)



Using an heuristic: Greedy Best-First Search

- Let's use domain knowledge: perform a UCS, but instead of considering costs, we consider an heuristic $h(n)$ that estimates, for each <u>node</u> the cost "to go" to the solution from there

- Greedy approach: we expand the goal that seems closest to the solution

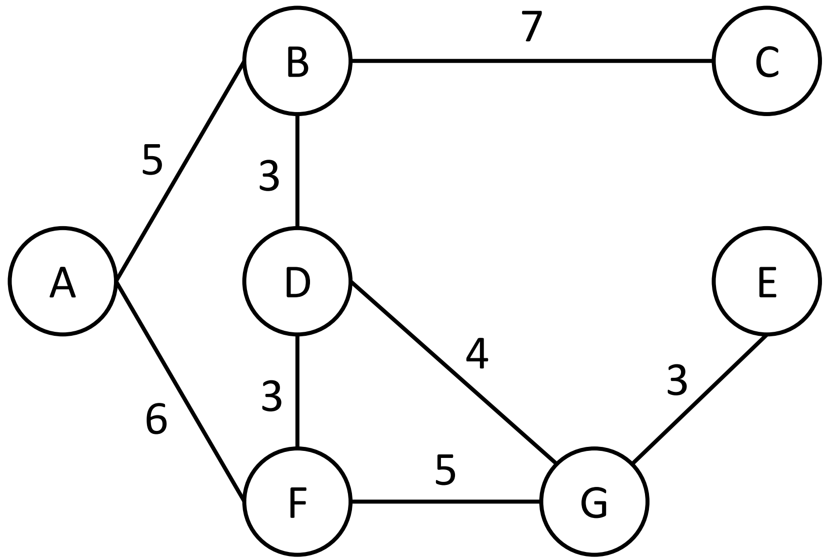
$$h(n) \leq \text{ Cost of the minimum path from n to the goal}$$

- The idea is to go as fast as possible towards the solution

Running example

- To present the various search algorithms, we will use this *problem instance* as our running example

State-transition graph:

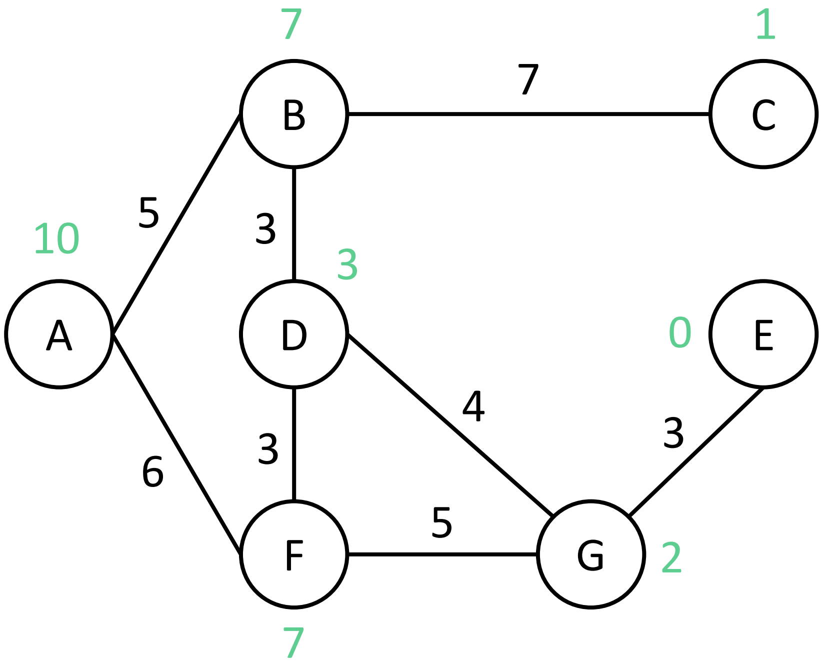


Initial state: A

Desired solution: any path to goal state E

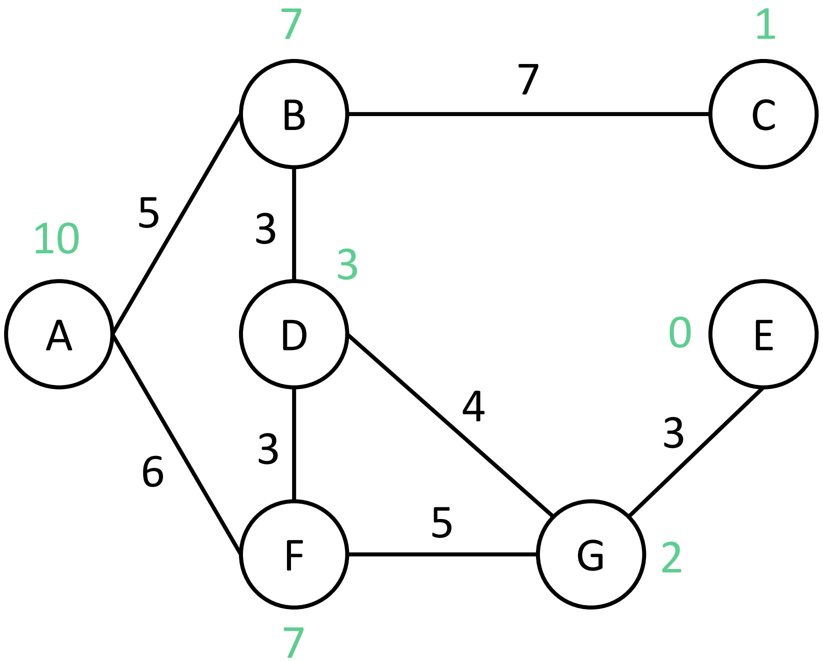
- It might be useful to think it as a map, but keep in mind that this interpretation does not hold for every instance

Greedy Best-First Search



node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

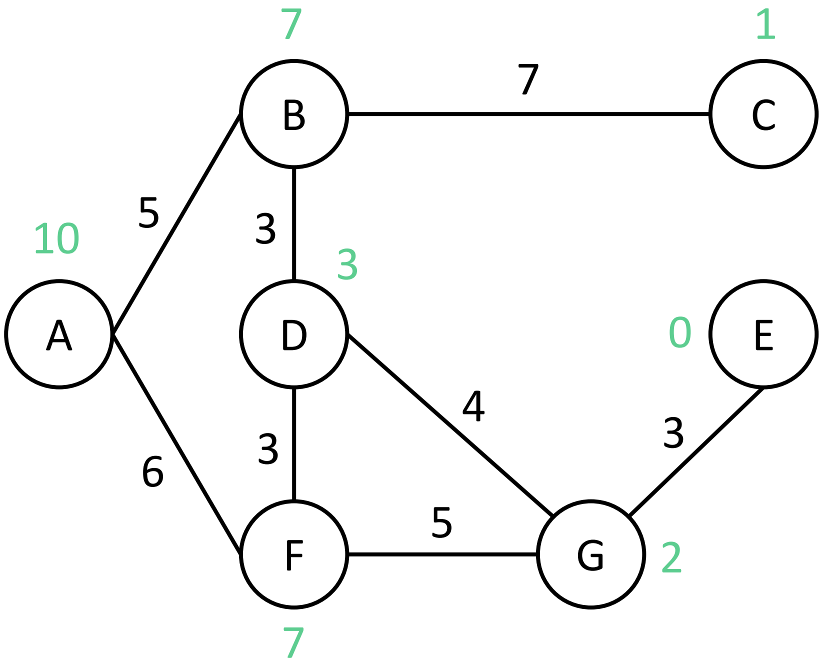
Greedy Best-First Search

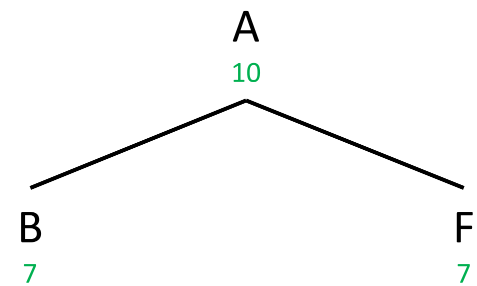


A
10

node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

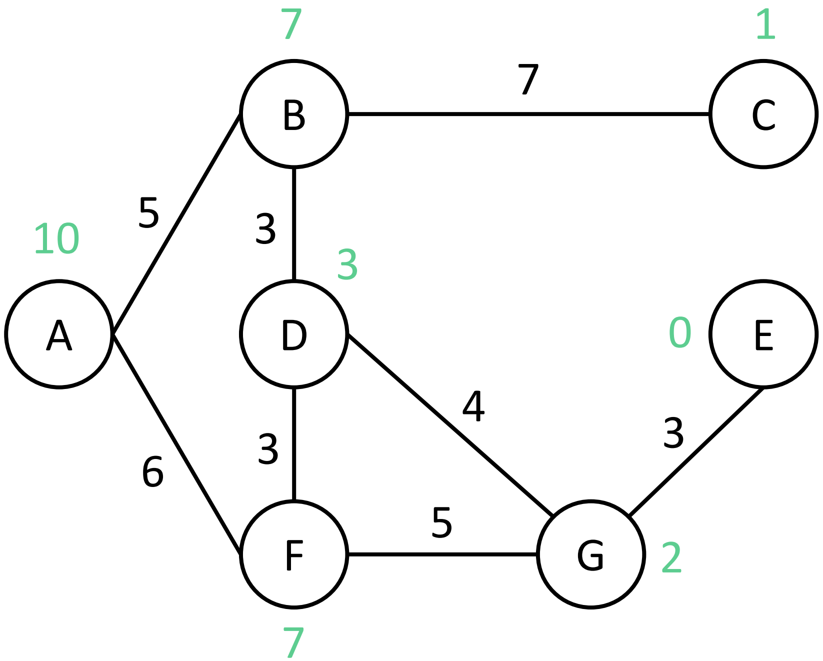
Greedy Best-First Search

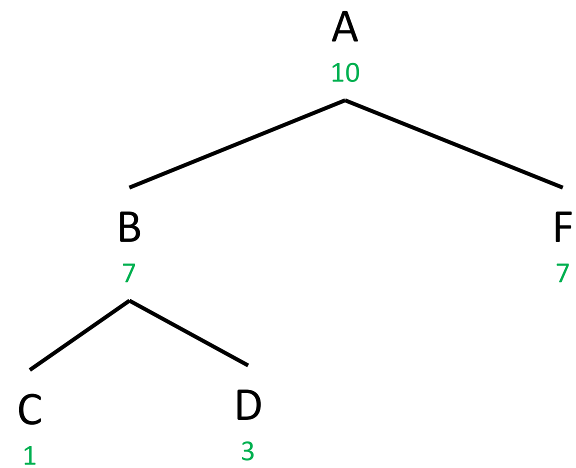




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

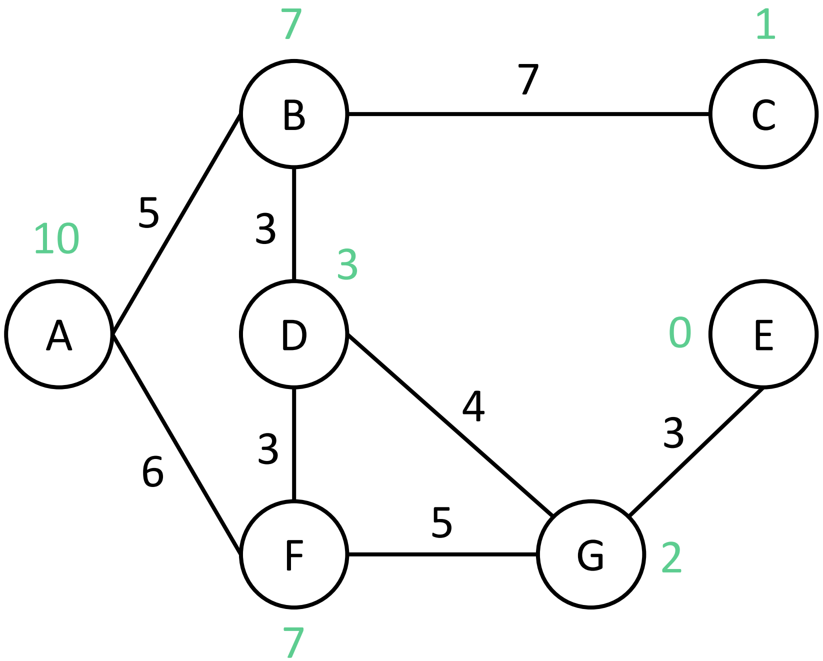
Greedy Best-First Search

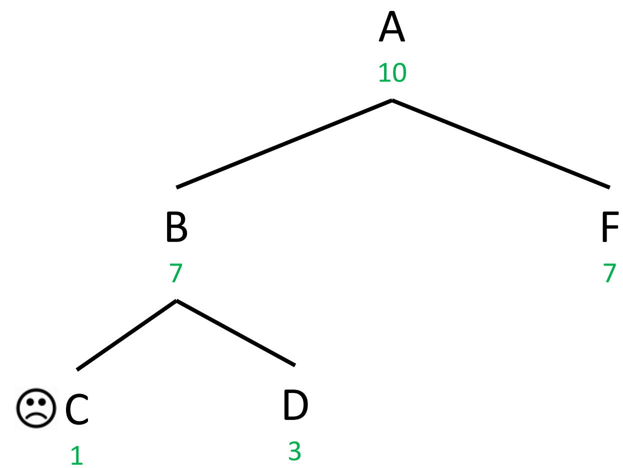




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

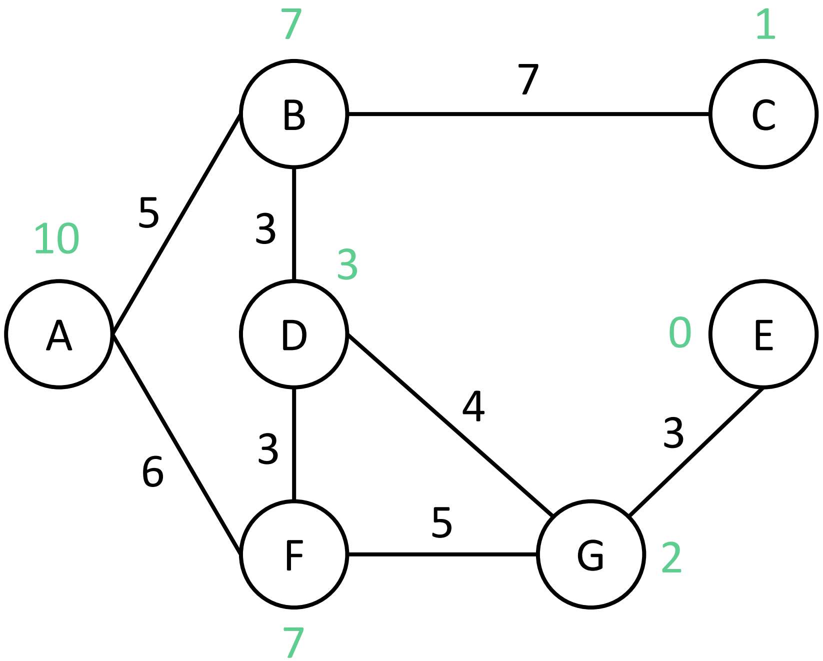
Greedy Best-First Search

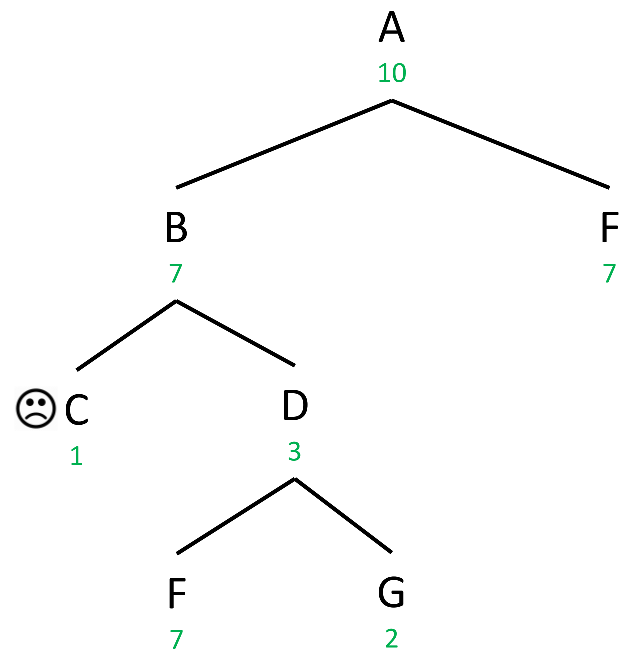




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

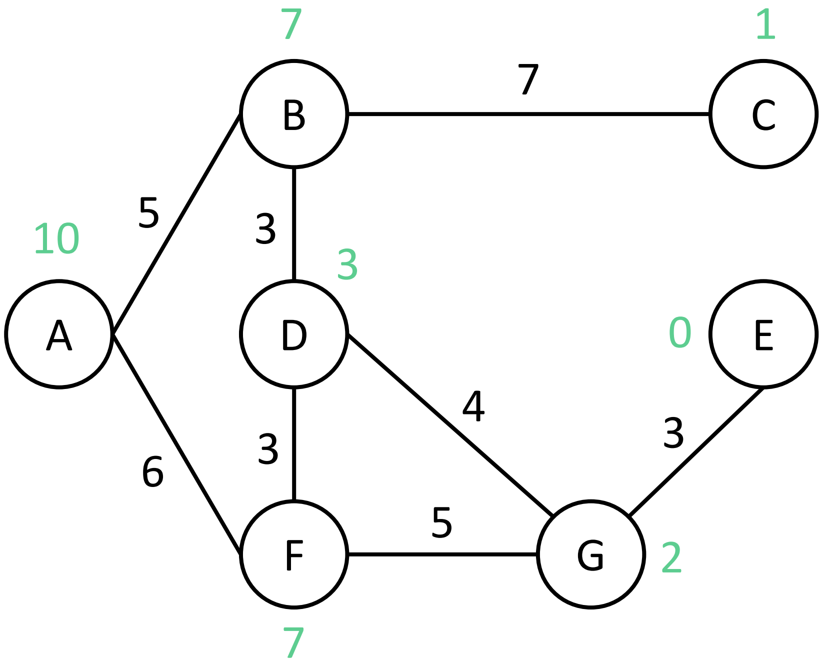
Greedy Best-First Search

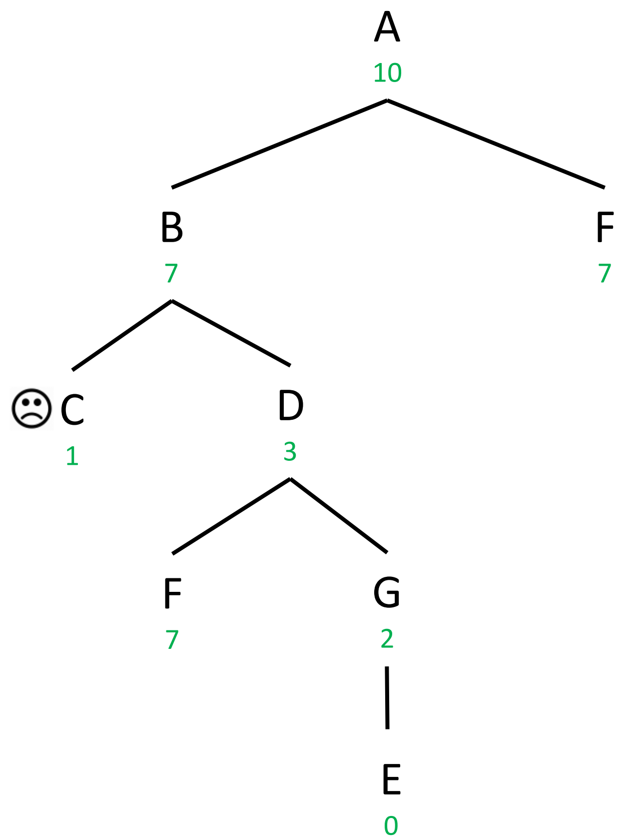




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

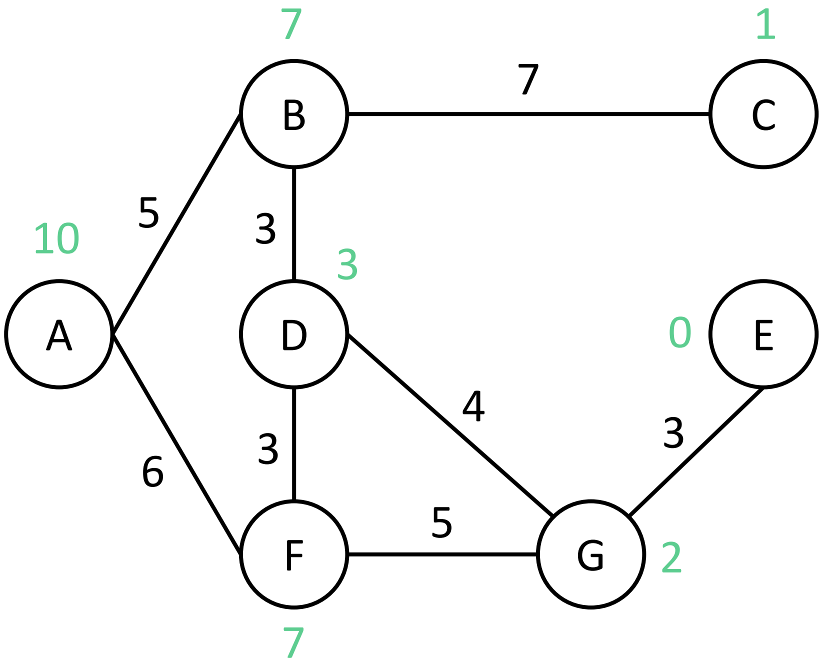
Greedy Best-First Search

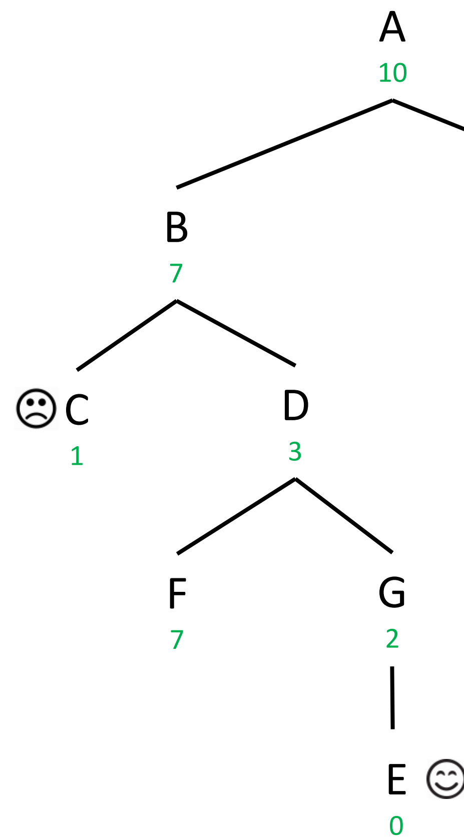




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

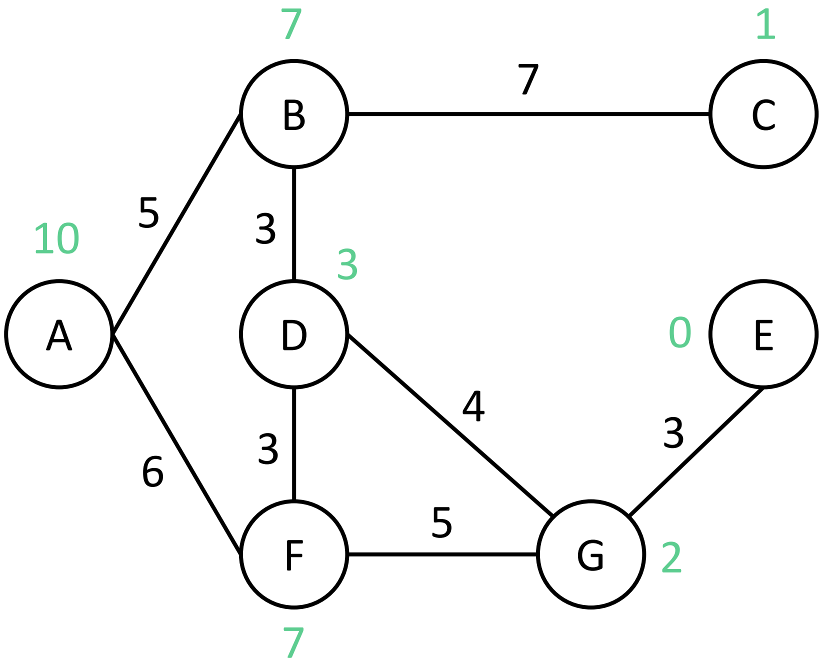
Greedy Best-First Search

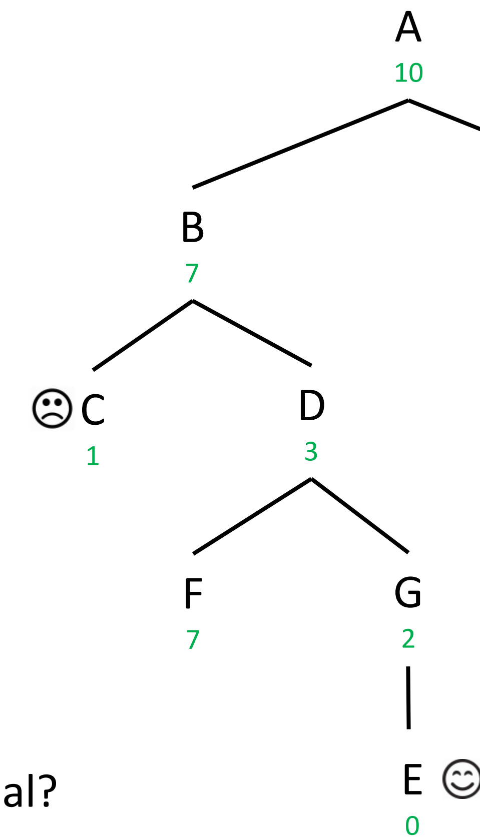




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

Greedy Best-First Search





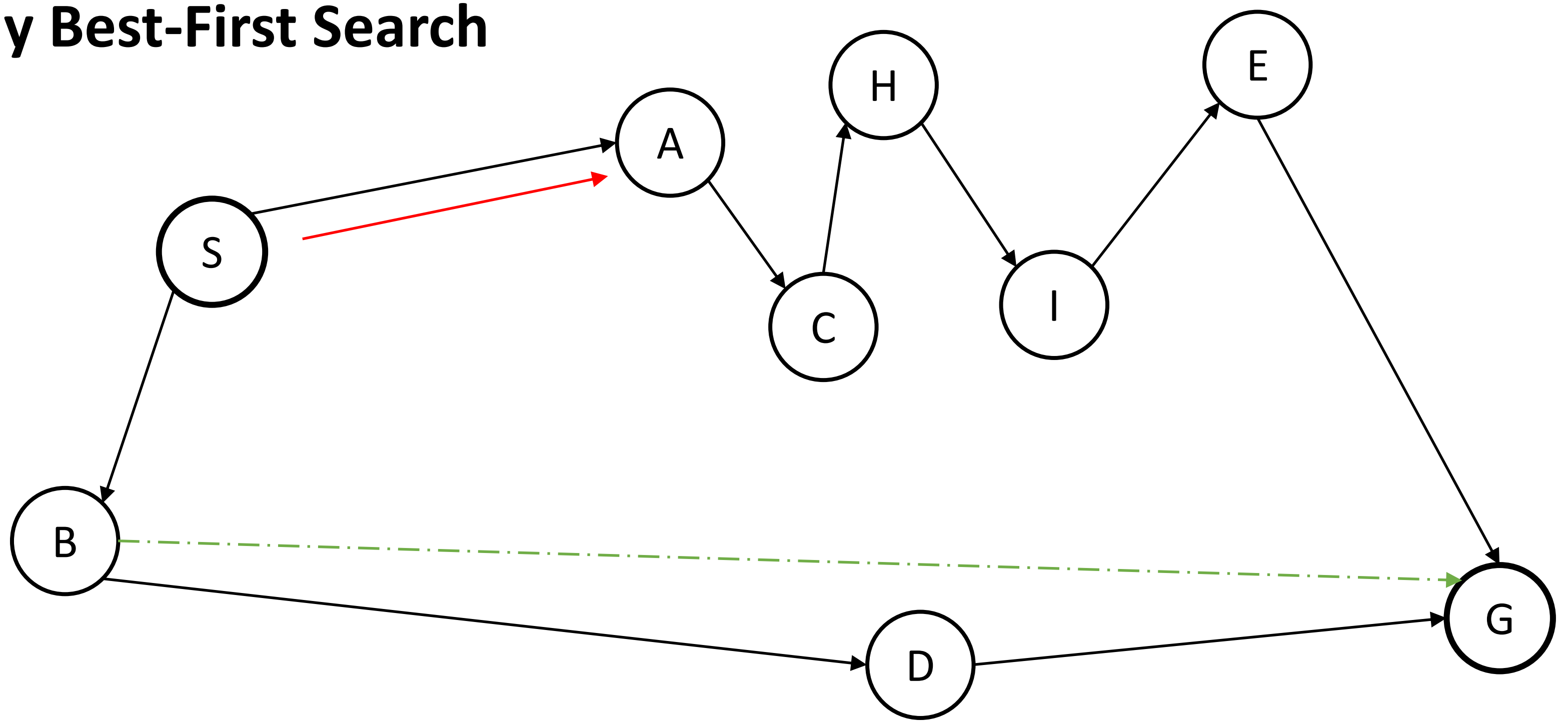
node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

- We have a solution, and fast but…is it optimal?
- Is this search strategy complete?
  Answer: Yes for finite spaces

If we call m the maximum depth of the search space

- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$

Greedy Best-First Search



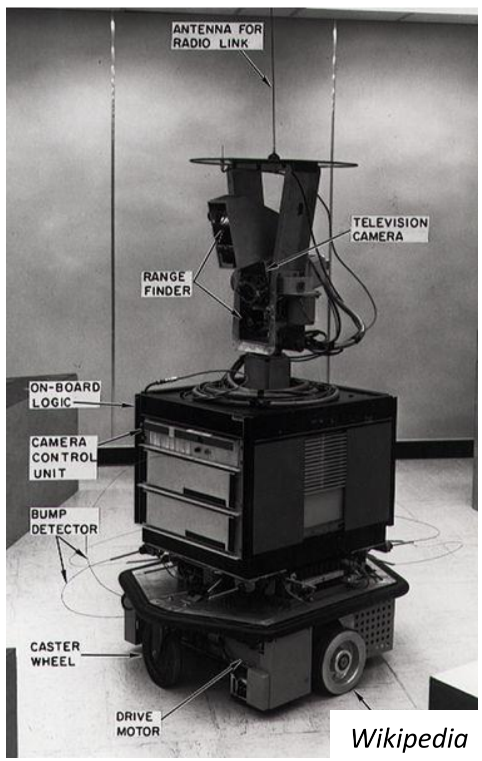
- We have a solution, and fast but…is it optimal?
- Is this search strategy complete?
  Answer: Yes for finite spaces

If we call m the maximum depth of the search space

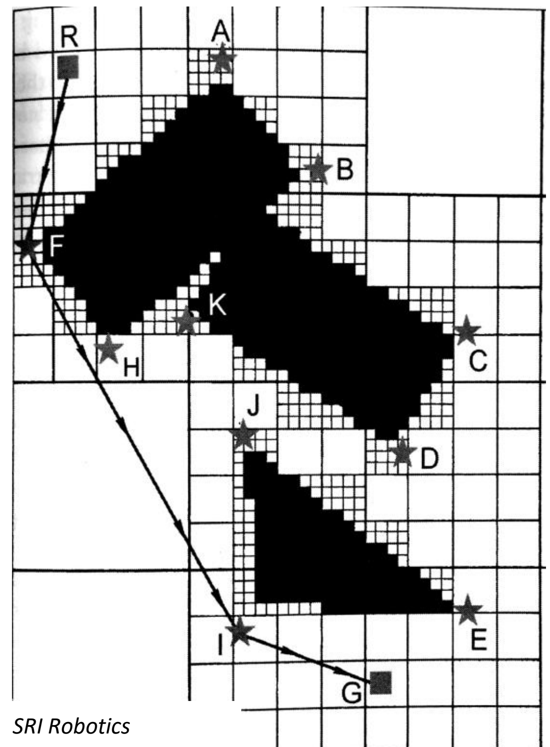
- Time complexity: $O(b^m)$
- Space complexity: $O(b^m)$

A*

- The informed version of UCS is called A*
- Very popular search algorithm
- It was born in the early days of mobile robotics when, in 1968, Nilsson, Hart, and Raphael had to face a practical problem with Shakey (one of the ancestors of today's mobile robots)



Wikipedia



SRI Robotics

A*

- The idea behind A* is simple: perform a UCS, but instead of considering accumulated costs consider the following:

Heuristic
("cost-to-go")

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

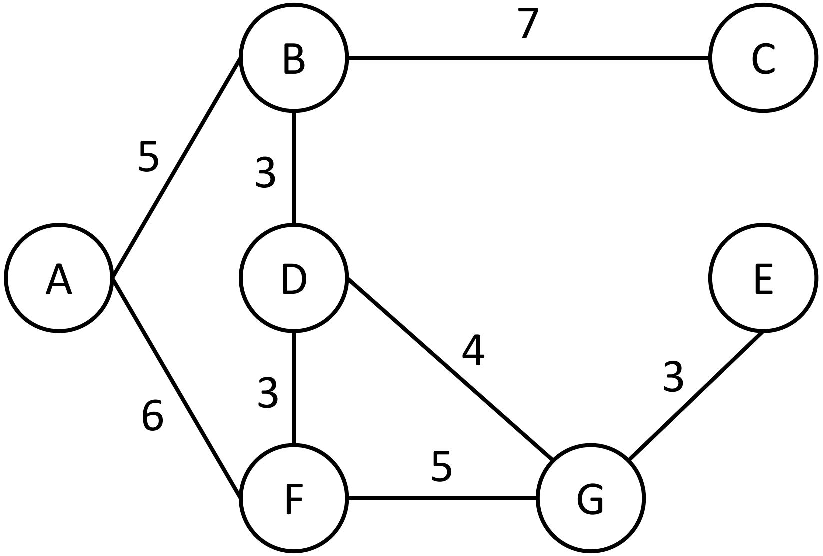
Cost accumulated
on the path to n
("cost-to-come")

- To guarantee that the search is sound and complete we need to require that the heuristic is **admissible**: it is an optimistic estimate or, more formally:

$$h(n) \leq \text{ Cost of the minimum path from n to the goal}$$

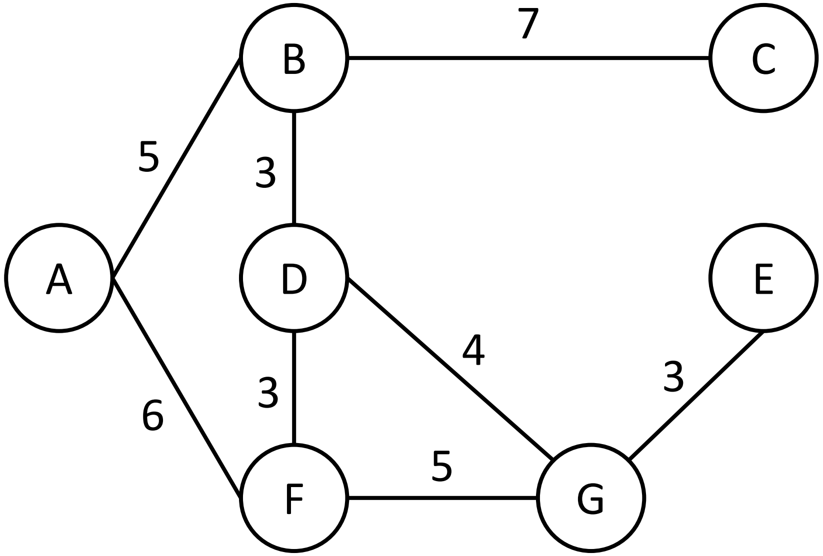
- If the heuristic is not admissible we might discard a path that could actually turn out to be better that the best candidate found so far

A*



node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

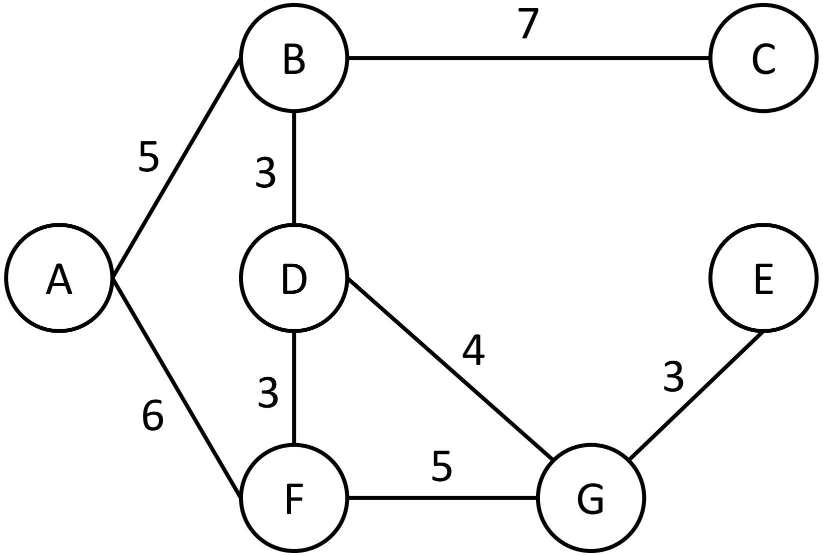
A*

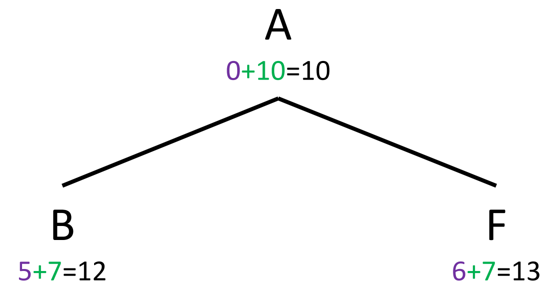


A
0+10=10

node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

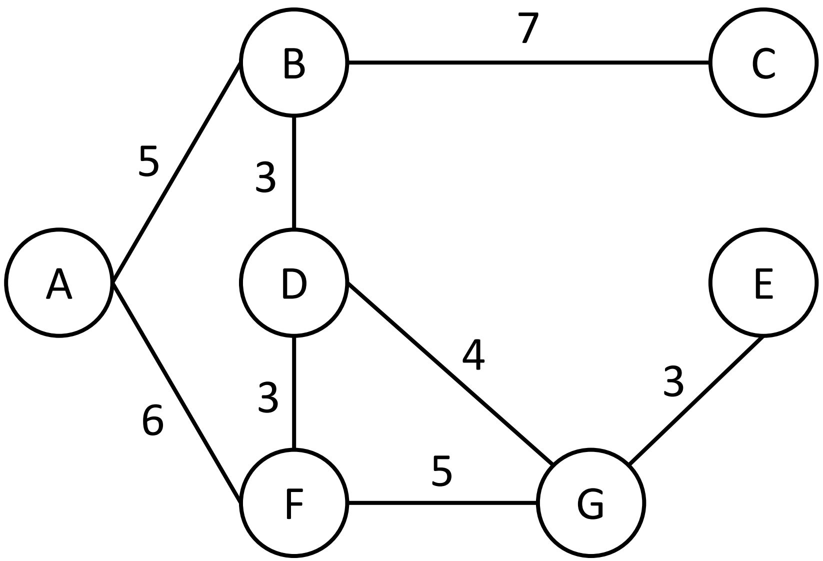
A*

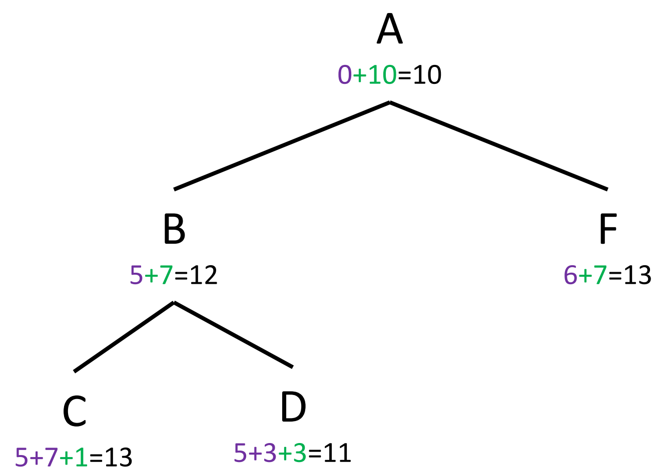




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

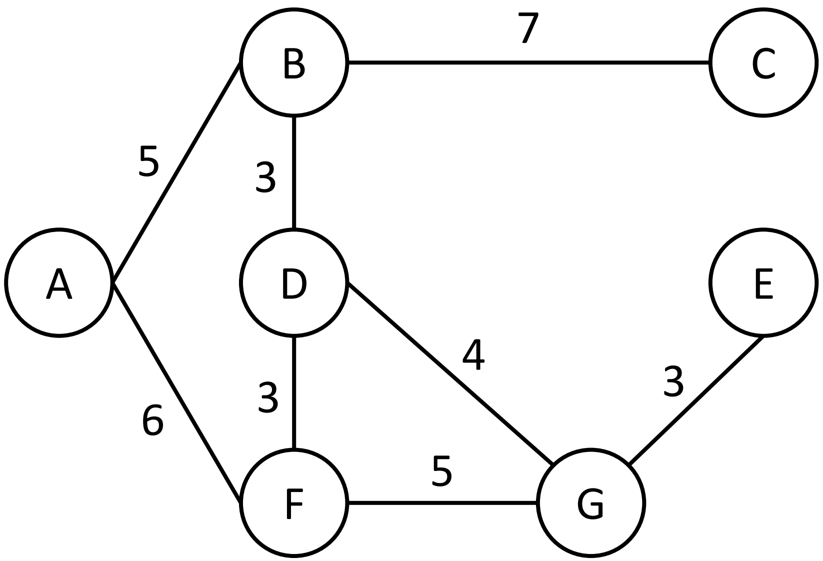
A*

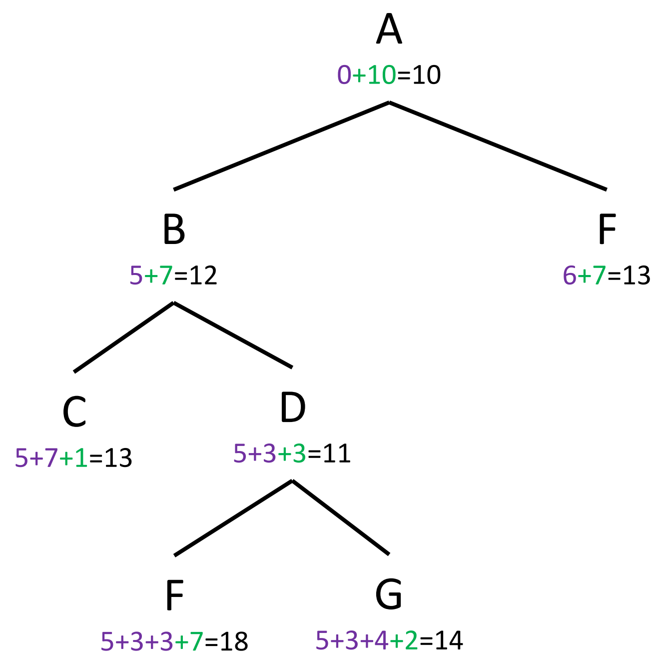




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

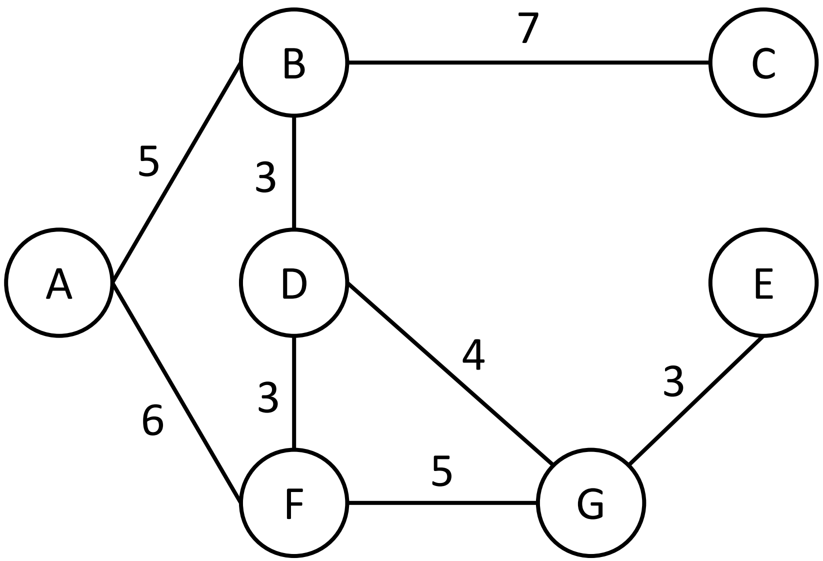
A*

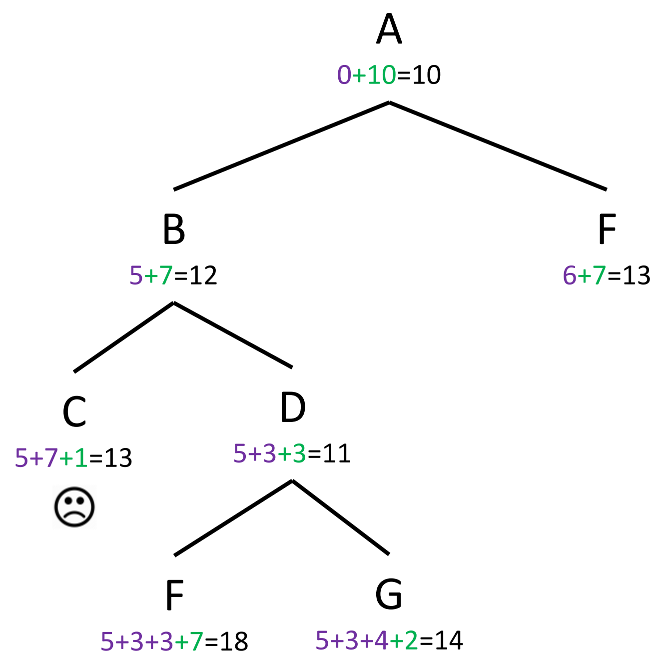




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

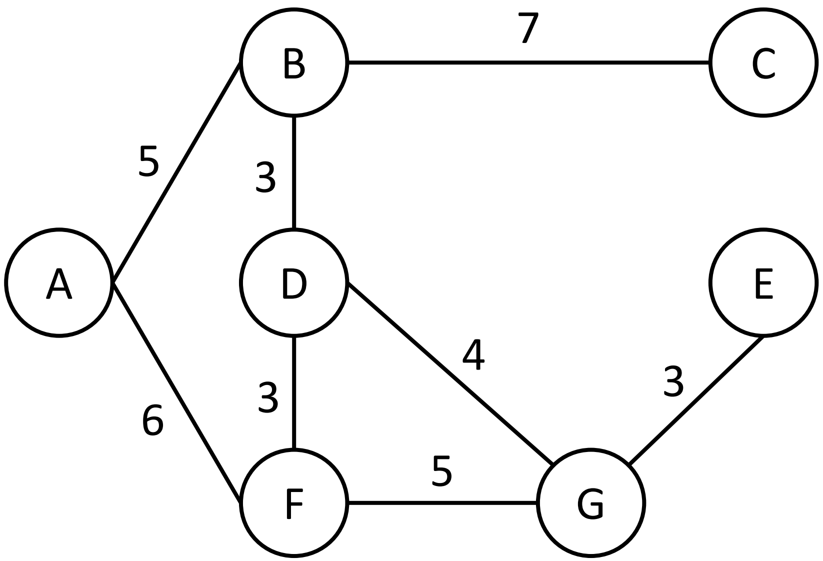
A*

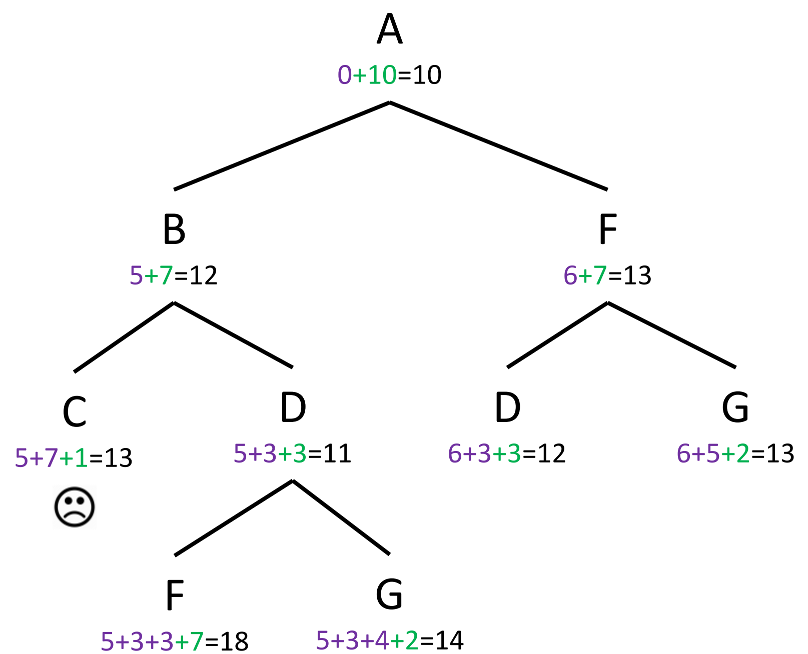




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

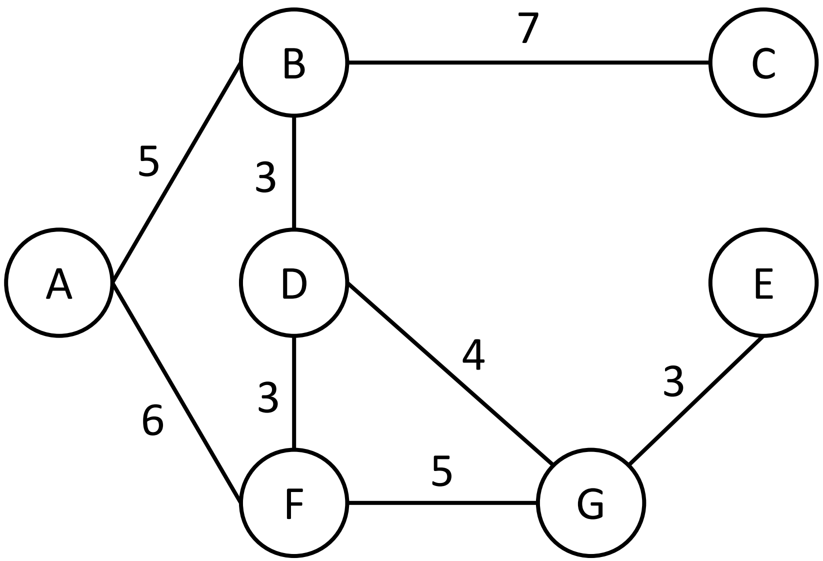
A*

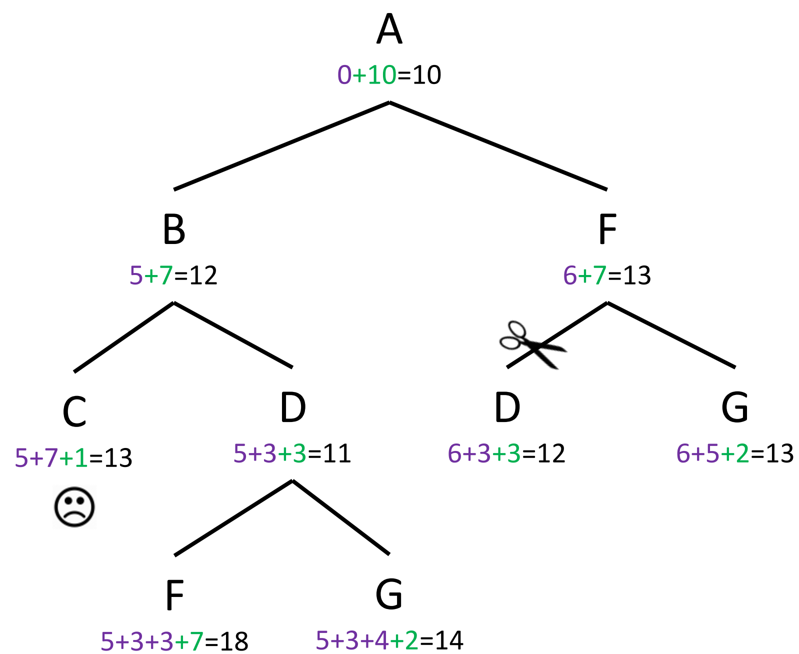




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

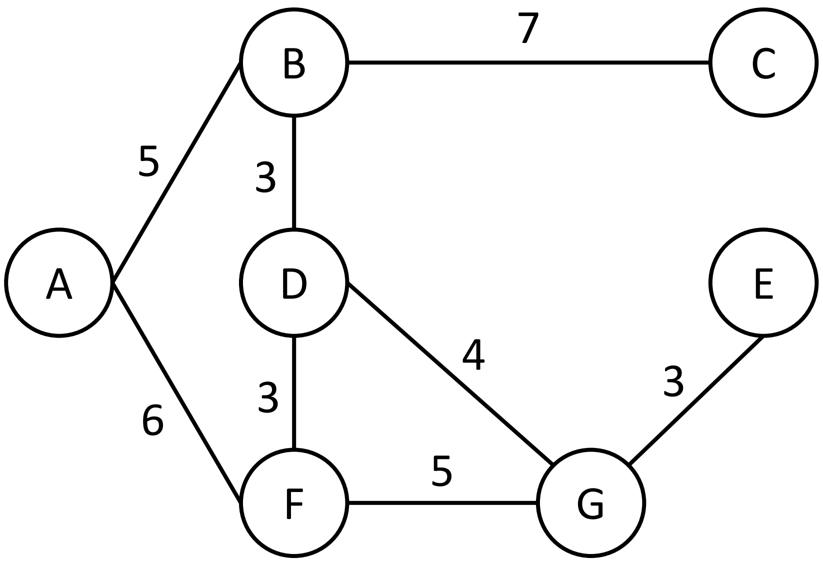
A*

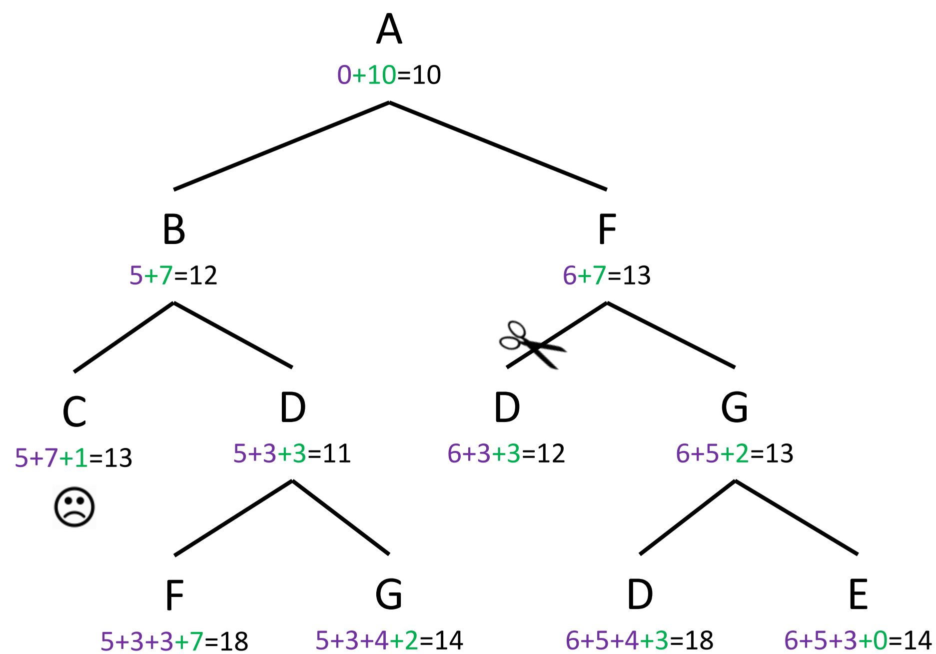




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

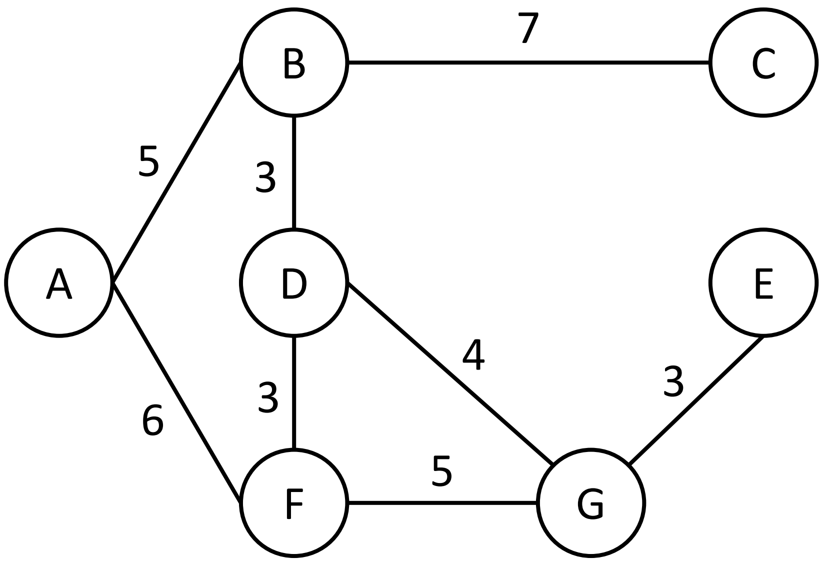
A*

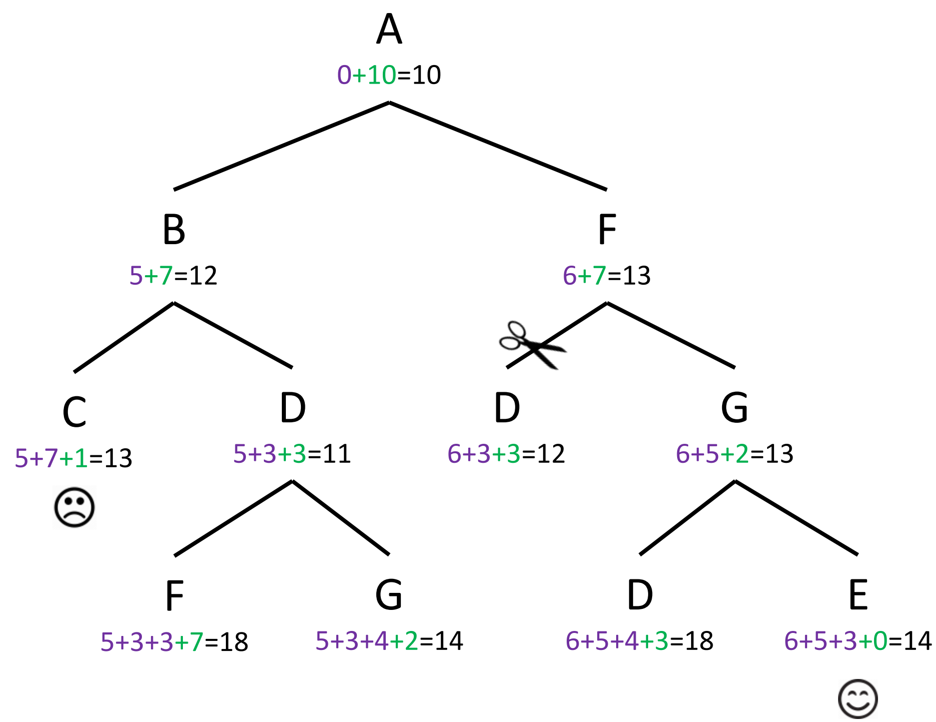




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

A*

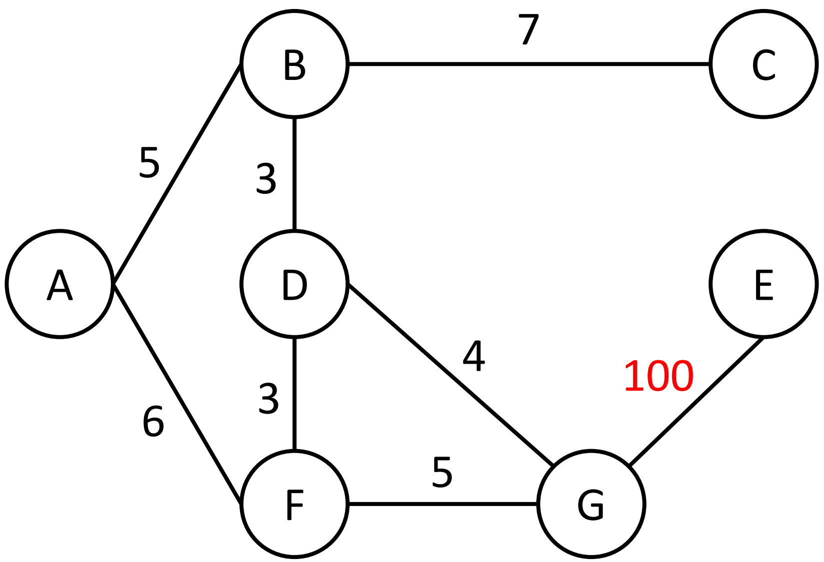




node v	$h(v)$
A	10
B	7
C	1
D	3
E	0
F	7
G	2

A*

- Problem: if we work with an extended list, admissibility is not enough!
- Let's consider this "pathological" instance:

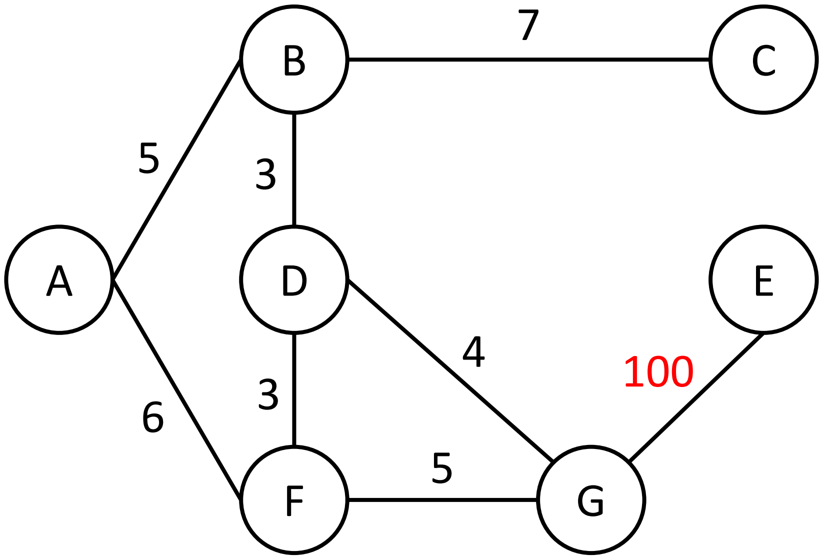


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!

- Let’s consider this “pathological" instance:



A

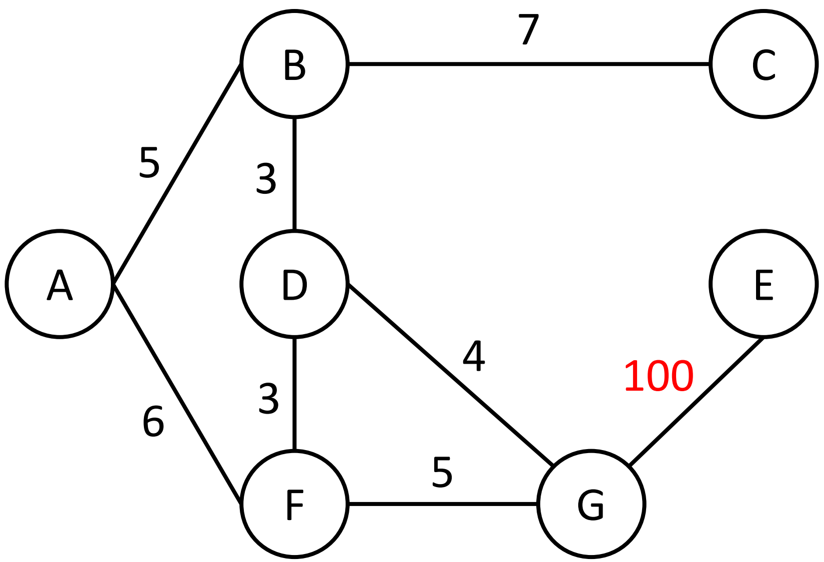
0+10=10

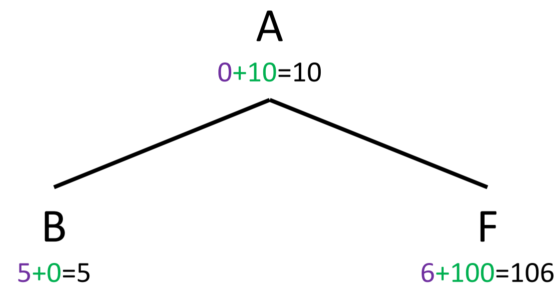
node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!

- Let's consider this "pathological" instance:



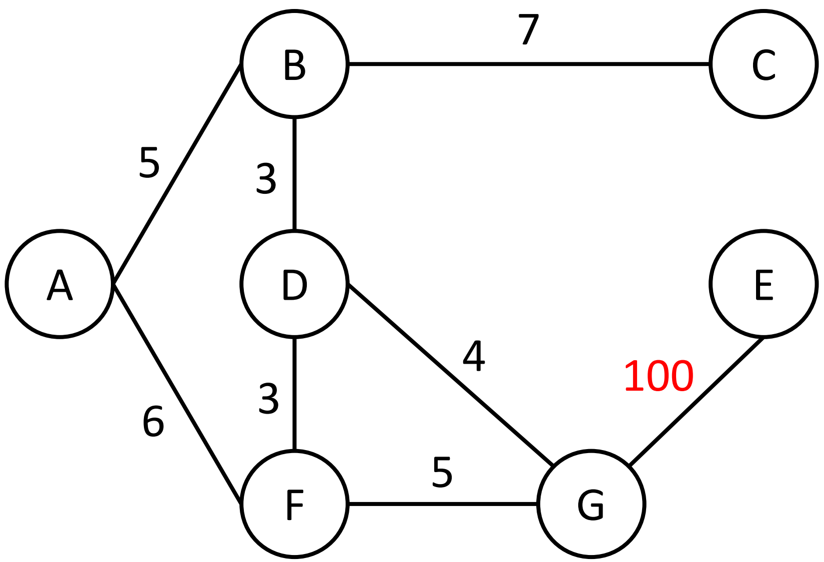


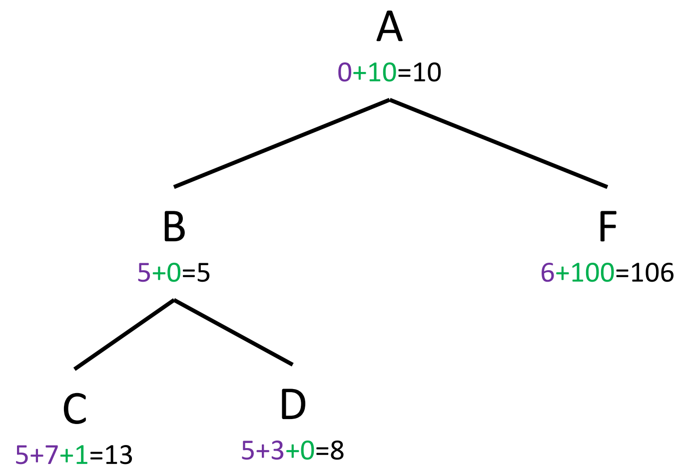
node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!

- Let's consider this "pathological" instance:



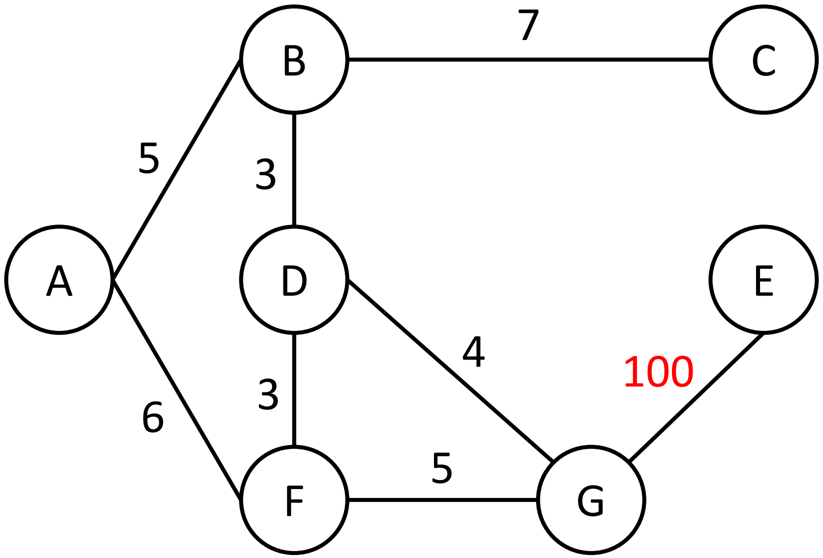


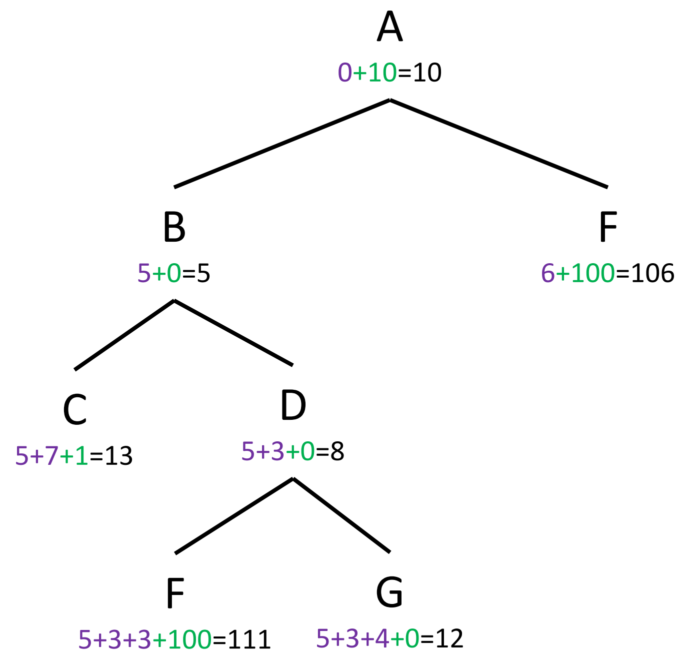
node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!

- Let's consider this "pathological" instance:



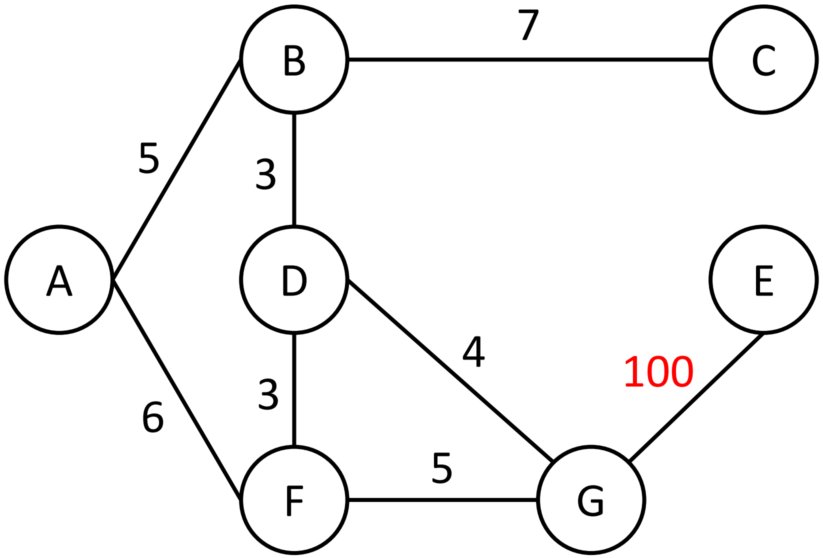


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

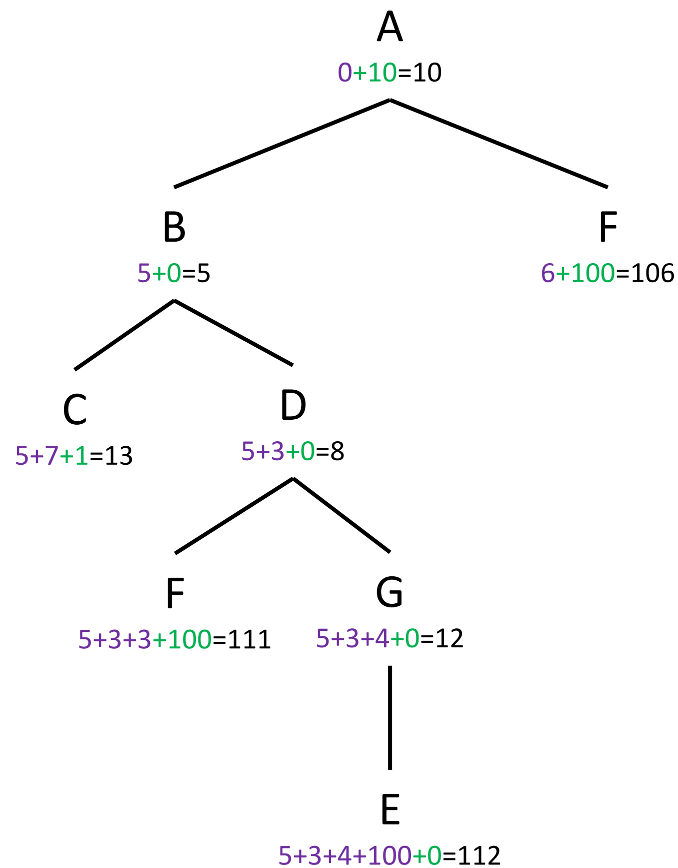
A*

- Problem: if we work with an extended list, admissibility is not enough!

- Let's consider this "pathological" instance:

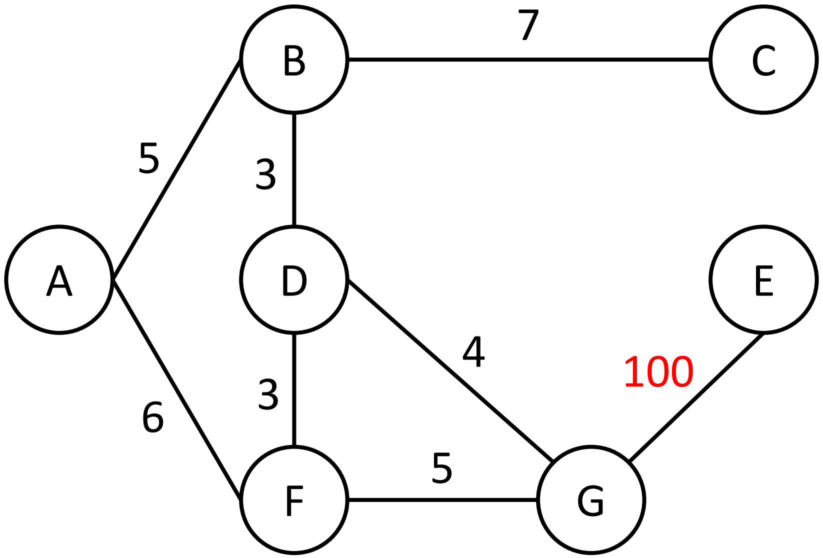


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

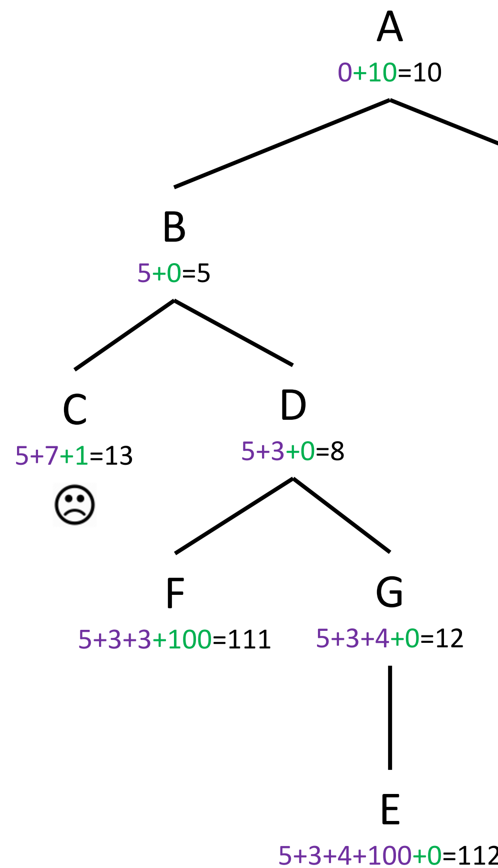


A*

- Problem: if we work with an extended list, admissibility is not enough!
- Let's consider this "pathological" instance:

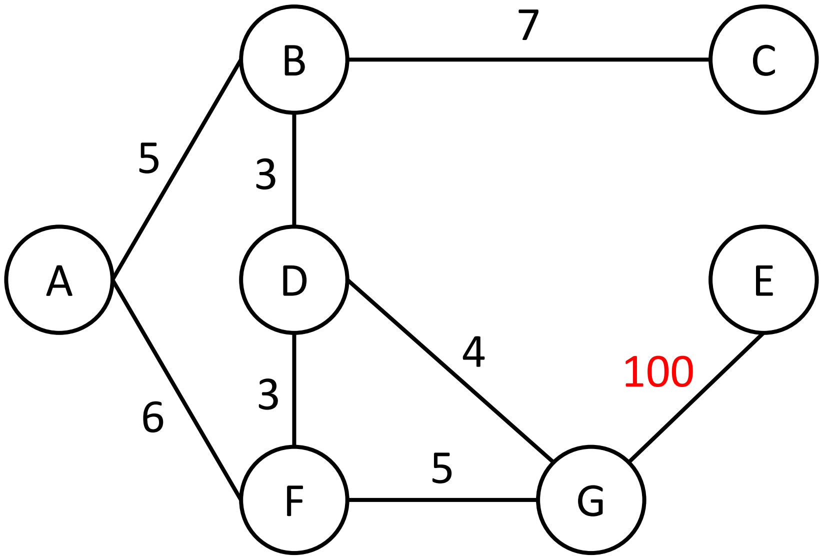


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

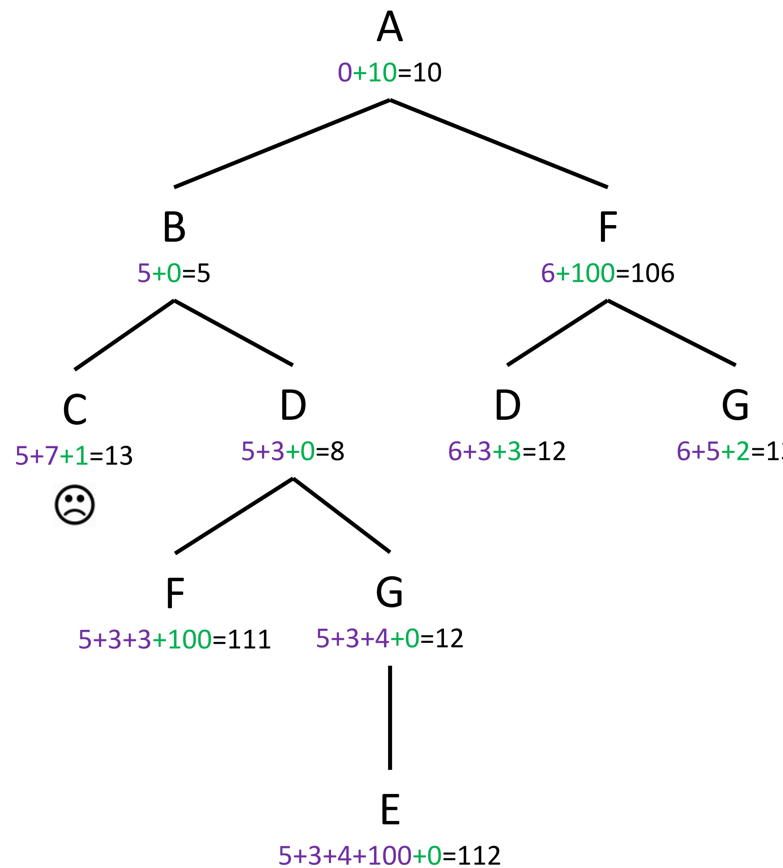


A*

- Problem: if we work with an extended list, admissibility is not enough!
- Let’s consider this “pathological" instance:



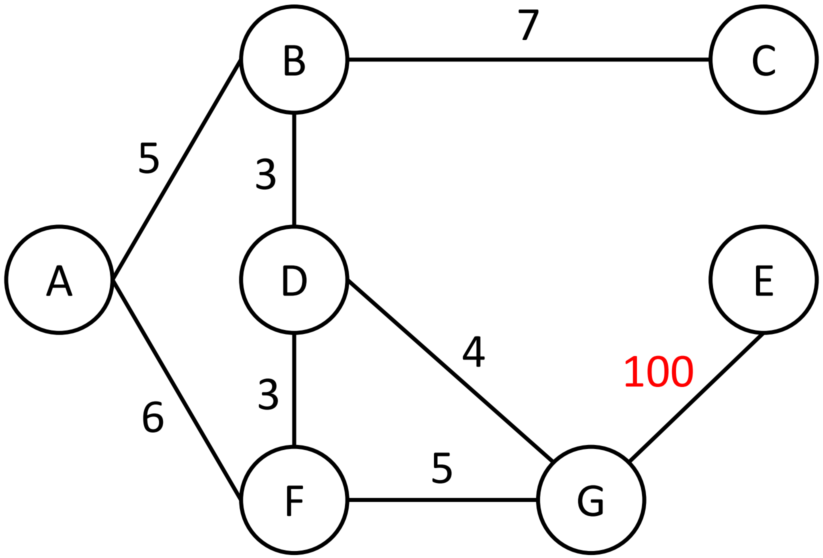
node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

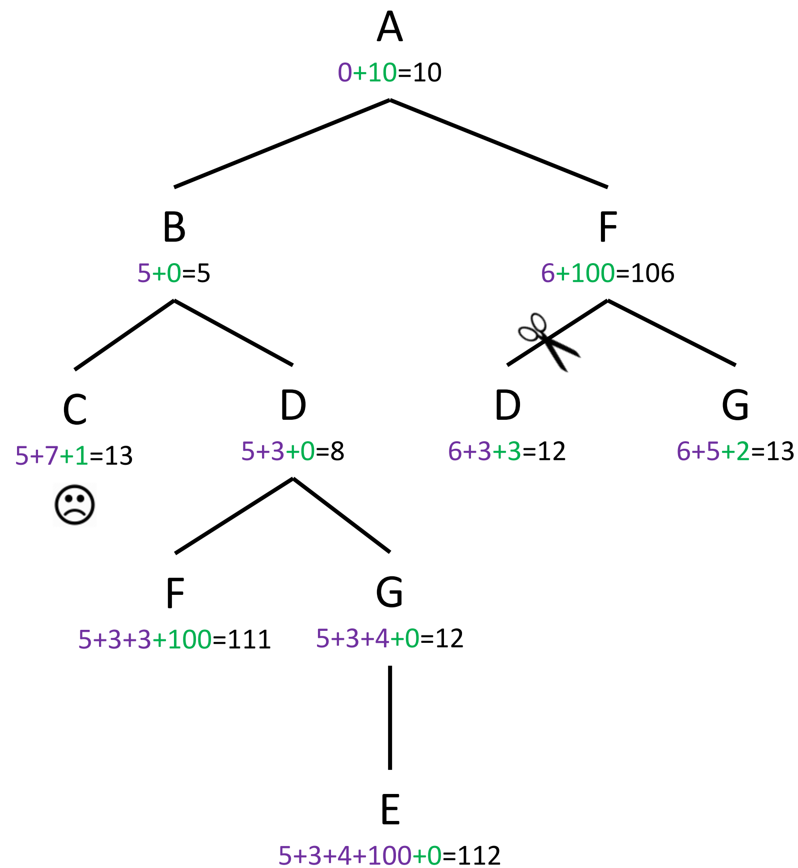


A*

- Problem: if we work with an extended list, admissibility is not enough!

- Let's consider this "pathological" instance:

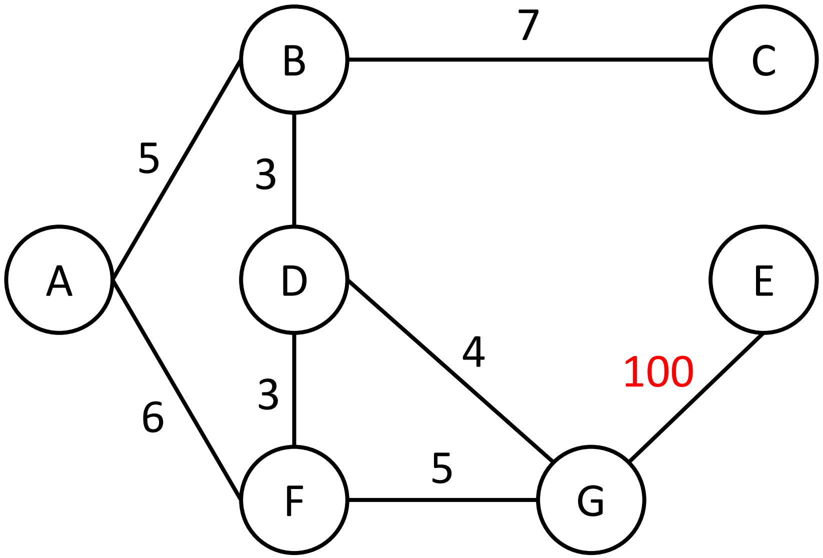


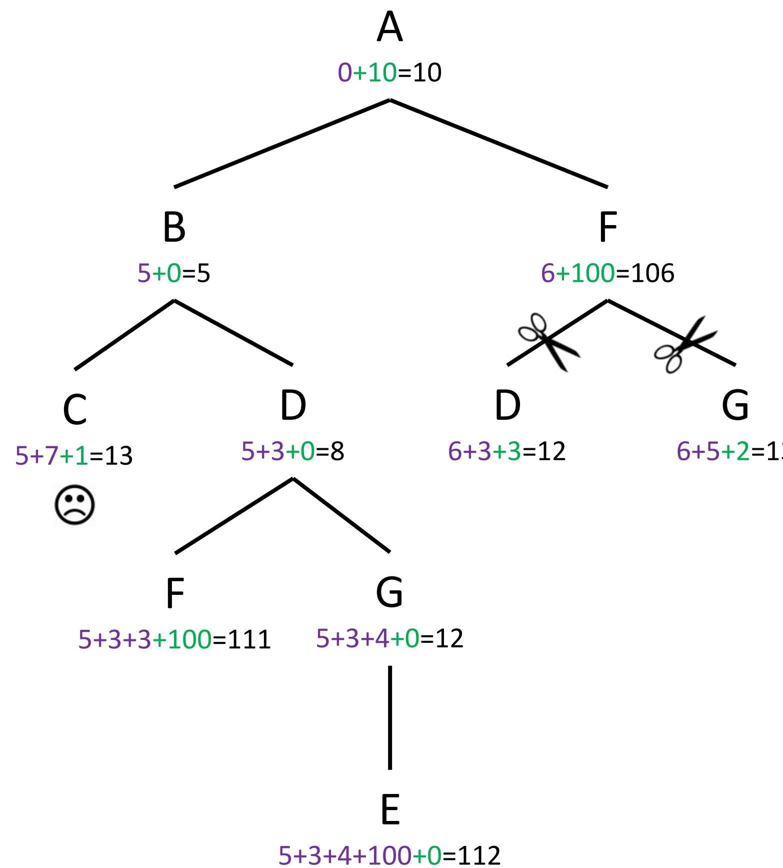


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!
- Let’s consider this “pathological" instance:

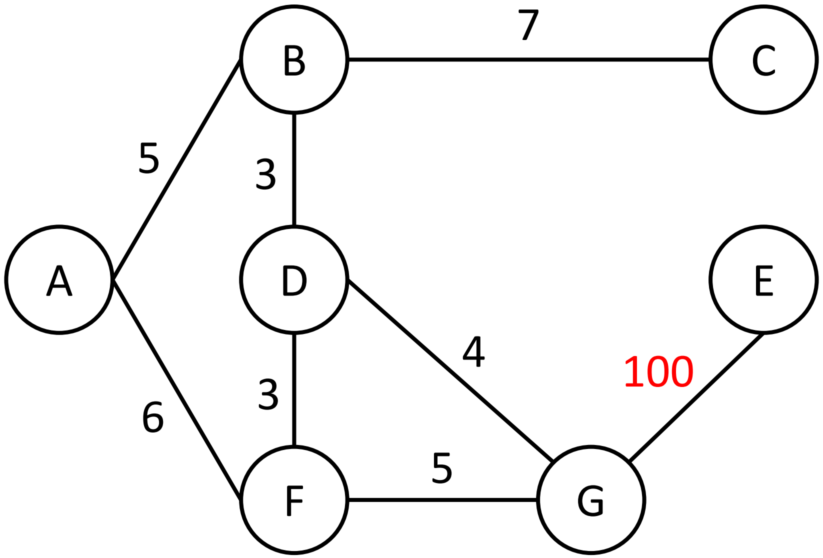


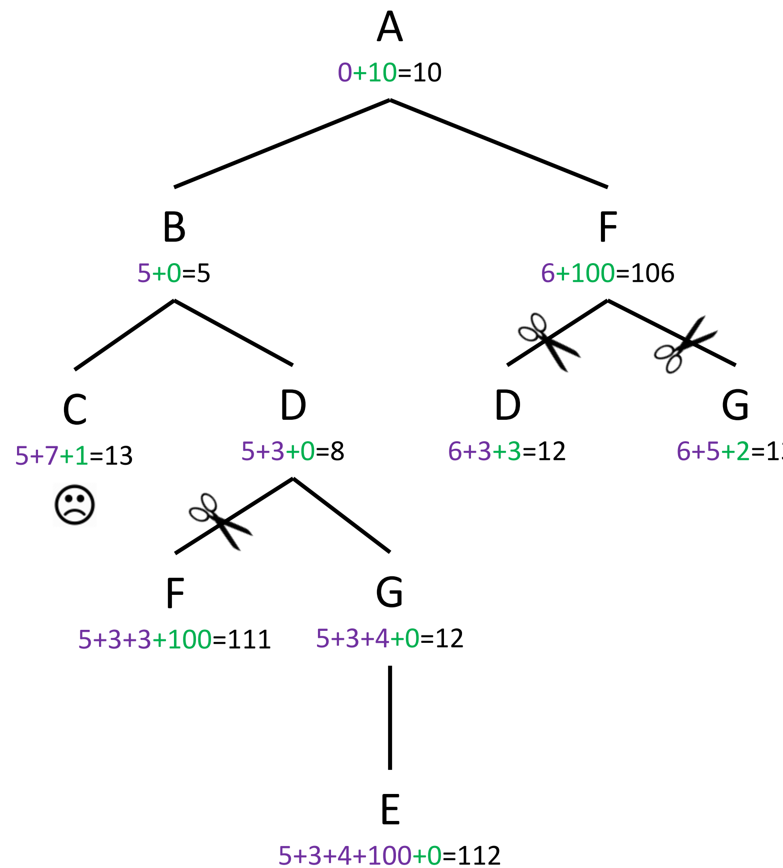


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!
- Let's consider this "pathological" instance:

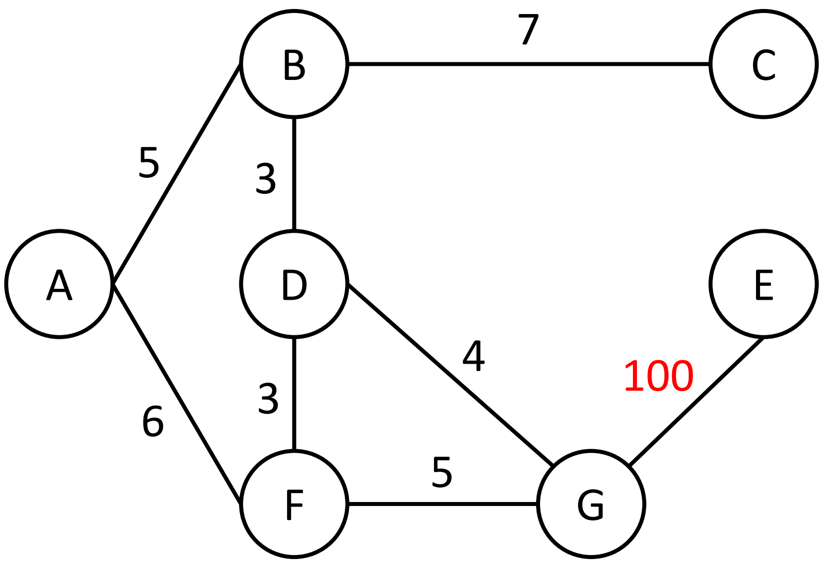


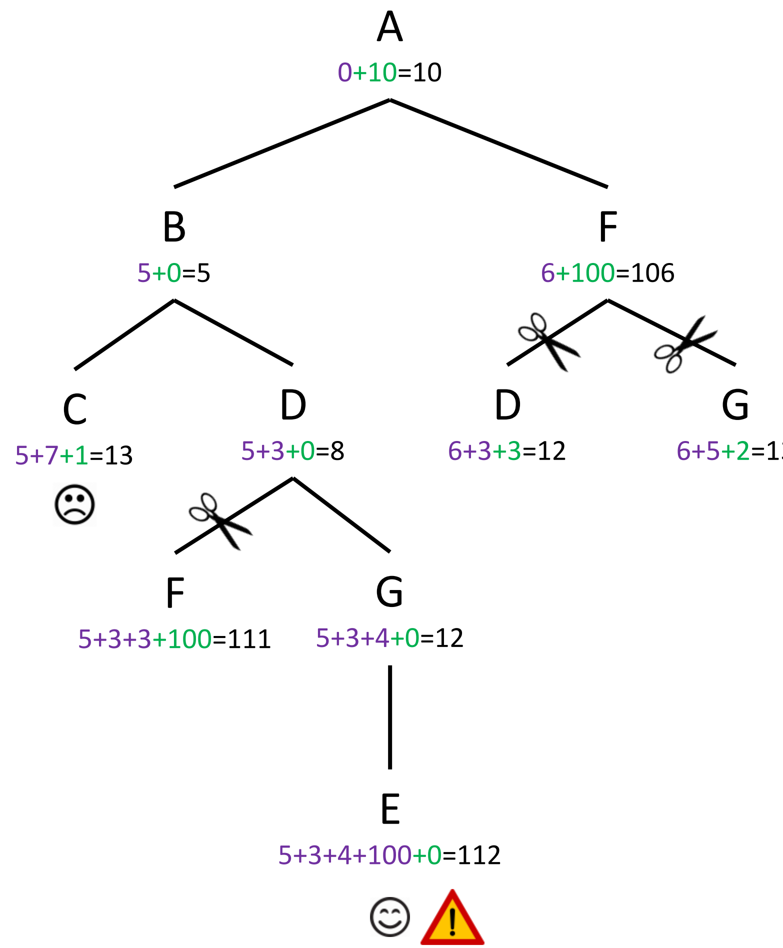


node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

A*

- Problem: if we work with an extended list, admissibility is not enough!
- Let's consider this "pathological" instance:

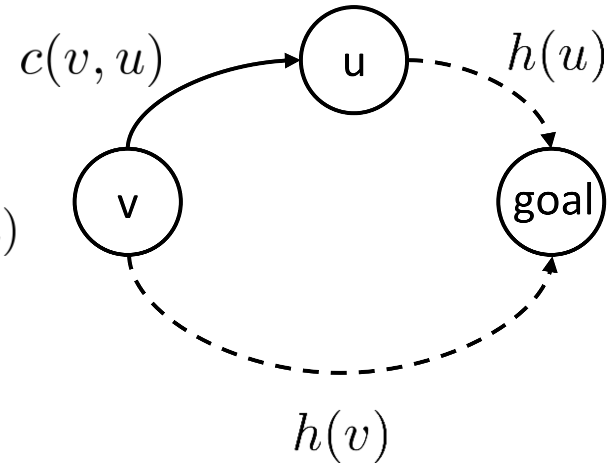




node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0

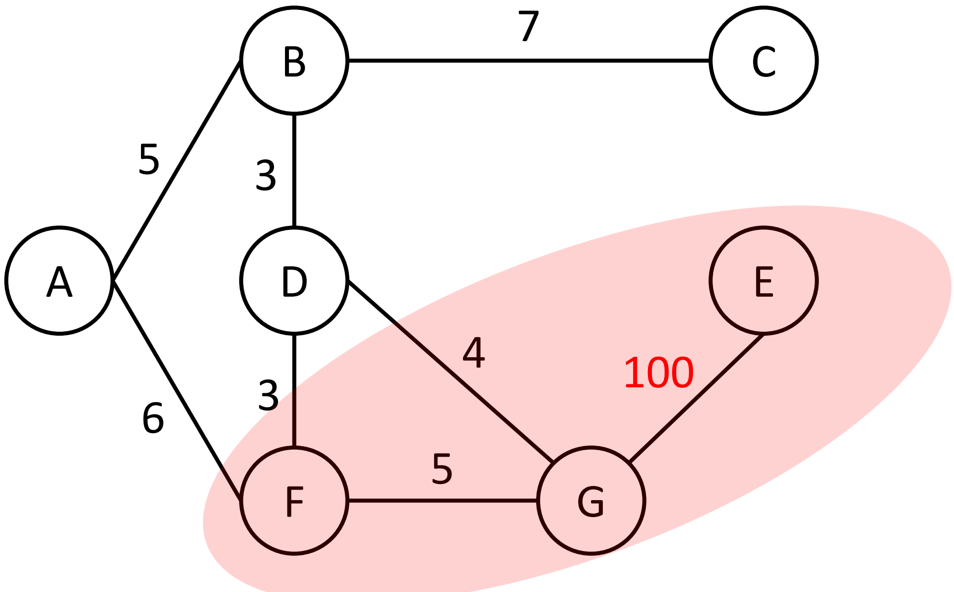
A*

- We need to require a stronger property: **consistency**
- For any connected nodes u and v: $h(v) \leq c(v,u) + h(u)$
- A consistent heuristic is also admissible
  (as it is a stricter requirement)

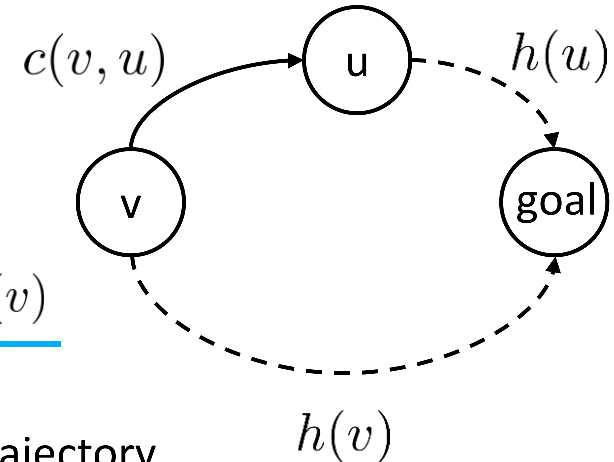


- It's a sort of triangle inequality, let's reconsider our pathological instance:

node v	$h(v)$
A	10
B	0
C	1
D	0
E	0
F	100
G	0



Optimality of A*



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

$$f(u) = \overline{g(u)} + h(u) = \overline{g(v) + c(v,u)} + h(u) \geq g(v) + h(v)$$

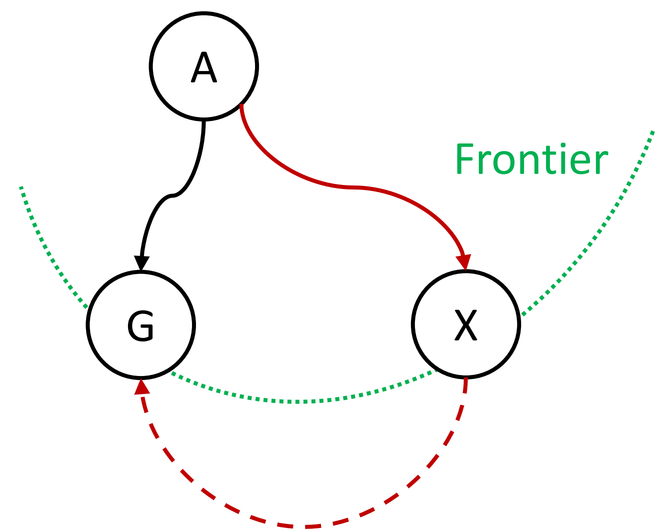
consistency

$f(u) \geq f(v)$ ⟶ f is non-decreasing along any search trajectory

Hypotheses:

1. A* selects from the frontier a node G that has been generated through a path p
2. p is not the optimal path to G

Given 2 and the frontier separation property, we know that there must exist a node X on the frontier that is on a better path to G



f is non-decreasing: $f(G) \geq f(X)$

A* selected G: $f(G) < f(X)$

When A selects a node for expansion, it discovers the optimal path to that node*

Building good heuristics

- Good heuristics allows to limit the number of states explored before finding the solution
- The “larger heuristics are better” principle is not a methodology to define a good heuristic

- Such a task, seems to be rather complex: heuristics deeply leverage the inner structure of a problem and have to satisfy a number of constraints (admissibility, consistency, efficiency) whose guarantee is not straightforward

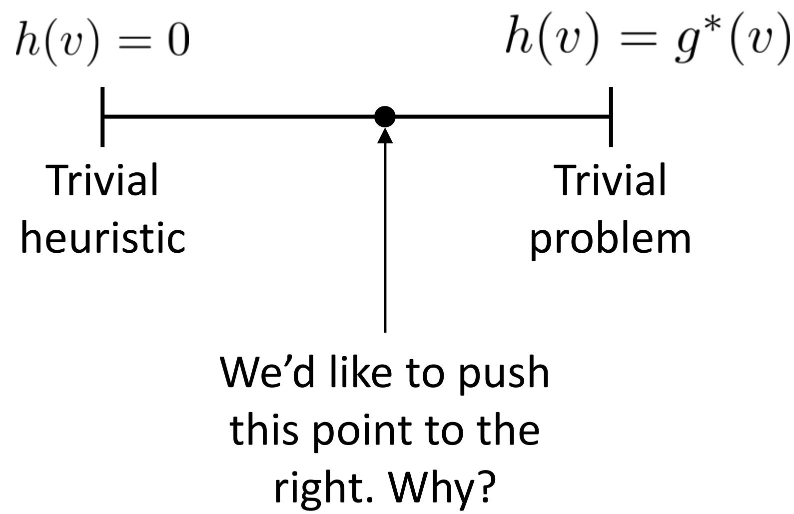
- When we adopted the straight-line distance in our route finding examples, we were sure it was a good heuristic

- Would it be possible to generalize what we did with the straight-line distance to define a method to *compute* heuristics for a problem?

- Good news: the answer is yes

Evaluating heuristics

- How to evaluate if an heuristic is good?



- A* will expand all nodes v such that: $f(v) < g^*(goal) \longrightarrow h(v) < g^*(goal) - g(v)$
- If, for any node v $h_1(v) \leq h_2(v)$

  then A* with h_2 will not expand more nodes than A* with h_1, in general h_2 is better (provided that is consistent and can be computed by an efficient algorithm)

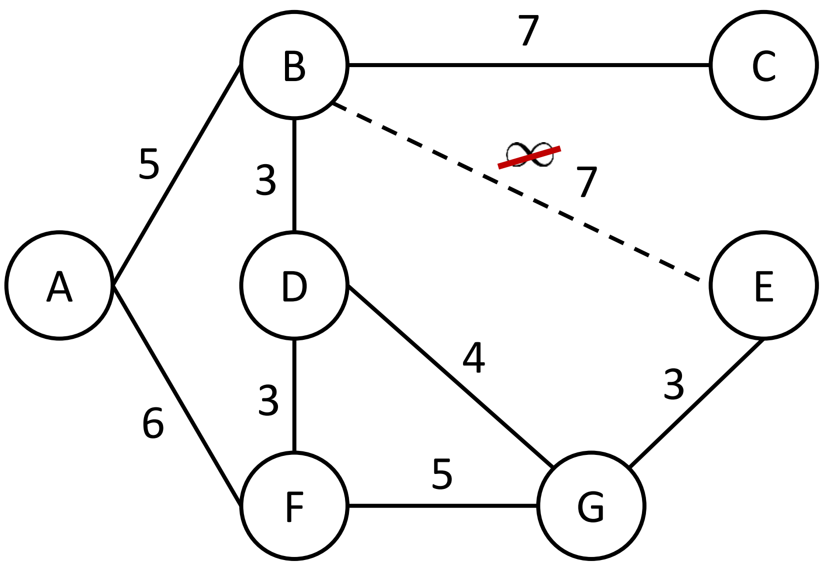
- If we have two consistent heuristics h_1 and h_2 we can define

$$h_3(v) = \max\{h_2(v), h_1(v)\}$$

Relaxed problems

- Given a problem P, a relaxation of P is an easier version of P where some constraints have been dropped

P Original problem $\xrightarrow{\text{Removing constraints}}$ $\hat{P}$ Relaxed problem



$$\hat{g}(v,u) \leq g(v,u)$$

Costs in the relaxation

Costs in the original problem

- In our route finding problems removing the constraint that movements should be over roads (links) means that some costs pass from an infinite value to a finite one (the straight-line distance)

Relaxed problems

- Idea:

Define a relaxation of P: $\hat{P}$ $\longrightarrow$ Apply A* to every node and get $\hat{h}^*(v)$ $\longrightarrow$ Set $h(v) = \hat{h}^*(v)$ in the original problem and run A*

- We can easily define a problem relaxation, it’s just matter of removing constraints/rewriting costs

- But what happens to soundness and completeness of A*?

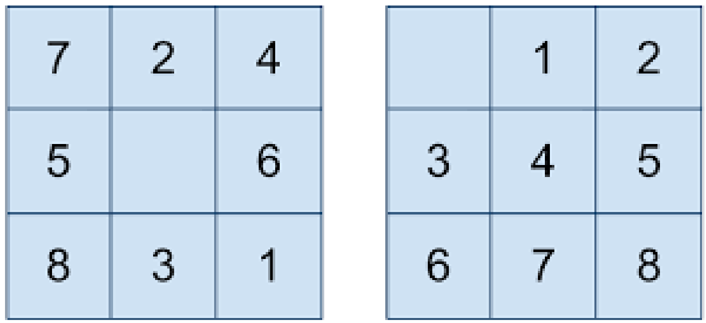
$\hat{h}^*(v) \leq \hat{g}(v,u) + \hat{h}^*(u)$ Path costs are optimal

$h(v) \leq \hat{g}(v,u) + h(u)$ From our idea

$\hat{g}(v,u) \leq g(v,u)$ From the definition of relaxation

$h(v) \leq g(v,u) + h(u)$ **h is consistent**

Heuristic function and relaxed problems: an example



Start State   Goal State

- Average solution: 22 steps and branching factor 3 = 10^{10} states using tree search
- h_1 = number of misplaced tiles ($h_1 = 8$ for the example)
- h_2 = sum of Manhattan distance of each tile from goal ($h_1 = 18$ for the example)
- Both heuristics are admissible (true solution costs 26)

How to evaluate A* and h? Compute the effective branching factor b*

Heuristic function and relaxed problems: an example

7	2	4
5		6
8	3	1

Start State

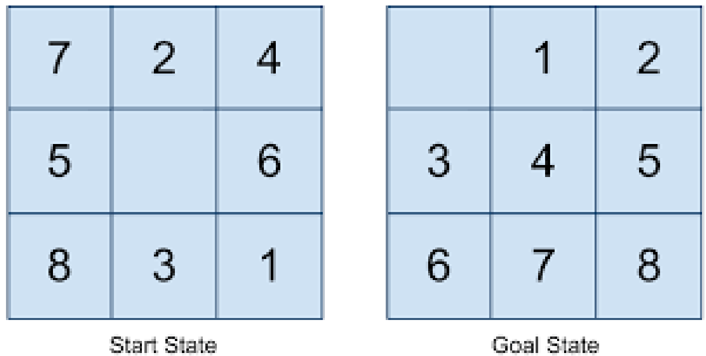
	1	2
3	4	5
6	7	8

Goal State

	Search Cost (nodes generated)			Effective Branching Factor		
d	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	3644035	227	73	2.78	1.42	1.24
14	–	539	113	–	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

- IDS = Iterative Deepening Search
- h_2 performs better than h_1 as it provides a higher estimate (still admissible and consistent, so lower than the true solution cost).

Heuristic function and relaxed problems: an example

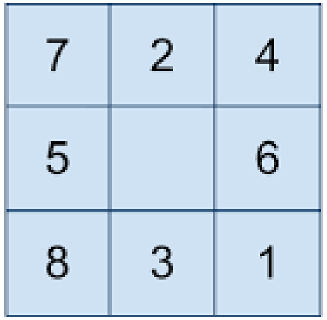


- A tile can move from square A to square B if
  A is horizontally or vertically adjacent to B **and** B is blank

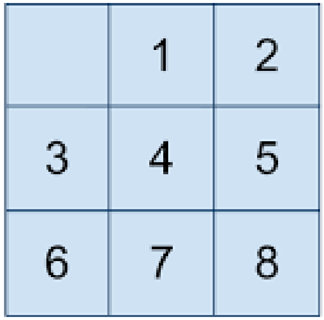
Relaxed problems:

1. A tile can move from square A to square B if A is adjacent to B
2. A tile can move from square A to square B if B is blank
3. A tile can move from square A to square B

Heuristic function and relaxed problems: an example

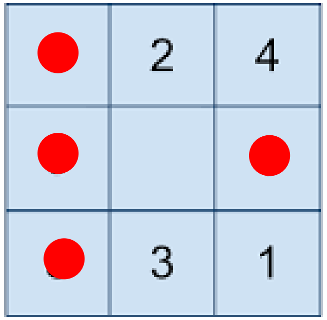


Start State

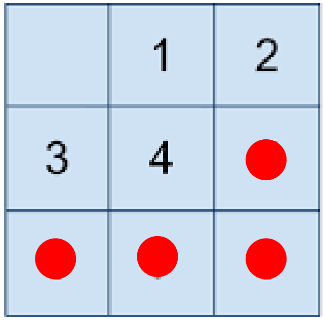


Goal State

Other heuristics can be derived from subproblems, as removing some of the tiles and solving an easier game: cost to solve the easier game = h



Start State



Goal State

Heuristic function and relaxed problems: an example

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

Other alternative to find heuristics are to:

- Learn from experience: play many 8-puzzles, learn a good heuristic
  e.g., neural nets
- Use domain knowledge to extract features:
  e.g. “number of misplaced tiles”

Informed vs non-informed search

- We can enrich DFS and BFS to obtain their an informed versions
- Both search methods break ties in lexicographical order, but it seems reasonable to do that in favor of nodes that are believed to be closer to the goal
- **Hill climbing**
  - A DFS where ties are broken in favor the node with smallest h
- **Beam** (of width w)
  - A BFS where at each level we keep the first w nodes in increasing order of h

References

- Russel S., Norvig P., Artificial Intelligence, a Modern Approach, III ED
- LaVallе, SM., Planning Algorithms
  http://lavalle.pl/planning/
- https://qiao.github.io/PathFinding.js/visual/
- https://www.redblobgames.com/pathfinding/a-star/introduction.html

Sistemi Intelligenti Avanzati
Corso di Laurea in Informatica, A.A. 2020-2021
Università degli Studi di Milan

Matteo Luperto
Dipartimento di Informatica
matteo.luperto@unimi.it