Problem solving by search II

Tomáš Svoboda

Department of Cybernetics, Vision for Robots and Autonomous Systems

February 26, 2019

Outline

- Graph search
- Heuristics (how to search faster)
- Greedy
- A*. A-star search.

Recap: Search

A tree search recap

function TREE_SEARCH(problem) **return** a solution or failure

initialize the frontier the initial state of the problem

loop

if the frontier is empty **then return** failure

else choose a node from frontier and remove from frontier

end if

if the node contains a goal state **then return** the solution

end if

Expand the node and add the resulting nodes to frontier

end loop

end function

A tree search recap

function TREE_SEARCH(problem) **return** a solution or failure
    initialize the frontier the initial state of the problem
    loop
        if the frontier is empty **then return** failure
        else choose a node from frontier and remove from frontier
        end if
        if the node contains a goal state **then return** the solution
        end if
        Expand the node and add the resulting nodes to frontier
    end loop
end function

A tree search recap

function TREE_SEARCH(problem) **return** a solution or failure
    initialize the frontier the initial state of the problem
    loop
        if the frontier is empty **then return** failure
        else choose a node from frontier and remove from frontier
        end if
        if the node contains a goal state **then return** the solution
        end if
        Expand the node and add the resulting nodes to frontier
    end loop
end function

A tree search recap

function TREE_SEARCH(problem) **return** a solution or failure
    initialize the frontier the initial state of the problem
    loop
        if the frontier is empty **then return** failure
        else choose a node from frontier and remove from frontier
        end if
        if the node contains a goal state **then return** the solution
        end if
        Expand the node and add the resulting nodes to frontier
    end loop
end function

A tree search recap

function TREE_SEARCH(problem) **return** a solution or failure
    initialize the frontier the initial state of the problem
    loop
        if the frontier is empty **then return** failure
        else choose a node from frontier and remove from frontier
        end if
        if the node contains a goal state **then return** the solution
        end if
        Expand the node and add the resulting nodes to frontier
    end loop
end function

A tree search recap

function TREE_SEARCH(problem) **return** a solution or failure
    initialize the frontier the initial state of the problem
    loop
        if the frontier is empty **then return** failure
        else choose a node from frontier and remove from frontier
        end if
        if the node contains a goal state **then return** the solution
        end if
        Expand the node and add the resulting nodes to frontier
    end loop
end function

A tree search recap

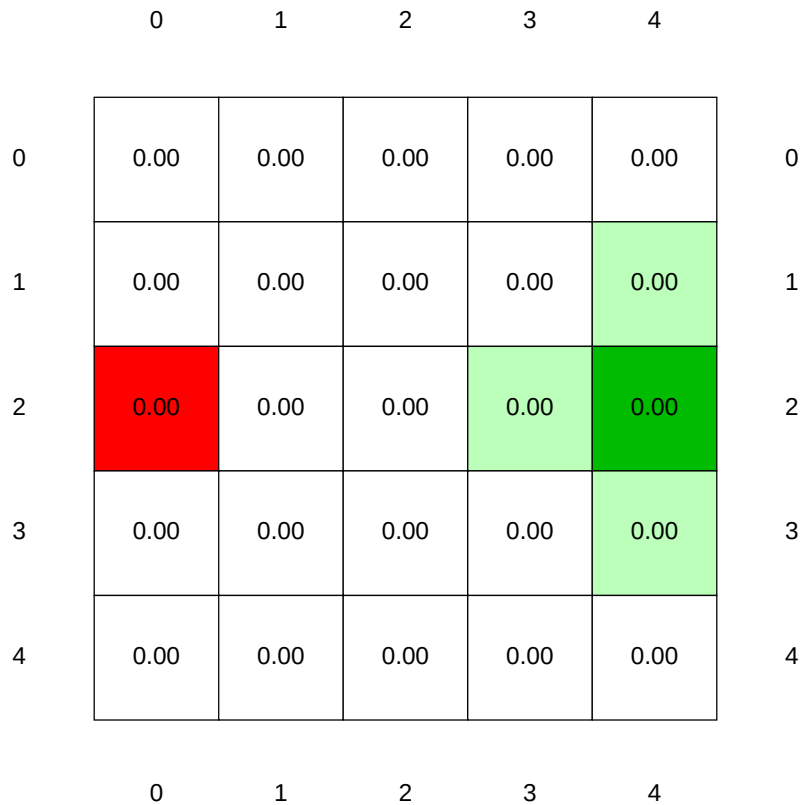
function TREE_SEARCH(problem) **return** a solution or failure
    initialize the frontier the initial state of the problem
    loop
        if the frontier is empty **then return** failure
        else choose a node from frontier and remove from frontier
        end if
        if the node contains a goal state **then return** the solution
        end if
        Expand the node and add the resulting nodes to frontier
    end loop
end function

A Maze, what could possibly go wrong?

	0	1	2	3	4	
0	0.00	0.00	0.00	0.00	0.00	0
1	0.00	0.00	0.00	0.00	0.00	1
2	0.00	0.00	0.00	0.00	0.00	2
3	0.00	0.00	0.00	0.00	0.00	3
4	0.00	0.00	0.00	0.00	0.00	4
	0	1	2	3	4	

Tree search the maze

```
function TREE_SEARCH(env) return a solution or failure
    initialize the frontier
    while frontier do
        node = frontier.pop()
        if goal in node then
            break
        end if
        nodes = env.expand(node.state)
        Add nodes to frontier
    end while
end function
```



A graph search

function GRAPH_SEARCH(env) **return** a solution or failure

init frontier by the start state

initialize the explored set to be empty

while frontier **do**

node = frontier.pop()

if goal in node **then** break

end if

nodes = env.expand(node.state)

add node to explored

for all nodes **do**

if *node not in explored (or in frontier)* **then**

add nodes to frontier

end if

end for

end while

end function

Do not forget: node is not the same as state!

A graph search

function GRAPH_SEARCH(env) **return** a solution or failure
    init frontier by the start state
    initialize the explored set to be empty
    while frontier **do**
        node = frontier.pop()
        if goal in node **then** break
        end if
        nodes = env.expand(node.state)
        add node to explored
        for all nodes **do**
            if *node not in explored (or in frontier)* **then**
                add nodes to frontier
            end if
        end for
    end while
end function

Do not forget: node is not the same as state!

A graph search

function GRAPH_SEARCH(env) **return** a solution or failure
    init frontier by the start state
    initialize the explored set to be empty
    while frontier **do**
        node = frontier.pop()
        if goal in node **then** break
        end if
        nodes = env.expand(node.state)
        add node to explored
        for all nodes **do**
            if *node not in explored (or in frontier)* **then**
                add nodes to frontier
            end if
        end for
    end while
end function

Do not forget: node is not the same as state!

A graph search

function GRAPH_SEARCH(env) **return** a solution or failure
    init frontier by the start state
    initialize the explored set to be empty
    while frontier **do**
        node = frontier.pop()
        if goal in node **then** break
        end if
        nodes = env.expand(node.state)
        add node to explored
        for all nodes **do**
            if *node not in explored (or in frontier)* **then**
                add nodes to frontier
            end if
        end for
    end while
end function

Do not forget: node is not the same as state!

A graph search

```
function GRAPH_SEARCH(env) return a solution or failure
    init frontier by the start state
    initialize the explored set to be empty
    while frontier do
        node = frontier.pop()
        if goal in node then break
        end if
        nodes = env.expand(node.state)
        add node to explored
        for all nodes do
            if node not in explored (or in frontier) then
                add nodes to frontier
            end if
        end for
    end while
end function
```

Do not forget: node is not the same as state!

A graph search

```
function GRAPH_SEARCH(env) return a solution or failure
    init frontier by the start state
    initialize the explored set to be empty
    while frontier do
        node = frontier.pop()
        if goal in node then break
        end if
        nodes = env.expand(node.state)
        add node to explored
        for all nodes do
            if node not in explored (or in frontier) then
                add nodes to frontier
            end if
        end for
    end while
end function
```

⚠ **Do not forget:** node is not the same as state!

The BFS graph search

```
function BFS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        explored.add(node.state)                    ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                if child_node contains Goal then return child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function
```

The BFS graph search

```
function BFS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        explored.add(node.state)                    ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                if child_node contains Goal then return child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function
```

The BFS graph search

function BFS_GRAPH_SEARCH(env) **return** a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty **do**
        node ← frontier.pop()
        explored.add(node.state) ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes **do**
            if child_node.state not in explored or in frontier **then**
                if child_node contains Goal **then return** child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function

The BFS graph search

```
function BFS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        explored.add(node.state)                    ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                if child_node contains Goal then return child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function
```

The BFS graph search

```
function BFS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        explored.add(node.state)                    ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                if child_node contains Goal then return child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function
```

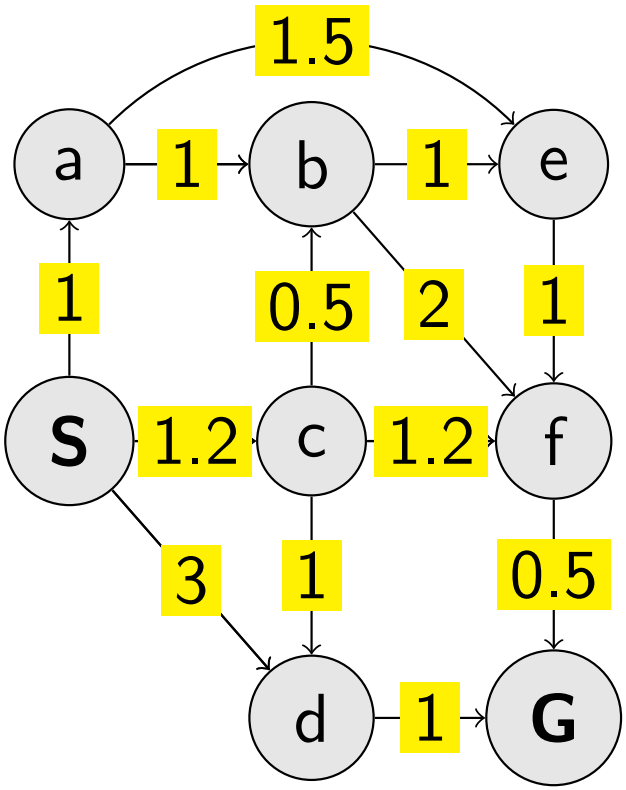
The BFS graph search

```
function BFS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        explored.add(node.state)                    ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                if child_node contains Goal then return child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function
```

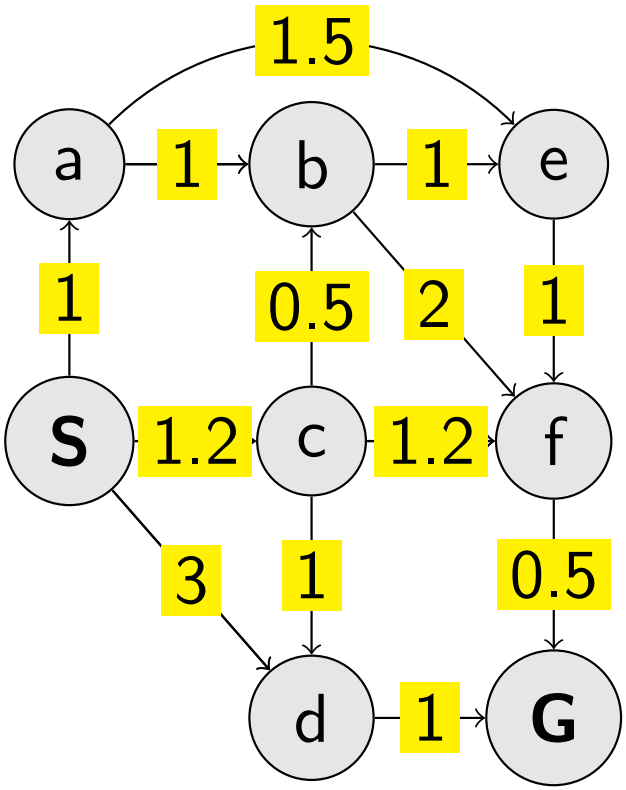
The BFS graph search

```
function BFS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← FIFOqueue(node)
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        explored.add(node.state)                    ▷ adding state not node!
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                if child_node contains Goal then return child_node
                end if
                fronter.insert(child_node)
            end if
        end for
    end while
end function
```

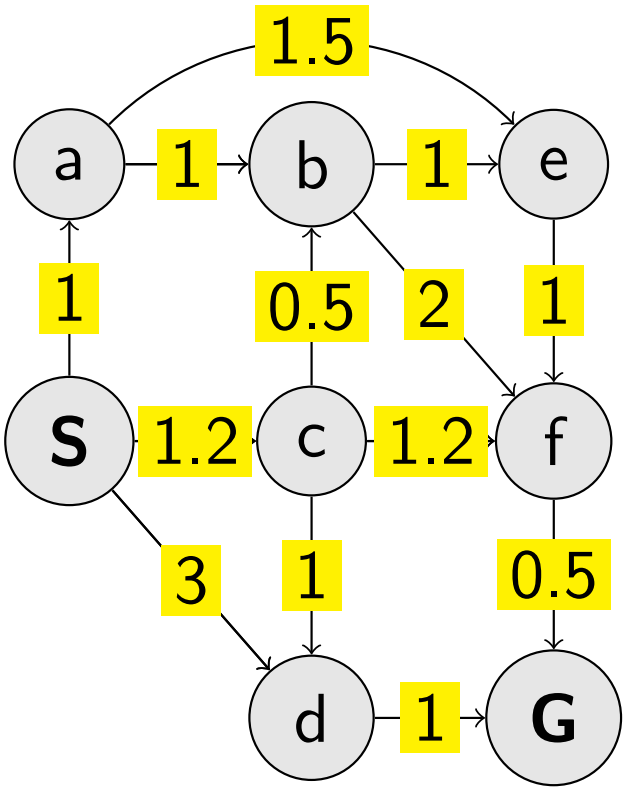
What about actual costs graph search?



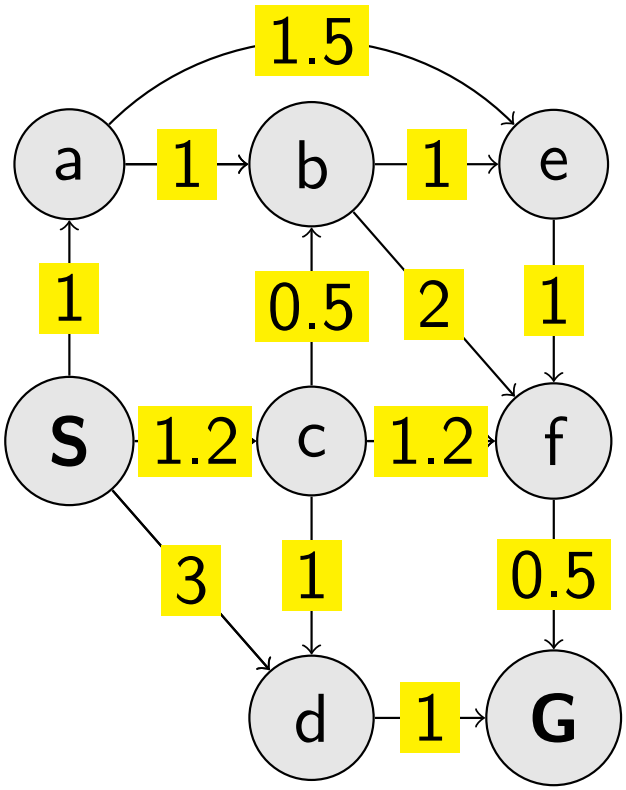
What about actual costs graph search?

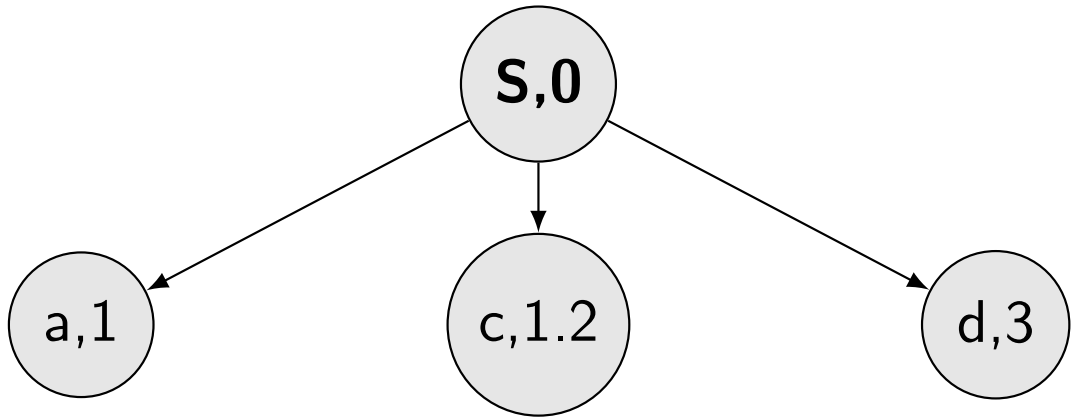


What about actual costs graph search?

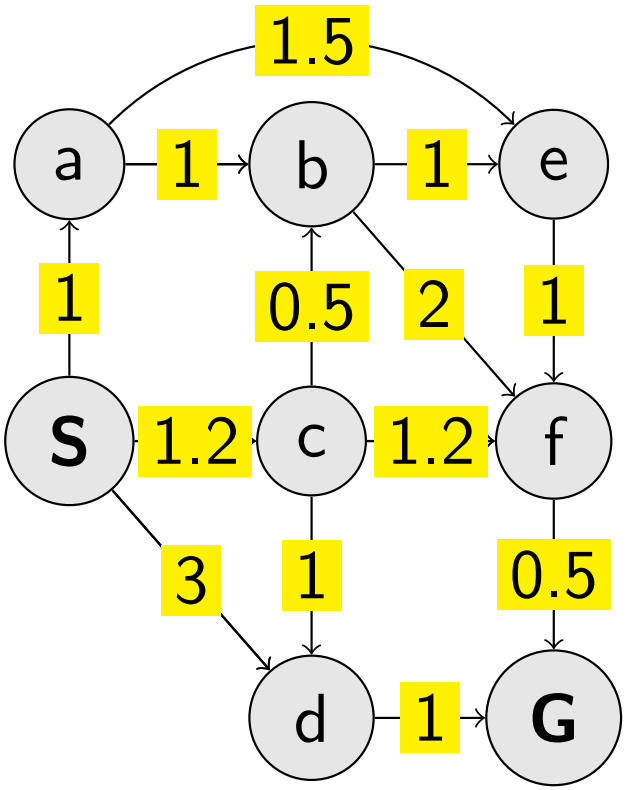


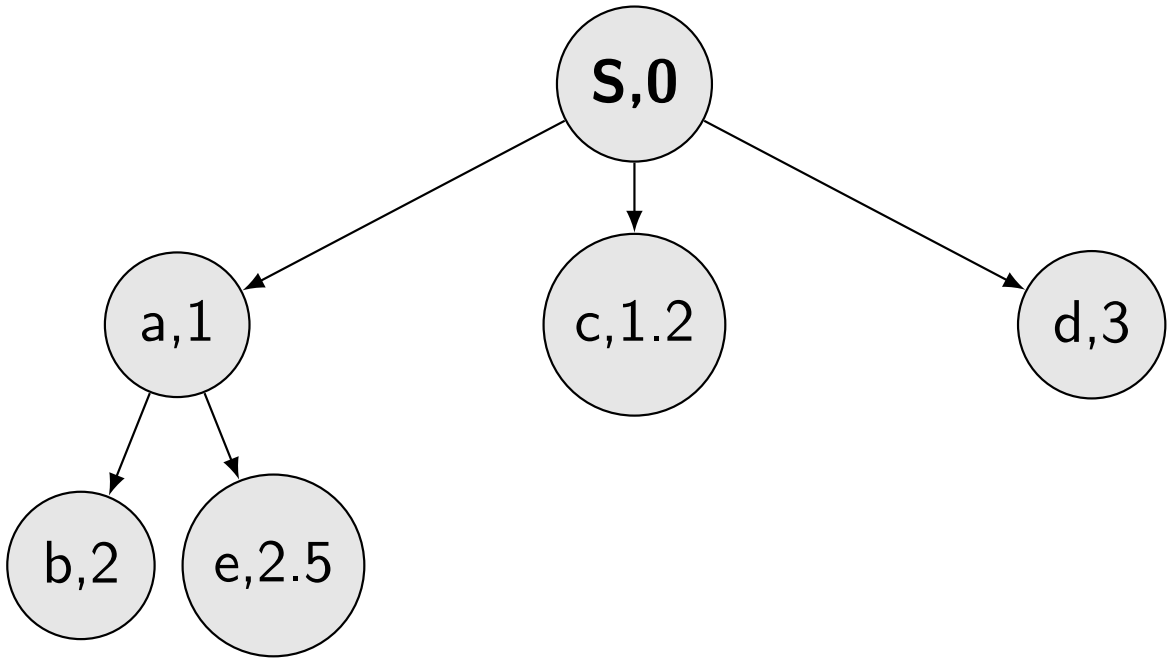
What about actual costs graph search?



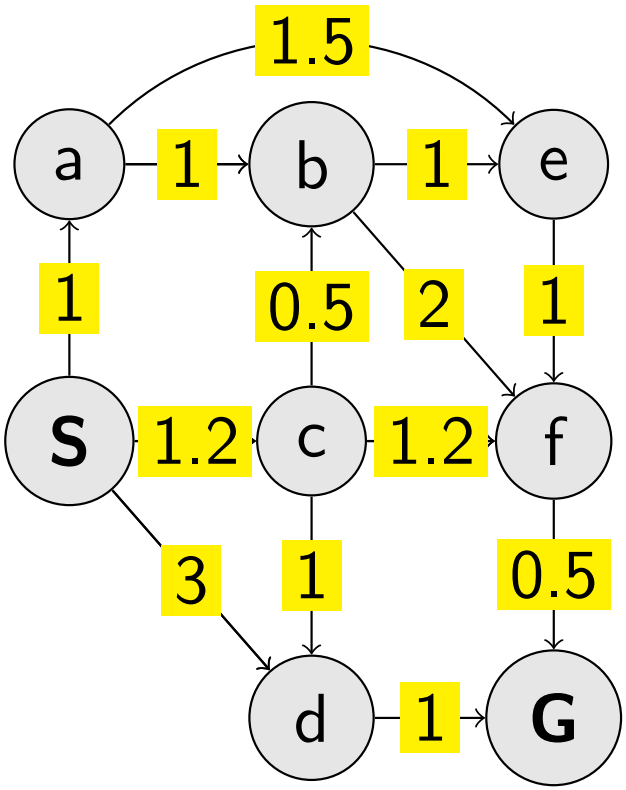


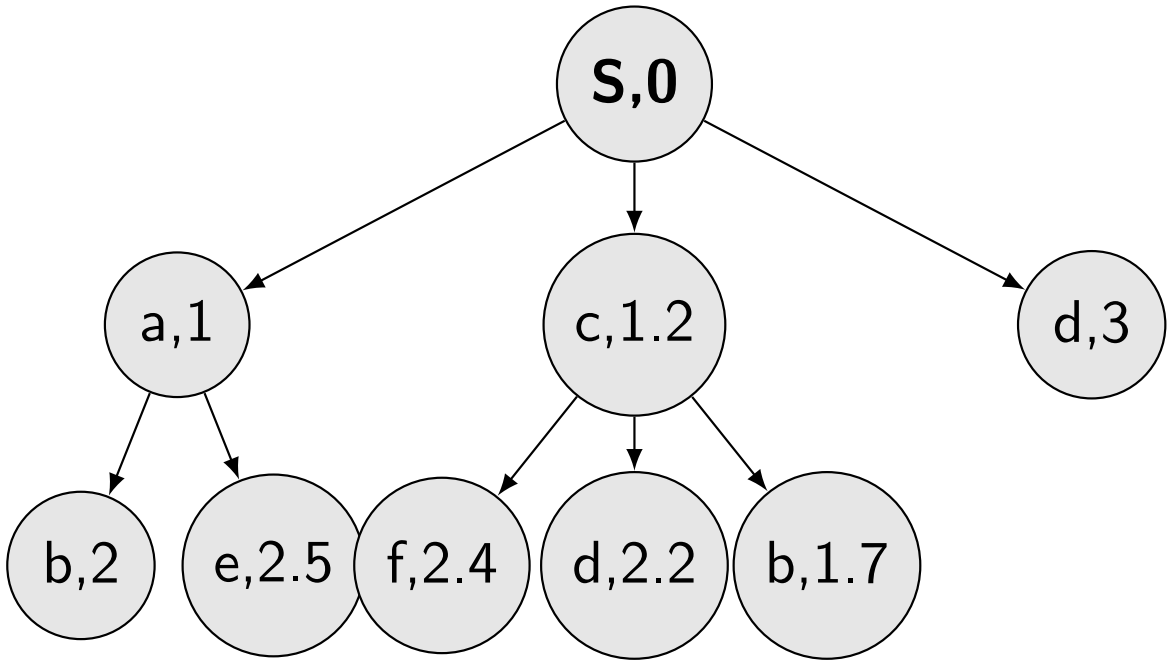
What about actual costs graph search?



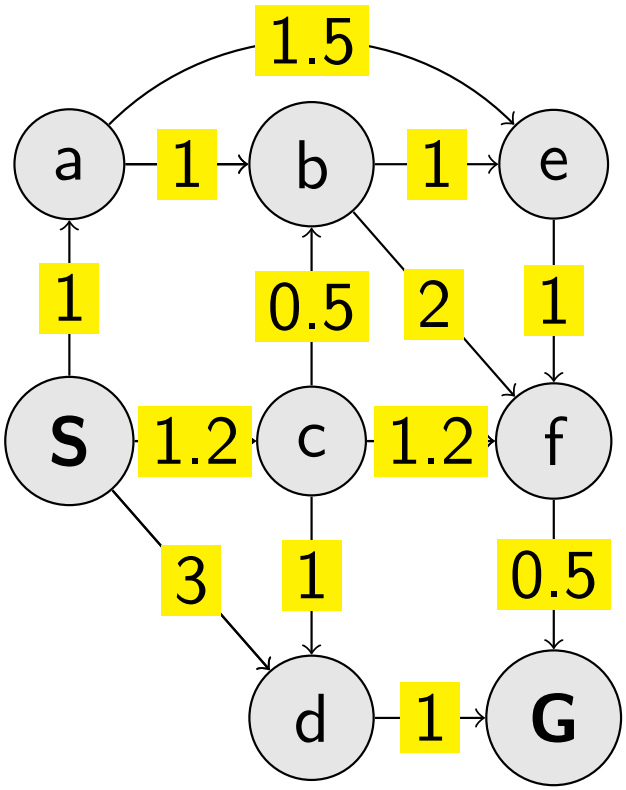


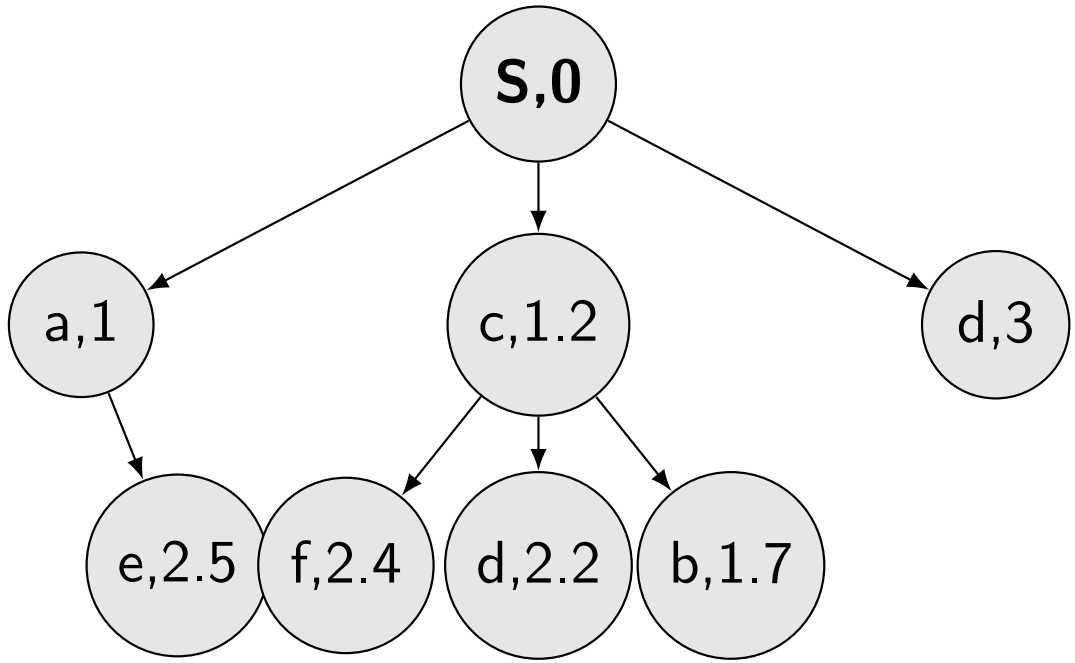
What about actual costs graph search?



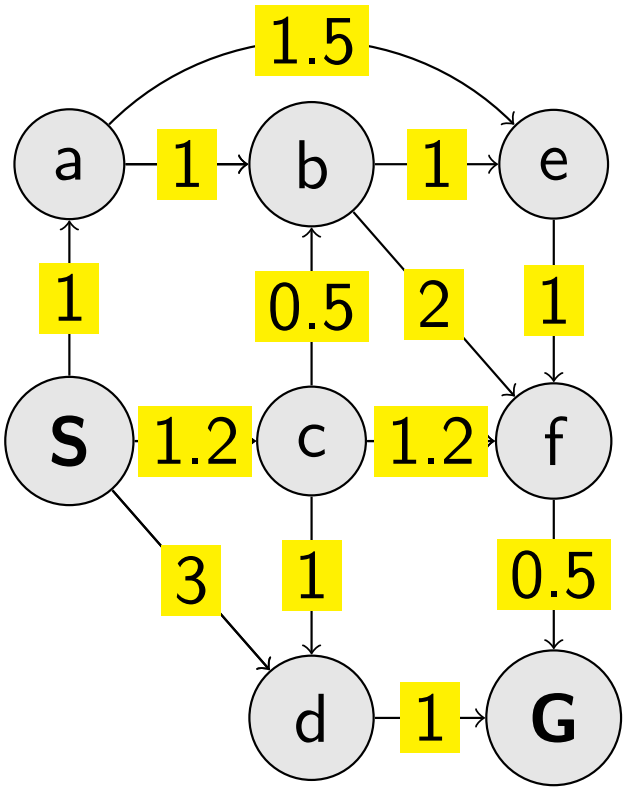


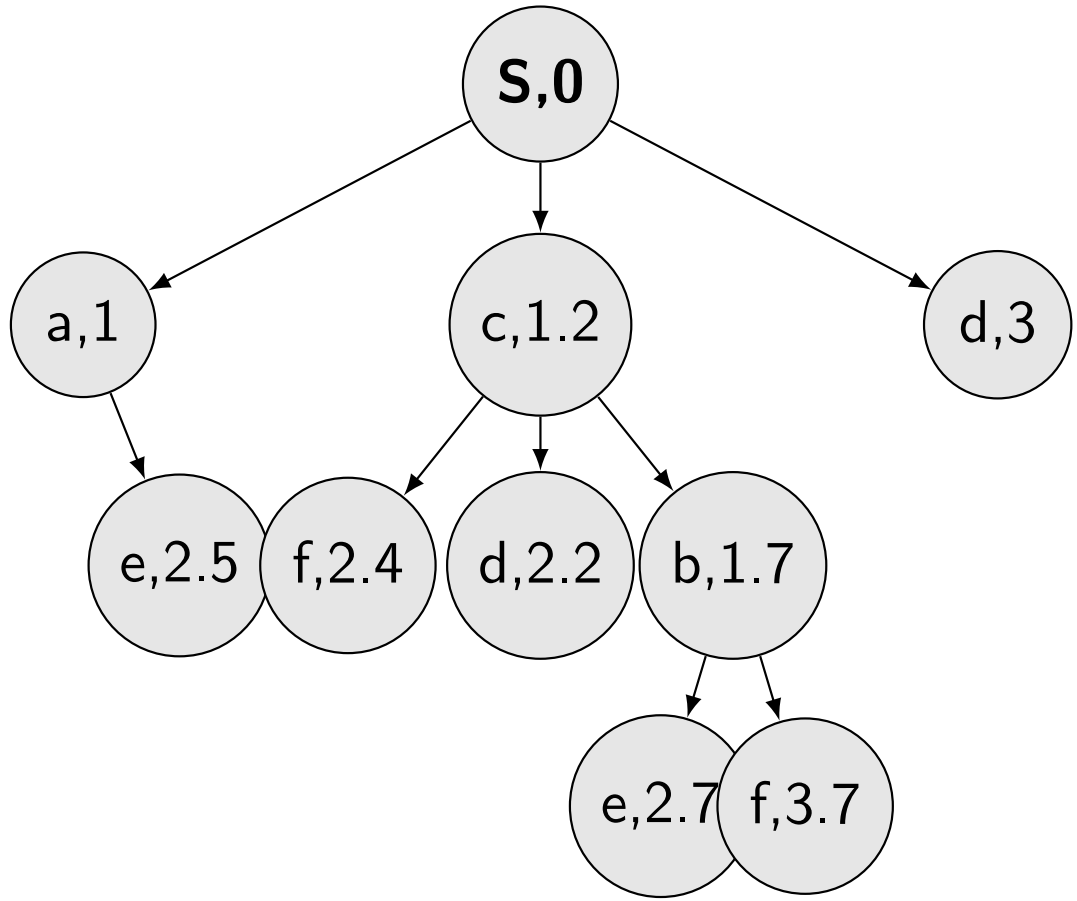
What about actual costs graph search?



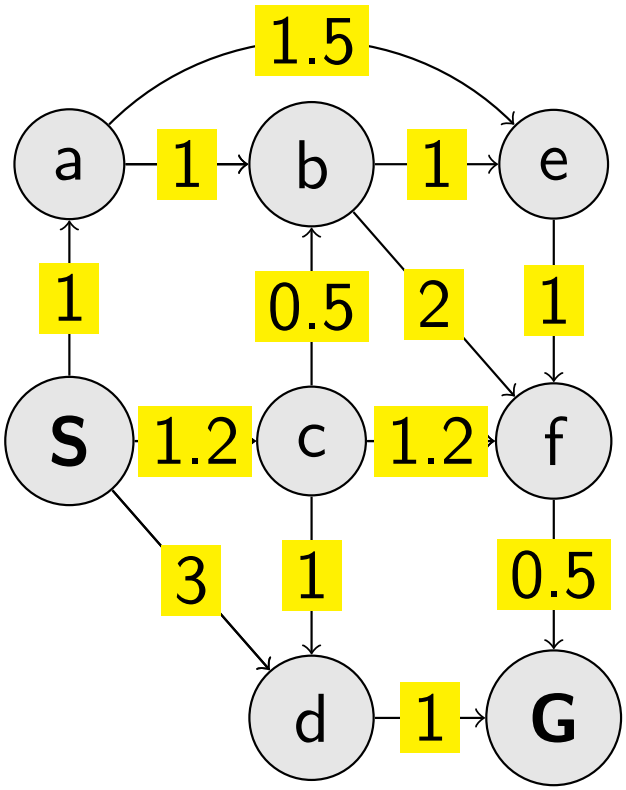


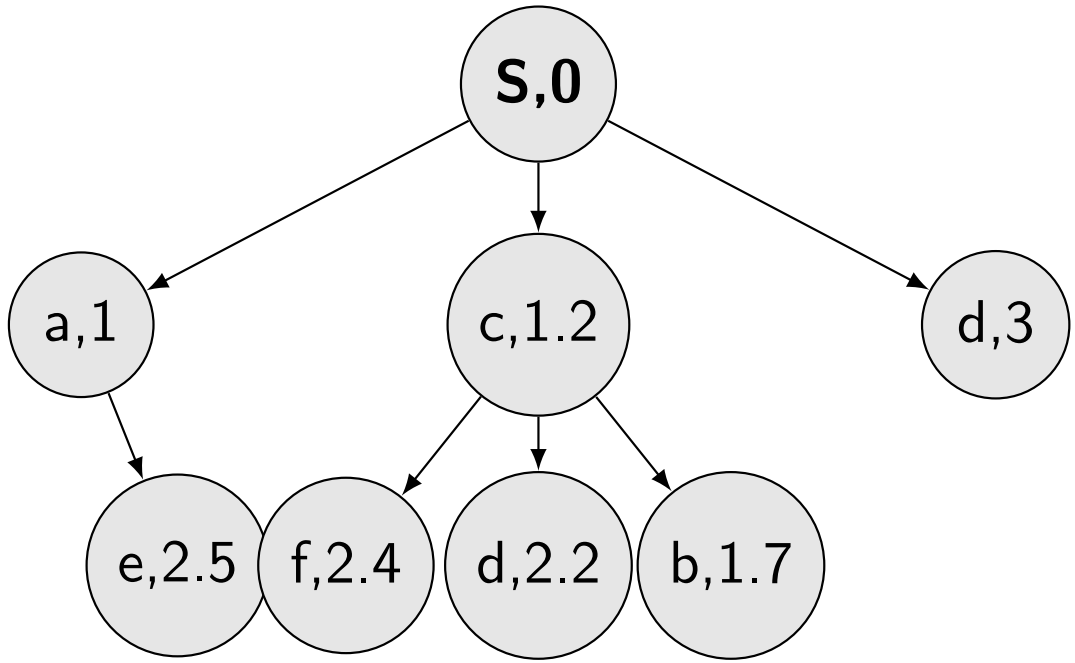
What about actual costs graph search?



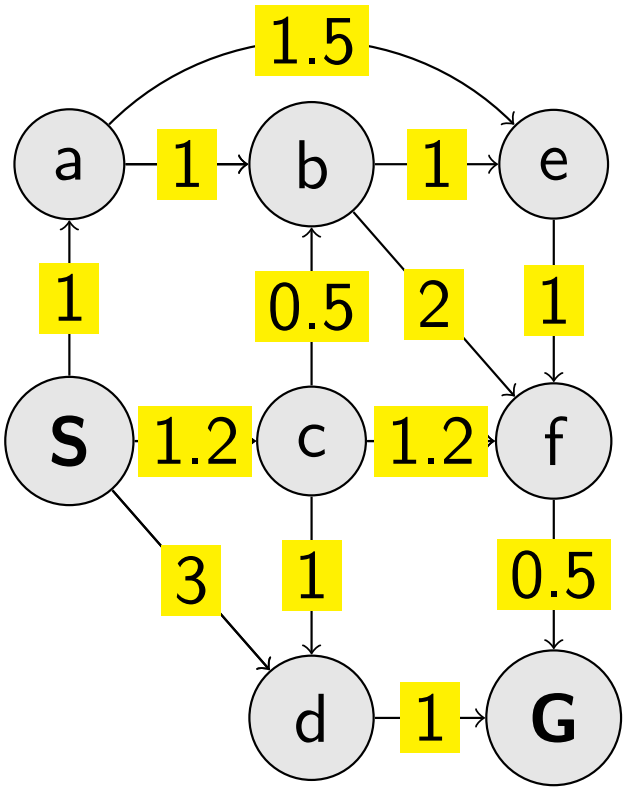


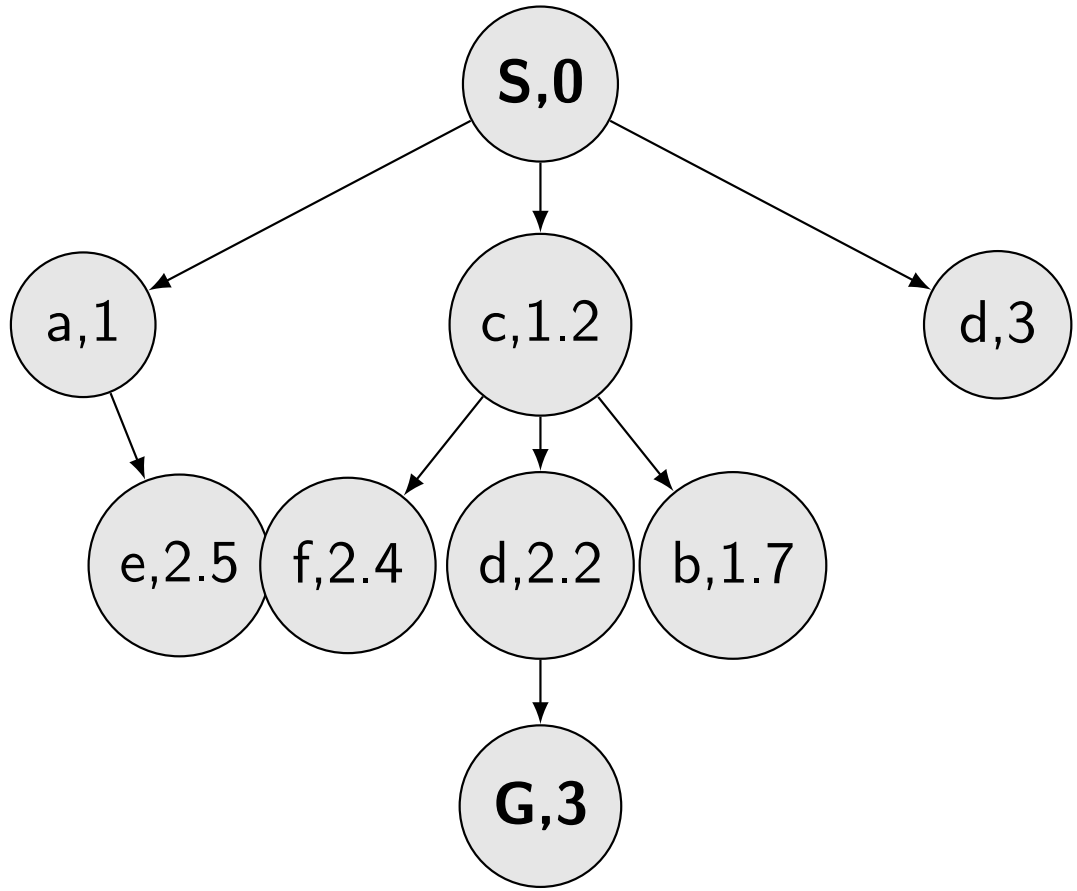
What about actual costs graph search?



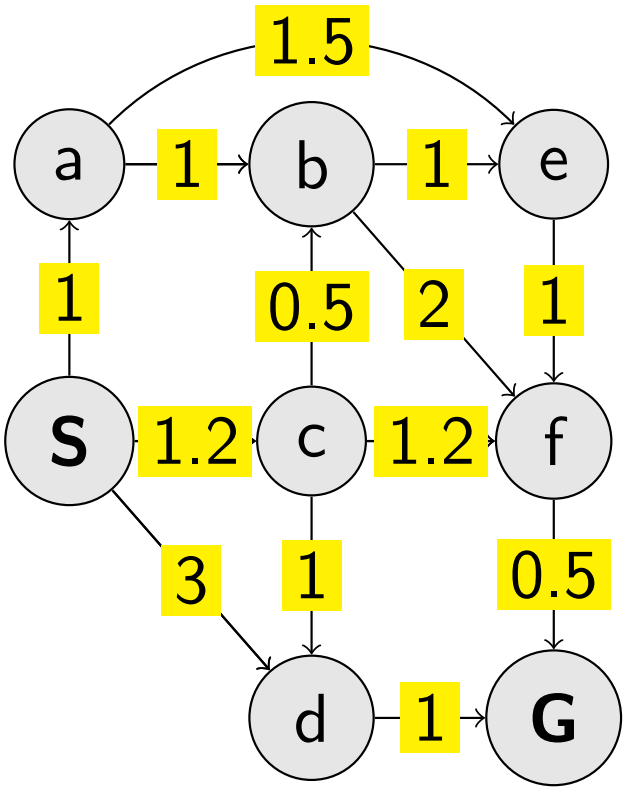


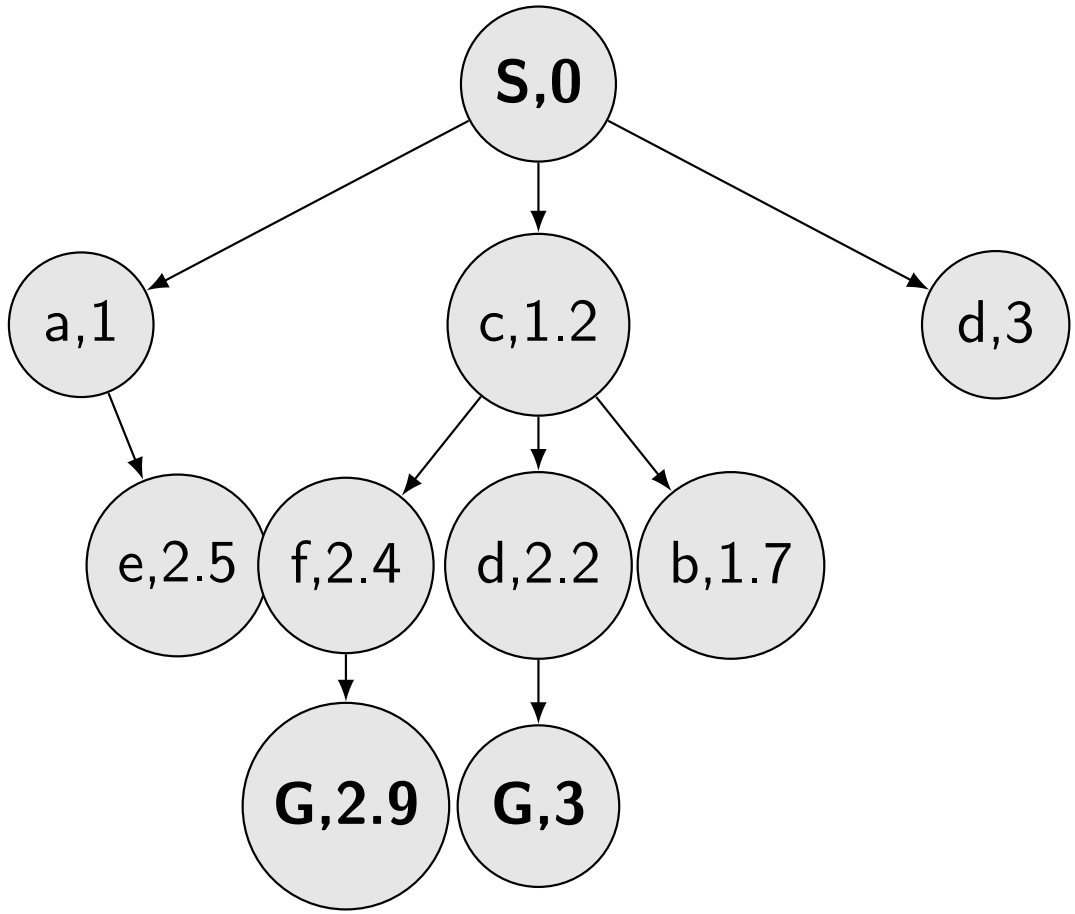
What about actual costs graph search?



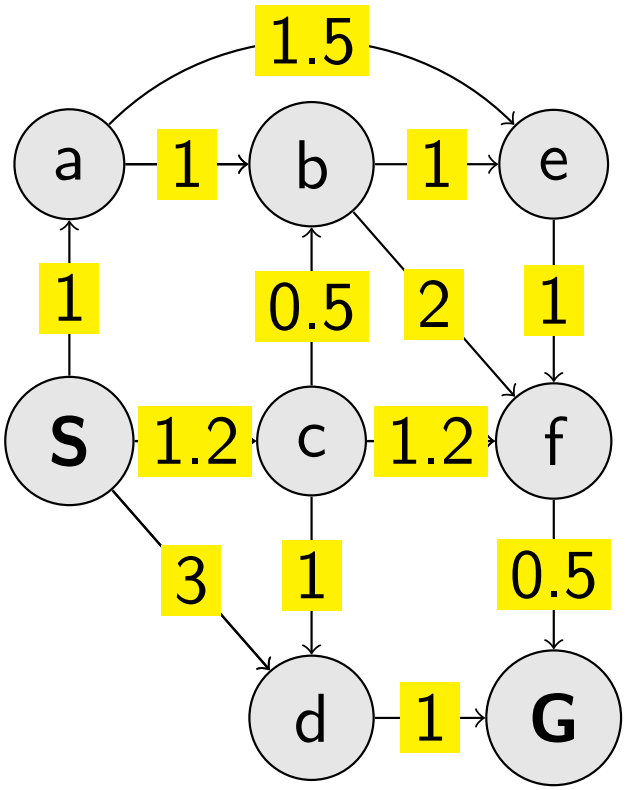


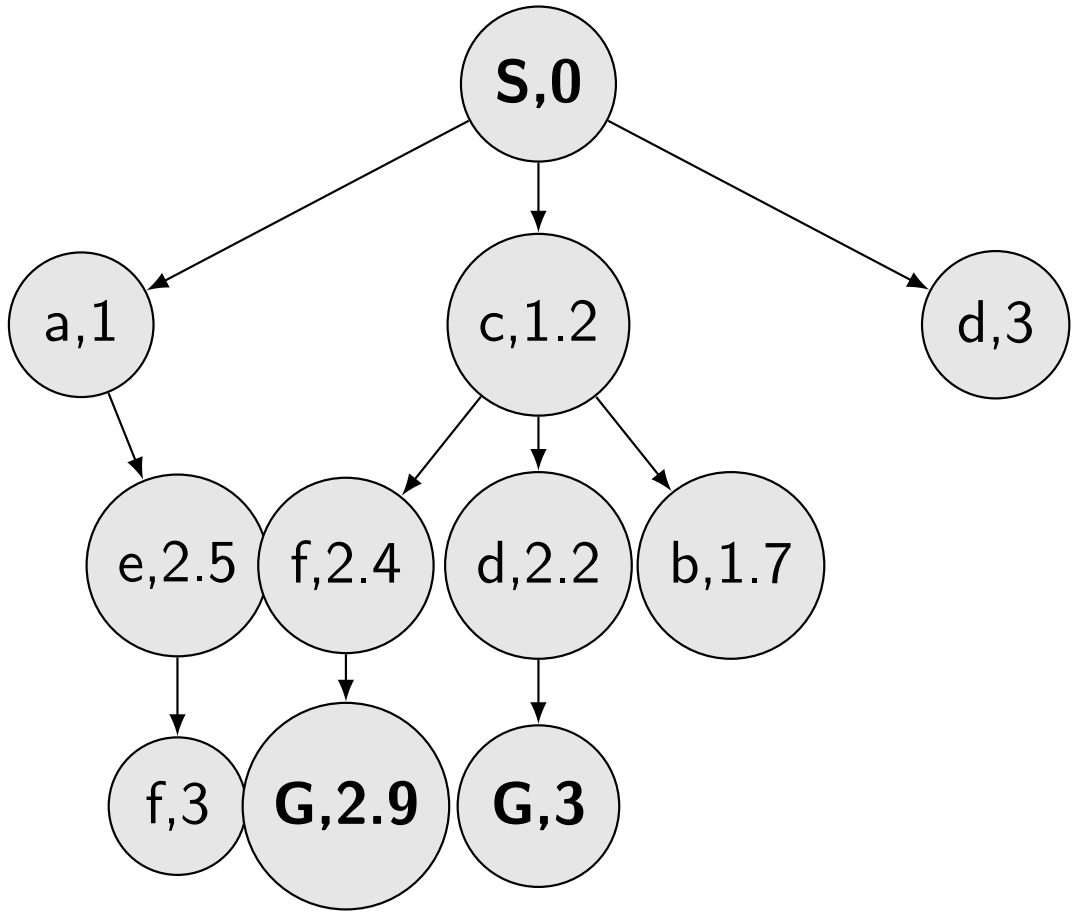
What about actual costs graph search?





What about actual costs graph search?





The UCS graph search

function UCS_GRAPH_SEARCH(env) **return** a solution or failure

```
node ← env.observe()
frontier ← priority_queue(node)                    ▷ path_cost for ordering
explored ← set()
while frontier not empty do
    node ← frontier.pop()
    if node contains Goal then return node                  ▷ check here!
    end if
    explored.add(node.state)
    child_nodes ← env.expand(node.state)
    for all child_nodes do
        if child_node.state not in explored or in frontier then
            fronter.insert(child_node)
        else if child_node.state in frontier with higher cost then
            replace that with the child_node
        end if
    end for
end while
```

The UCS graph search

function UCS_GRAPH_SEARCH(env) **return** a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node) ▷ path_cost for ordering
    explored ← set()
    while frontier not empty **do**
        node ← frontier.pop()
        if node contains Goal **then return** node ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes **do**
            if child_node.state not in explored or in frontier **then**
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost **then**
                replace that with the child_node
            end if
        end for
    end while

The UCS graph search

function UCS_GRAPH_SEARCH(env) **return** a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node) ▷ path_cost for ordering
    explored ← set()
    while frontier not empty **do**
        node ← frontier.pop()
        if node contains Goal **then return** node ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes **do**
            if child_node.state not in explored or in frontier **then**
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost **then**
                replace that with the child_node
            end if
        end for
    end while

The UCS graph search

```
function UCS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node)                    ▷ path_cost for ordering
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        if node contains Goal then return node                  ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost then
                replace that with the child_node
            end if
        end for
    end while
```

The UCS graph search

```
function UCS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node)                    ▷ path_cost for ordering
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        if node contains Goal then return node                  ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost then
                replace that with the child_node
            end if
        end for
    end while
```

The UCS graph search

```
function UCS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node)                    ▷ path_cost for ordering
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        if node contains Goal then return node                  ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost then
                replace that with the child_node
            end if
        end for
    end while
```

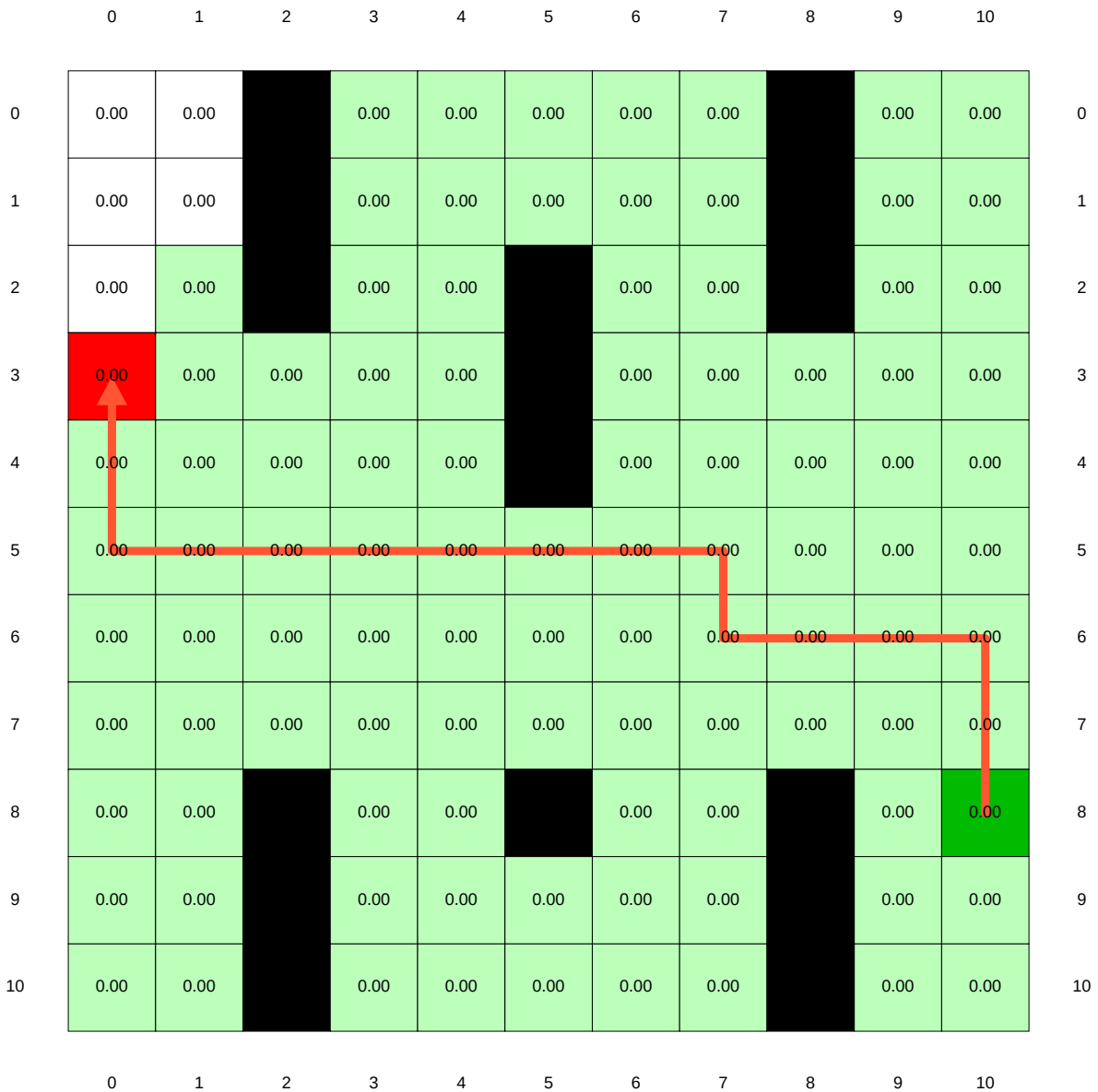
The UCS graph search

```
function UCS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node)                    ▷ path_cost for ordering
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        if node contains Goal then return node                  ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost then
                replace that with the child_node
            end if
        end for
    end while
```

The UCS graph search

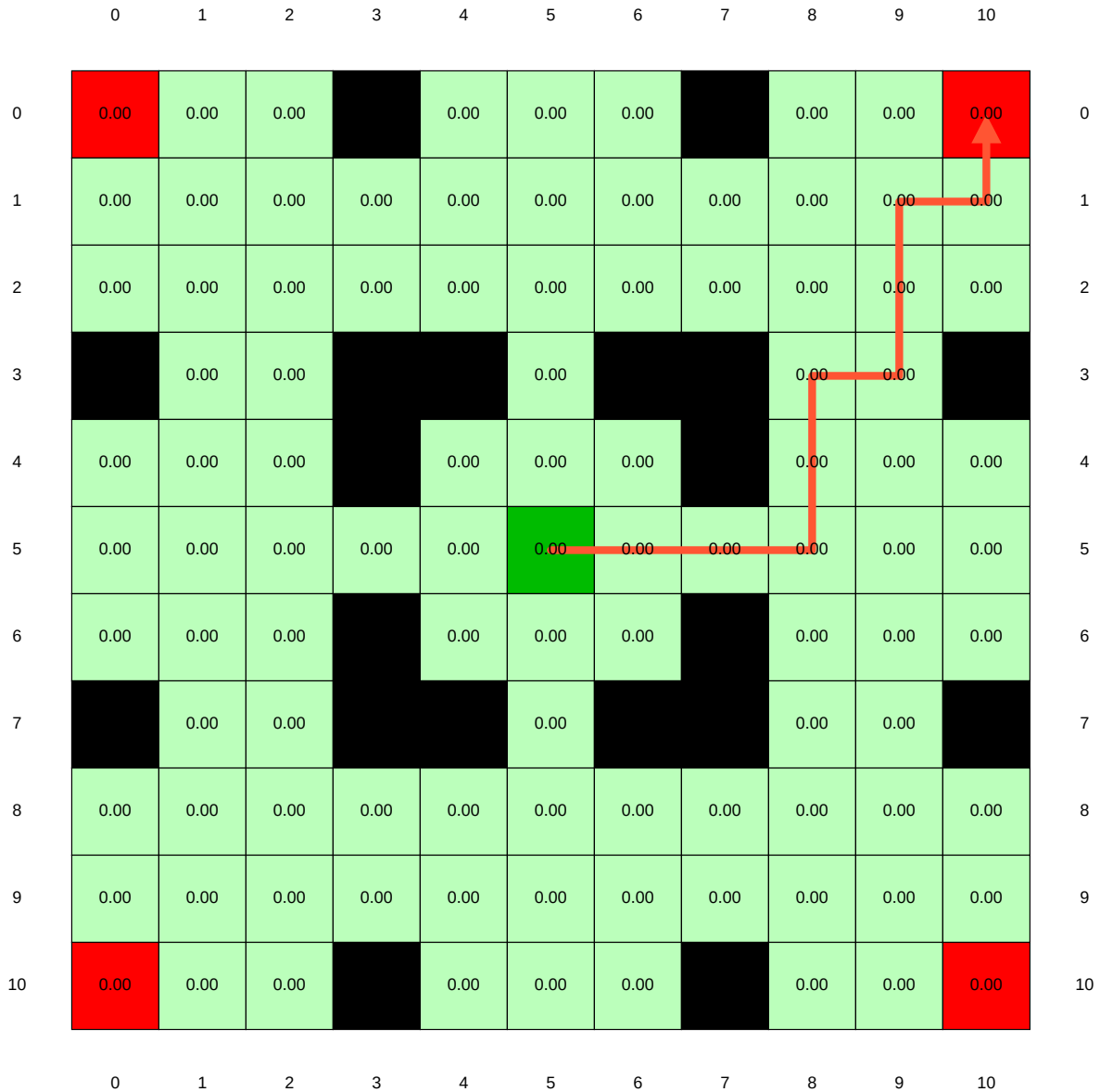
```
function UCS_GRAPH_SEARCH(env) return a solution or failure
    node ← env.observe()
    frontier ← priority_queue(node)                    ▷ path_cost for ordering
    explored ← set()
    while frontier not empty do
        node ← frontier.pop()
        if node contains Goal then return node                  ▷ check here!
        end if
        explored.add(node.state)
        child_nodes ← env.expand(node.state)
        for all child_nodes do
            if child_node.state not in explored or in frontier then
                fronter.insert(child_node)
            else if child_node.state in frontier with higher cost then
                replace that with the child_node
            end if
        end for
    end while
```

Few examples of search strategies so far



Run the demos.

What is wrong with UCS and other strategies?



Run the demo.

Node selection, take argmin $f(n)$

- DFS: $f(n) = -n.depth$
- BFS: $f(n) = n.depth$
- UCS: $f(n) = n.path\_cost$

The good: frontier as a priority queue The bad: All the $f(n)$ correspond to the cost from n to the start - only backward cost.

Node selection, take argmin $f(n)$

- DFS: $f(n) = -n.depth$
- BFS: $f(n) = n.depth$
- UCS: $f(n) = n.path\_cost$

The good: frontier as a priority queue The bad: All the $f(n)$ correspond to the cost from n to the start - only backward cost.

Node selection, take argmin $f(n)$

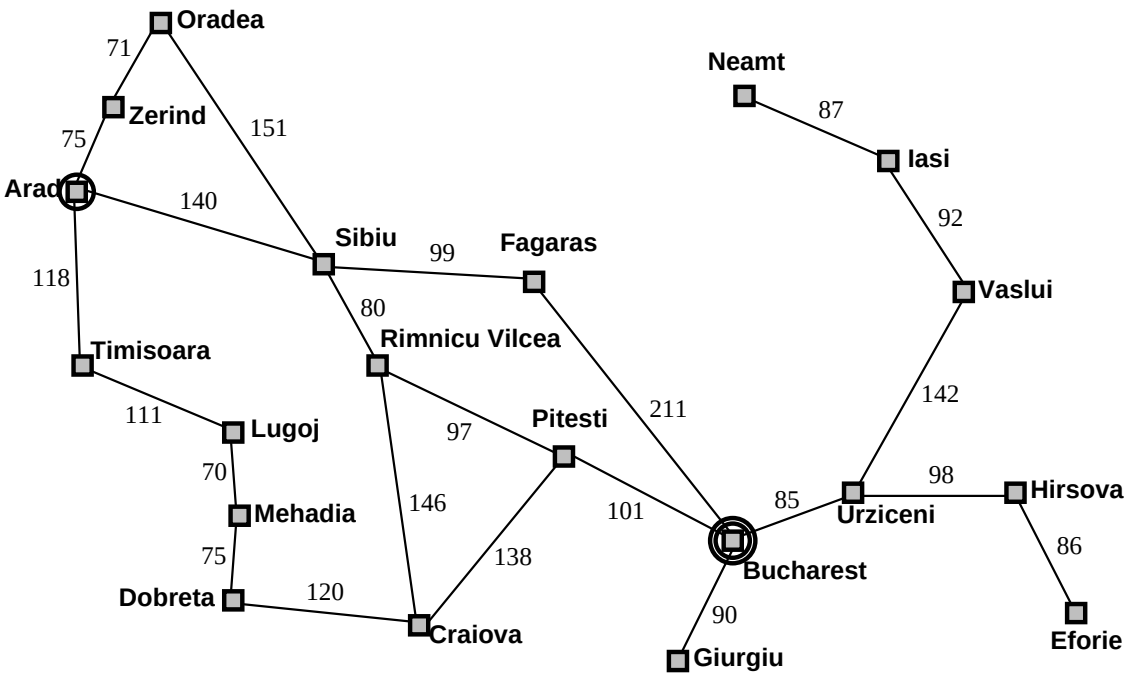
- DFS: $f(n) = -n.depth$
- BFS: $f(n) = n.depth$
- UCS: $f(n) = n.path\_cost$

The good: frontier as a priority queue The bad: All the $f(n)$ correspond to the cost from n to the start - only backward cost.

Heuristics

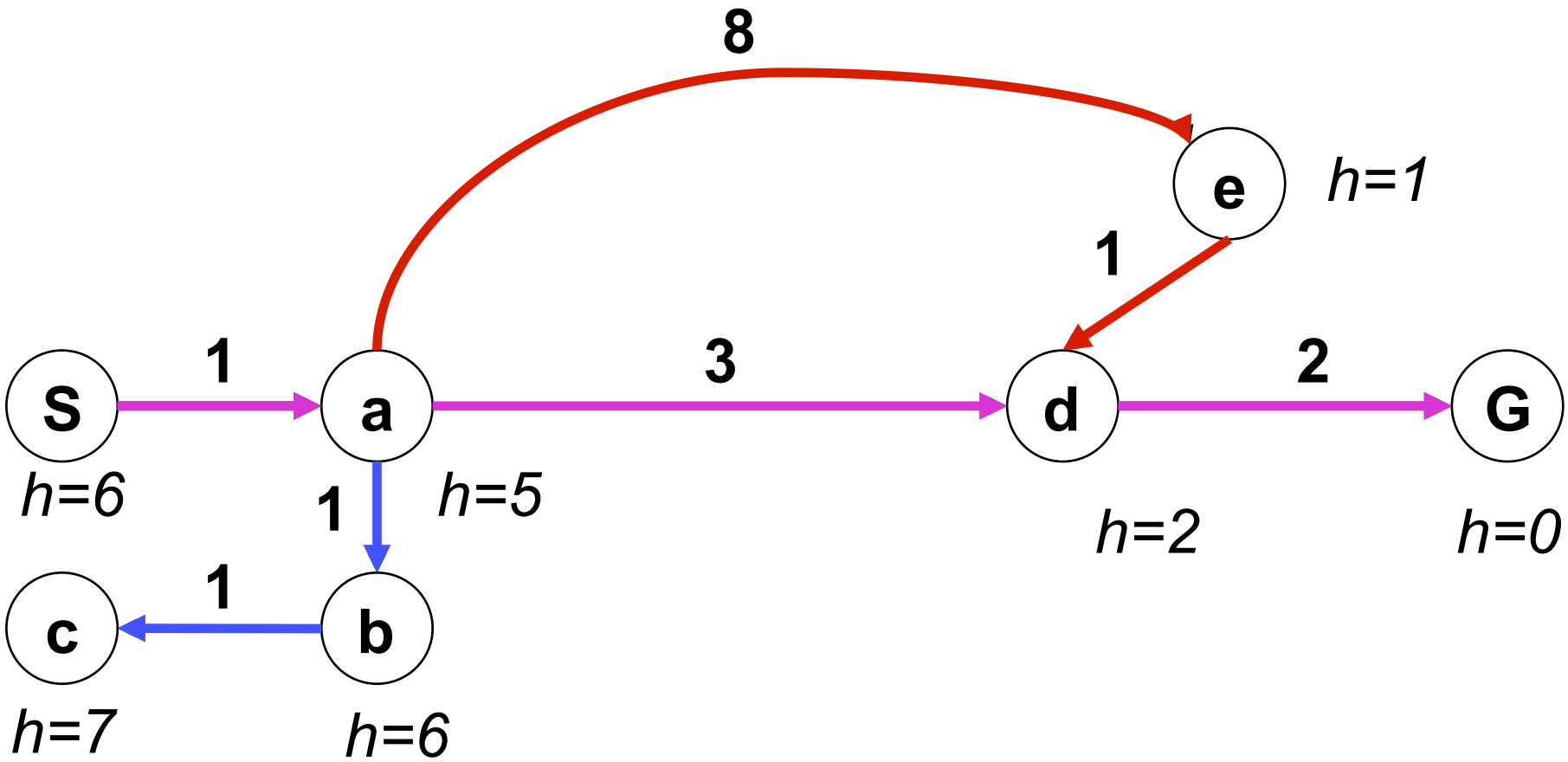
- A function that estimates how close a state to the goal.
- Designed for a particular problem.
- Examples:
- We will use $h(n)$ - heuristic value of node n

Example of hueristics



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Greedy, take the node argmin $h(n)$

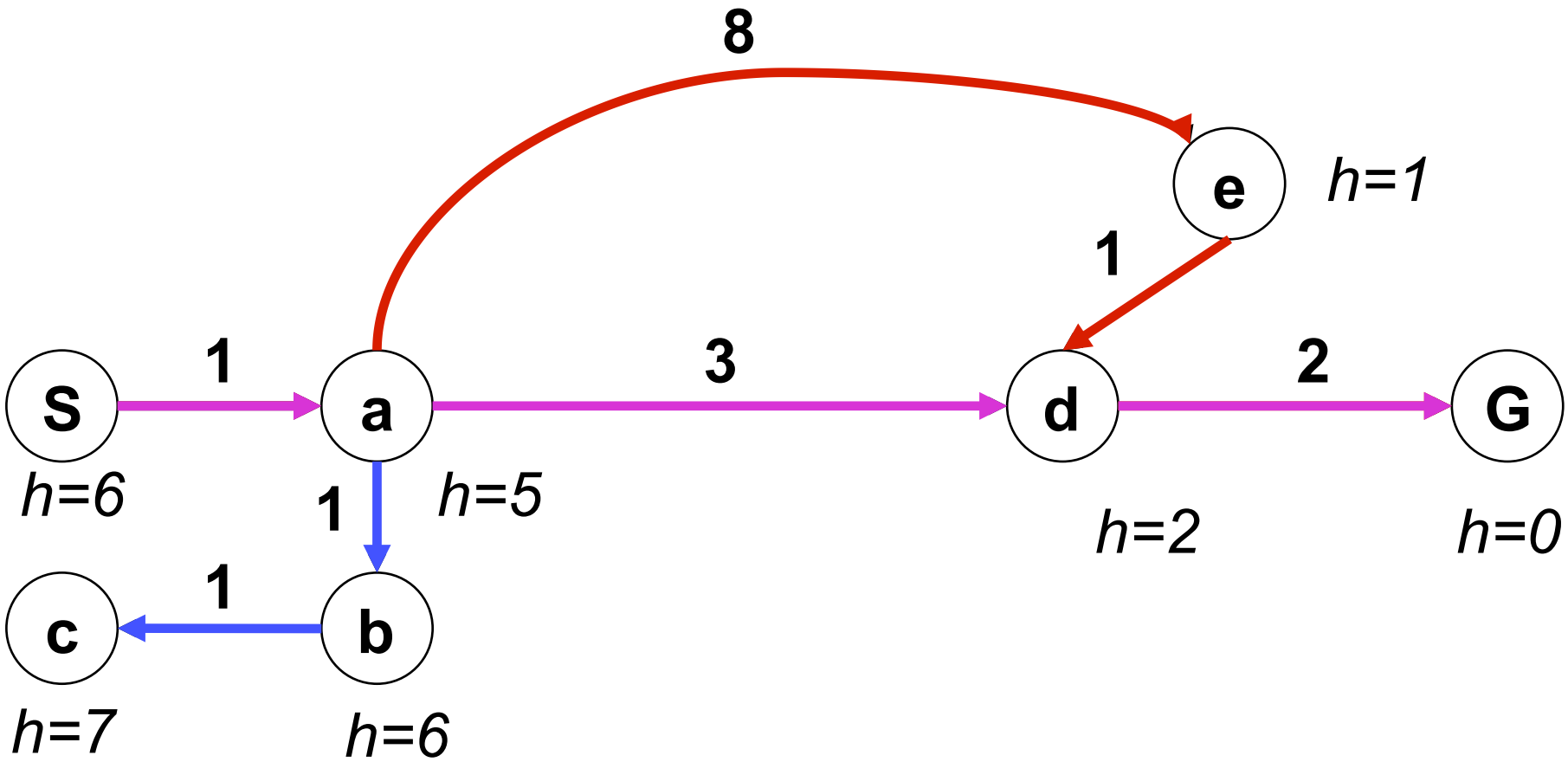


[1]

What is wrong (and nice) with the Greedy?

[1] Graph example: Ted Grenager

Greedy, take the node argmin $h(n)$

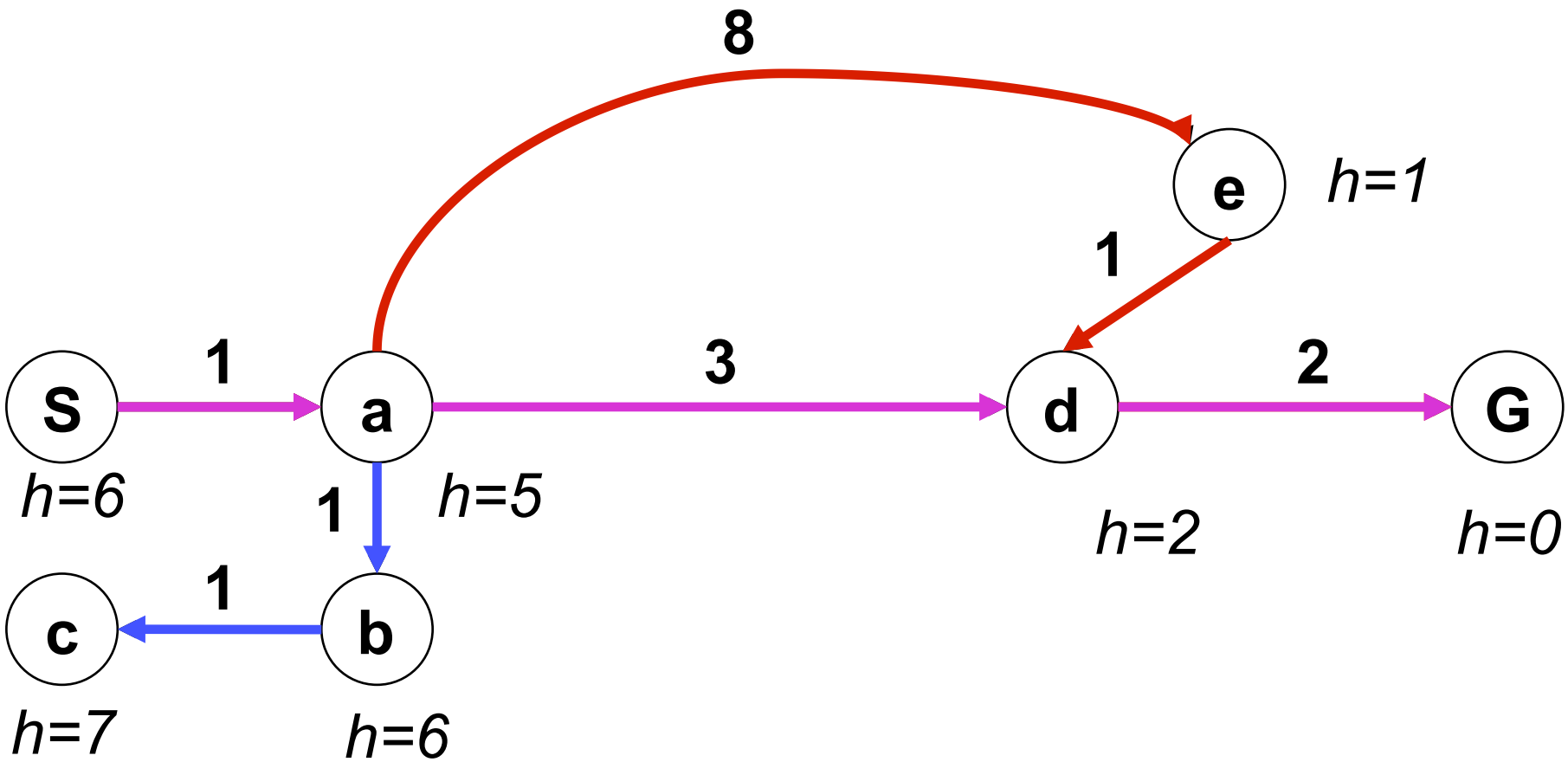


[1]

What is wrong (and nice) with the Greedy?

[1] Graph example: Ted Grenager

A* combines UCS and Greedy

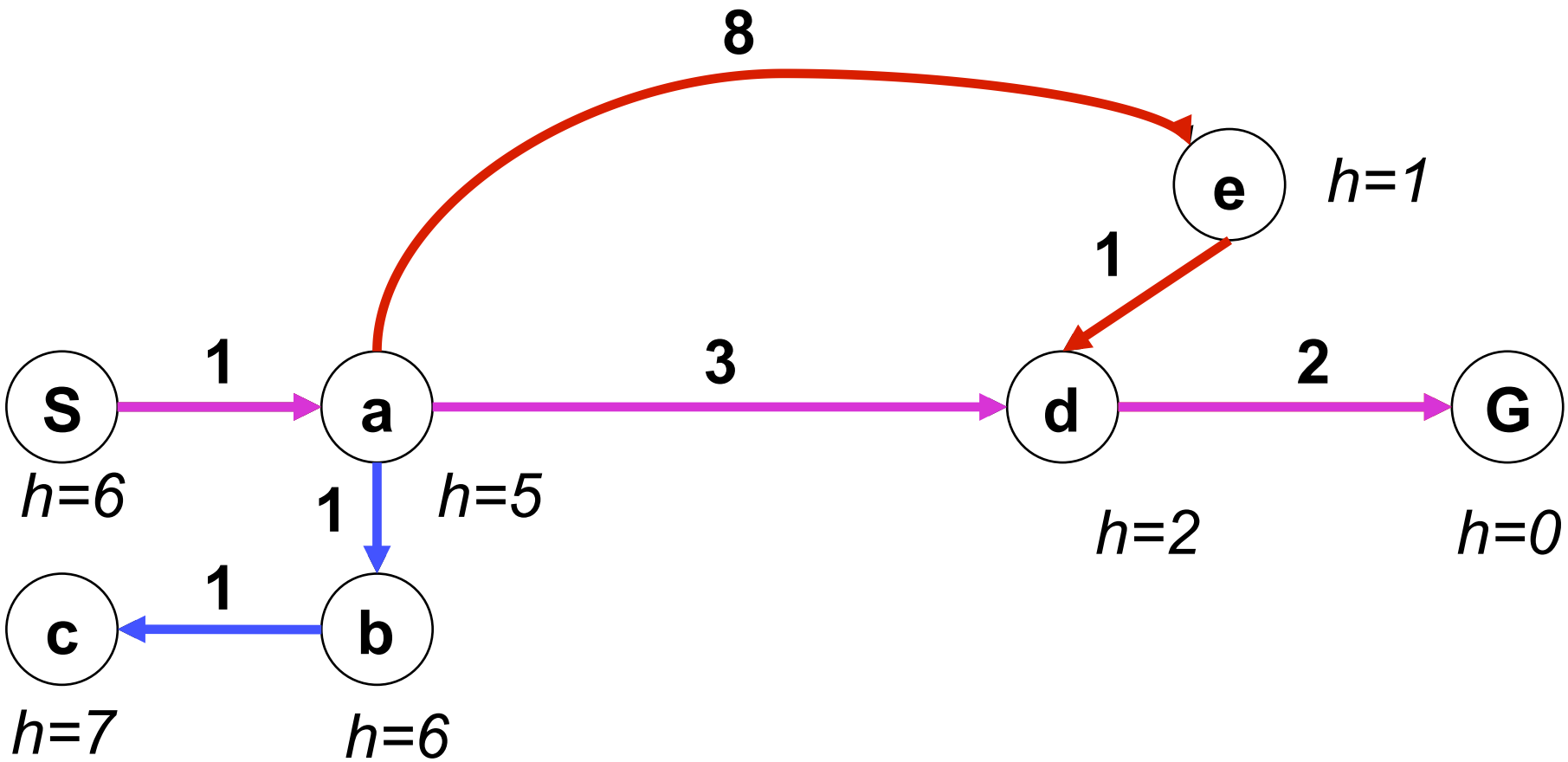


UCS orders by backward (path) cost $g(n)$

Greedy uses heuristics (goal proximity) $h(n)$

A* orders nodes by: $f(n) = g(n) + h(n)$

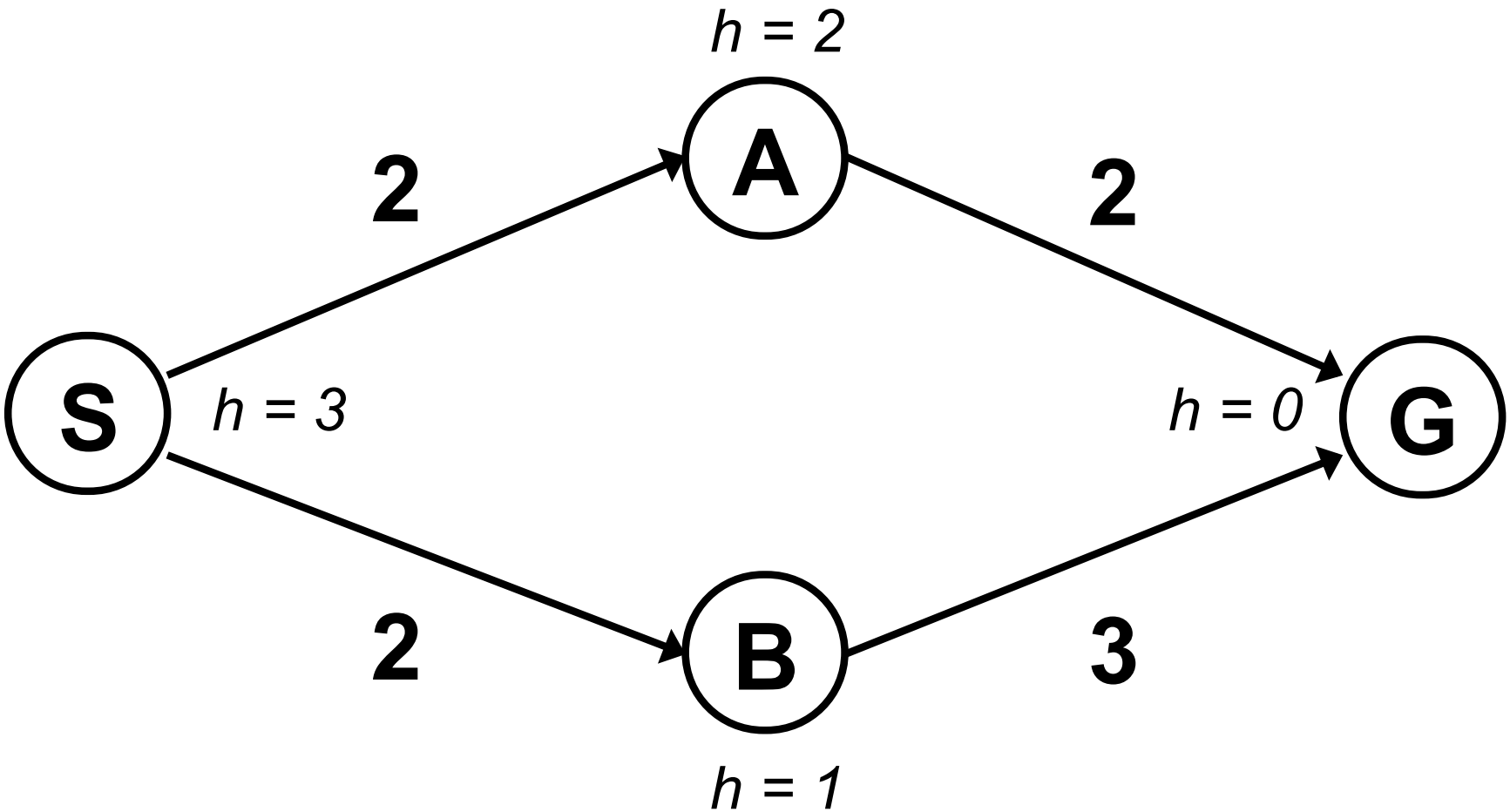
A* combines UCS and Greedy



UCS orders by backward (path) cost $g(n)$
Greedy uses heuristics (goal proximity) $h(n)$

A* orders nodes by: $f(n) = g(n) + h(n)$

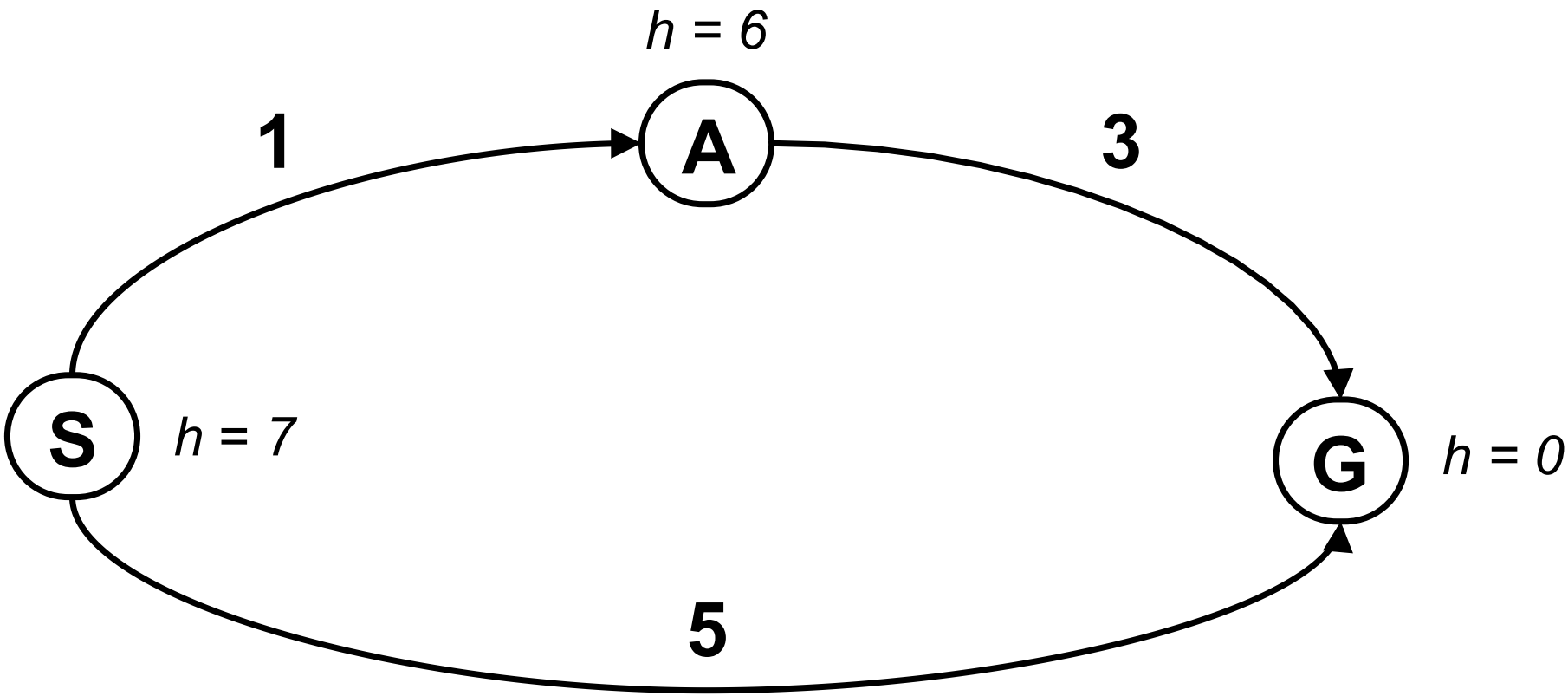
When to stop A*



2

[2] Graph example: Dan Klein and Pieter Abbeel

Is A* optimal?

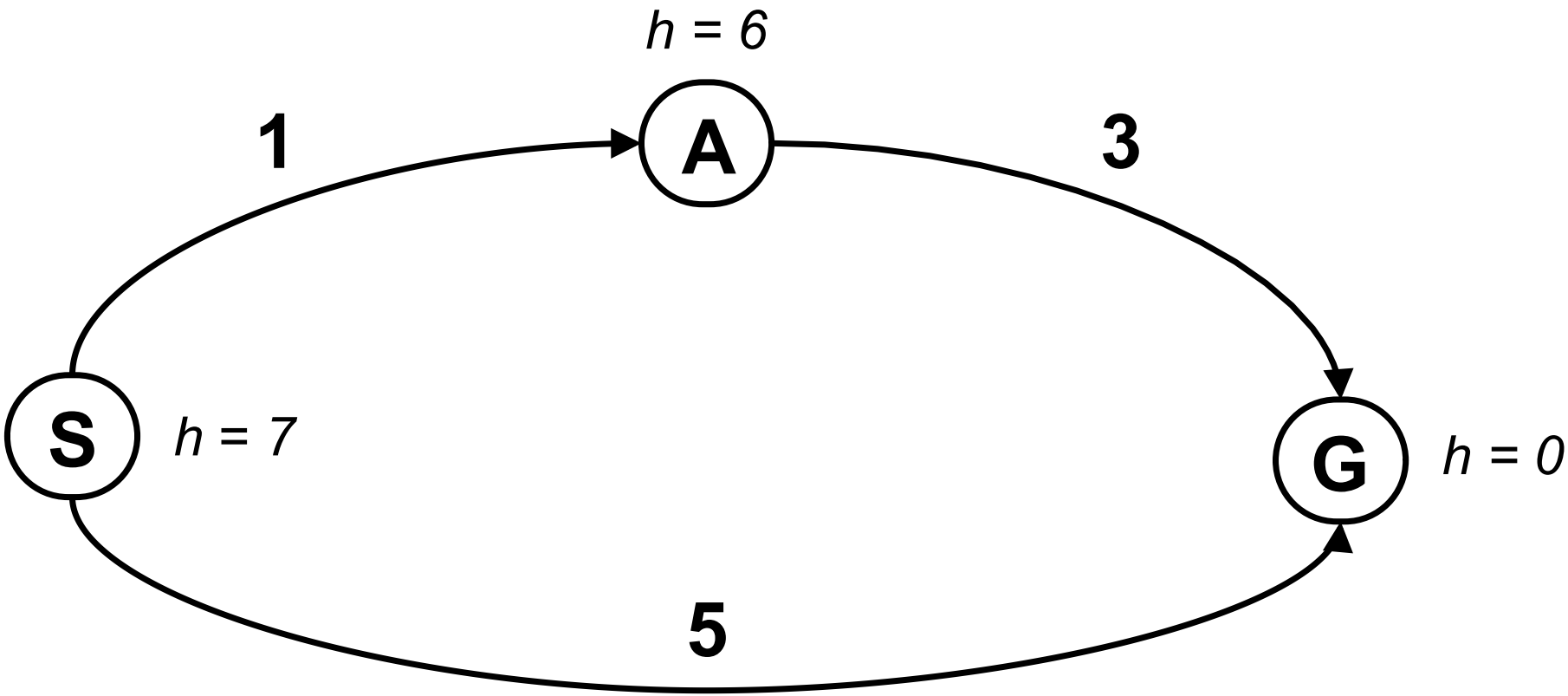


3

What is the problem?

[3] Graph example: Dan Klein and Pieter Abbeel

Is A* optimal?



3

What is the problem?

[3] Graph example: Dan Klein and Pieter Abbeel

Admissible heuristics

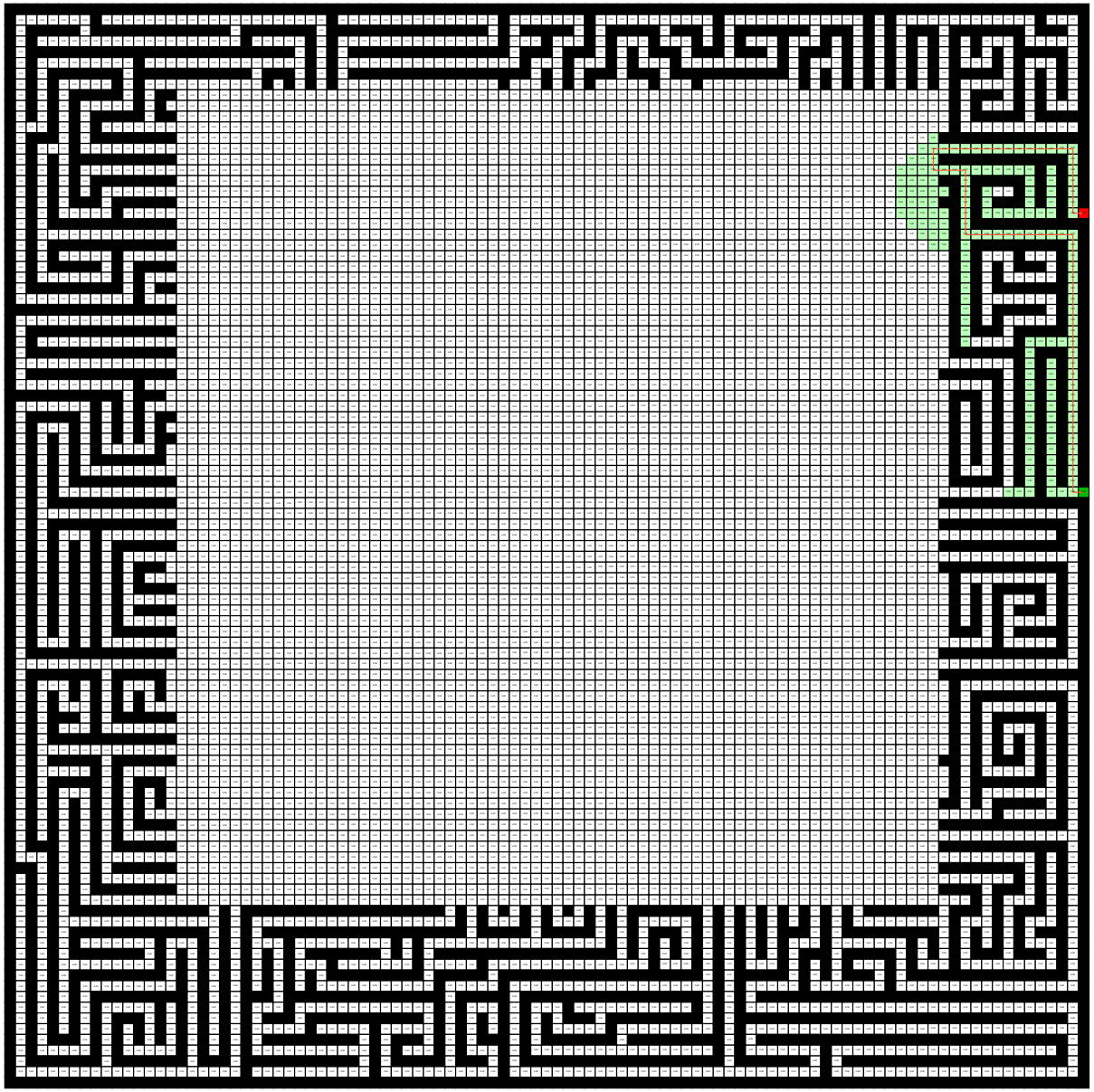
A heuristic function h is admissible if:

$$0 \leq h(n) \leq h^*(n)$$

where $h^*(n)$ is the true cost of going from n to the nearest goal.

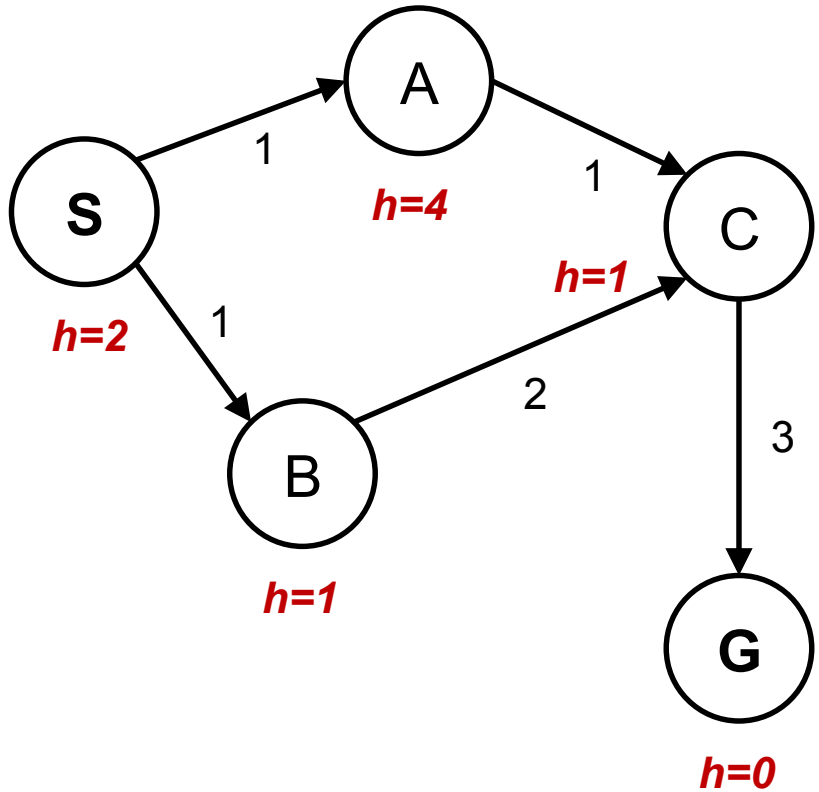
Optimality of A* tree search

Properties - does heuristic matter?



A* graph search

```
function GRAPH_SEARCH(env)
    frontier.insert(startnode)
    explored = set()
    while frontier do
        node = frontier.pop()
        if goal in node then break
        end if
        nodes = env.expand(node.state)
        explored.add(node)
        for all nodes do
            if node not in explored then
                frontier.insert(node)
            end if
        end for
    end while
end function
```

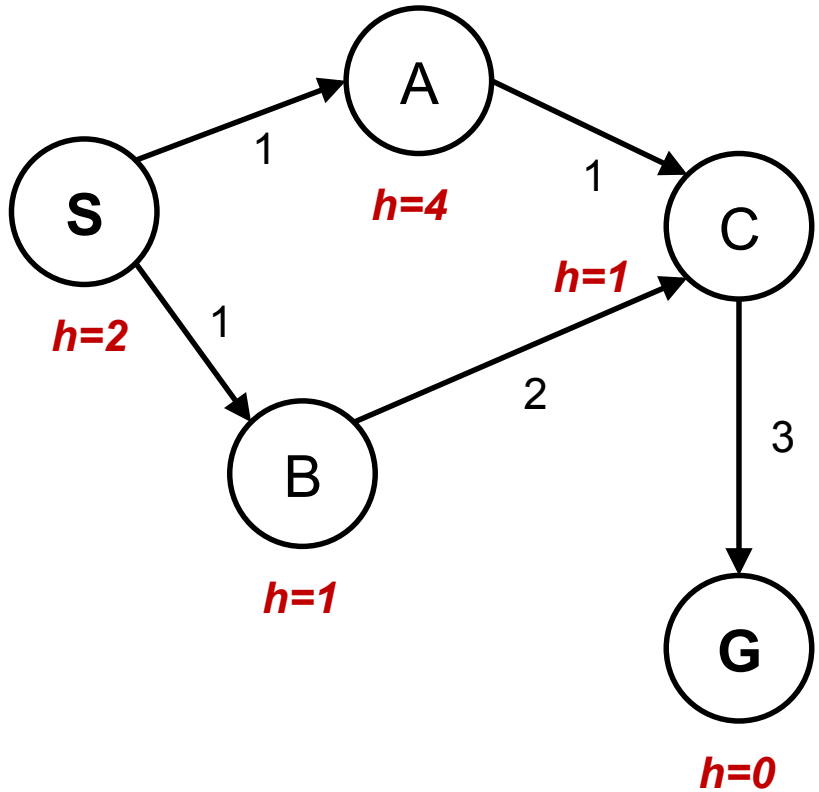


Graph example: Dan Klein and Pieter Abbeel

What went wrong?

A* graph search

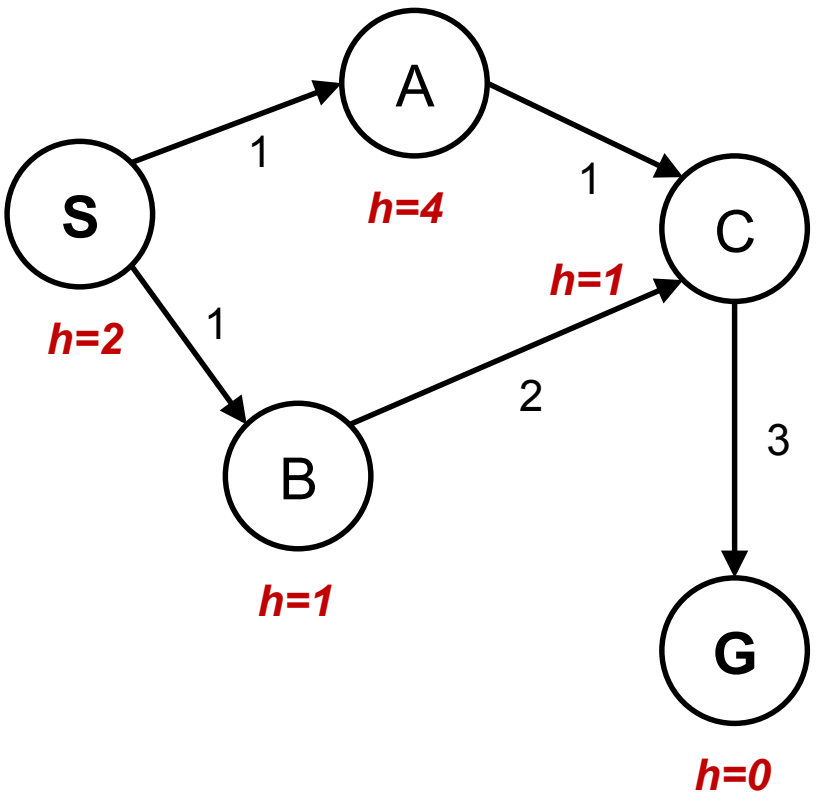
```
function GRAPH_SEARCH(env)
    frontier.insert(startnode)
    explored = set()
    while frontier do
        node = frontier.pop()
        if goal in node then break
        end if
        nodes = env.expand(node.state)
        explored.add(node)
        for all nodes do
            if node not in explored then
                frontier.insert(node)
            end if
        end for
    end while
end function
```



Graph example: Dan Klein and Pieter Abbeel

What went wrong?

Consistent heuristics

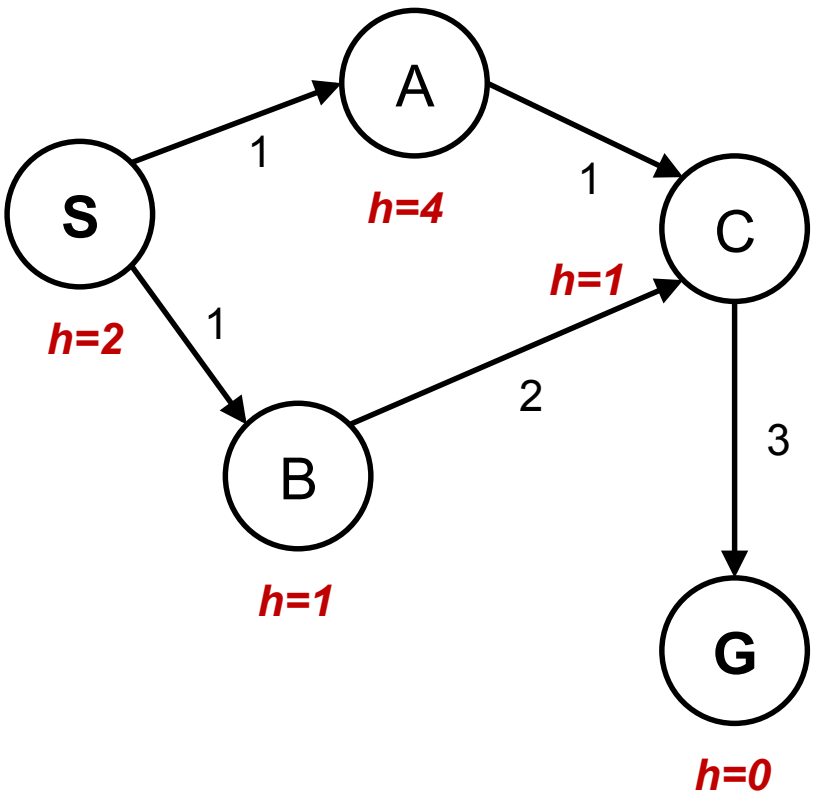


Admissible h:
$h(A) \leq$ true cost $A \to G$

Consistent h:
$h(A) - h(C) \leq$ true cost $A \to C$

f along a path never decreases!

Consistent heuristics



Admissible h:

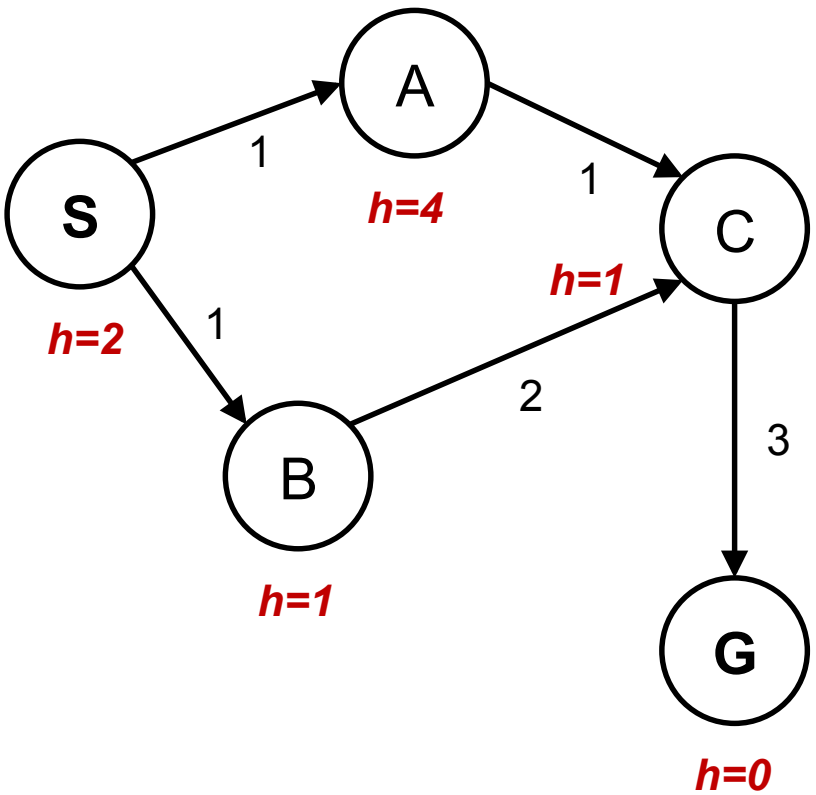
$h(A) \leq$ true cost $A \rightarrow G$

Consistent h:

$h(A) - h(C) \leq$ true cost $A \rightarrow C$

f along a path never decreases!

Consistent heuristics



Admissible h:

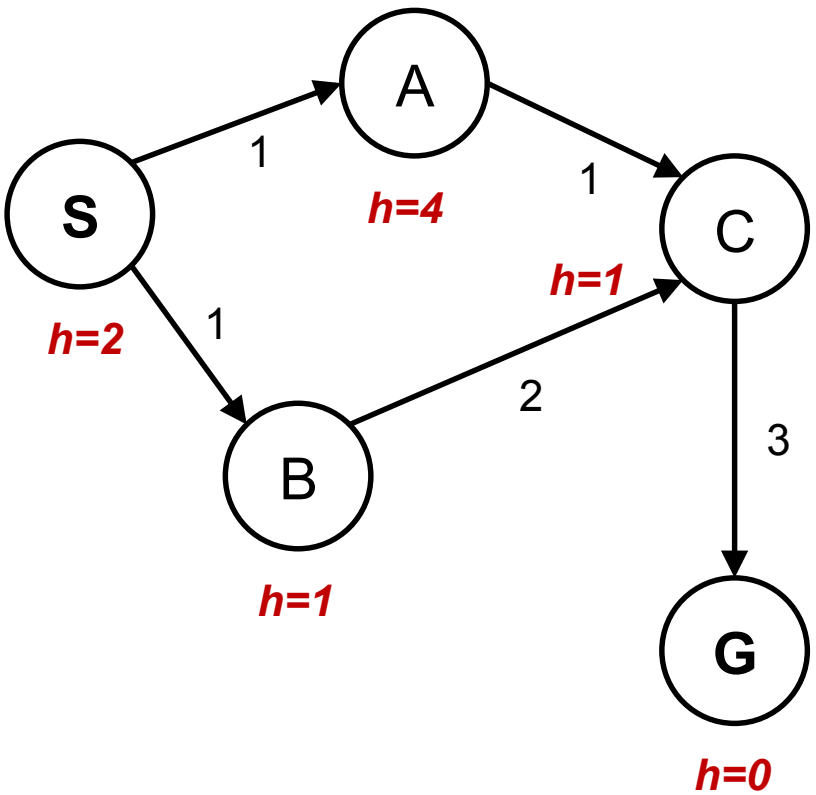
$h(A) \leq$ true cost $A \to G$

Consistent h:

$h(A) - h(C) \leq$ true cost $A \to C$

f along a path never decreases!

Consistent heuristics



Admissible h:
$h(A) \leq$ true cost $A \to G$

Consistent h:
$h(A) - h(C) \leq$ true cost $A \to C$

f along a path never decreases!

Optimality of A*

- admissible h for tree search
- consistent h for graph search
- What about UCS?
- Are all consistent heuristics also admissible?
  $h(A) - h(C) \leq \text{cost}(A \to C)$

Optimality of A*

- admissible h for tree search
- consistent h for graph search
- What about UCS?
- Are all consistent heuristics also admissible?
  $h(A) - h(C) \leq \text{cost}(A \to C)$

Optimality of A*

- admissible h for tree search
- consistent h for graph search
- What about UCS?
- Are all consistent heuristics also admissible?

$$h(A) - h(C) \leq \text{cost}(A \to C)$$

Optimality of A*

- admissible h for tree search
- consistent h for graph search
- What about UCS?
- Are all consistent heuristics also admissible?
  $h(A) - h(C) \leq \text{cost}(A \rightarrow C)$

How to find a heuristics?

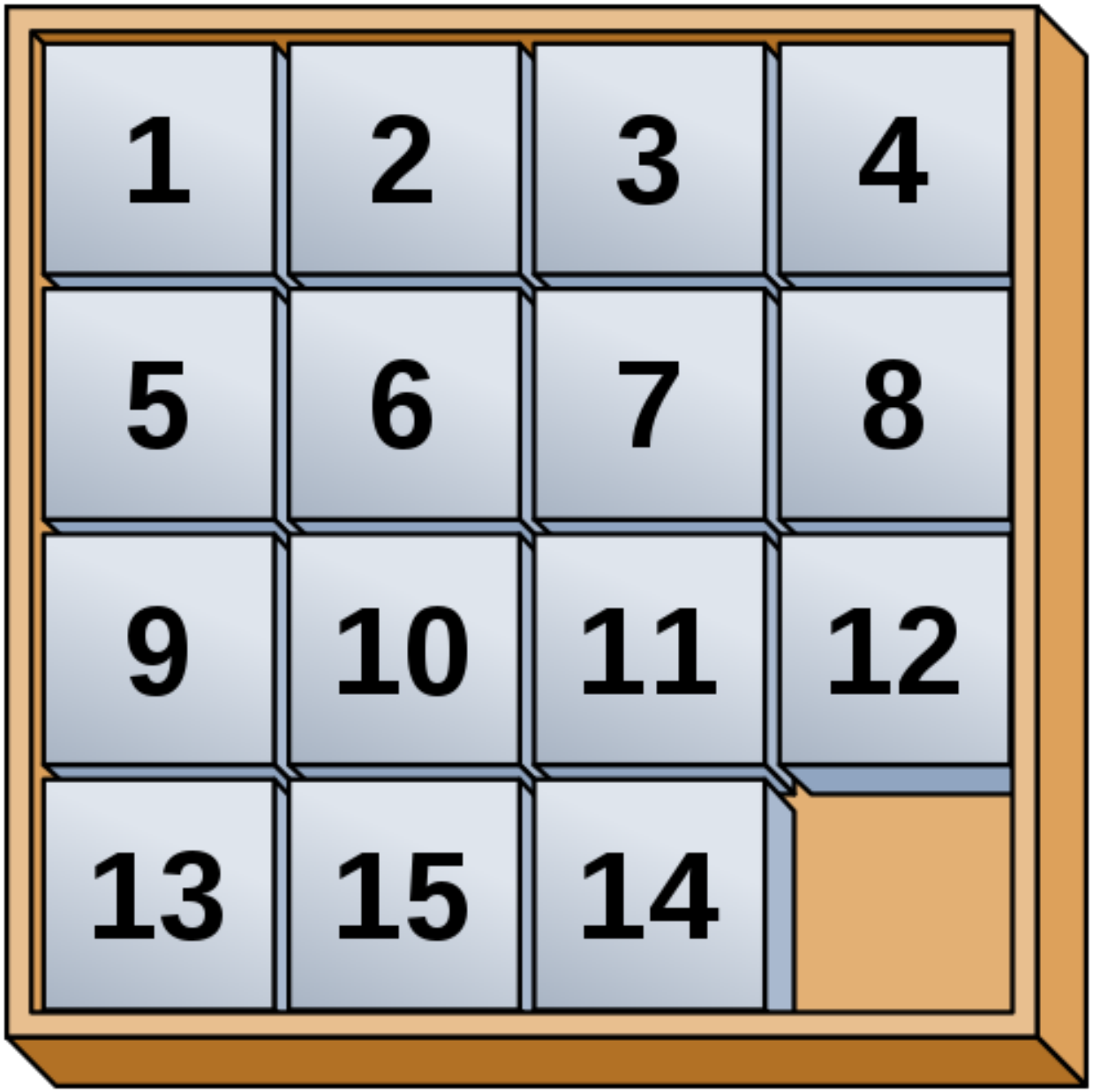


Image from: https://commons.wikimedia.org/wiki/File:15-puzzle-loyd.svg

Which heuristics is the best?