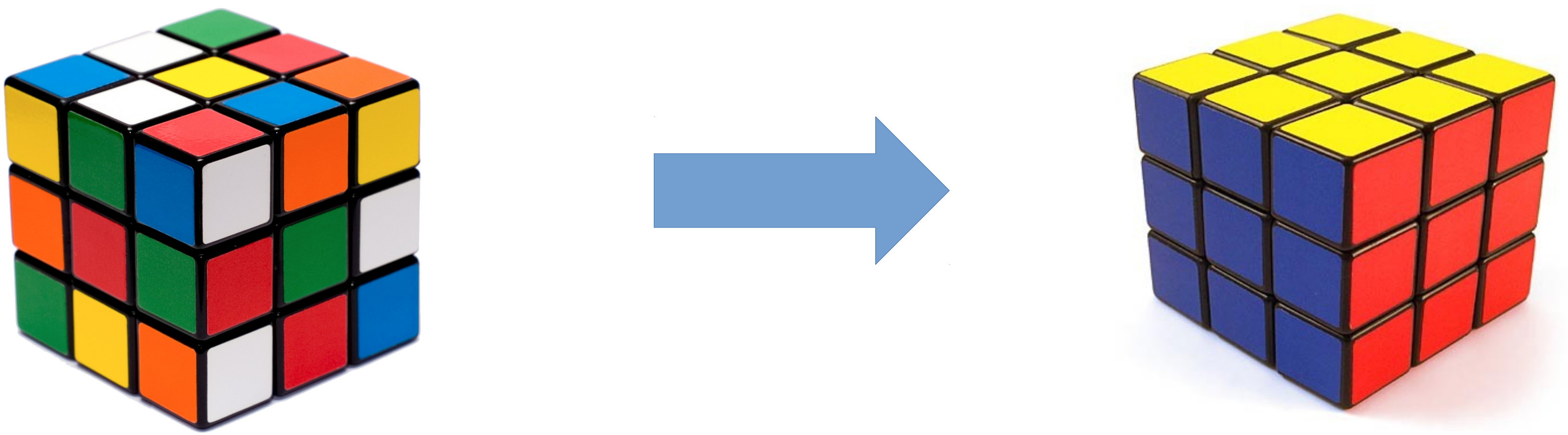
Graph Search

Rob Platt
Northeastern University

Some images and slides are used from:
AIMA
CS188 UC Berkeley

What is graph search?



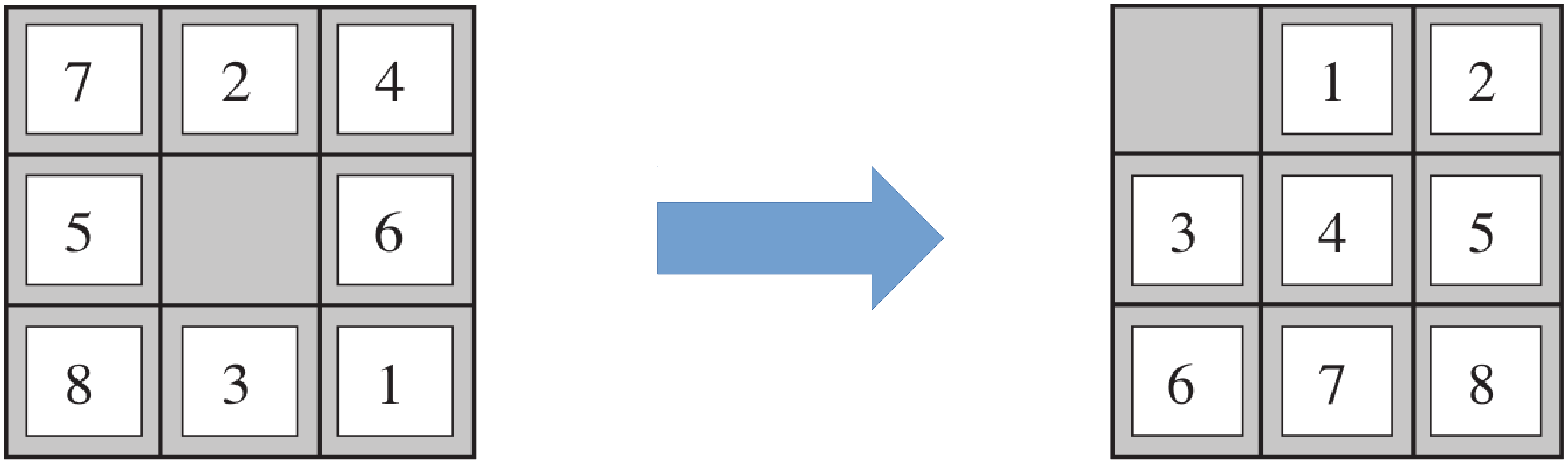
Start state

Goal state

Graph search: find a path from start to goal

- what are the states?
- what are the actions (transitions)?
- how is this a graph?

What is graph search?



Start state

Goal state

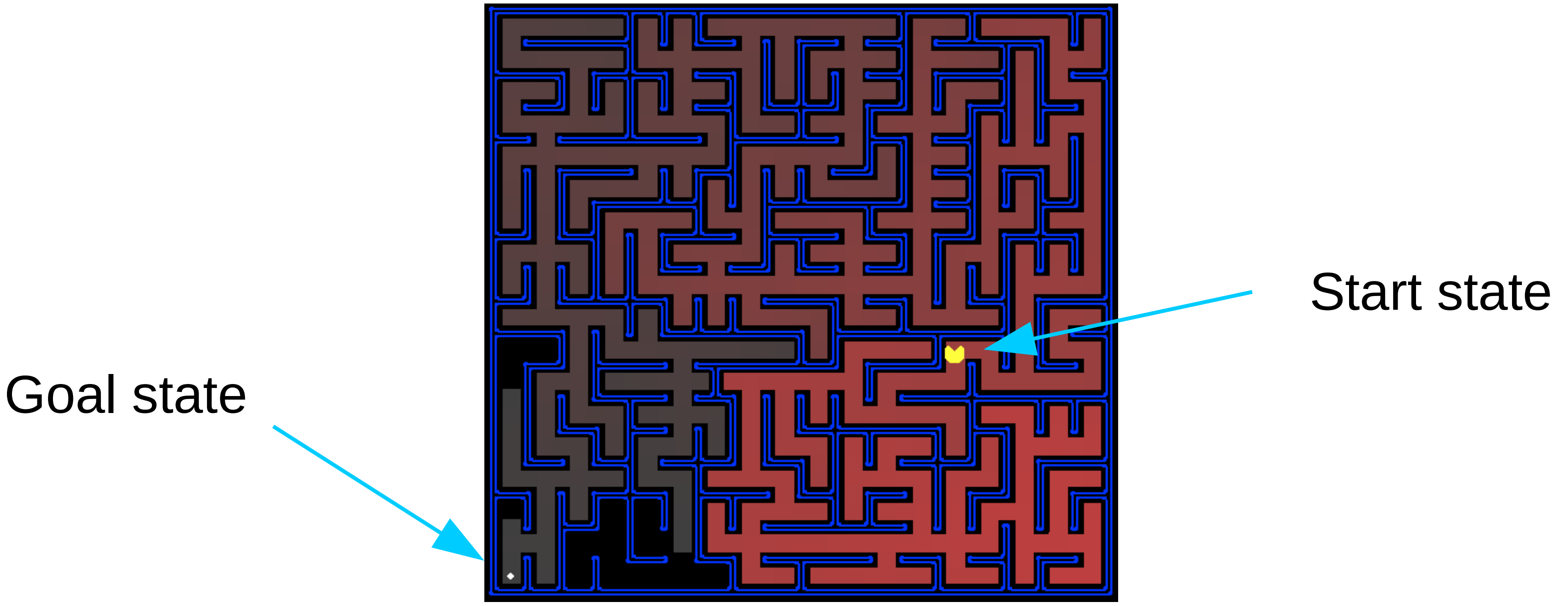
Graph search: find a path from start to goal

– what are the states?

– what are the actions (transitions)?

– how is this a graph?

What is graph search?



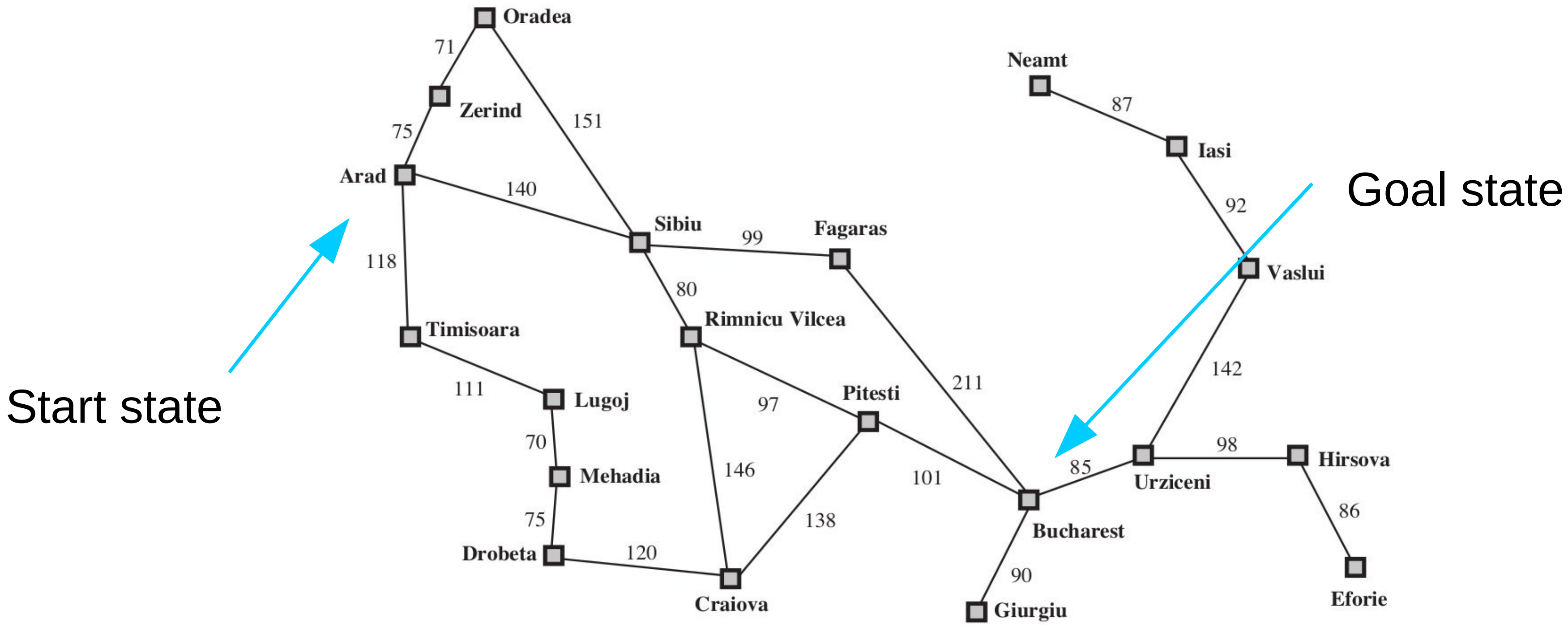
Graph search: find a path from start to goal

– what are the states?

– what are the actions (transitions)?

– how is this a graph?

What is graph search?



Graph search: find a path from start to goal

– what are the states?

– what are the actions (transitions)?

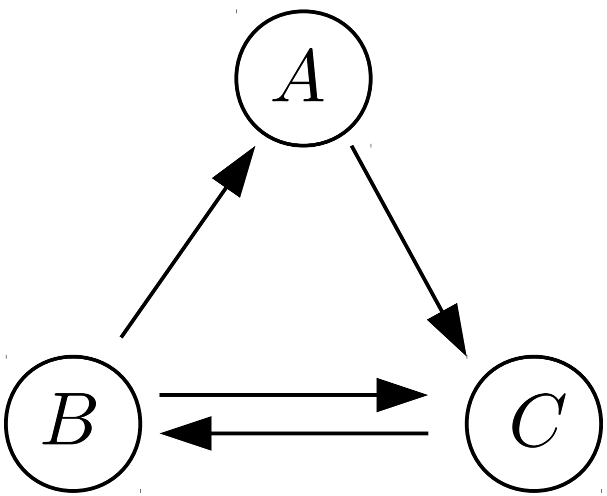
– how is this a graph?

What is a graph?

Graph: $G = (V, E)$

Vertices: V

Edges: E



Directed graph

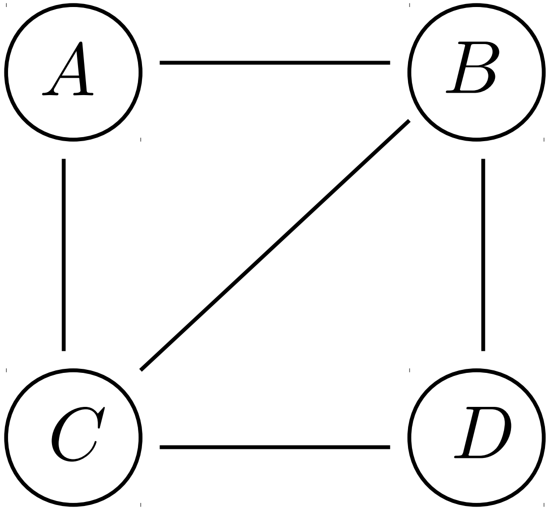
$$V = \{A, B, C\}$$
$$E = \{(B, A), (A, C), (B, C), (C, B)\}$$

What is a graph?

Graph: $G = (V, E)$

Vertices: V

Edges: E



<u>Undirected</u> graph

$$V = \{A, B, C, D\}$$

$$E = \{\{A, C\}, \{A, B\}, \{C, D\}, \{B, D\}, \{C, B\}\}$$

What is a graph?

Graph: $G = (V, E)$

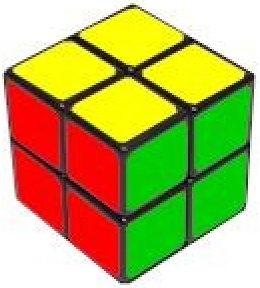
Vertices: $V \longleftarrow$ Also called *states*

Edges: $E \longleftarrow$ Also called *transitions*

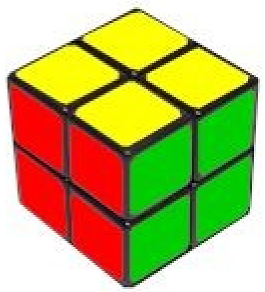
Defining a graph: example

$V = ?$

$E = ?$



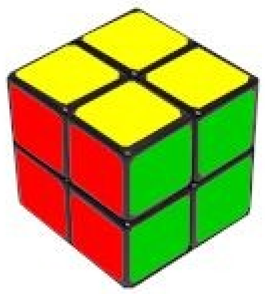
Defining a graph: example



$V = ?$ ← How many states?

$E = ?$

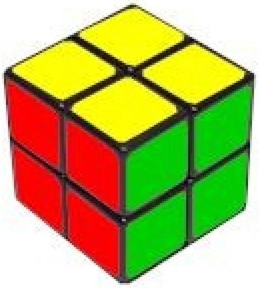
Defining a graph: example



$V = ?$ $\longleftarrow$ $|V| = 8! \times 3^8$

$E = ?$

Defining a graph: example



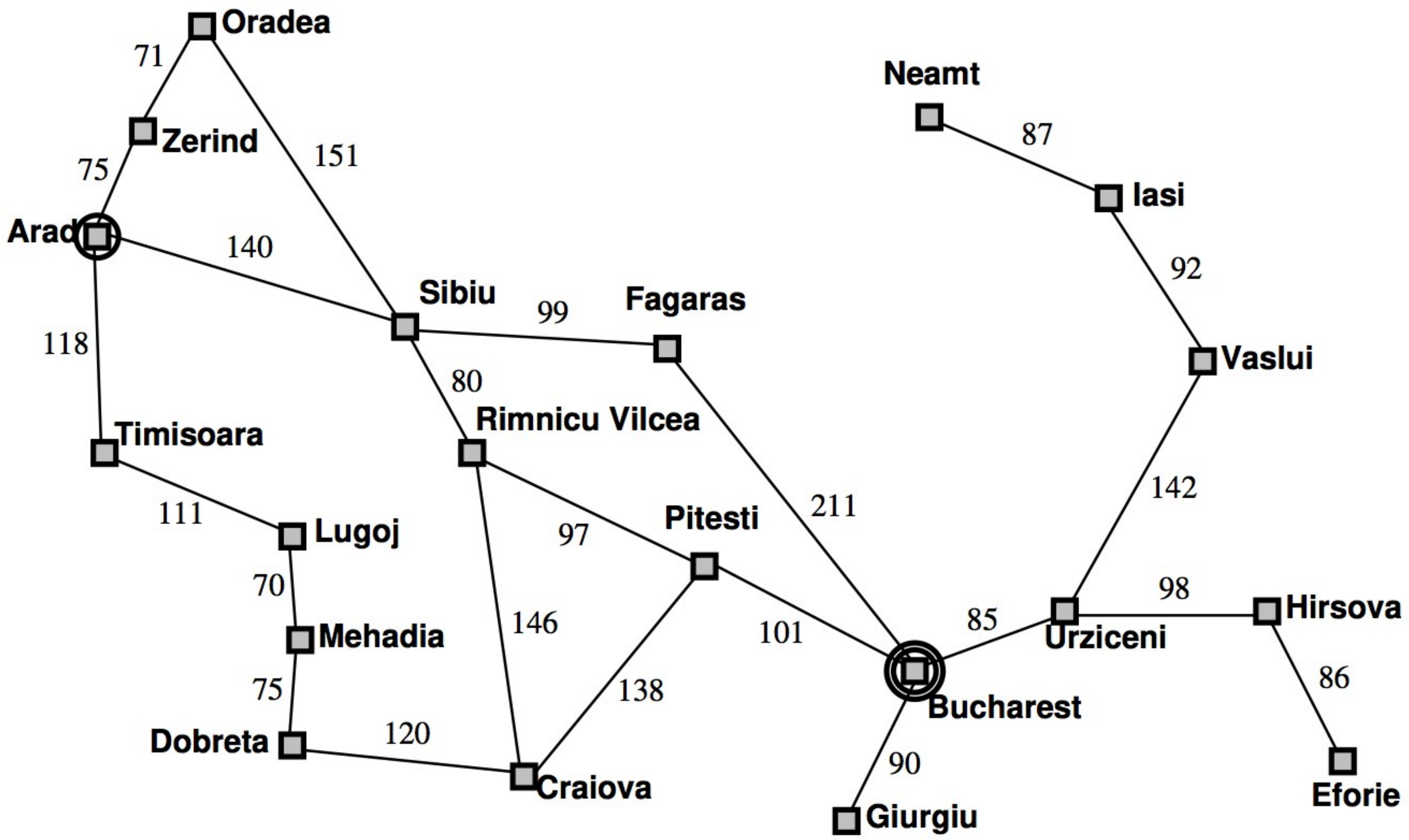
$V = ?$

$E = ?$ ← Pairs of states that are “connected” by one turn of the cube.

Example: Romania

- On holiday in Romania; currently in Arad. Flight leaves tomorrow from Bucharest
- Formulate goal: Be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Graph search



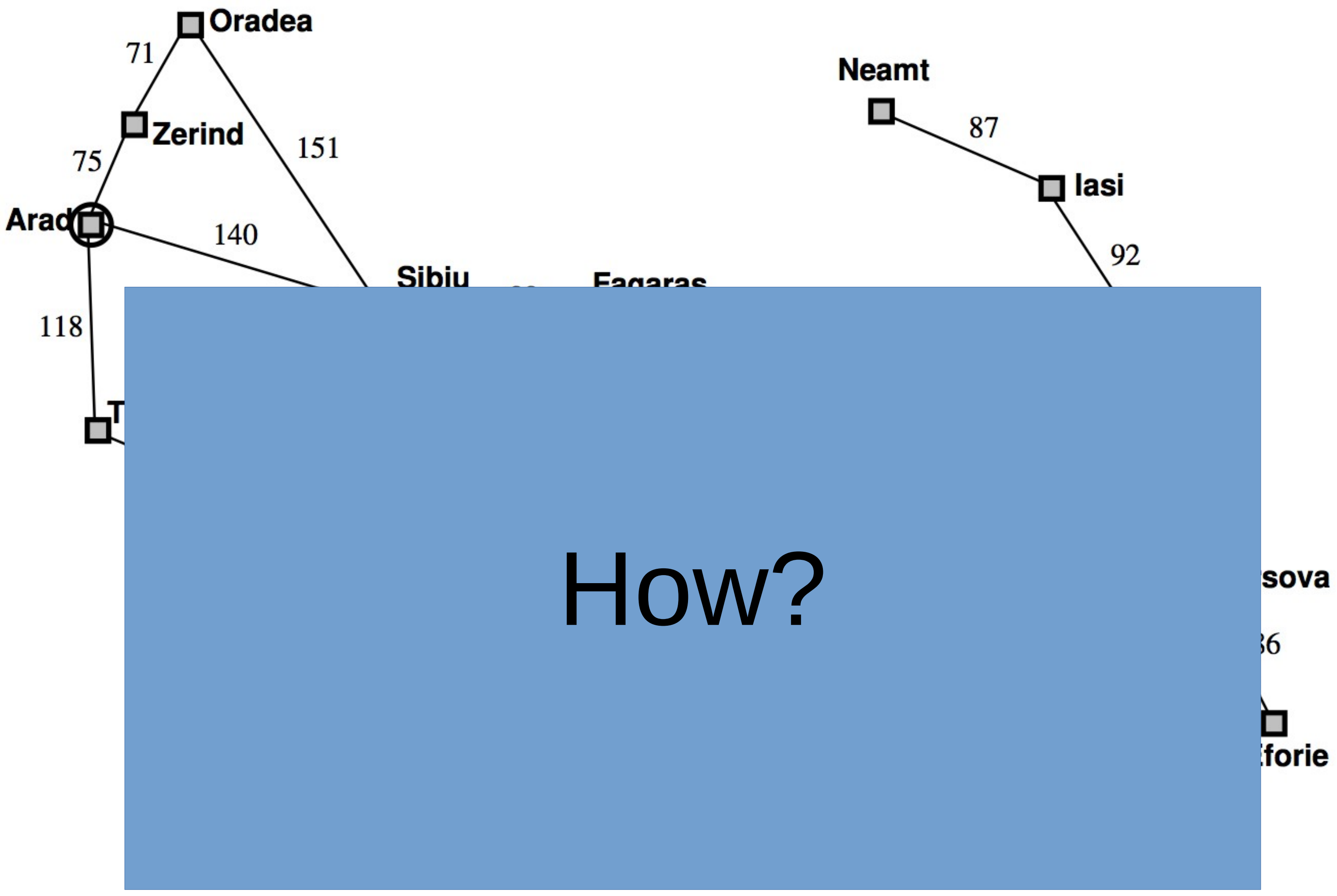
<u>Given</u>: a graph, *G*

<u>Problem</u>: find a path from A to B

– A: start state

– B: goal state

Graph search



Problem: find a path from A to B

– A: start state

– B: goal state

Problem formulation

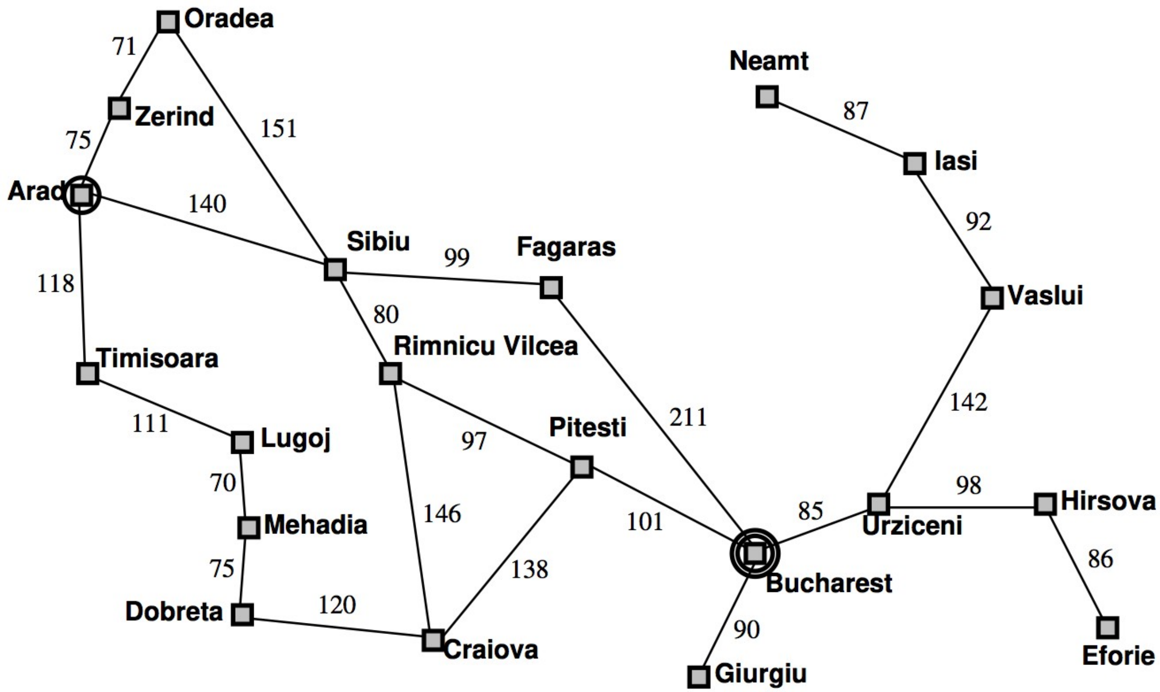
A problem is defined by four items:

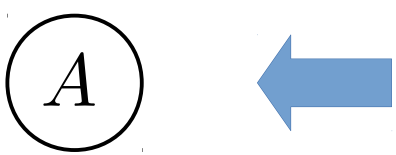
- initial state e.g., “at Arad”
- successor function $S(x)$ = set of action–state pairs

  e.g., $S(\text{Arad}) = \{\langle \text{Arad} \rightarrow \text{Zerind}, \text{Zerind}\rangle, \ldots\}$
- goal test, can be explicit, e.g., x = “at Bucharest” implicit, e.g., $NoDirt(x)$
- path cost (additive)

  e.g., sum of distances, number of actions executed, etc. $c(x, a, y)$ is the step cost, assumed to be ≥ 0
- A solution is a sequence of actions leading from the initial state to a goal state

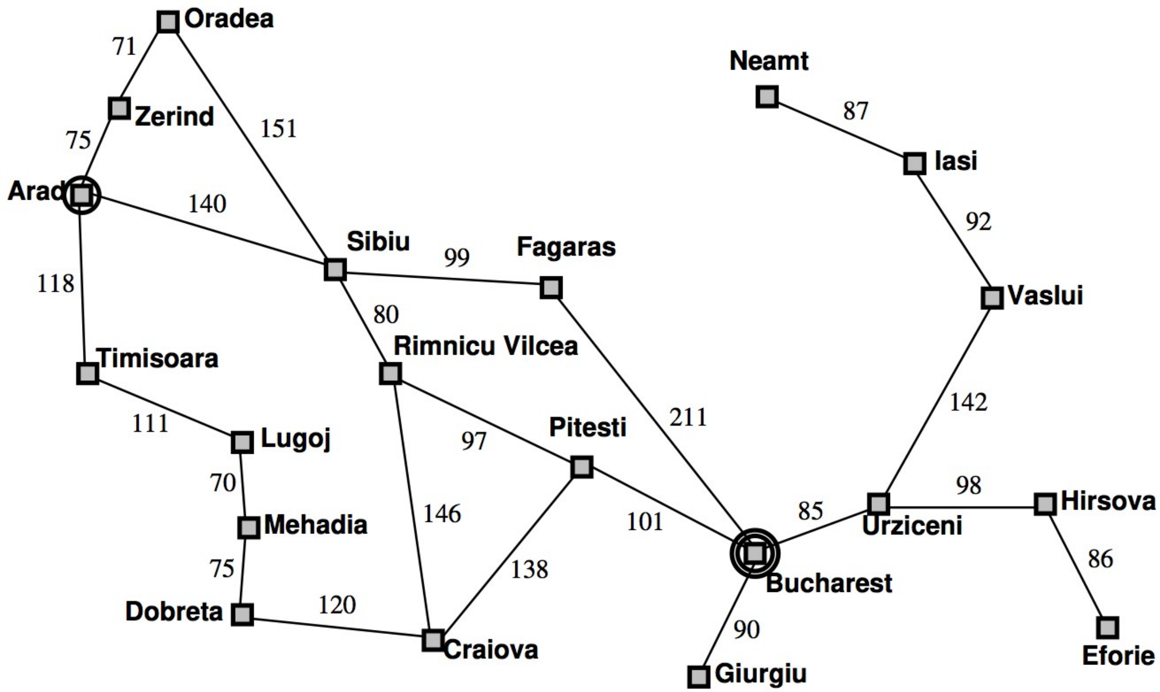
A search tree

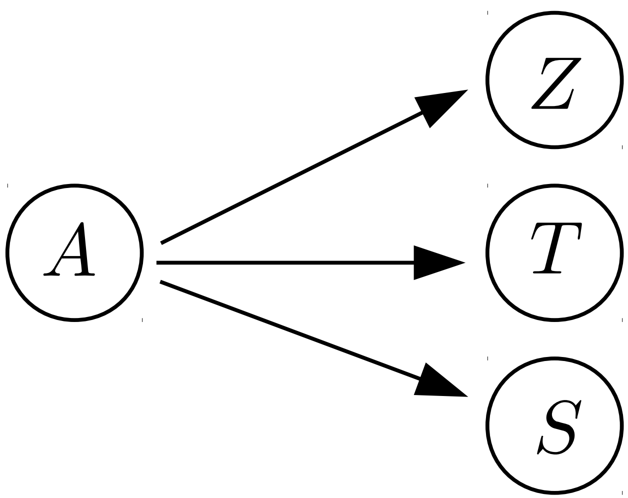




Start at A

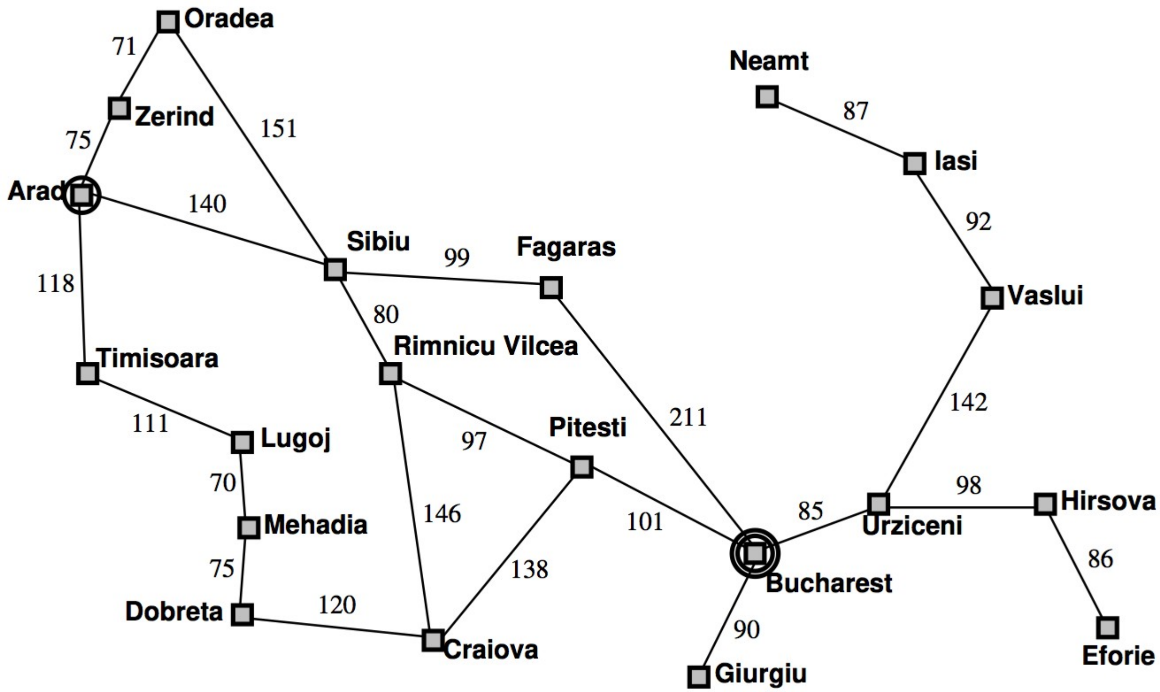
A search tree

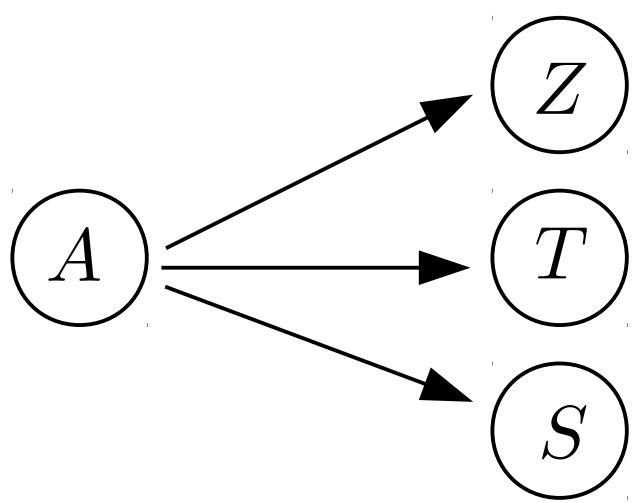




Successors of *A*

A search tree



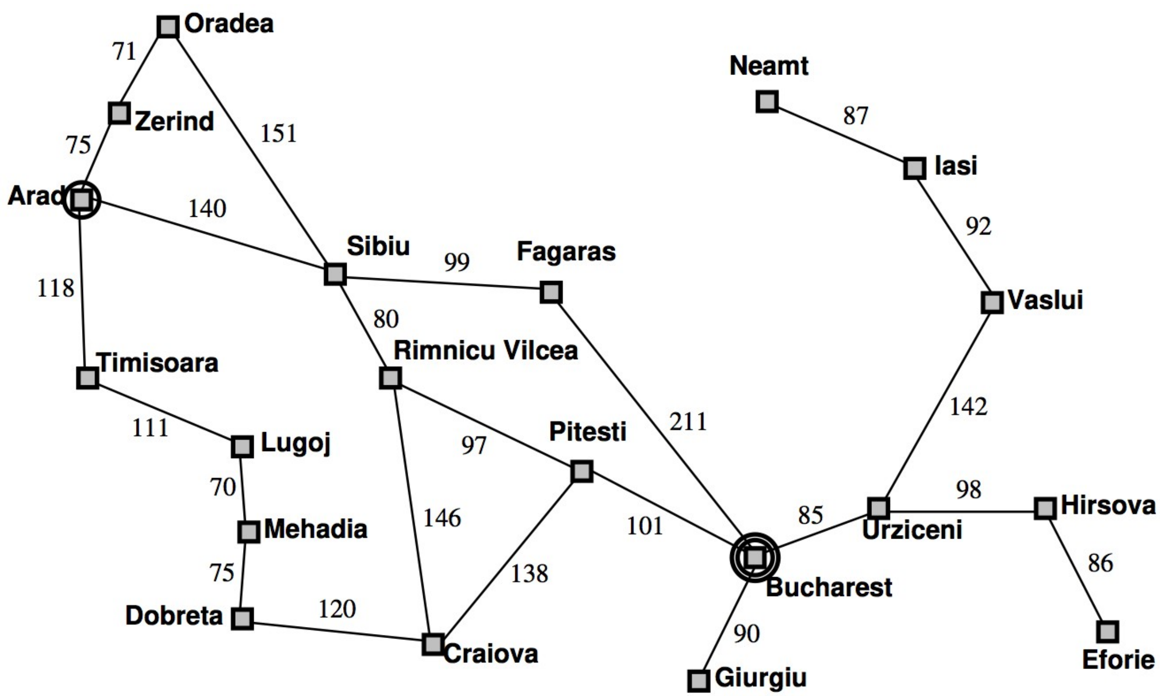


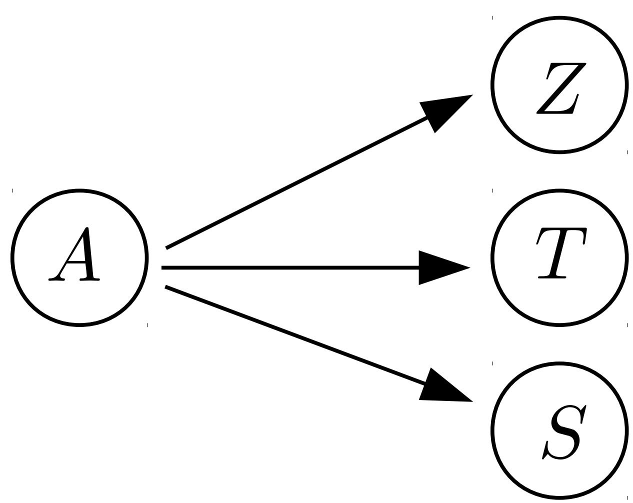
Successors of *A*

parent

children

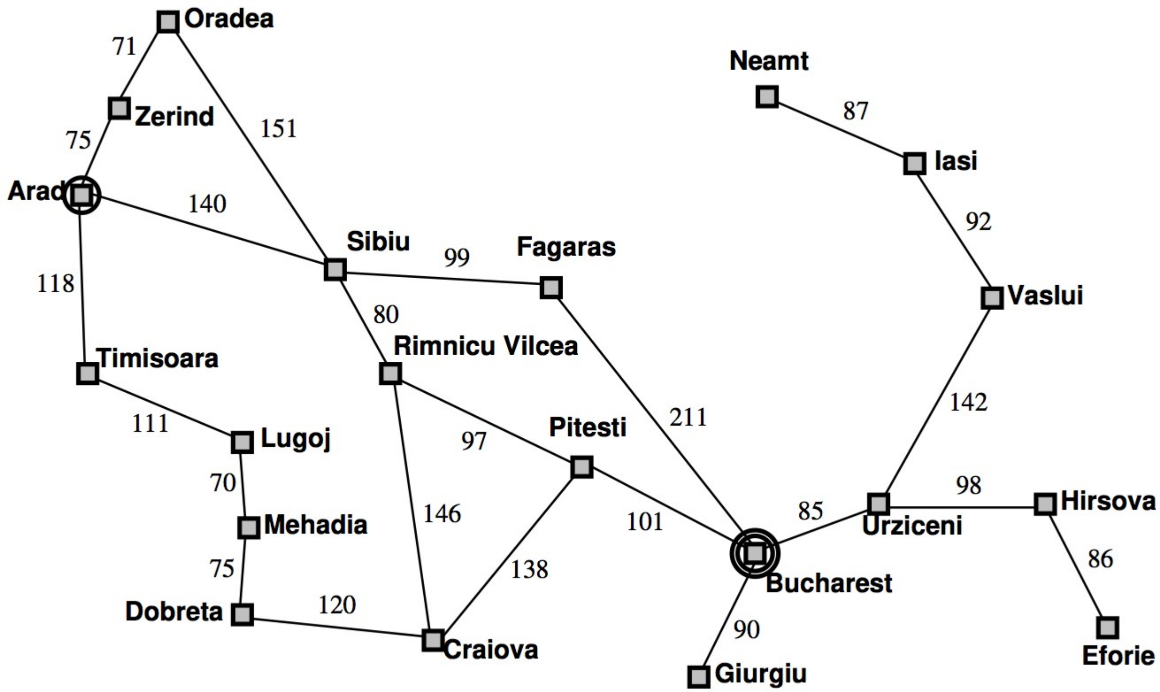
A search tree

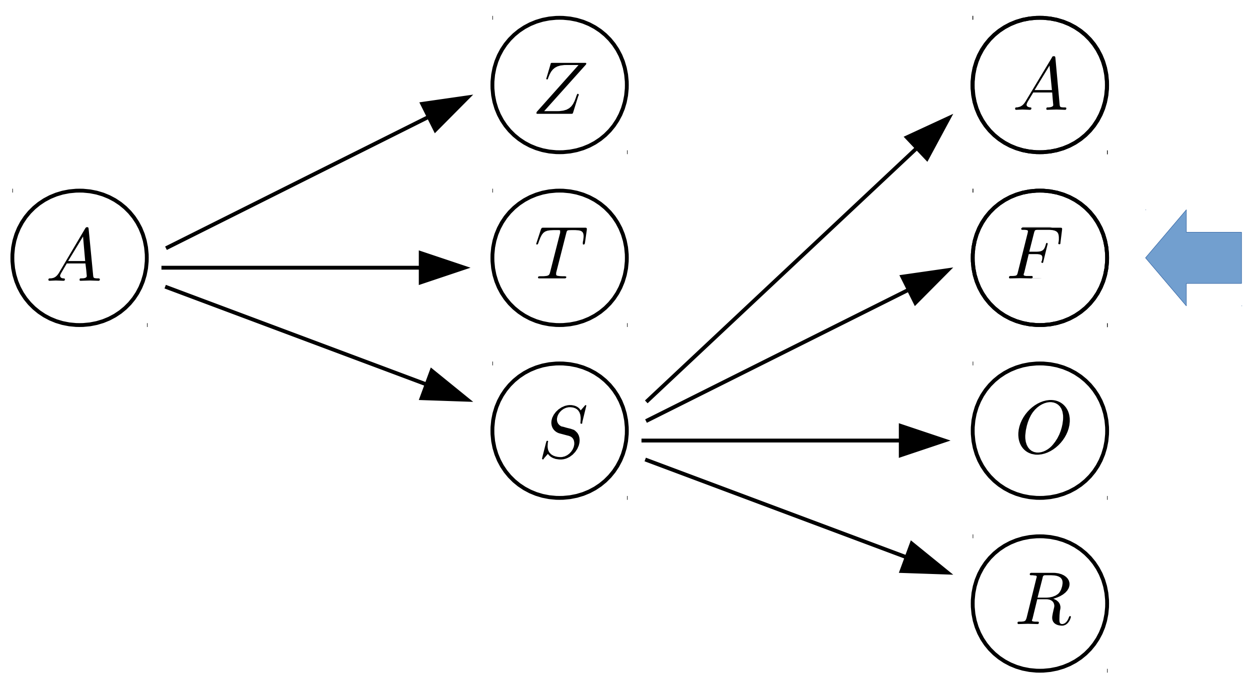




Let's expand *S* next

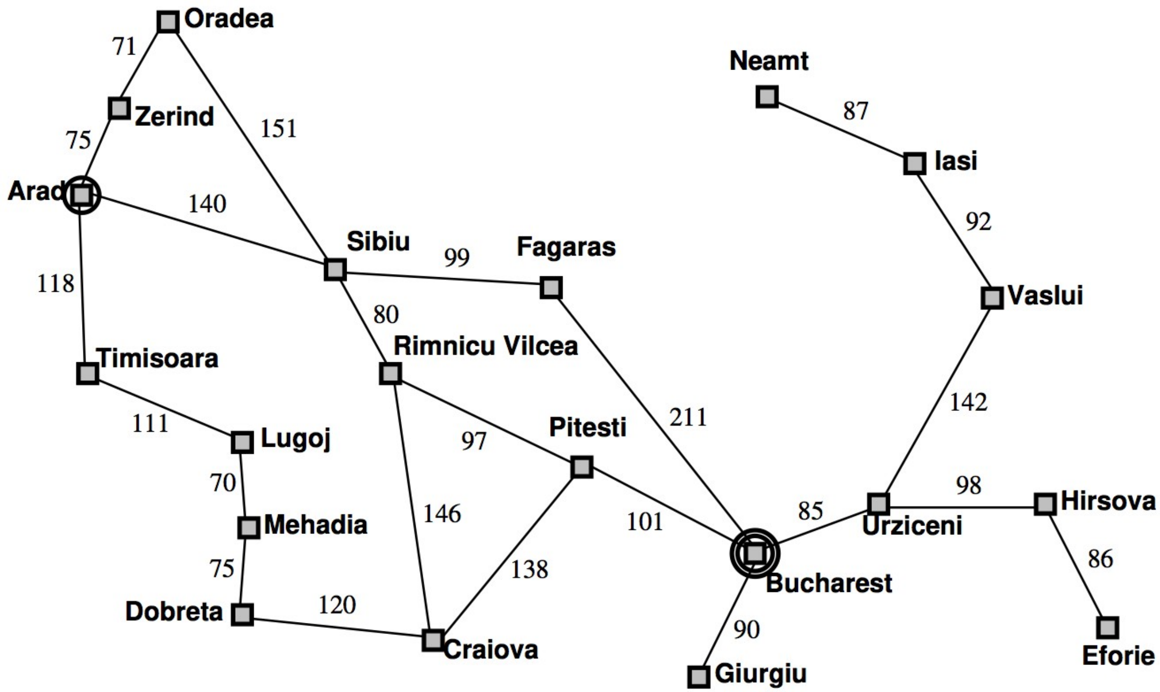
A search tree

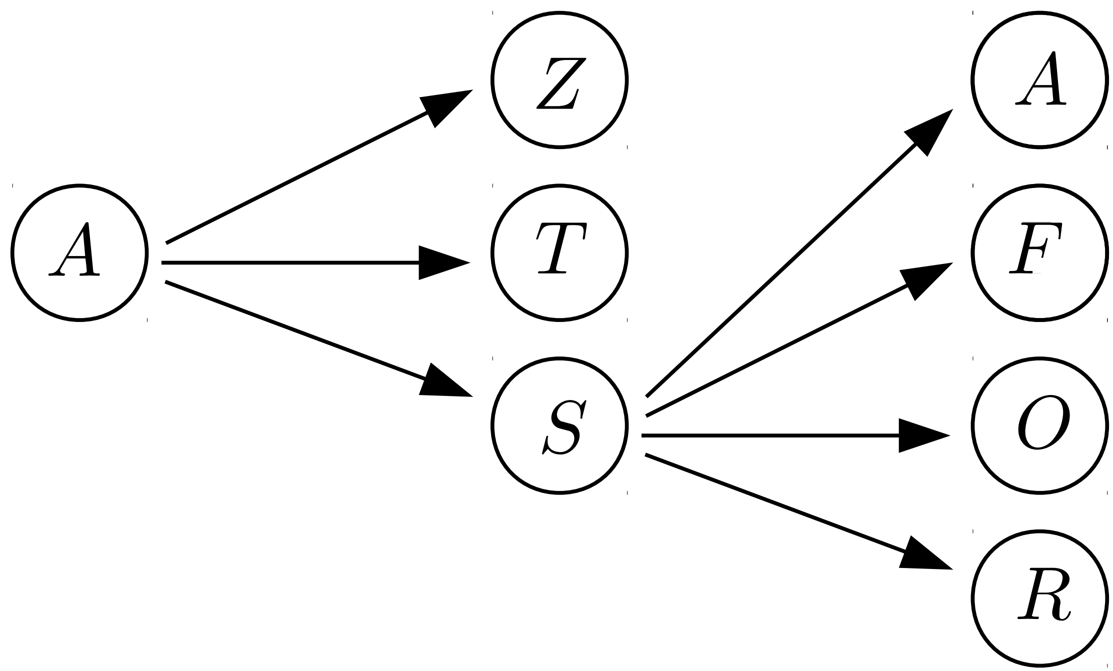




Successors of *S*

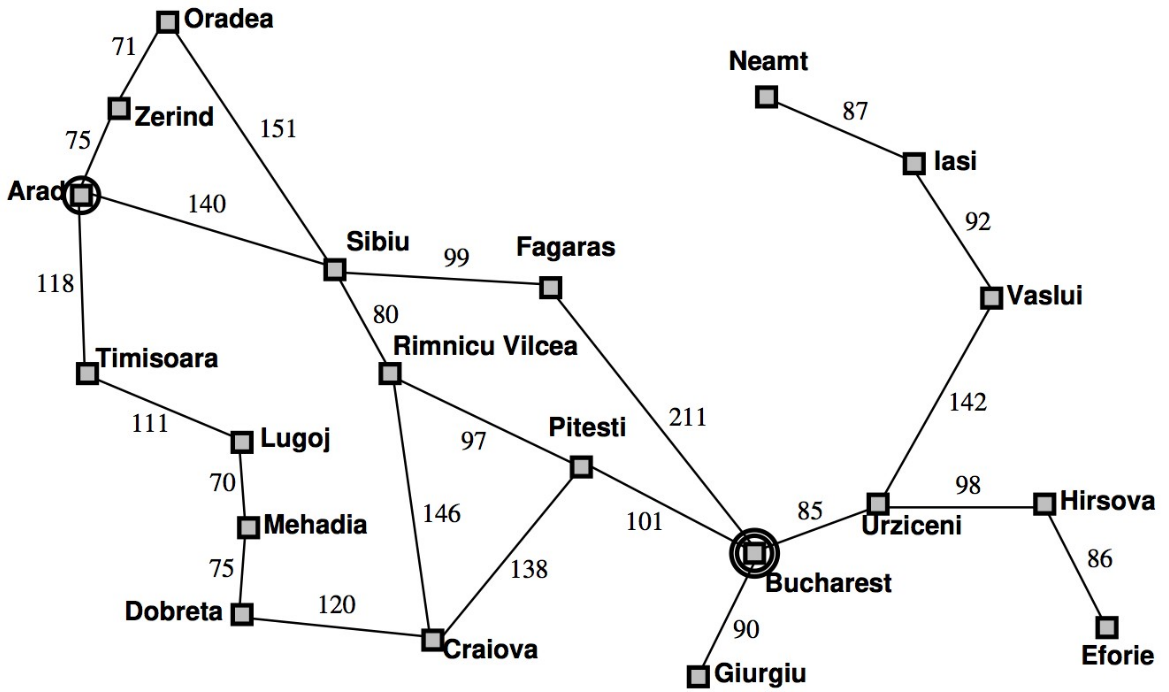
A search tree

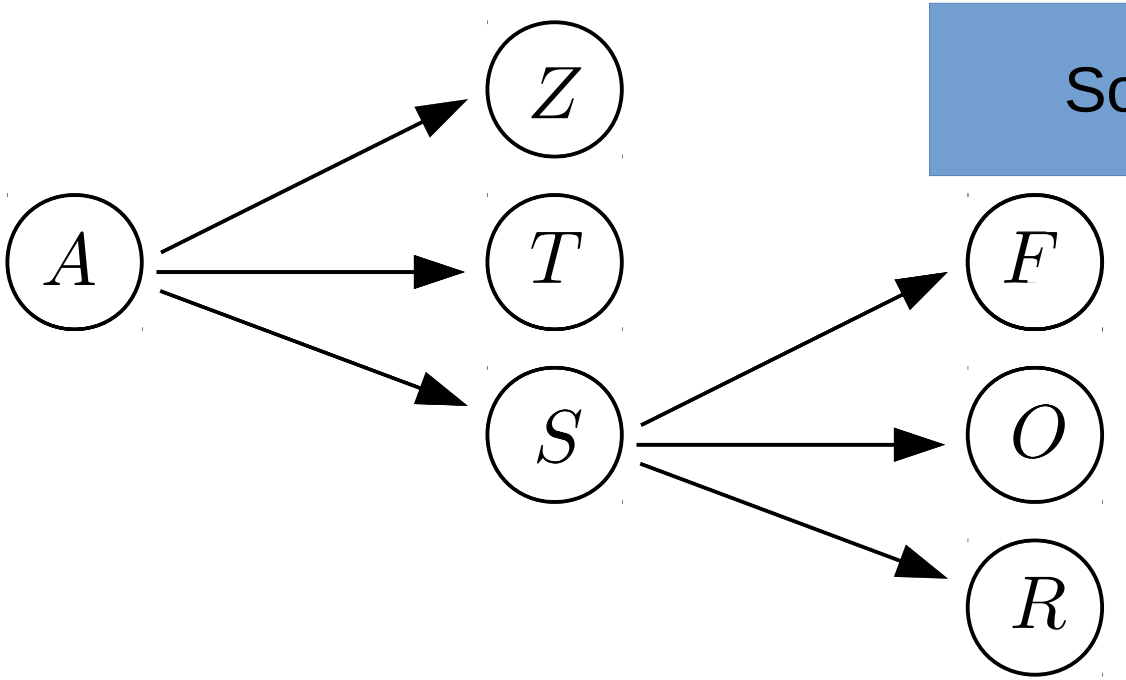




A was already visited!

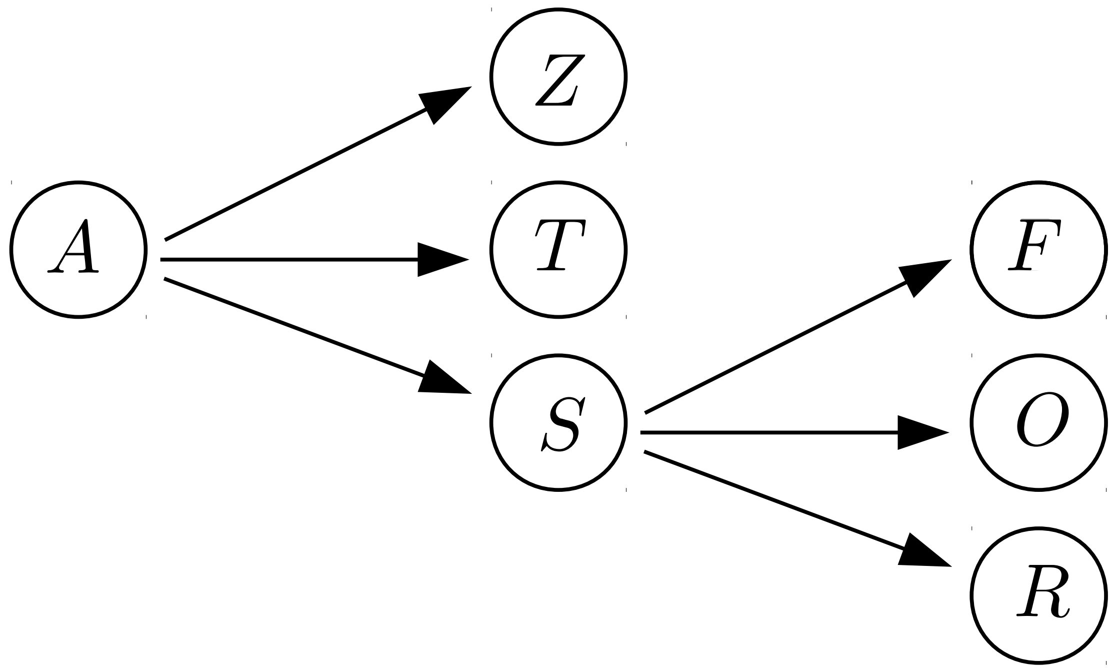
A search tree





So, prune it!

A search tree



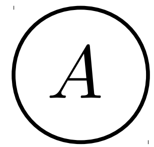
<u>In what order should we expand states?</u>

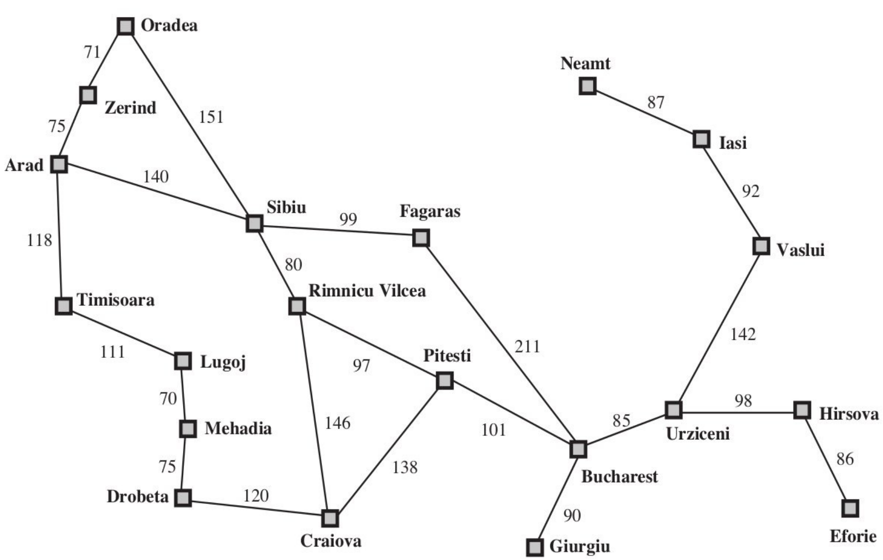
– here, we expanded *S*, but we could also have expanded *Z* or *T*

– different search algorithms expand in different orders

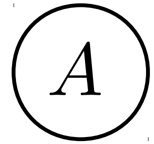
Breadth first search (BFS)

Breadth first search (BFS)

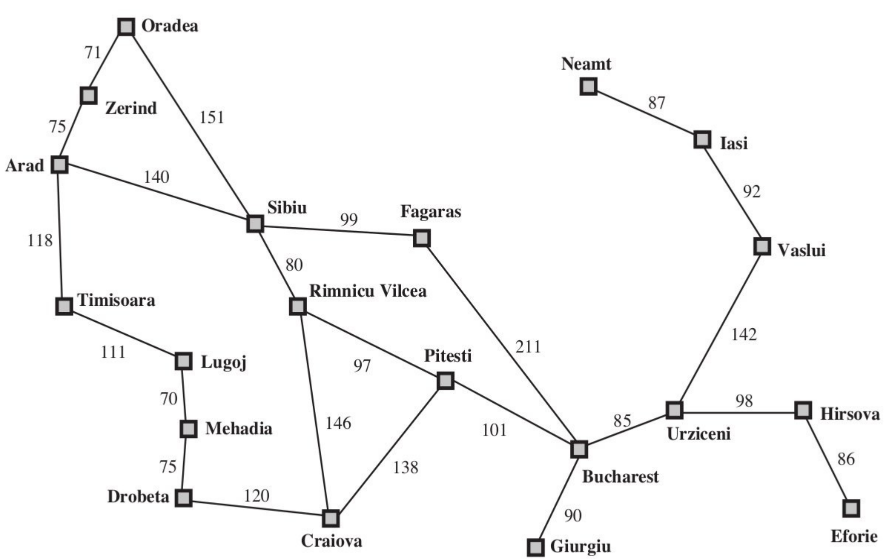




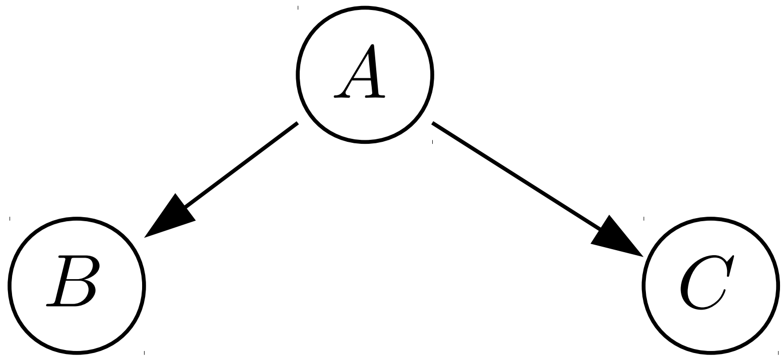
Breadth first search (BFS)

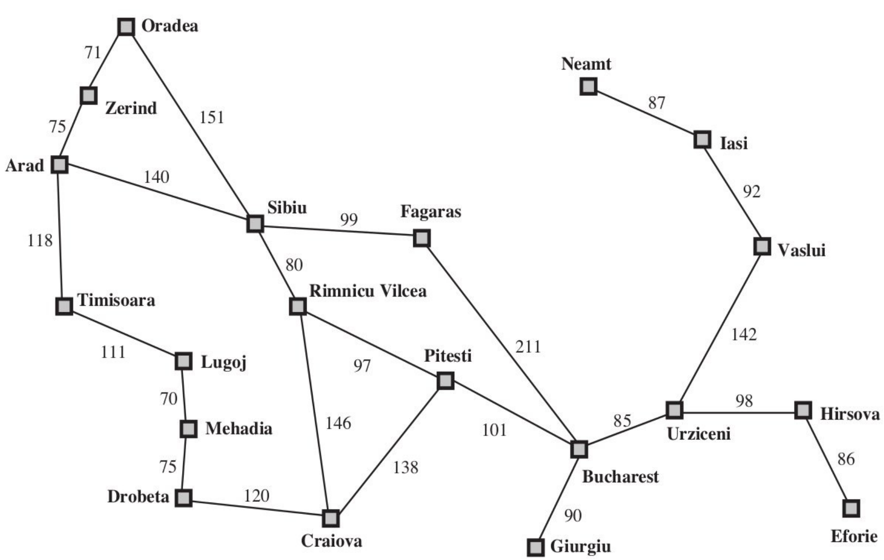


Start node

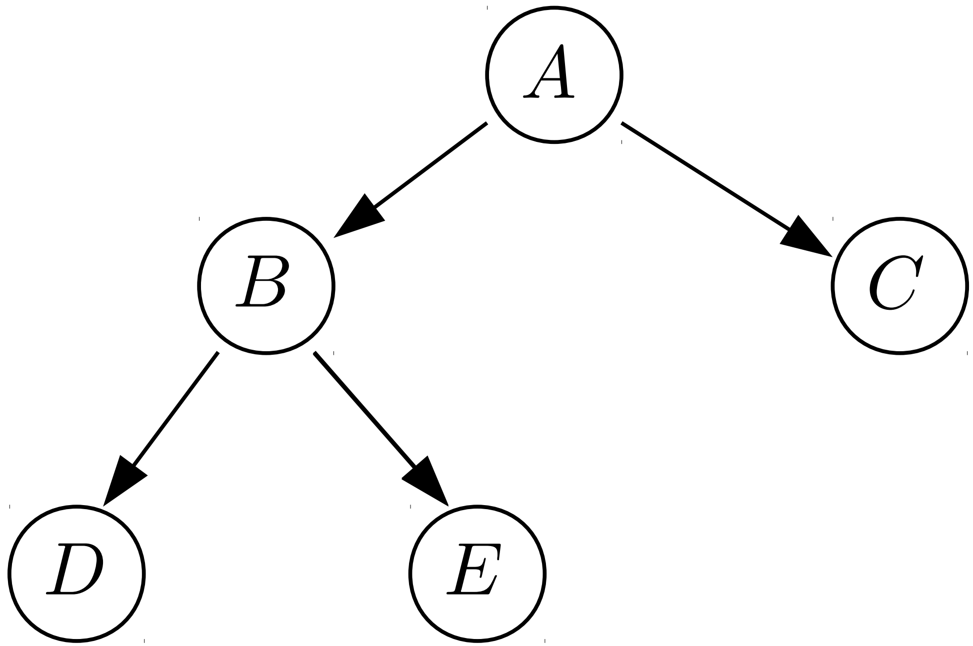


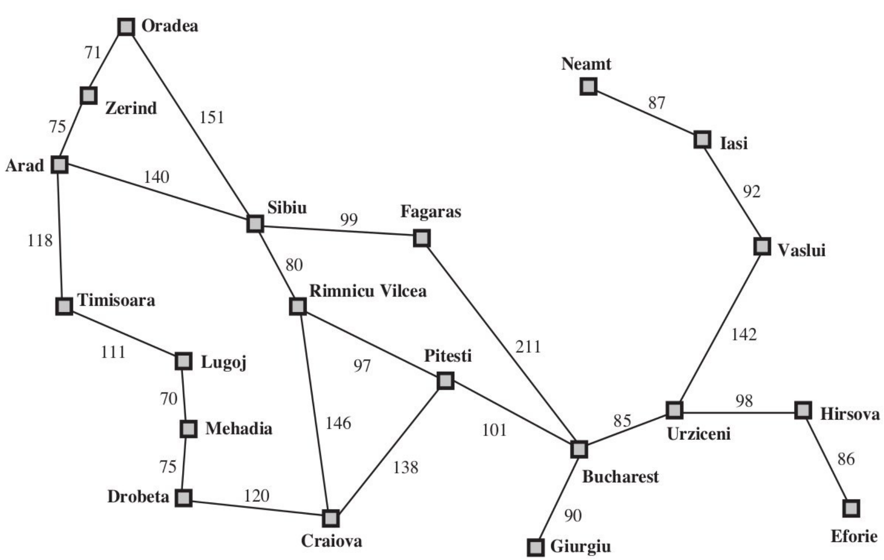
Breadth first search (BFS)



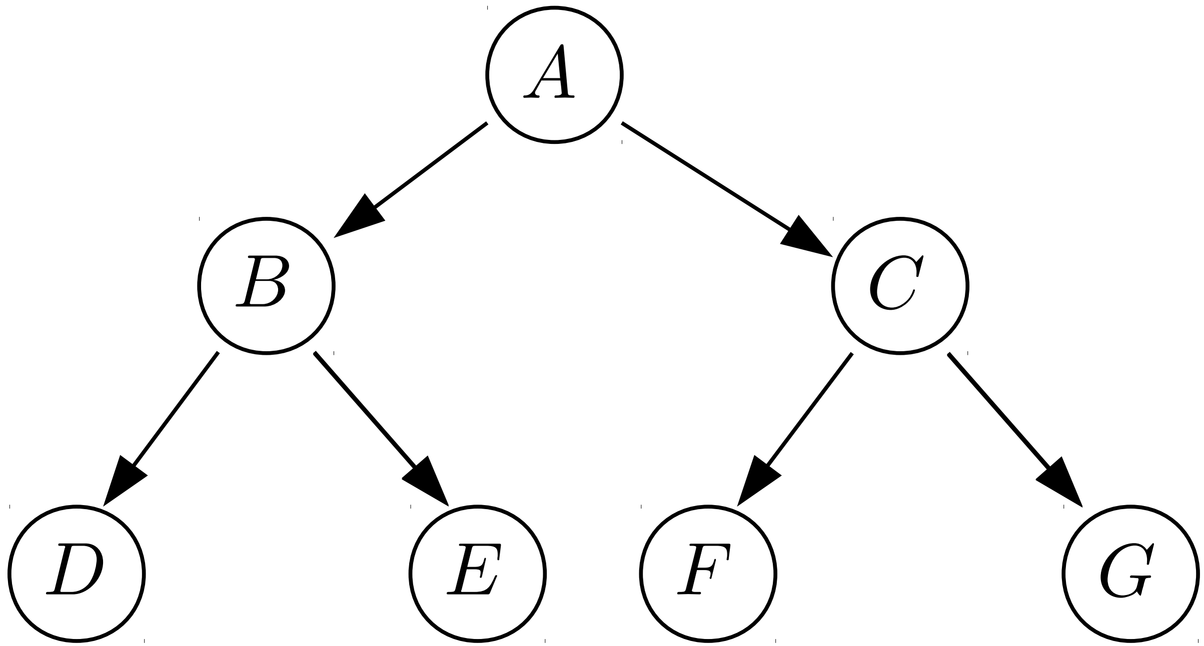


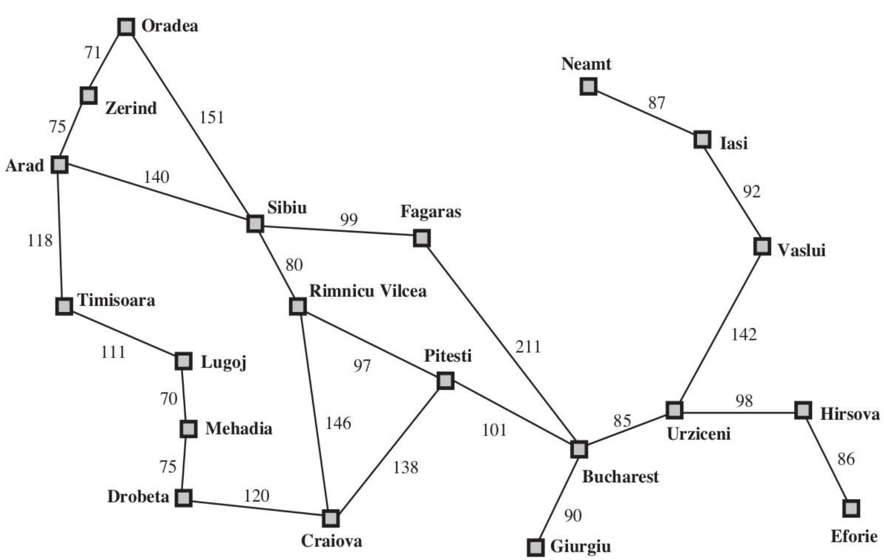
Breadth first search (BFS)





Breadth first search (BFS)





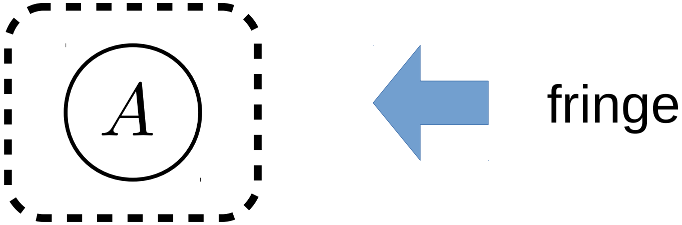
Breadth first search (BFS)

Fringe

We're going to maintain a queue called the fringe

– initialize the fringe as an empty queue

Breadth first search (BFS)

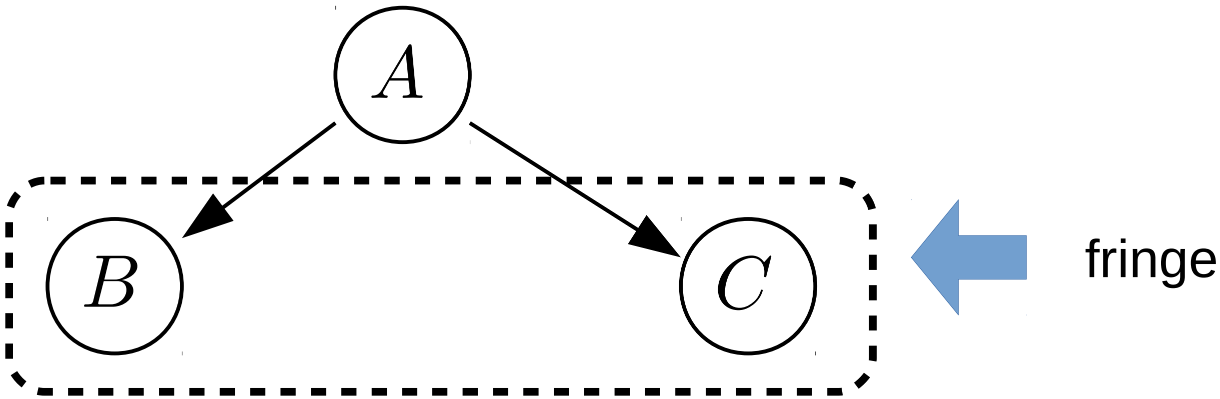


<u>Fringe</u>
A

– add *A* to the fringe

Breadth first search (BFS)

<u>Fringe</u>
B
C

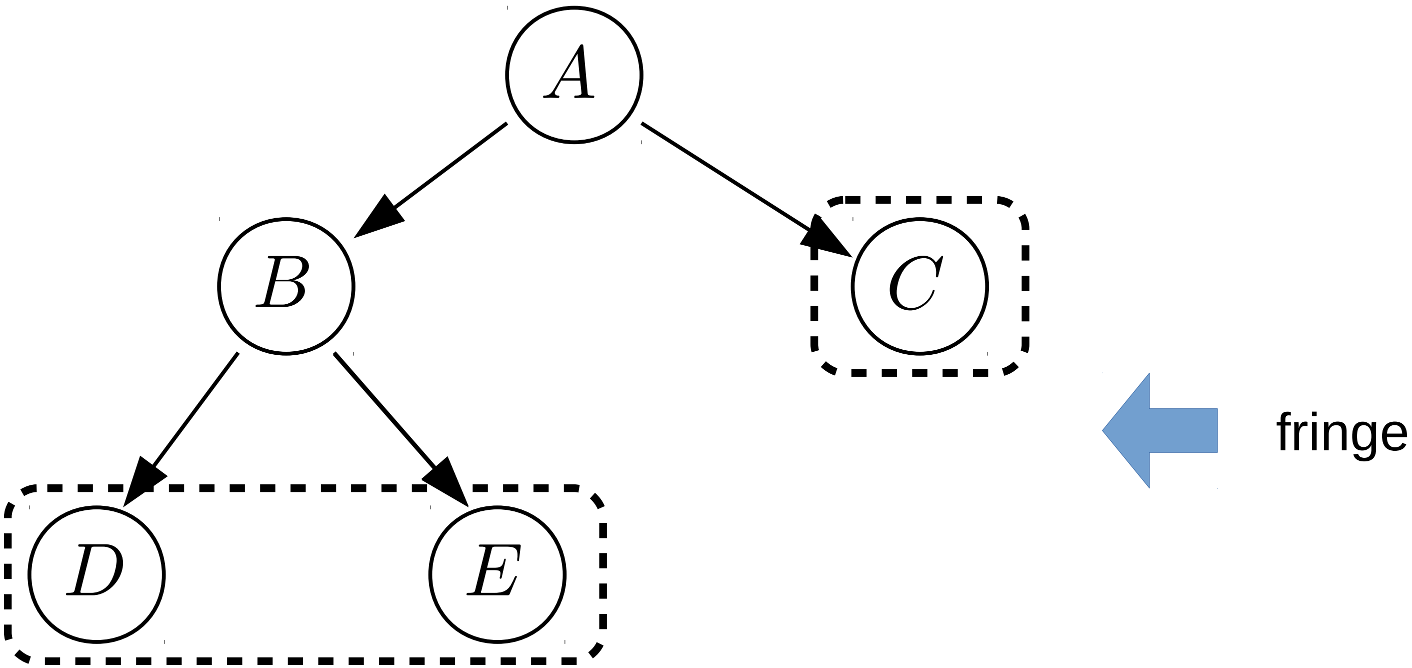


-- remove *A* from the fringe

-- add successors of *A* to the fringe

Breadth first search (BFS)

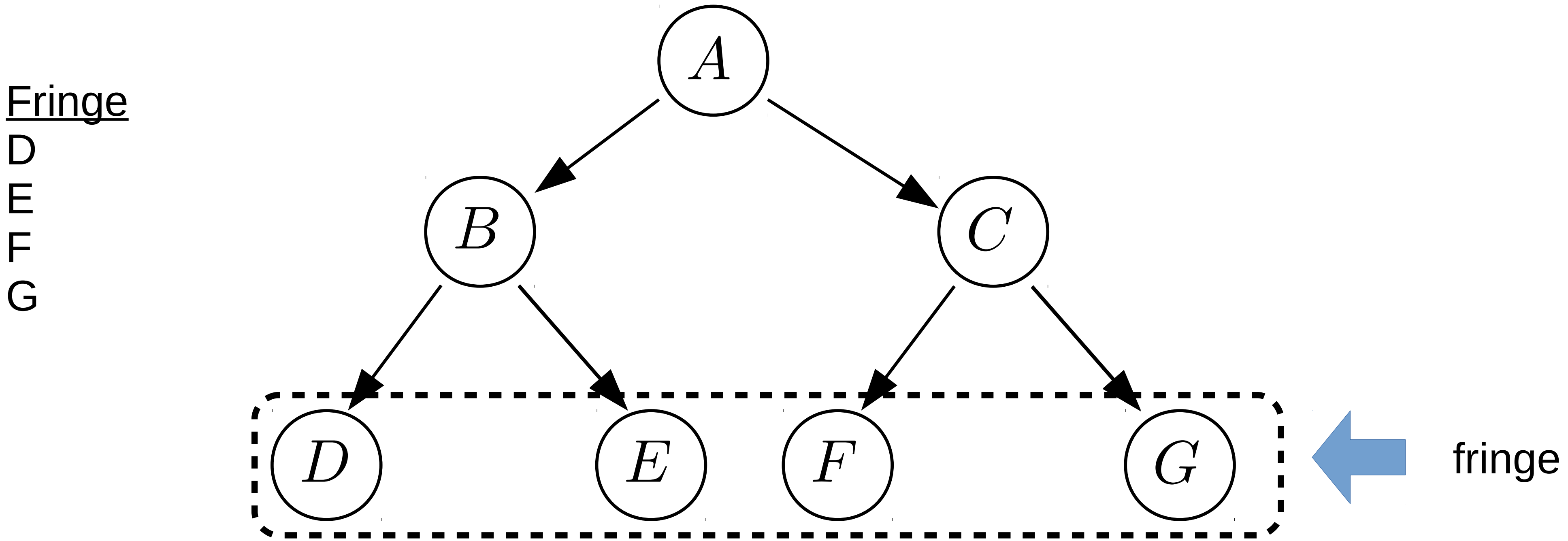
<u>Fringe</u>
C
D
E



-- remove *B* from the fringe

-- add successors of *B* to the fringe

Breadth first search (BFS)

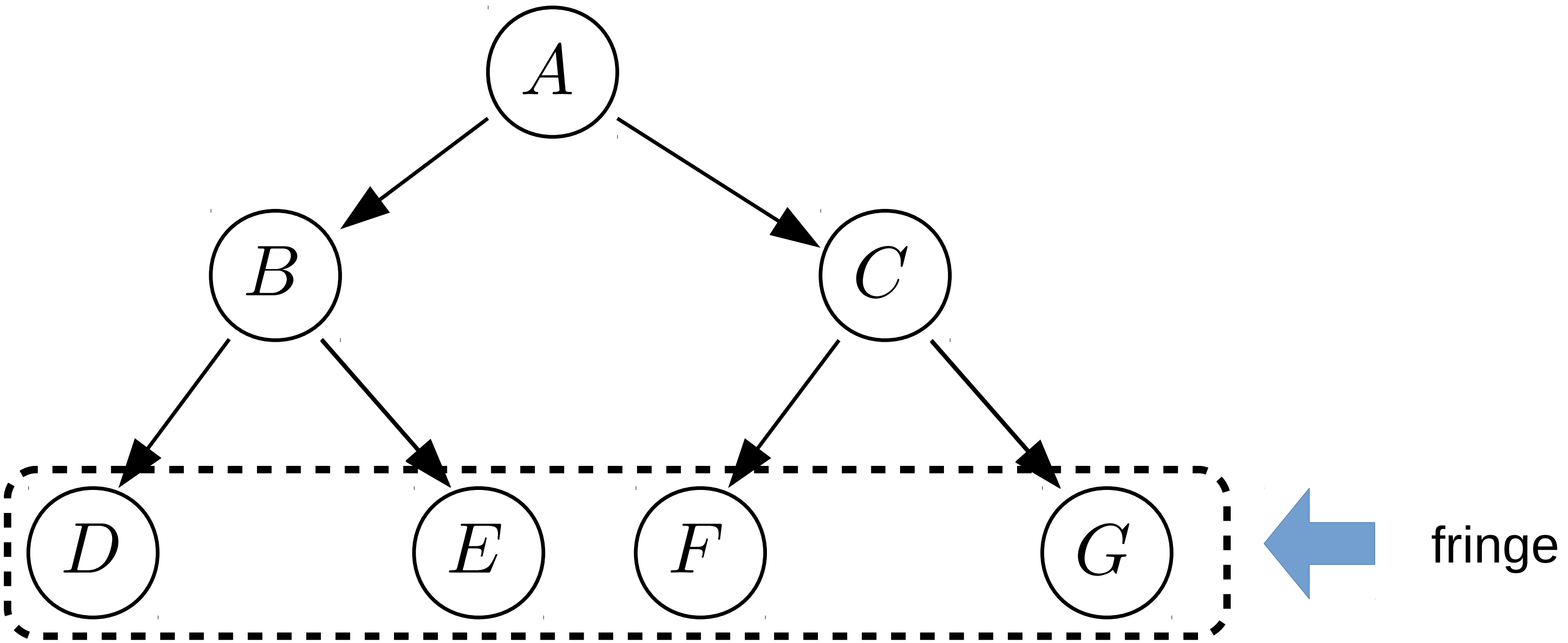


-- remove *C* from the fringe

-- add successors of *C* to the fringe

Breadth first search (BFS)

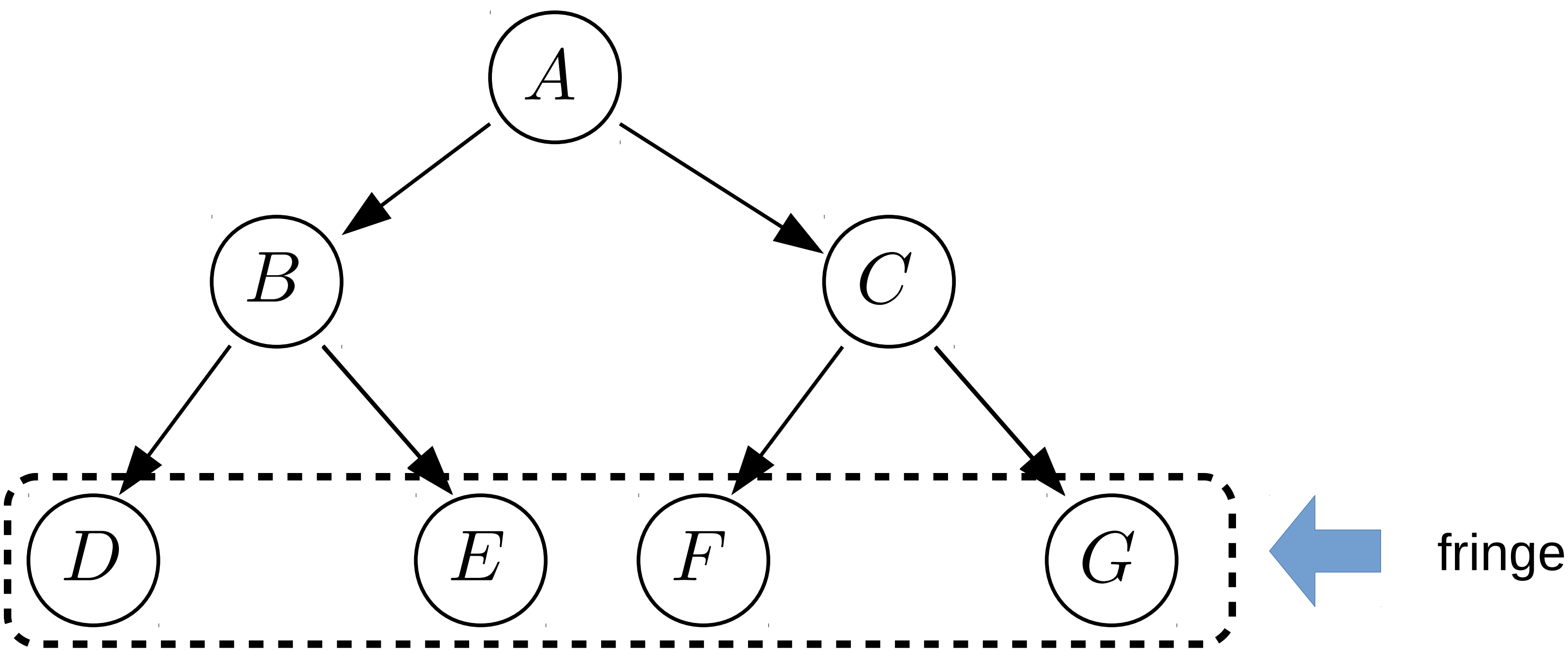
<u>Fringe</u>
D
E
F
G



Which state gets removed next from the fringe?

Breadth first search (BFS)

Fringe
D
E
F
G

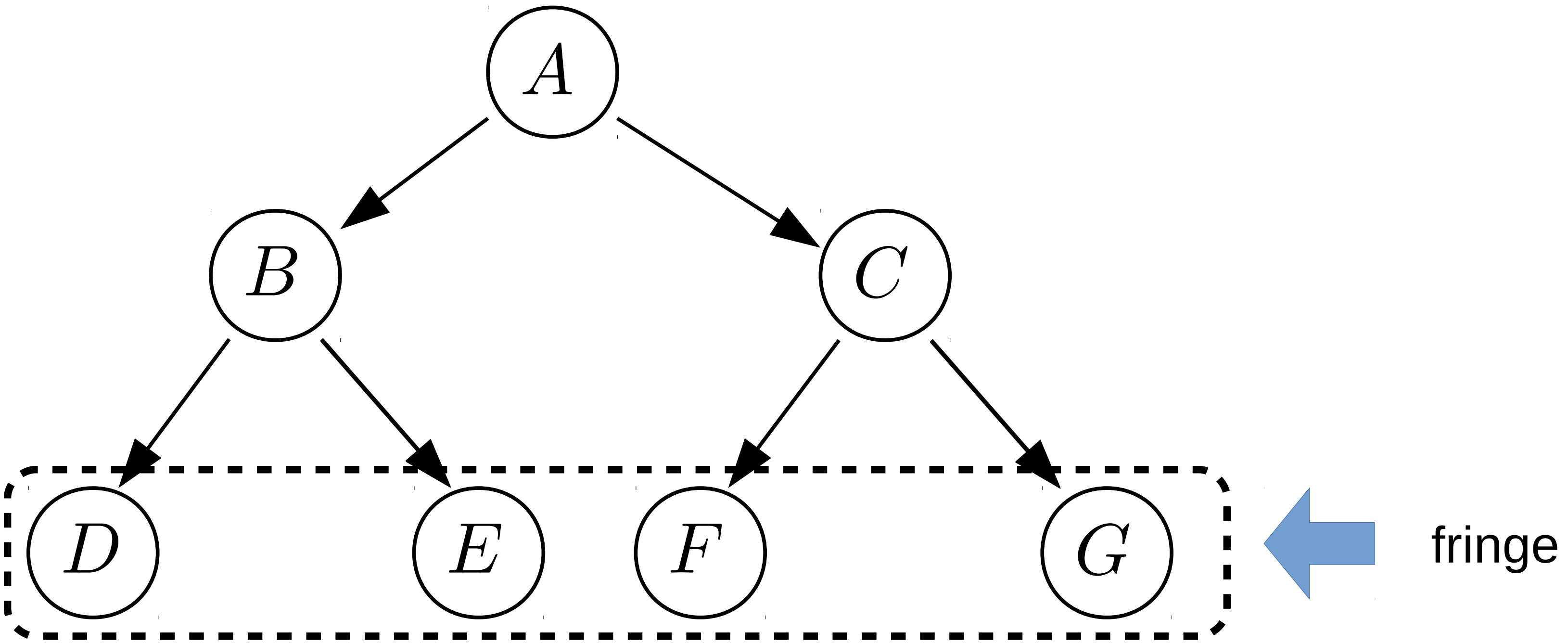


Which state gets removed next from the fringe?

What kind of a queue is this?

Breadth first search (BFS)

<u>Fringe</u>
D
E
F
G



Which state gets removed next from the fringe?

What kind of a queue is this?

FIFO Queue!
(first in first out)

Breadth first search (BFS)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)   /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
          child ← CHILD-NODE(problem, node, action)
          if child.STATE is not in explored or frontier then
              if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
              frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Breadth first search (BFS)

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
   node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
   if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
   frontier ← a FIFO queue with node as the only element
   explored ← an empty set
   loop do
      if EMPTY?(frontier) then return failure
      node ← POP(frontier)  /* chooses the shallowest node in frontier */
      add node.STATE to explored
      for each action in problem.ACTIONS(node.STATE) do
         child ← CHILD-NODE(problem, node, action)
         if child.STATE is not in explored or frontier then
            if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
            frontier ← INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

What is the purpose of the *explored* set?

BFS Properties

Is BFS <u>complete</u>?
– is it guaranteed to find a solution if one exists?

BFS Properties

Is BFS complete?
– is it guaranteed to find a solution if one exists?

What is the time complexity of BFS?
– how many states are expanded before finding a sol'n?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = ???

BFS Properties

Is BFS <u>complete</u>?
– is it guaranteed to find a solution if one exists?

What is the <u>time complexity</u> of BFS?
– how many states are expanded before finding a solution?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = $O(b^d)$

BFS Properties

Is BFS <u>complete</u>?
– is it guaranteed to find a solution if one exists?

What is the <u>time complexity</u> of BFS?
– how many states are expanded before finding a solution?
    – b: branching factor
    – d: depth of shallowest solution
    – complexity = $O(b^d)$

What is the <u>space complexity</u> of BFS?
– how much memory is required?
    – complexity = ???

BFS Properties

Is BFS <u>complete</u>?
– is it guaranteed to find a solution if one exists?

What is the <u>time complexity</u> of BFS?
– how many states are expanded before finding a solution?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = $O(b^d)$

What is the <u>space complexity</u> of BFS?
– how much memory is required?
  – complexity = $O(b^d)$

BFS Properties

Is BFS <u>complete</u>?
– is it guaranteed to find a solution if one exists?

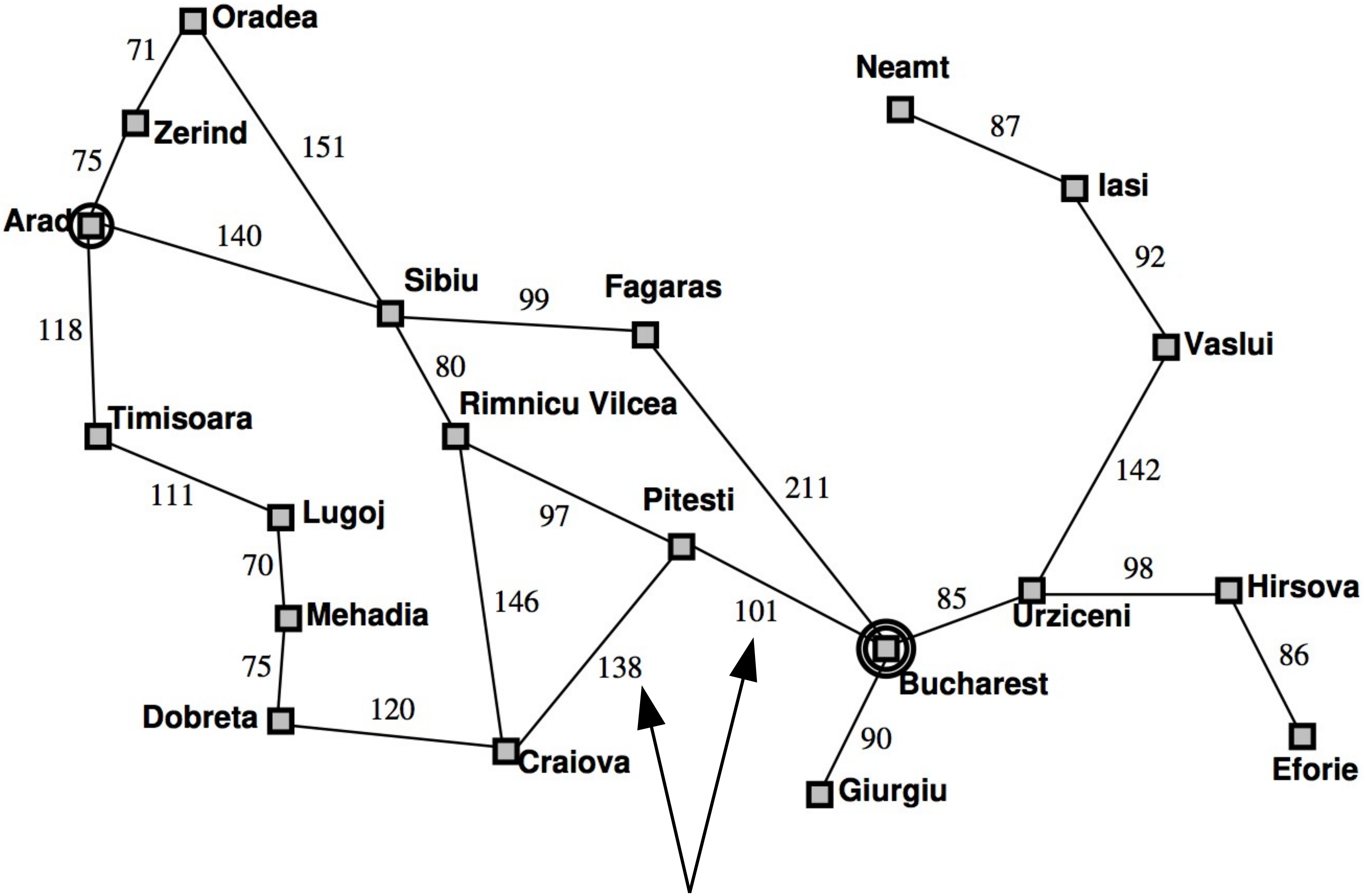
What is the <u>time complexity</u> of BFS?
– how many states are expanded before finding a solution?
  – b: branching factor
  – d: depth of shallowest solution
  – complexity = $O(b^d)$

What is the <u>space complexity</u> of BFS?
– how much memory is required?
  – complexity = $O(b^d)$

Is BFS optimal?
– is it guaranteed to find the best solution (shortest path)?

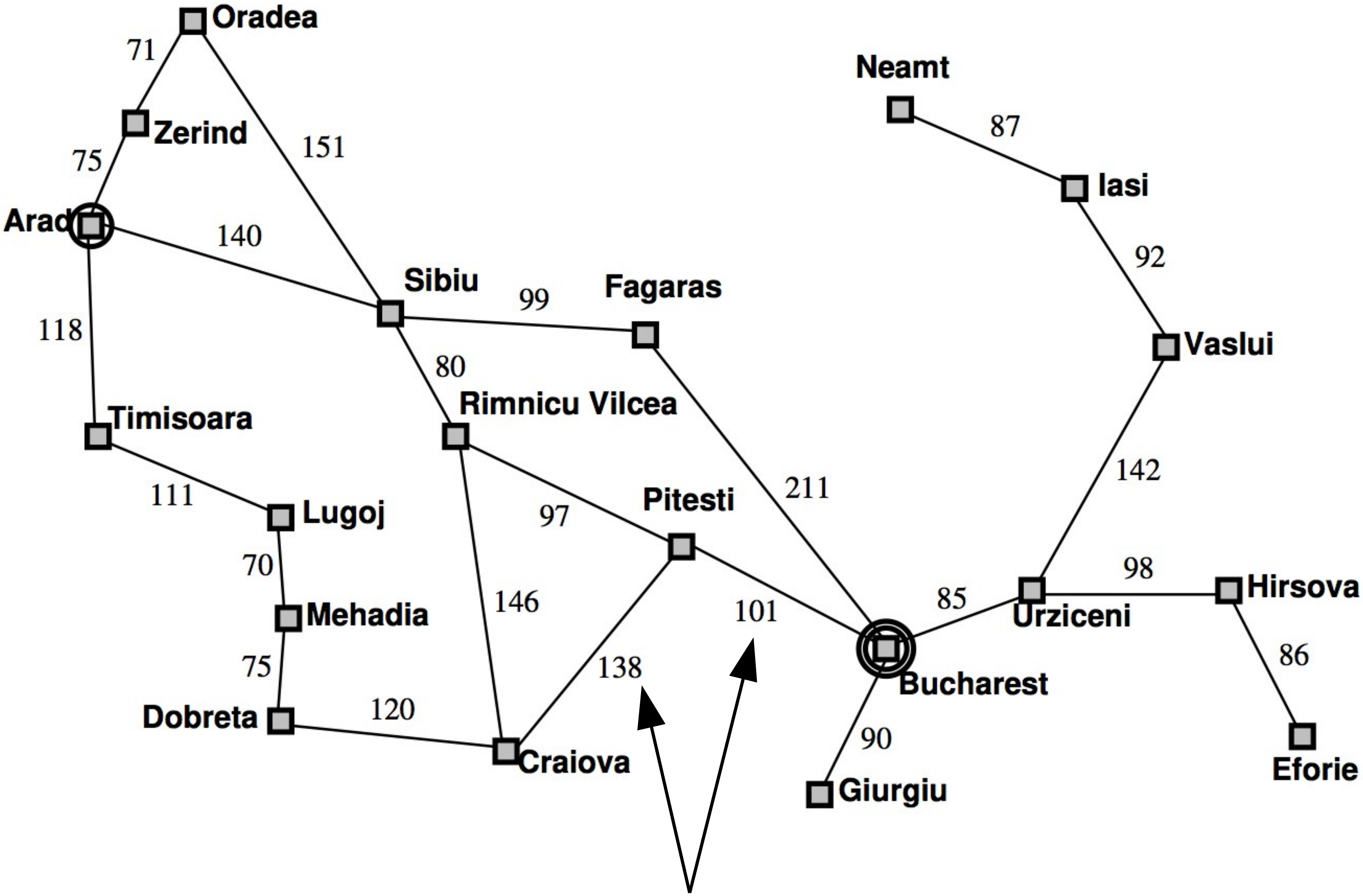
Uniform Cost Search (UCS)

Uniform Cost Search (UCS)



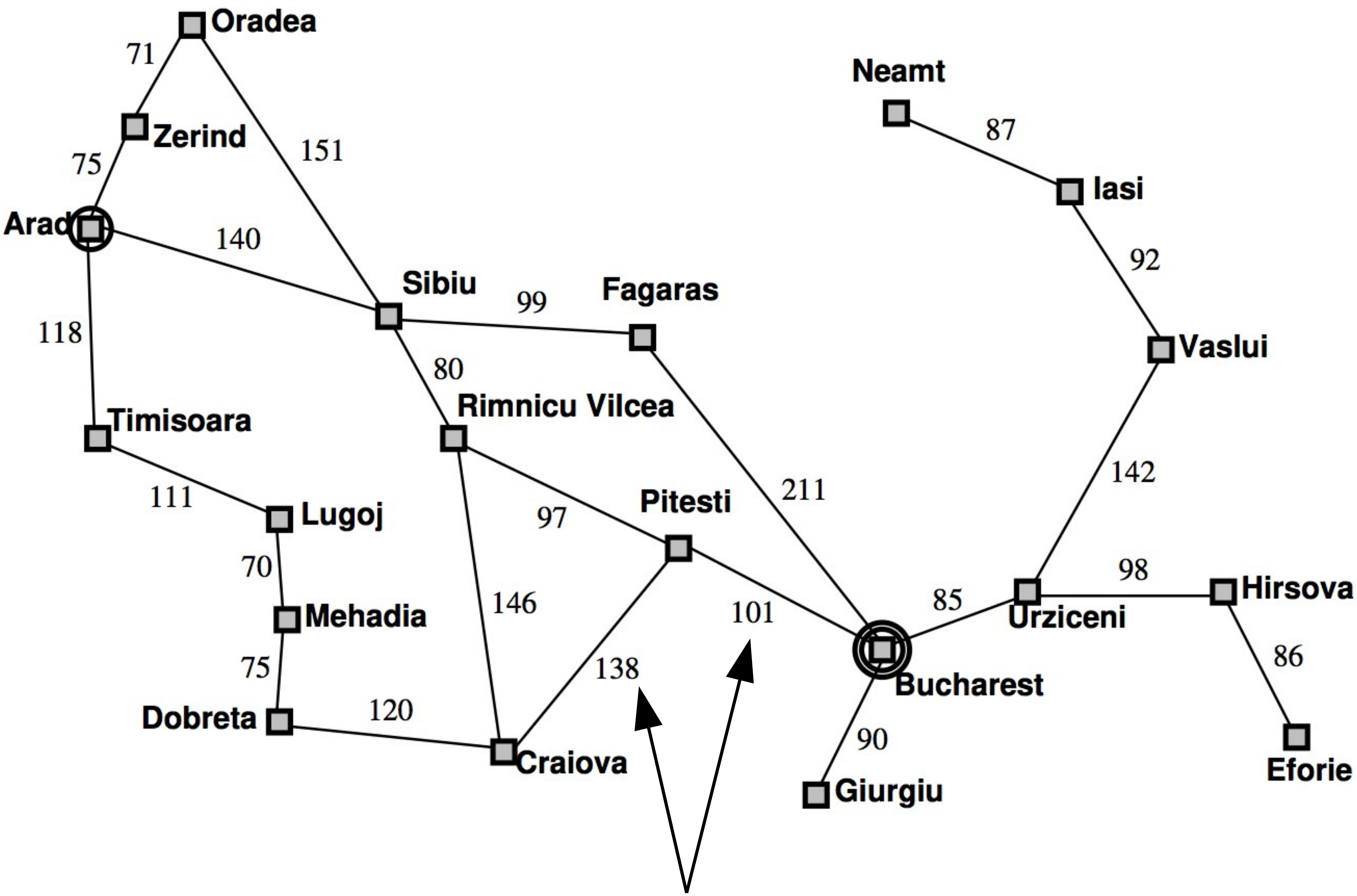
Notice the distances between cities

Uniform Cost Search (UCS)



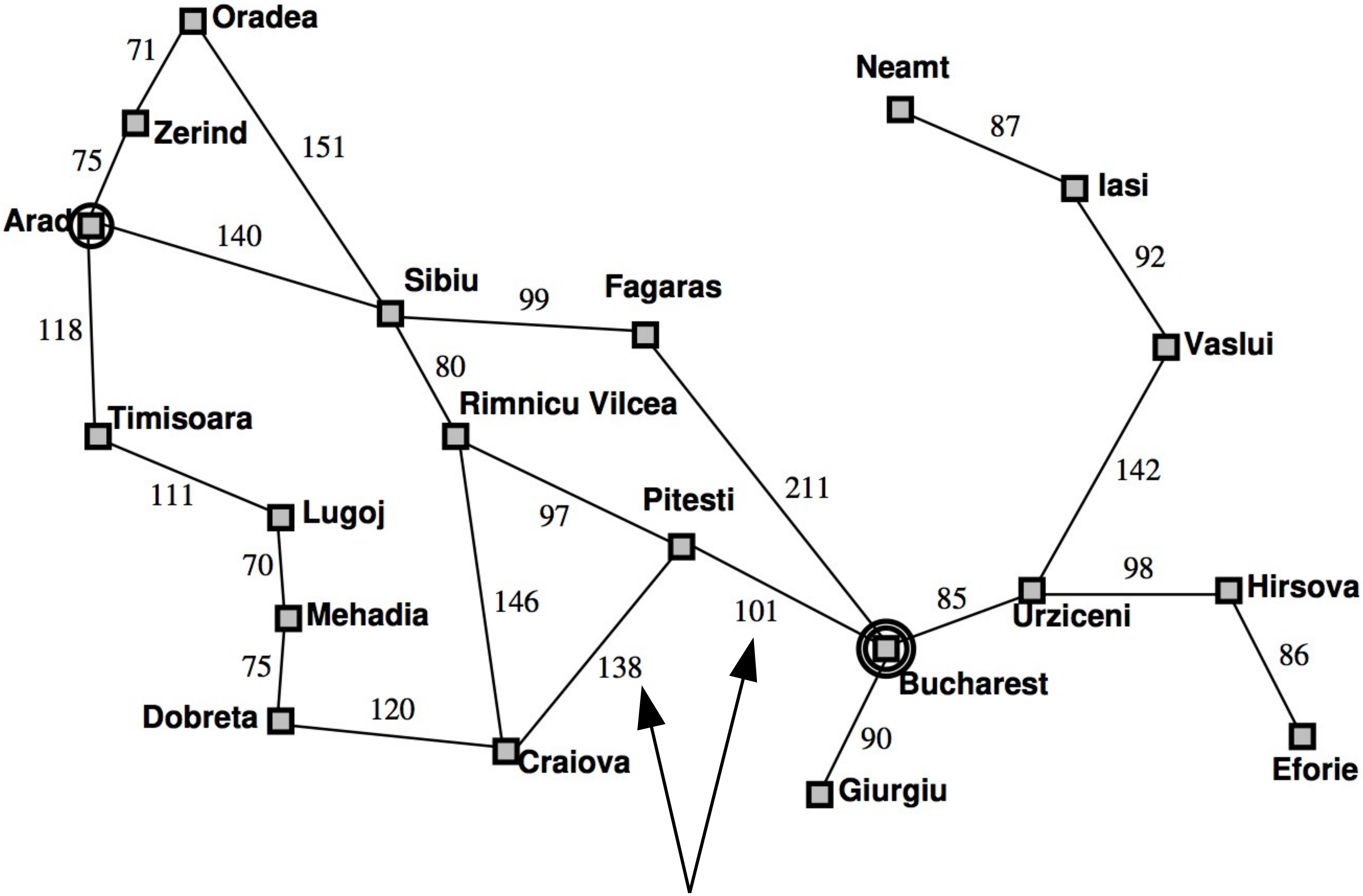
Notice the distances between cities
– does BFS take these distances into account?

Uniform Cost Search (UCS)



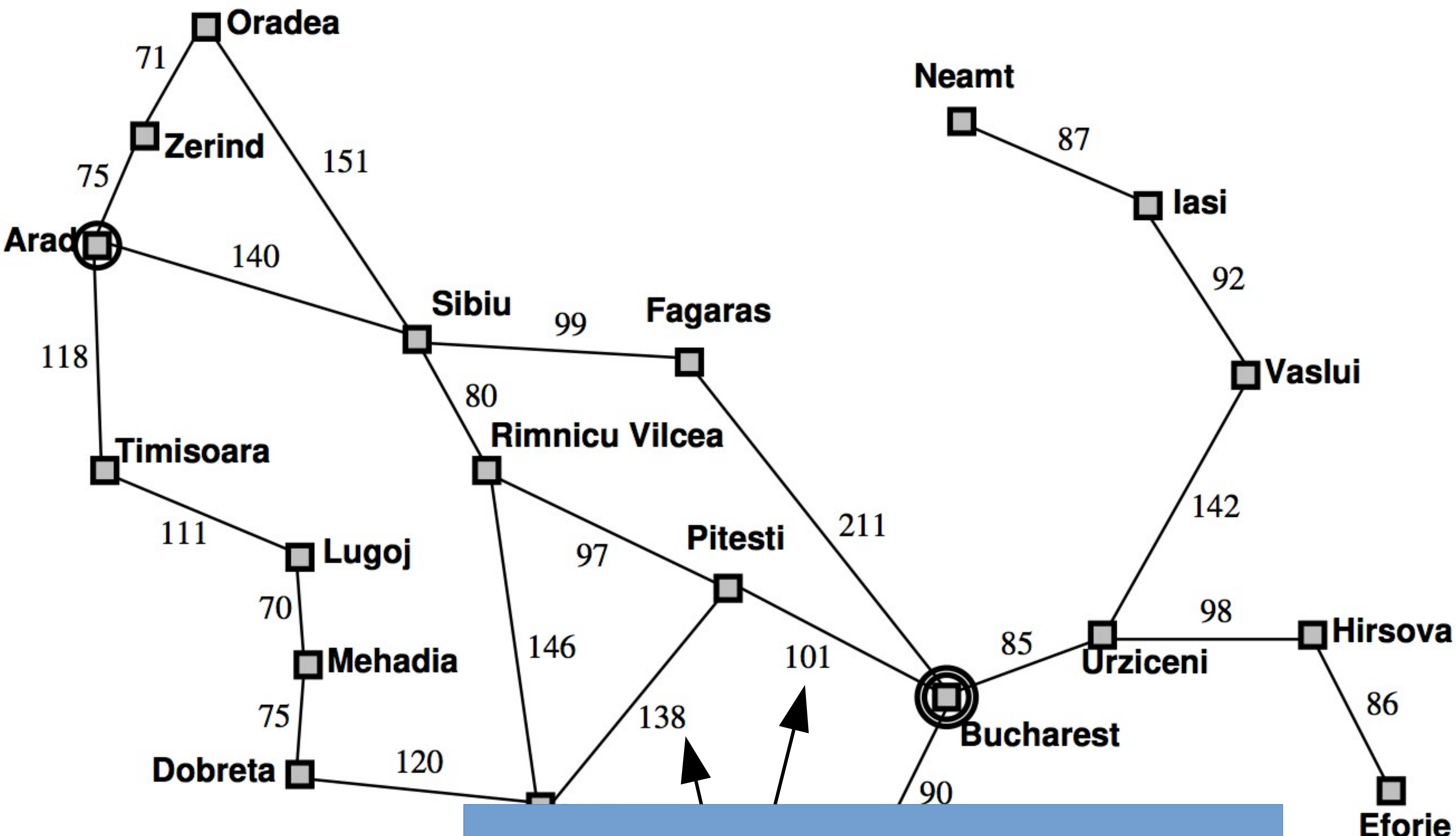
Notice the distances between cities
– does BFS take these distances into account?
– does BFS find the path w/ shortest milage?

Uniform Cost Search (UCS)



Notice the distances between cities
– does BFS take these distances into account?
– does BFS find the path w/ shortest milage?
– compare S-F-B with S-R-P-B. Which costs less?

Uniform Cost Search (UCS)



How do we fix this?

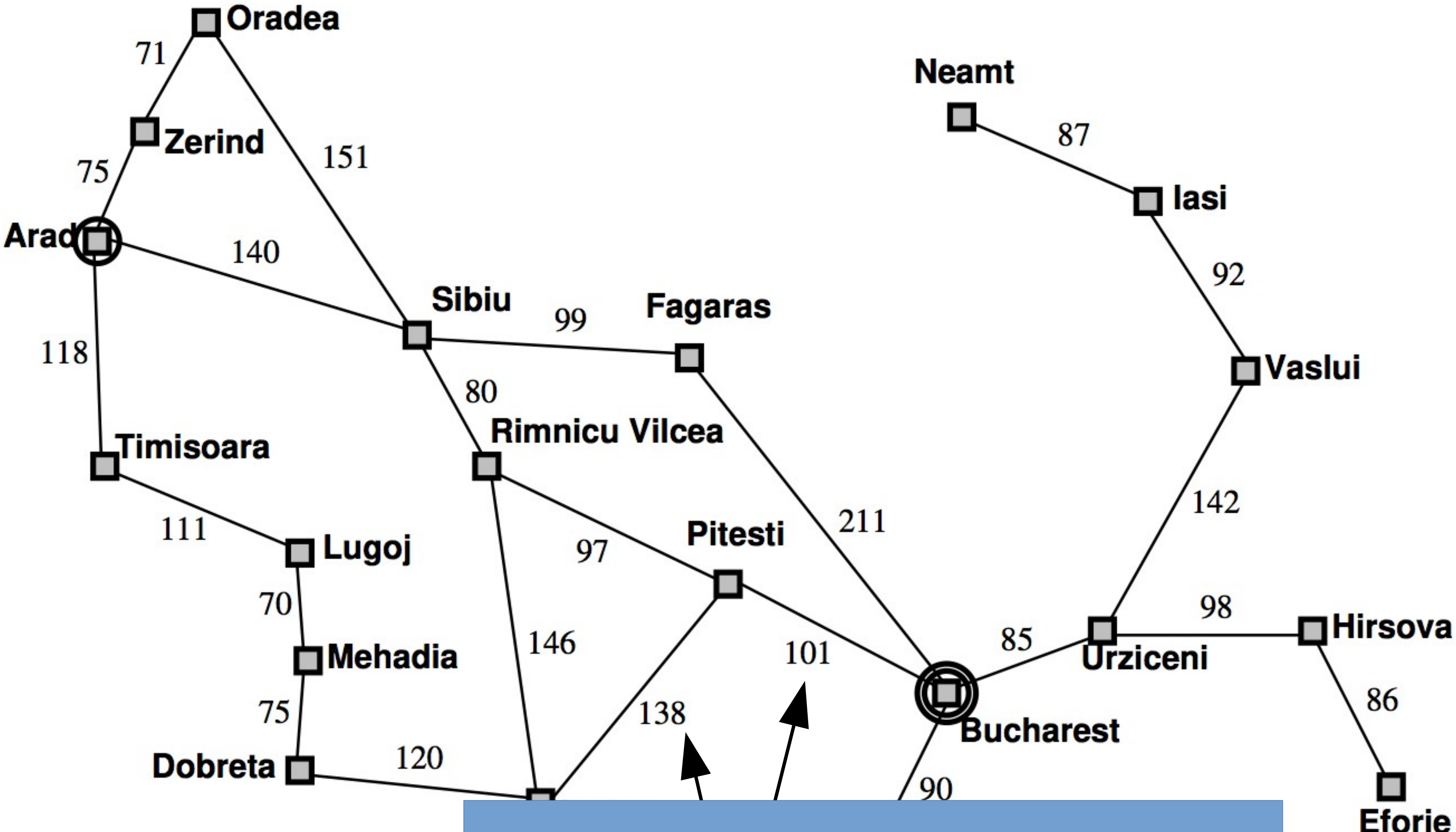
Notic

– do unt?

– do ?

– co s less?

Uniform Cost Search (UCS)



How do we fix this?
<u>UCS!</u>

Notic

– do … unt?

– do … ?

– co … s less?

Uniform Cost Search (UCS)

Same as BFS except: expand node w/ smallest <u>path cost</u>

Length of path

Uniform Cost Search (UCS)

Same as BFS except: expand node w/ smallest <u>path cost</u>

Length of path

Cost of going from state *A* to *B*: $c(A, B)$

Minimum cost of path going from start state to *B*: $g(B)$

Uniform Cost Search (UCS)

Same as BFS except: expand node w/ smallest path cost

Length of path

Cost of going from state *A* to *B:* $c(A, B)$

Minimum cost of path going from start state to *B:* $g(B)$

BFS: expands states in order of hops from start

UCS: expands states in order of $g(s)$

Uniform Cost Search (UCS)

Same as BFS except: expand node w/ smallest <u>path cost</u>

Length of path

Cost of going from state *A* to *B:* $c(A, B)$

Minimum cost of path going from start state to *B:* $g(B)$

BFS: exp

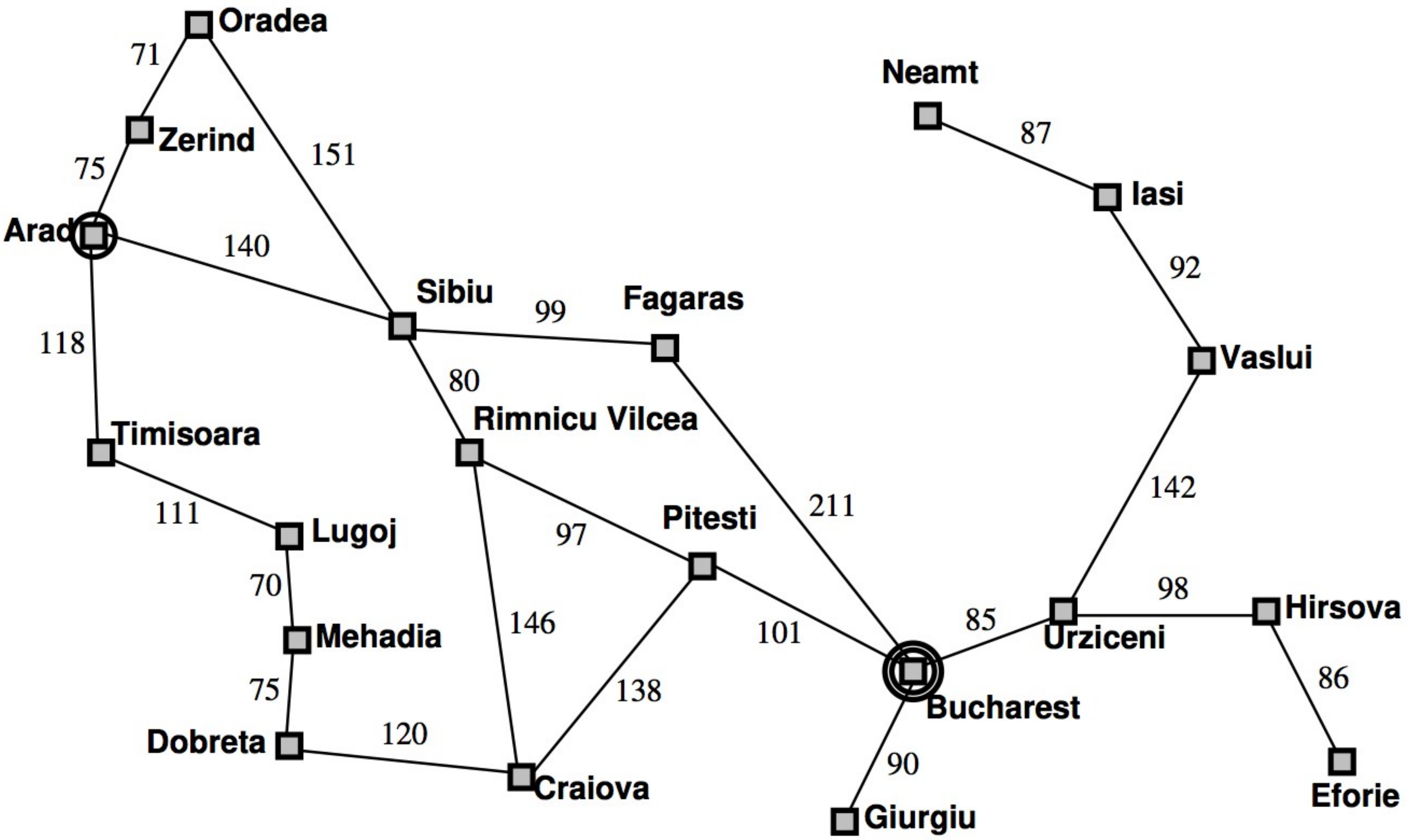
UCS: ex

How?

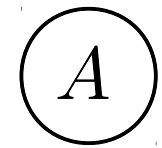
Uniform Cost Search (UCS)

Simple answer: change the FIFO to a priority queue
– the priority of each element in the queue is its path cost.

Uniform Cost Search (UCS)



UCS

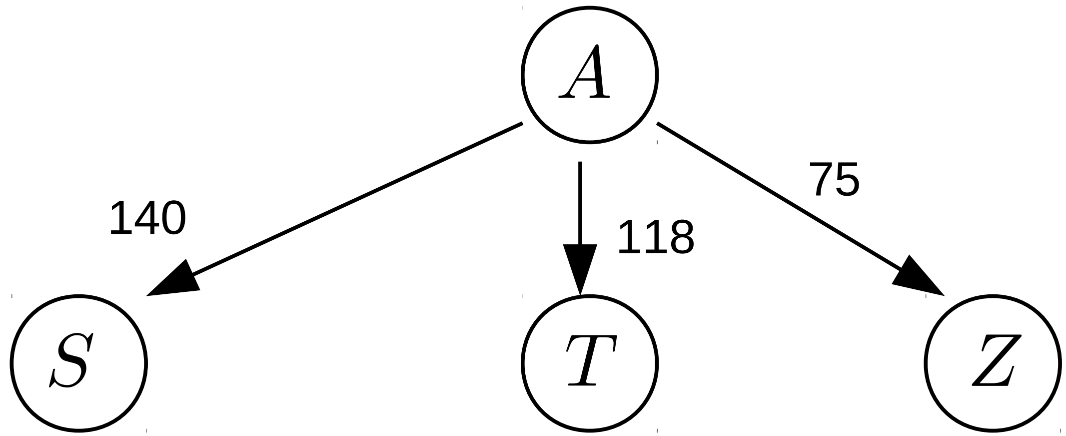


Fringe	Path Cost
A	0

Explored set:

UCS

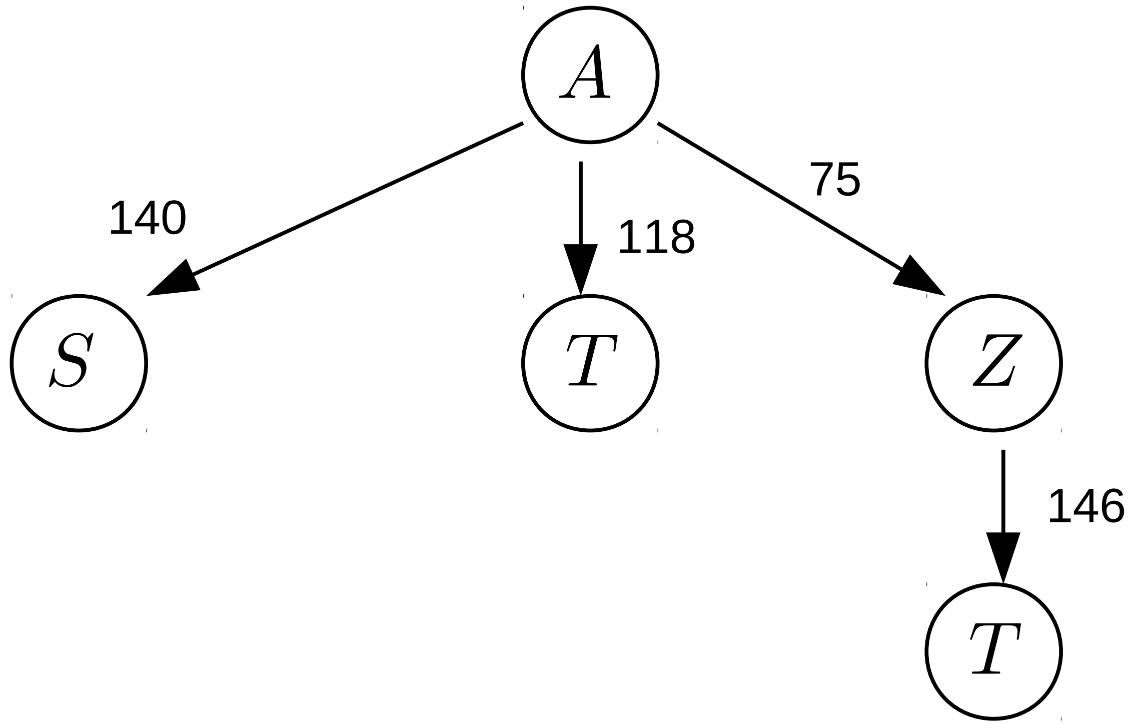
Fringe	Path Cost
~~A~~	~~0~~
S	140
T	118
Z	75



Explored set: A

UCS

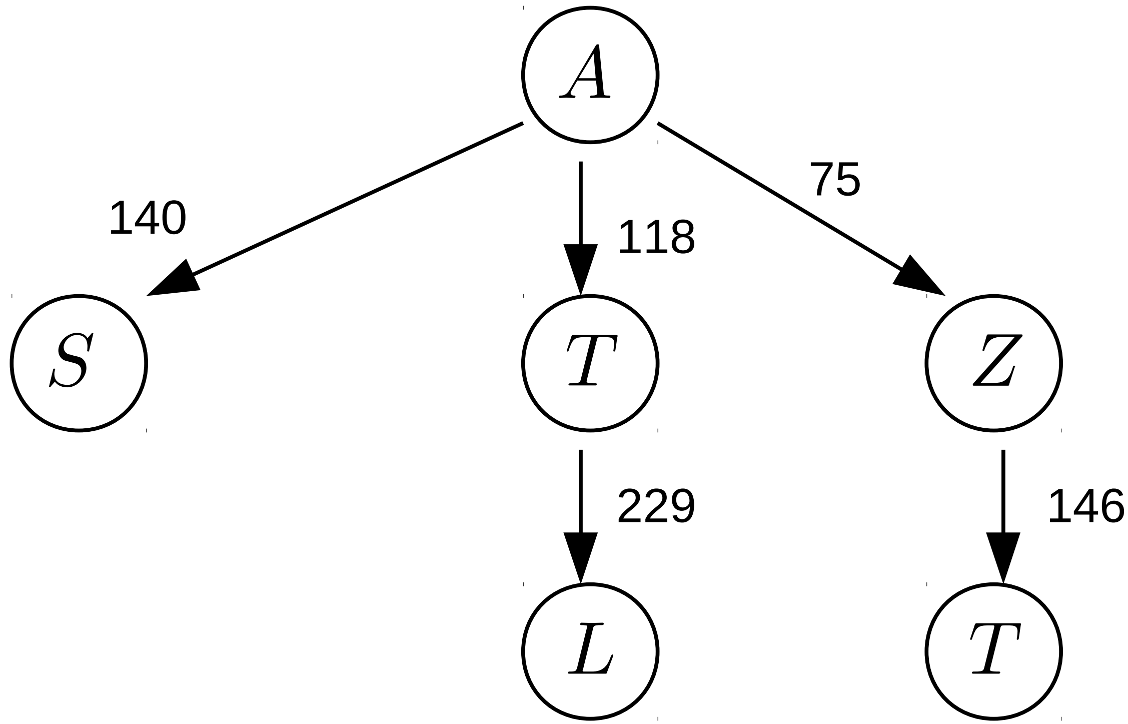
Fringe	Path Cost
~~A~~	~~0~~
S	140
T	118
~~Z~~	~~75~~
T	146



Explored set: A, Z

UCS

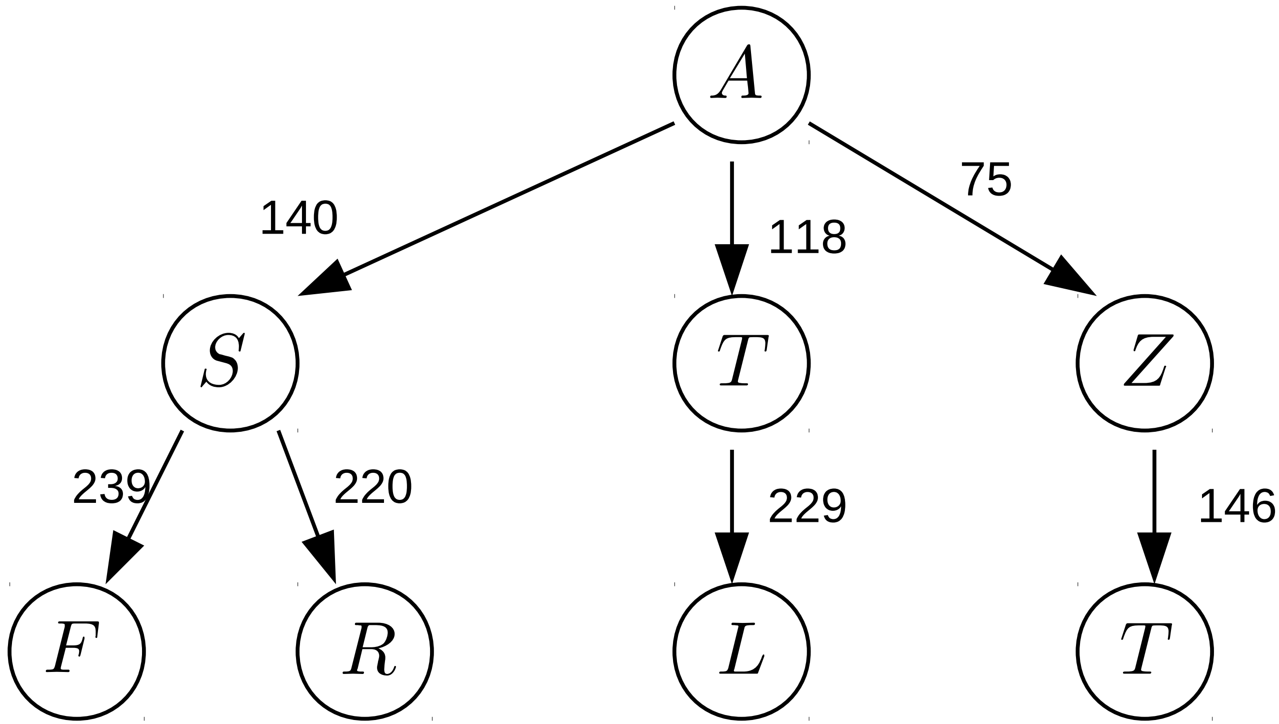
Fringe	Path Cost
~~A~~	~~0~~
S	140
~~T~~	~~118~~
~~Z~~	~~75~~
T	146
L	229



Explored set: A, Z, T

UCS

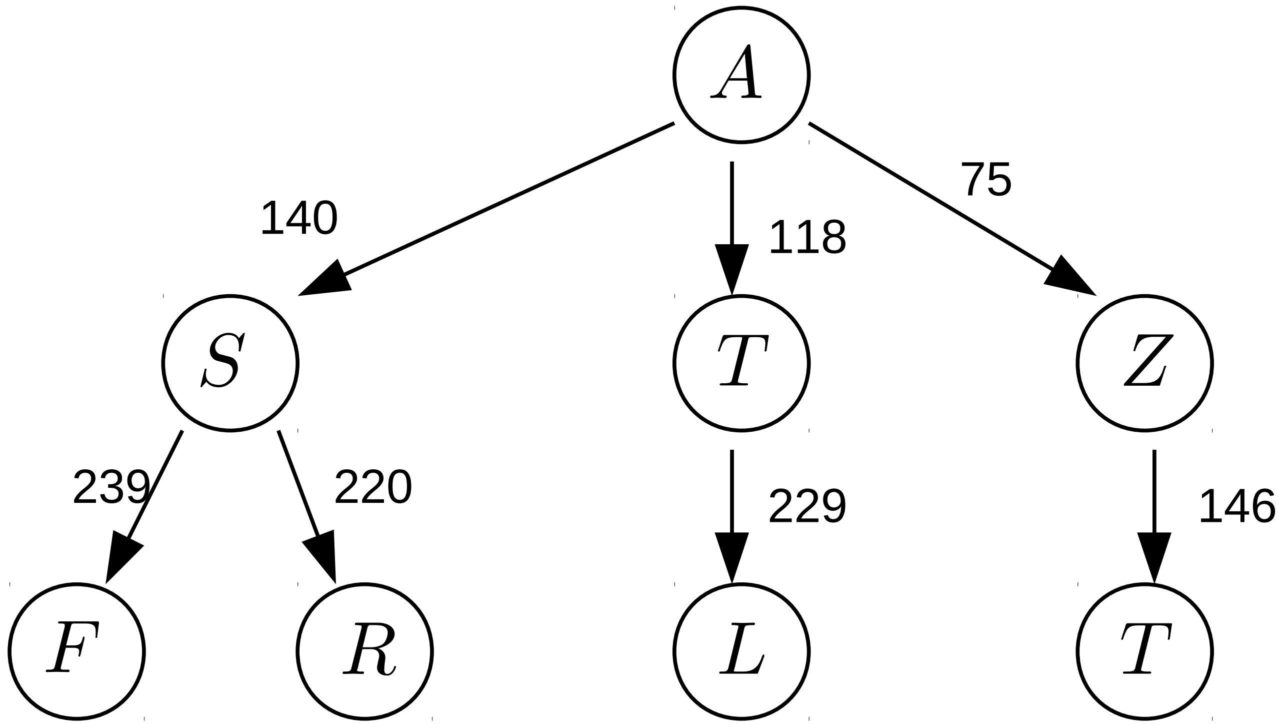
Fringe	Path Cost
~~A~~	~~0~~
~~S~~	~~140~~
~~T~~	~~118~~
~~Z~~	~~75~~
T	146
L	229
F	239
R	220



Explored set: A, Z, T, S

UCS

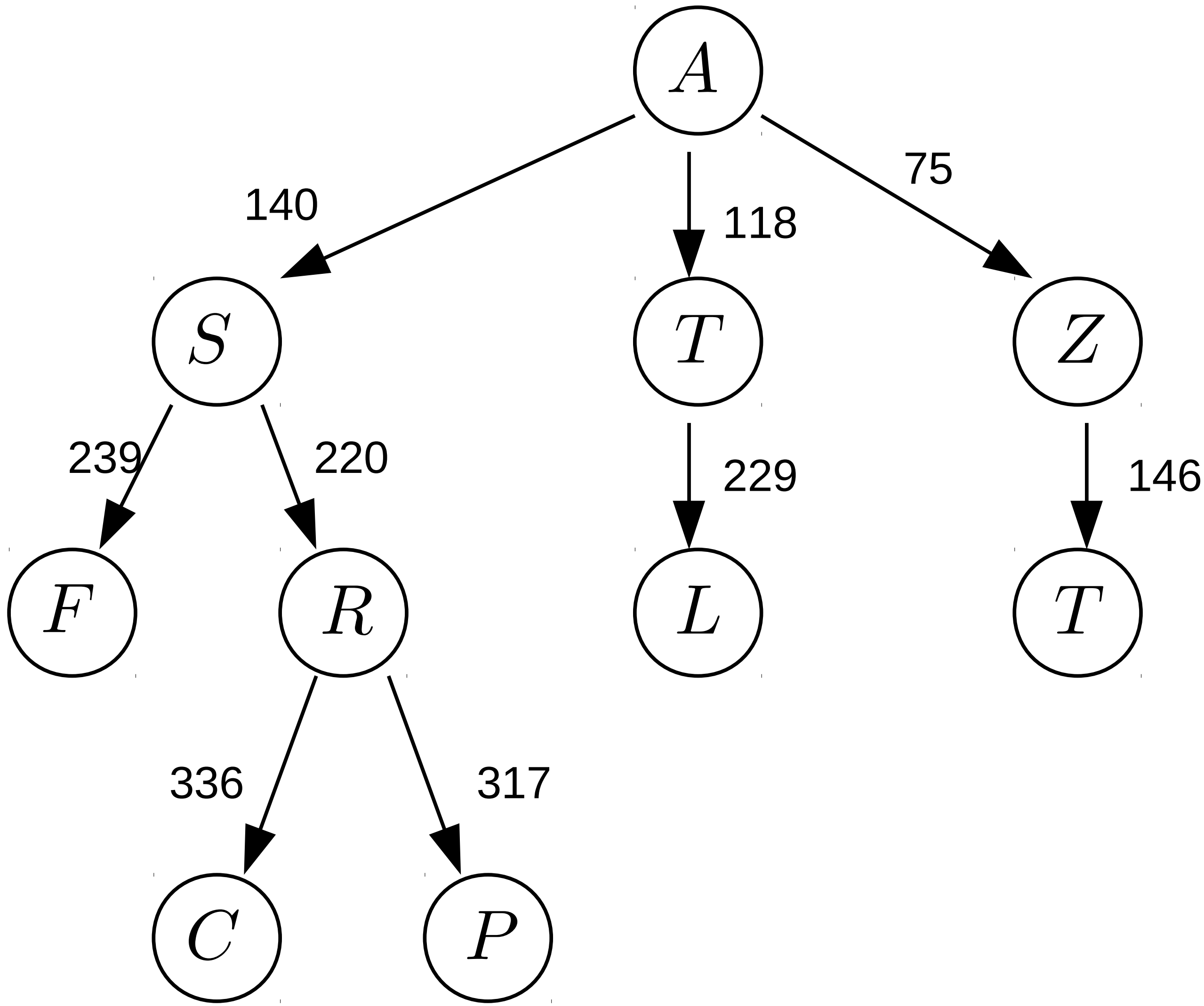
Fringe	Path Cost
~~A~~	~~0~~
~~S~~	~~140~~
~~T~~	~~118~~
~~Z~~	~~75~~
~~T~~	~~146~~
L	229
F	239
R	220



Explored set: A, Z, T, S

UCS

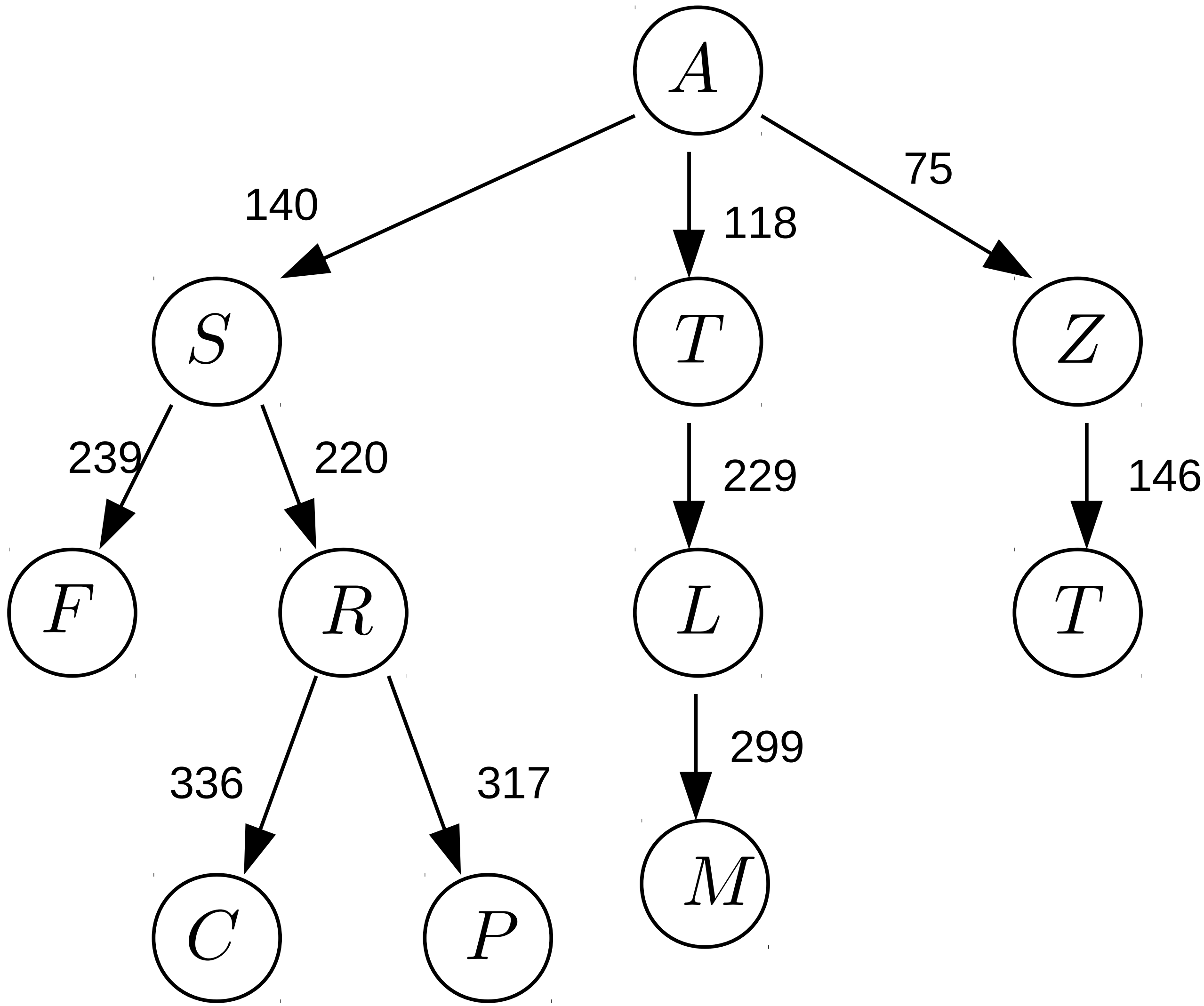
Fringe	Path Cost
~~A~~	~~0~~
~~S~~	~~140~~
~~T~~	~~118~~
~~Z~~	~~75~~
~~T~~	~~146~~
L	229
F	239
~~R~~	~~220~~
C	336
P	317



Explored set: A, Z, T, S, R

UCS

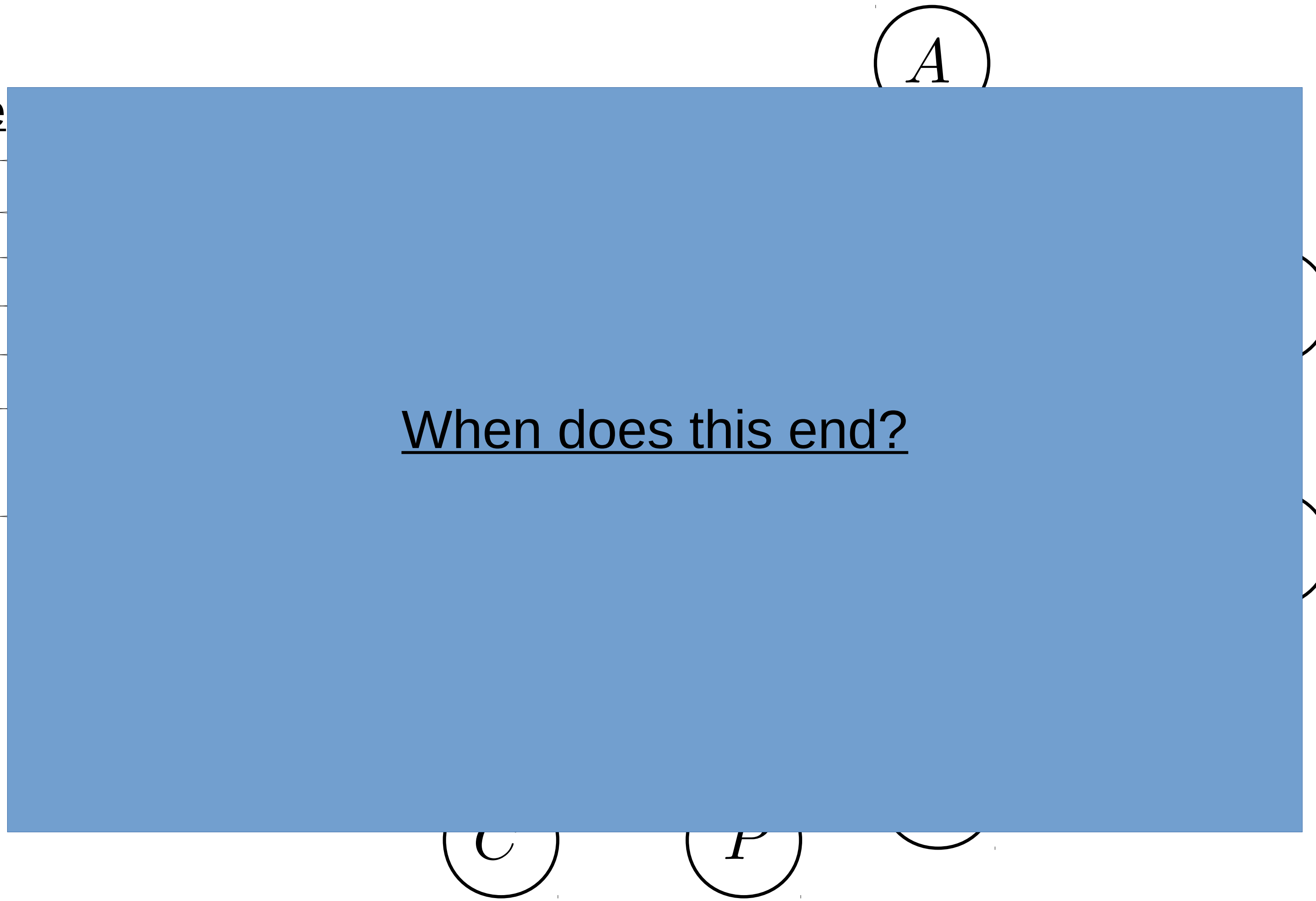
Fringe	Path Cost
~~A~~	~~0~~
~~S~~	~~140~~
~~T~~	~~118~~
~~Z~~	~~75~~
~~T~~	~~146~~
~~L~~	~~229~~
F	239
~~R~~	~~220~~
C	336
P	317
M	299



Explored set: A, Z, T, S, R, L

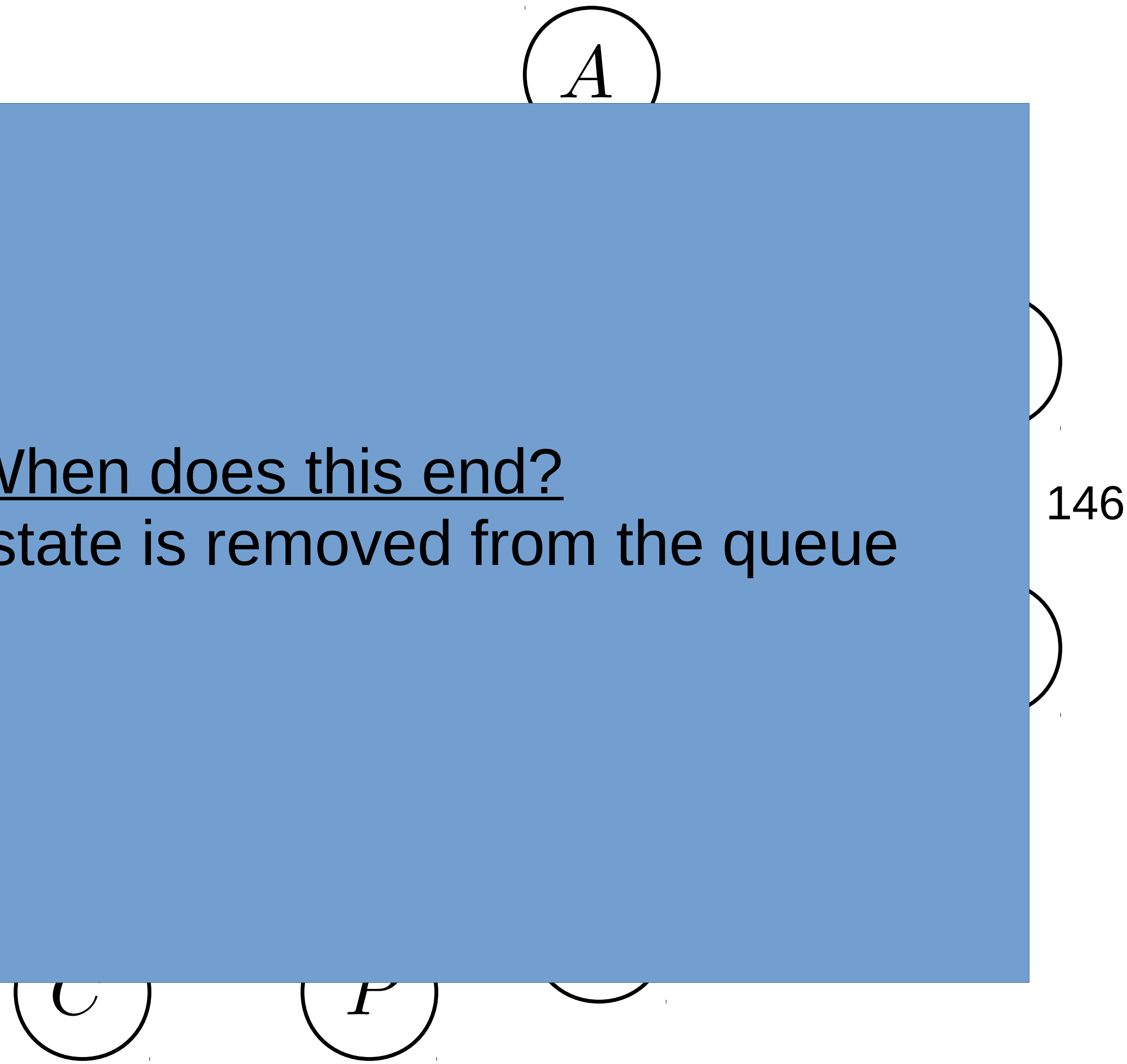
UCS

Fringe
~~A~~
~~S~~
~~T~~
~~Z~~
~~T~~
~~L~~
F
~~R~~
C
P
M



Explored set: A, Z, T, S, R, L

UCS



<u>Fringe</u>
~~A~~
~~S~~
~~T~~
~~Z~~
~~T~~
~~L~~
F
~~R~~
C
P
M

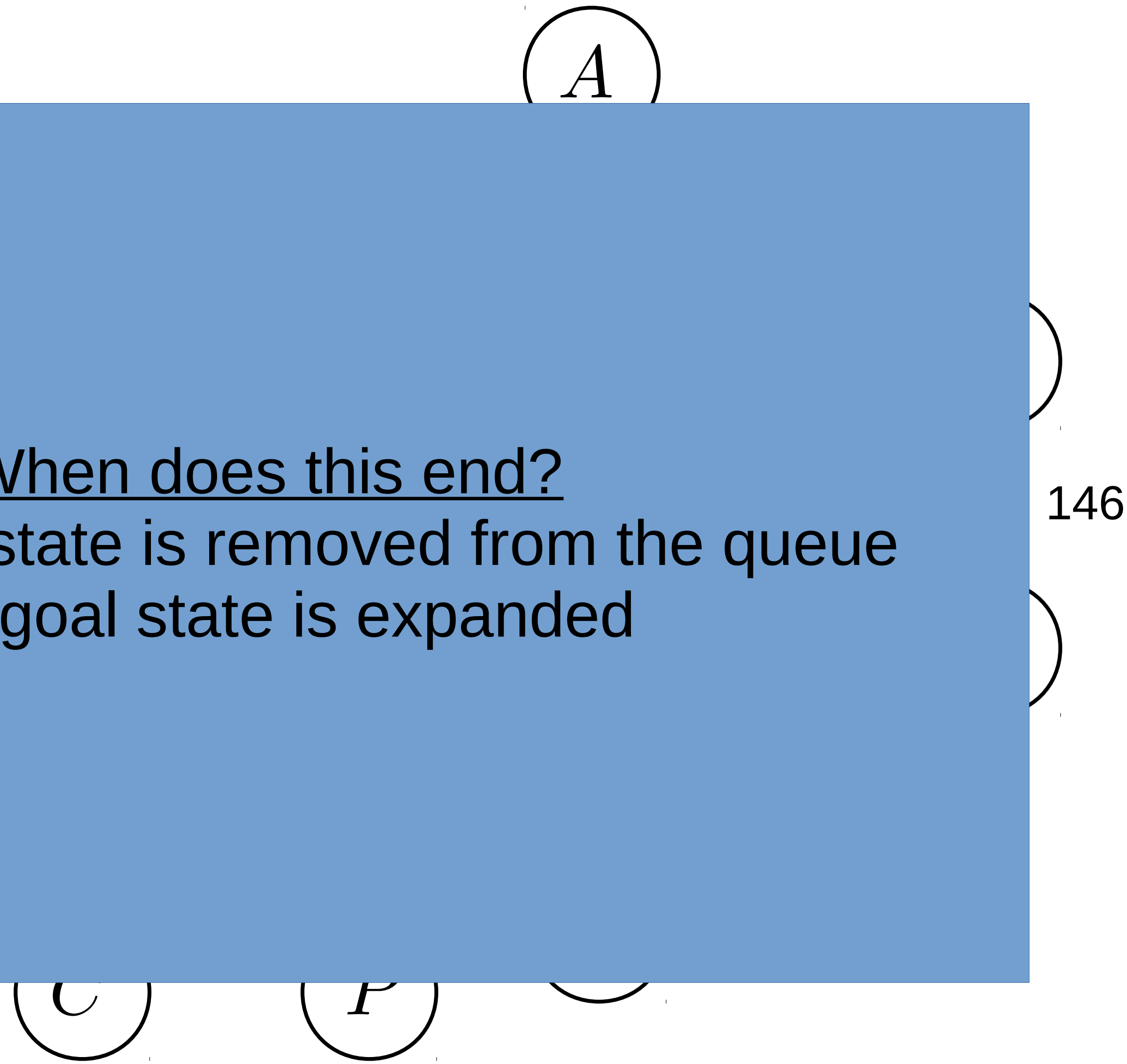
<u>When does this end?</u>
– when the goal state is removed from the queue

Explored set: A, Z, T, S, R, L

UCS

<u>Fringe</u>

~~A~~
~~S~~
~~T~~
~~Z~~
~~T~~
~~L~~
F
~~R~~
C
P
M



<u>When does this end?</u>

– when the goal state is removed from the queue
– NOT when the goal state is expanded

Explored set: A, Z, T, S, R, L

UCS

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier ← a priority queue ordered by PATH-COST, with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
     if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           frontier ← INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
           replace that frontier node with child
```

Figure 3.14 Uniform-cost search on a graph. The algorithm is identical to the general graph search algorithm in Figure 3.7, except for the use of a priority queue and the addition of an extra check in case a shorter path to a frontier state is discovered. The data structure for *frontier* needs to support efficient membership testing, so it should combine the capabilities of a priority queue and a hash table.

UCS Properties

Is UCS <u>complete</u>?

– is it guaranteed to find a solution if one exists?

What is the <u>time complexity</u> of UCS?

– how many states are expanded before finding a solution?

- – b: branching factor
- – C*: cost of optimal solution
- – e: min one-step cost
- – complexity = $O(b^{C^*/e})$

What is the <u>space complexity</u> of BFS?

– how much memory is required?

- – complexity = $O(b^{C^*/e})$

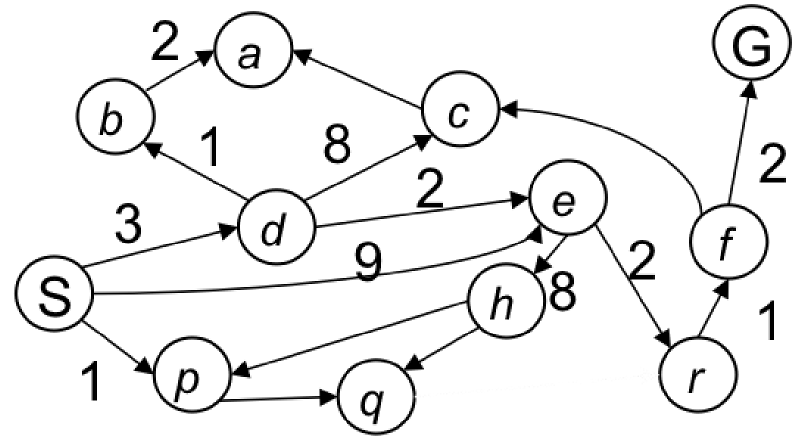
Is BFS optimal?

– is it guaranteed to find the best solution (shortest path)?

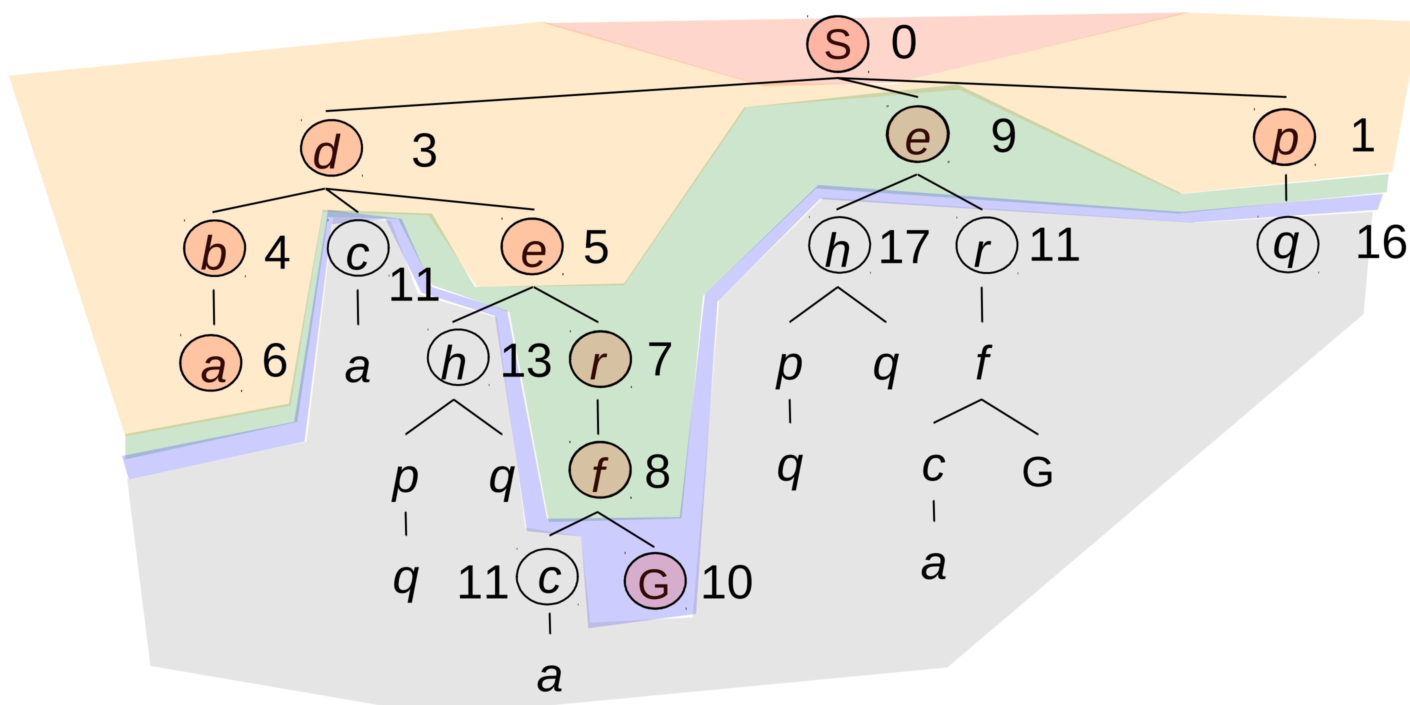
UCS vs BFS

Strategy: expand cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



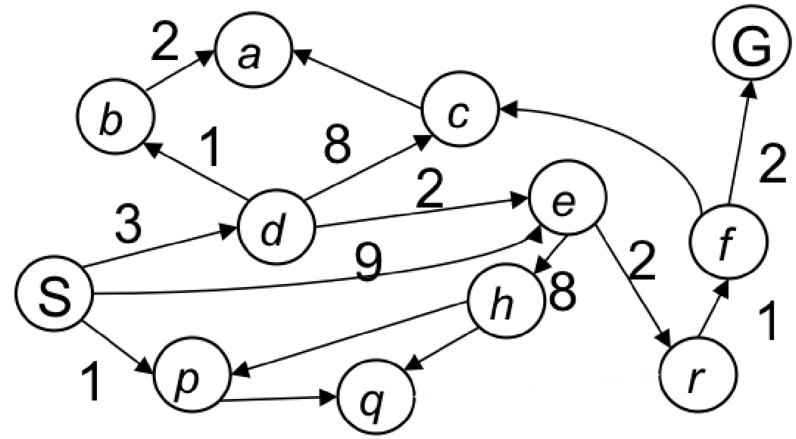
Cost contours

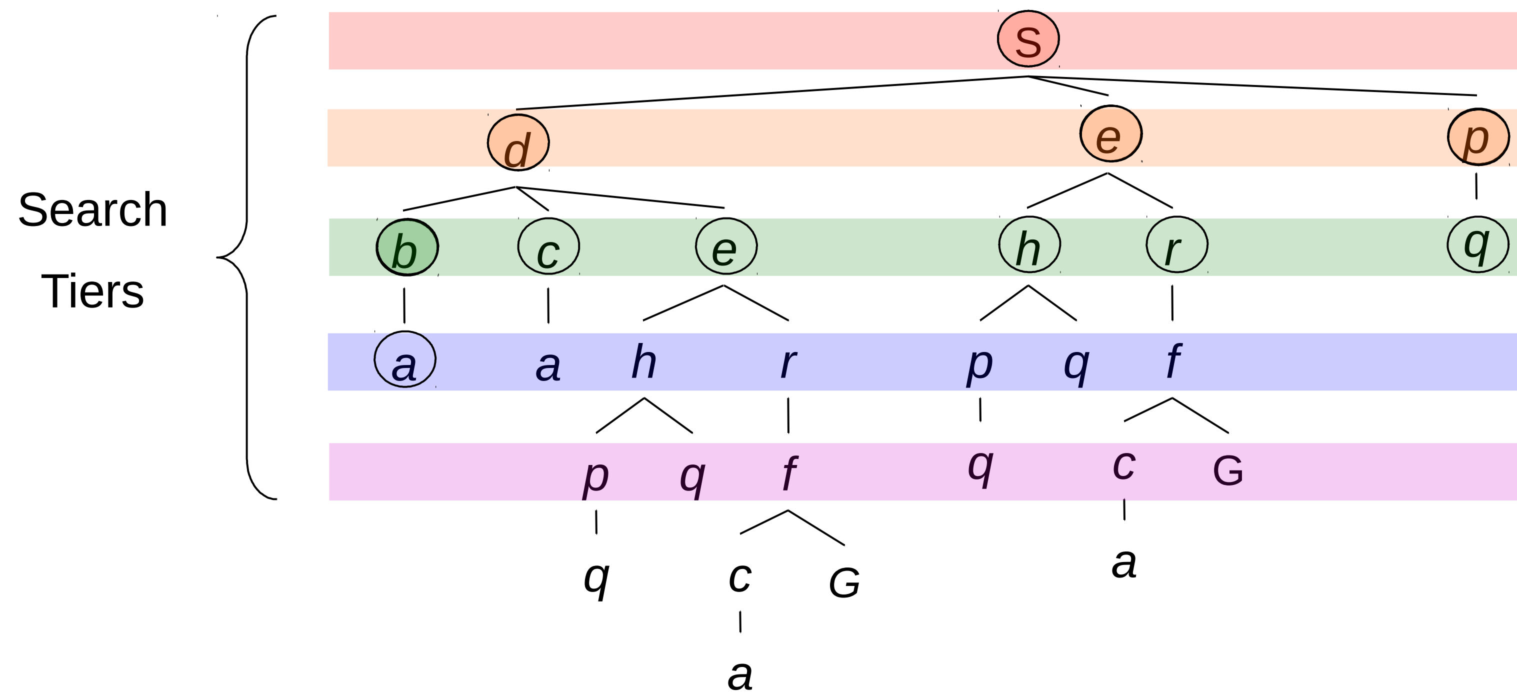


UCS vs BFS

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue





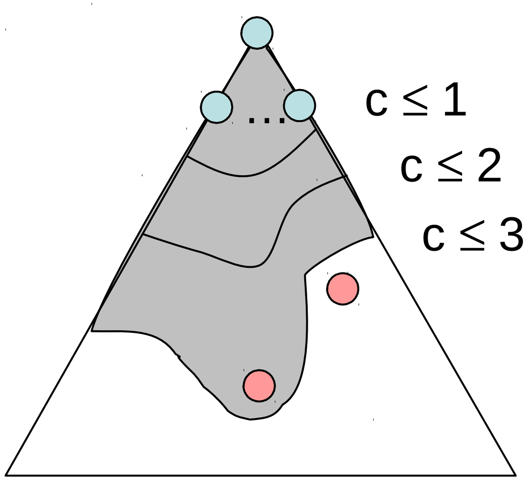
UCS vs BFS

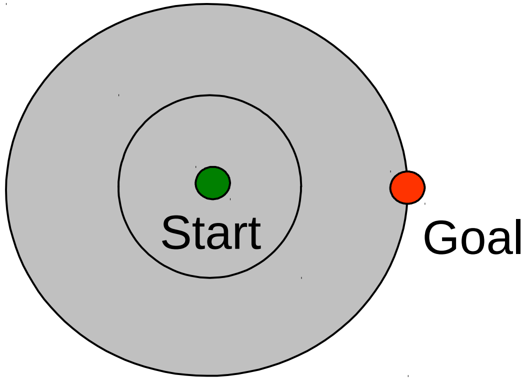
- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

- We’ll fix that soon!





Depth First Search (DFS)

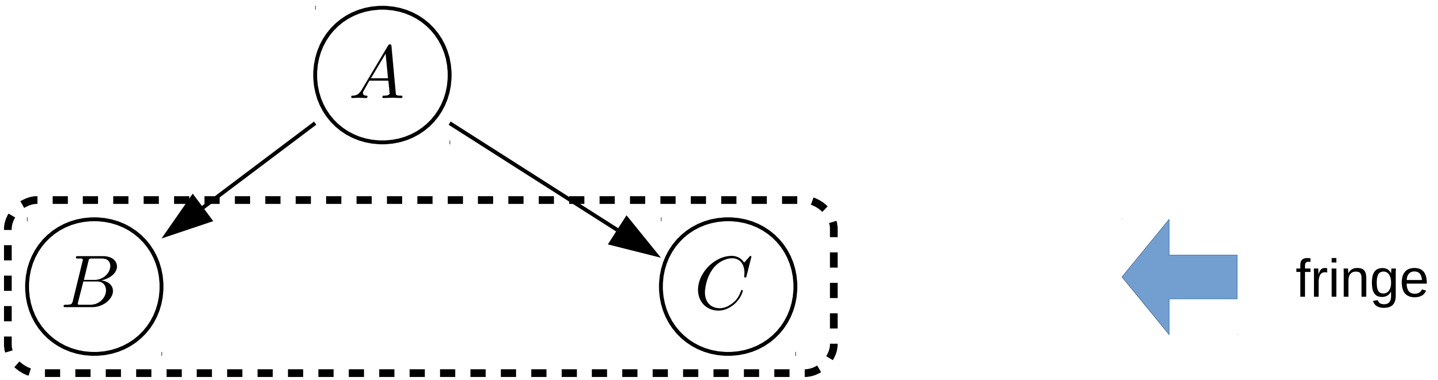
DFS

Fringe
A

DFS

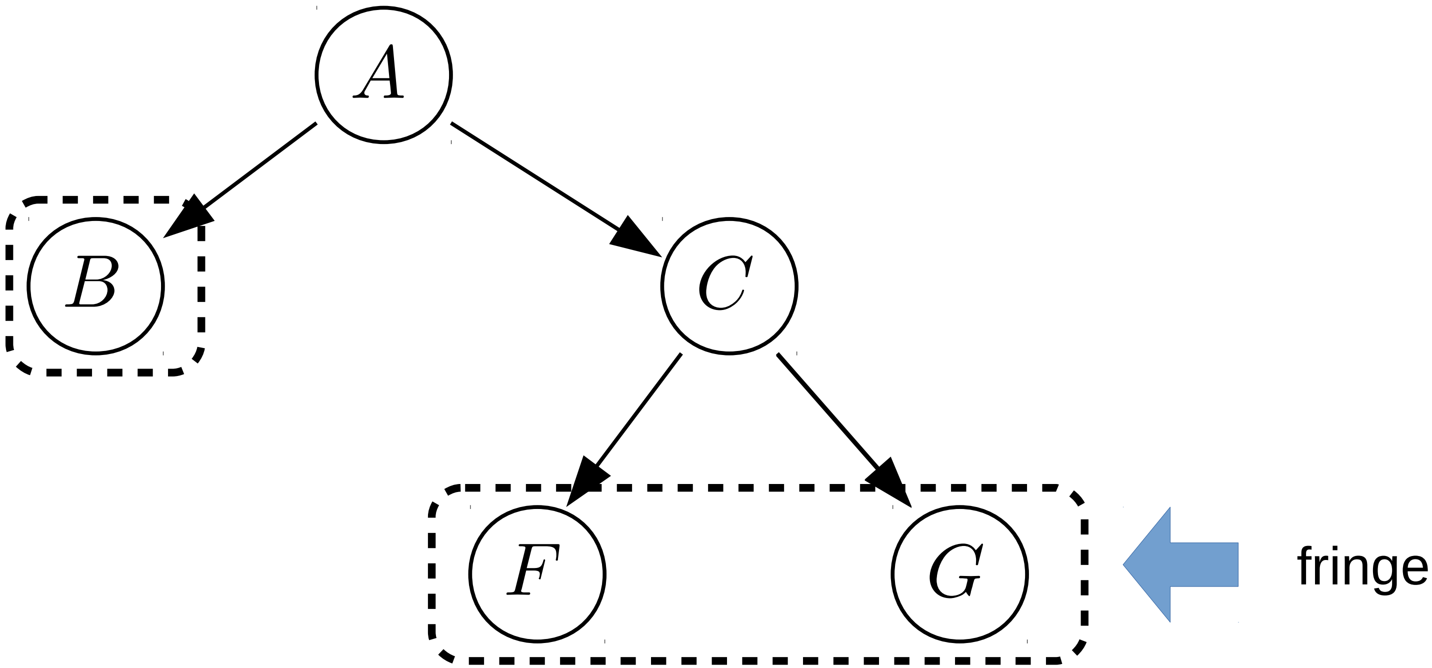
Fringe

~~A~~
B
C



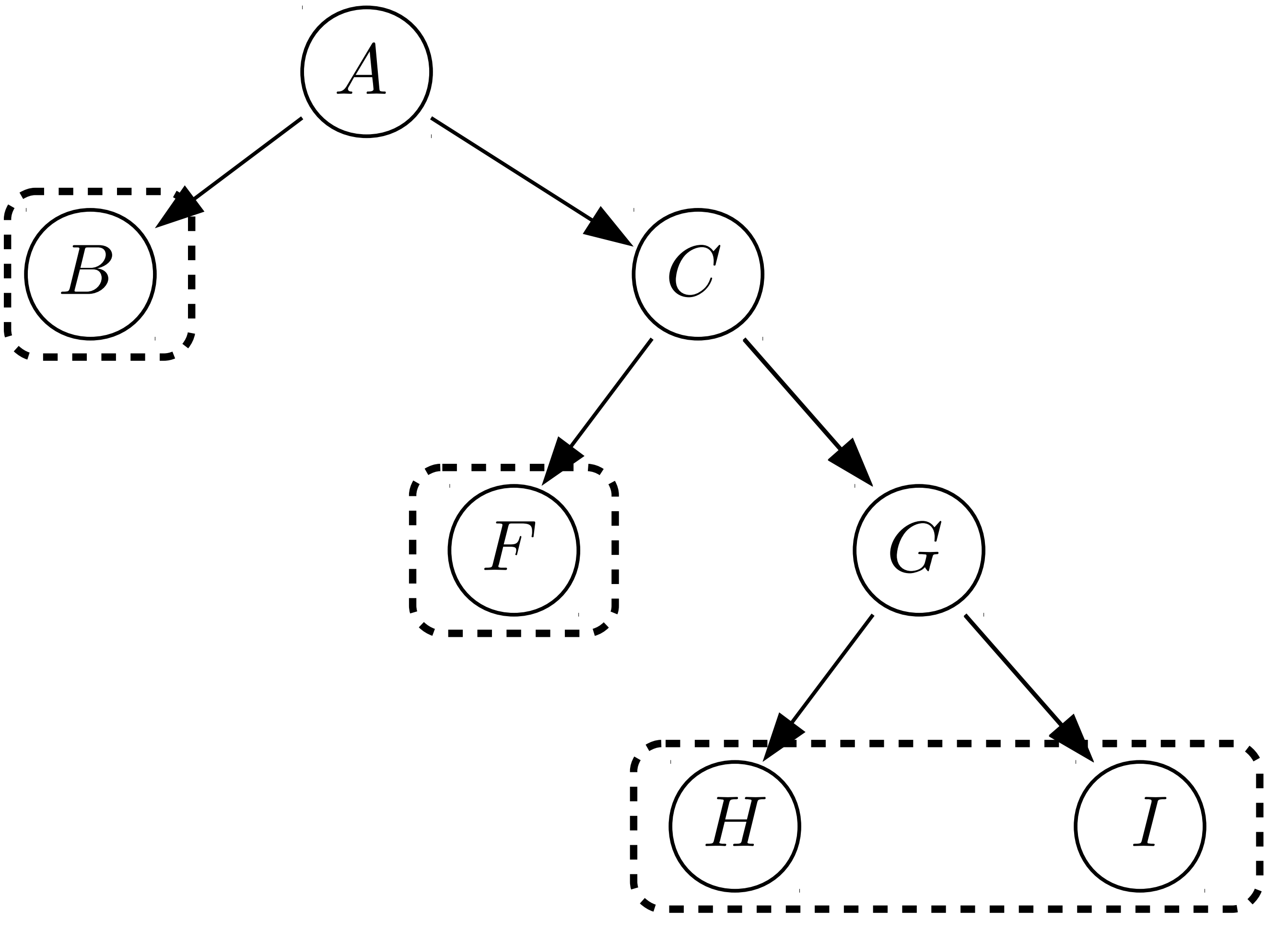
DFS

<u>Fringe</u>
~~A~~
B
~~C~~
F
G



DFS

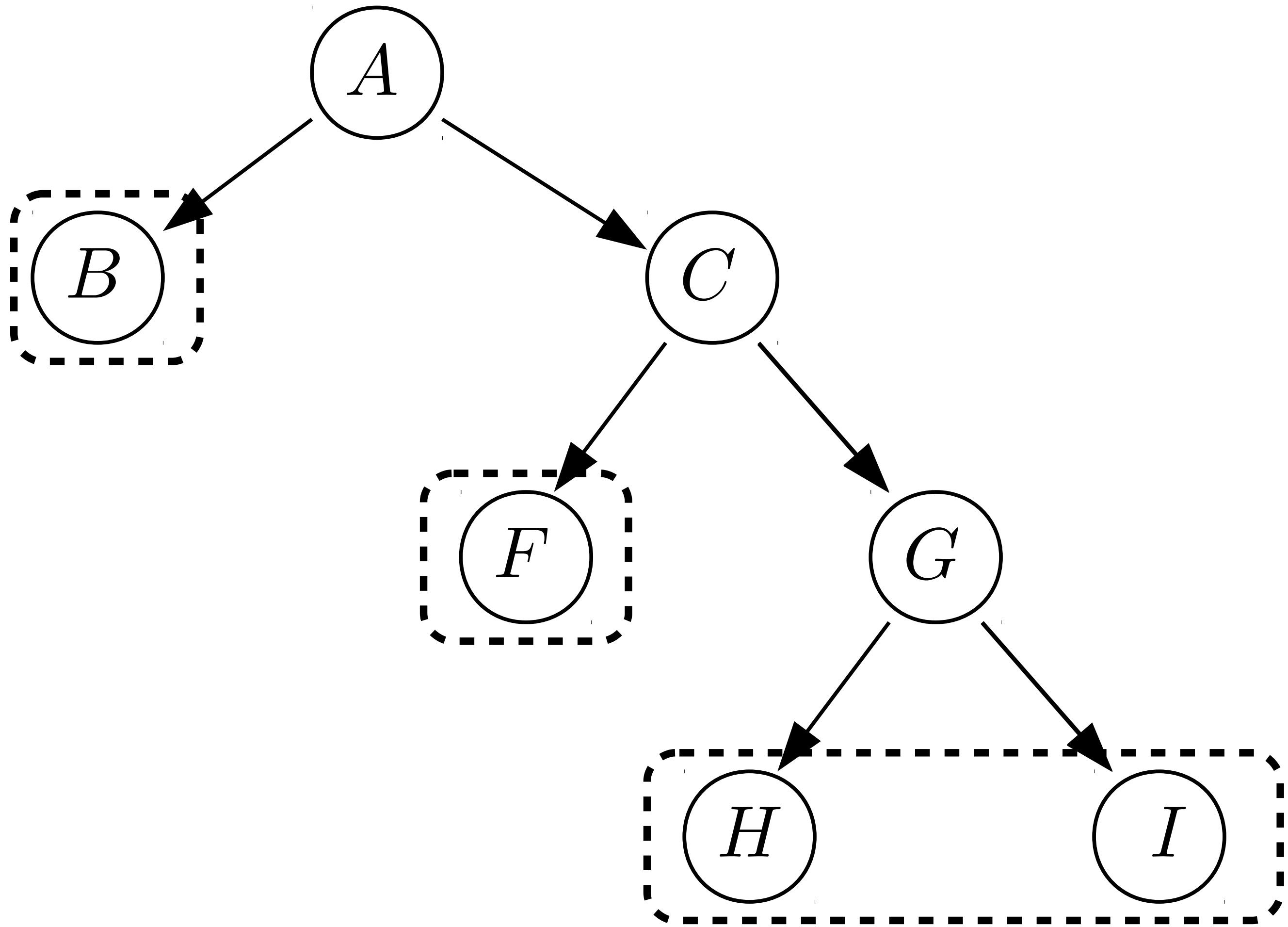
<u>Fringe</u>
~~A~~
B
~~C~~
F
~~G~~
H
I



DFS

Fringe

~~A~~
B
~~C~~
F
~~G~~
H
I

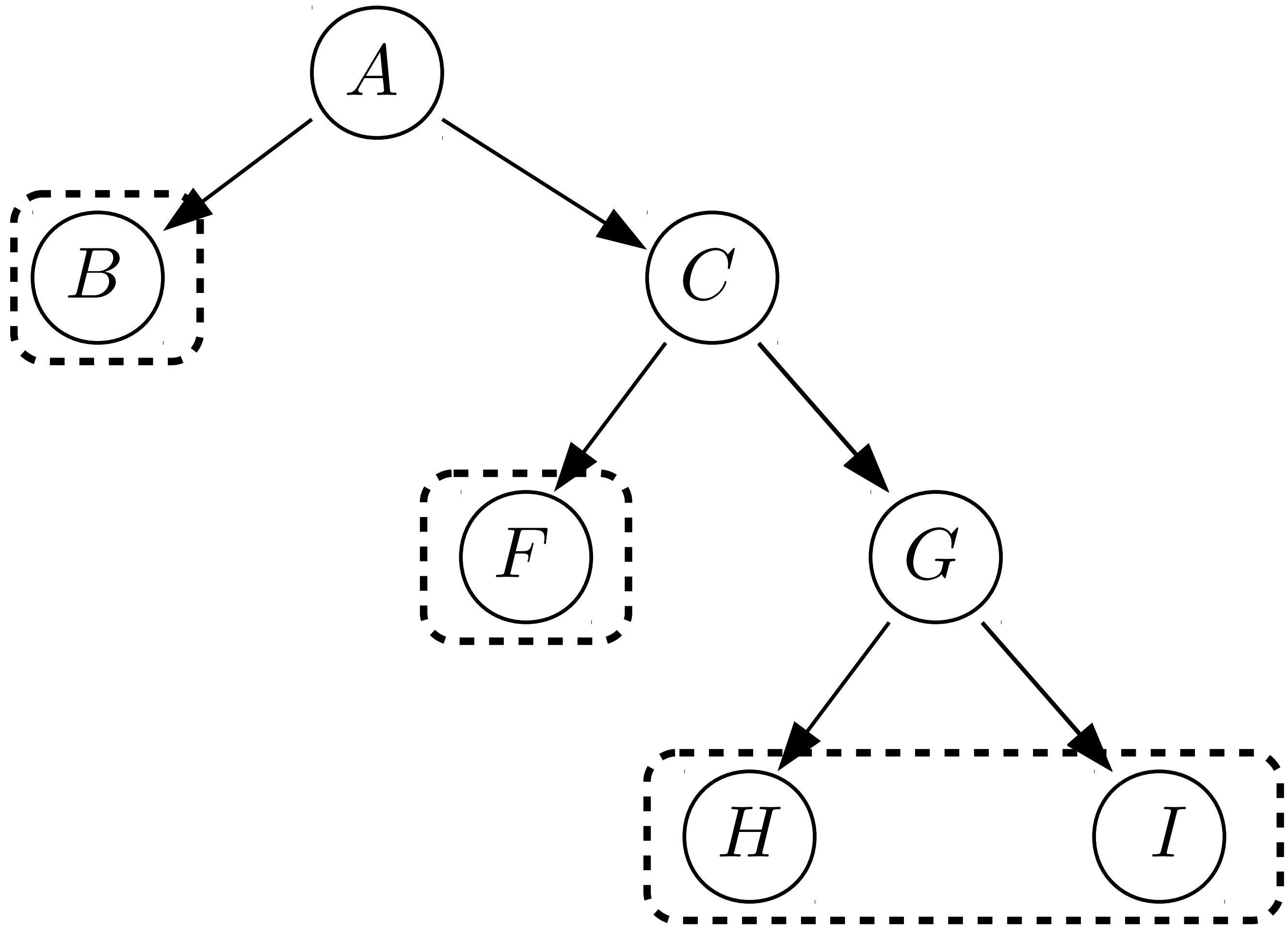


Which state gets removed next from the fringe?

DFS

<u>Fringe</u>

~~A~~
B
~~C~~
F
~~G~~
H
I

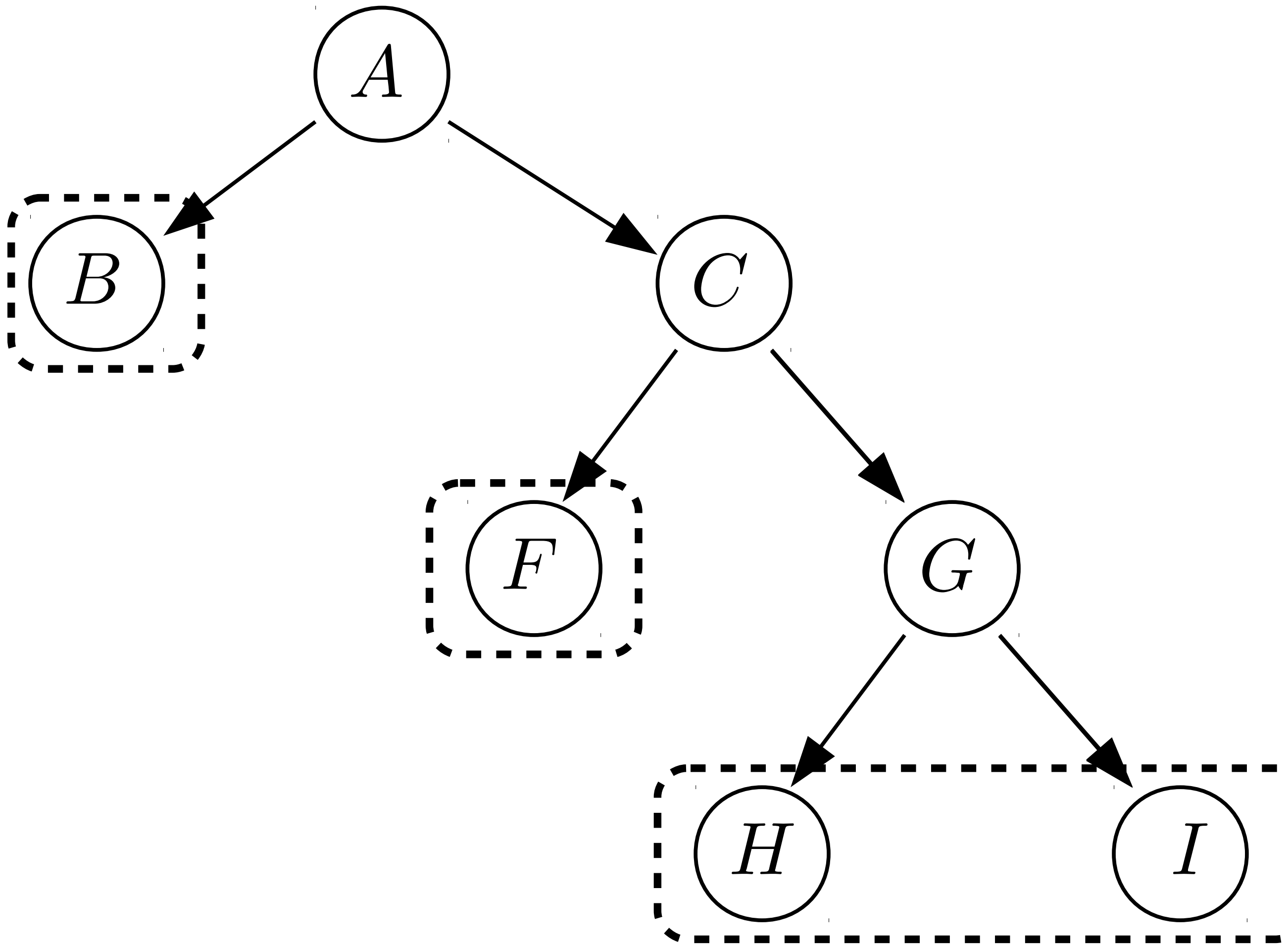


Which state gets removed next from the fringe?

What kind of a queue is this?

DFS

Fringe
~~A~~
B
~~C~~
F
~~G~~
H
I

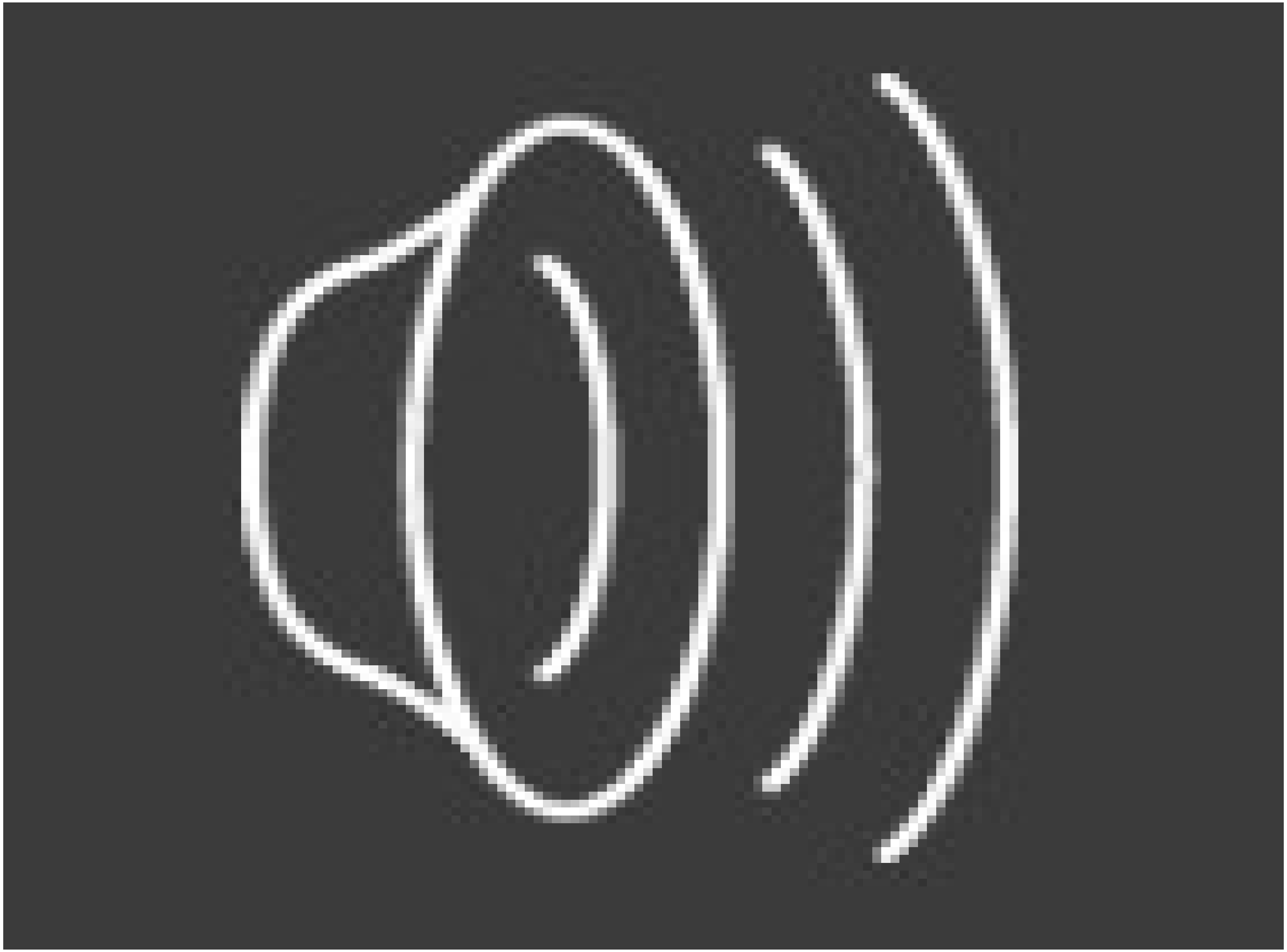


Which state gets removed next from the fringe?

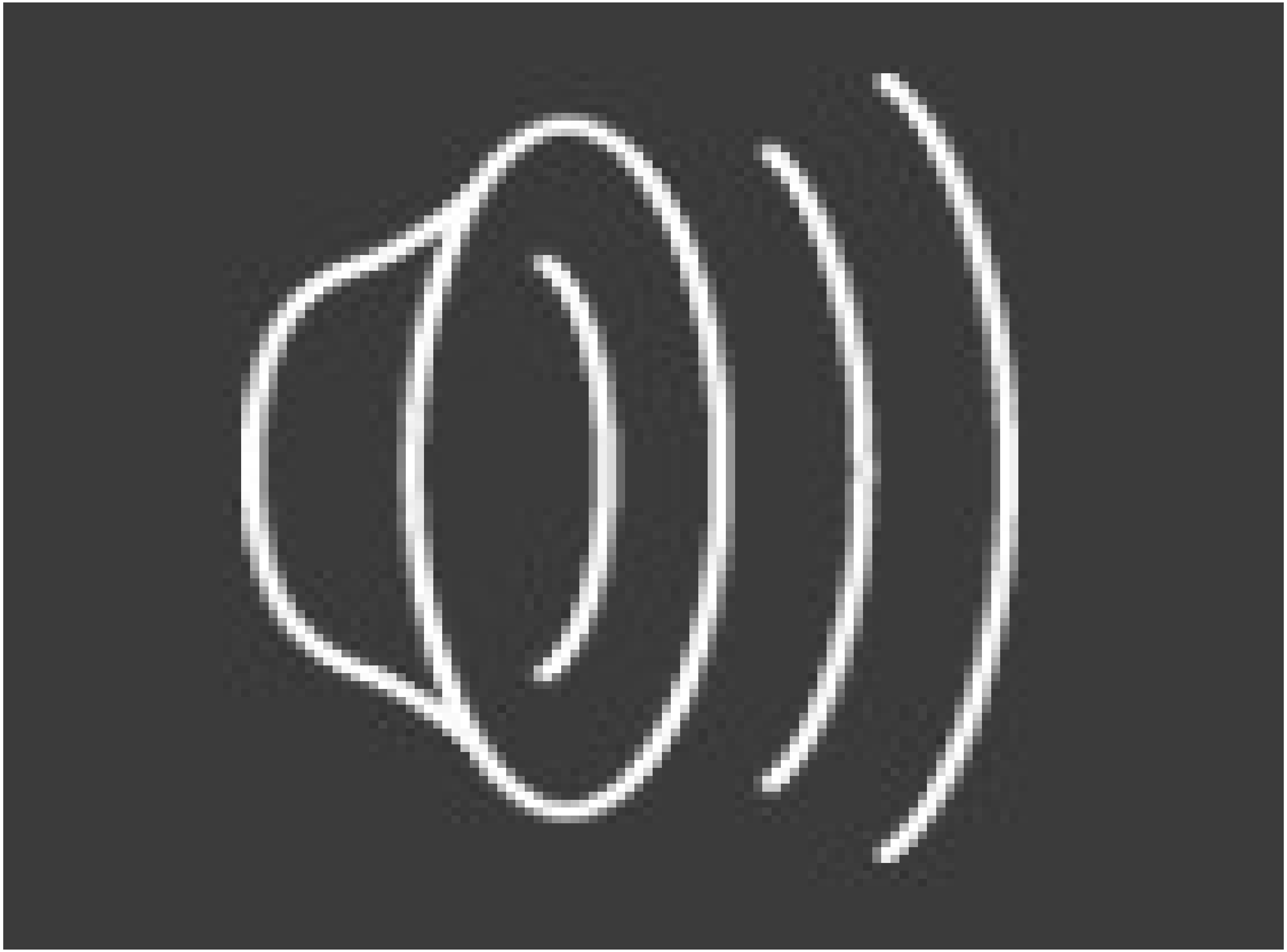
What kind of a queue is this?

LIFO Queue!
(last in first out)

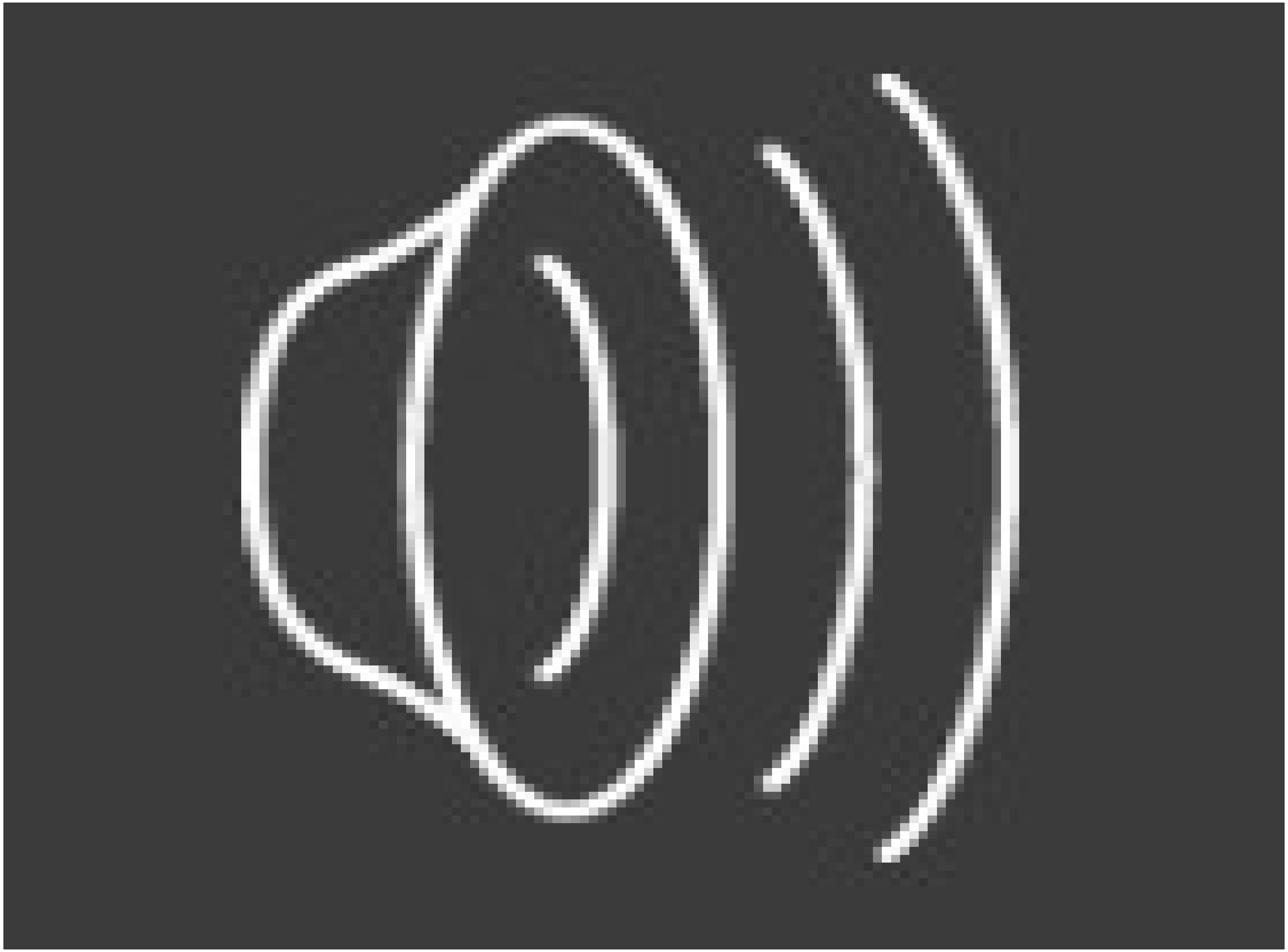
DFS vs BFS: which one is this?



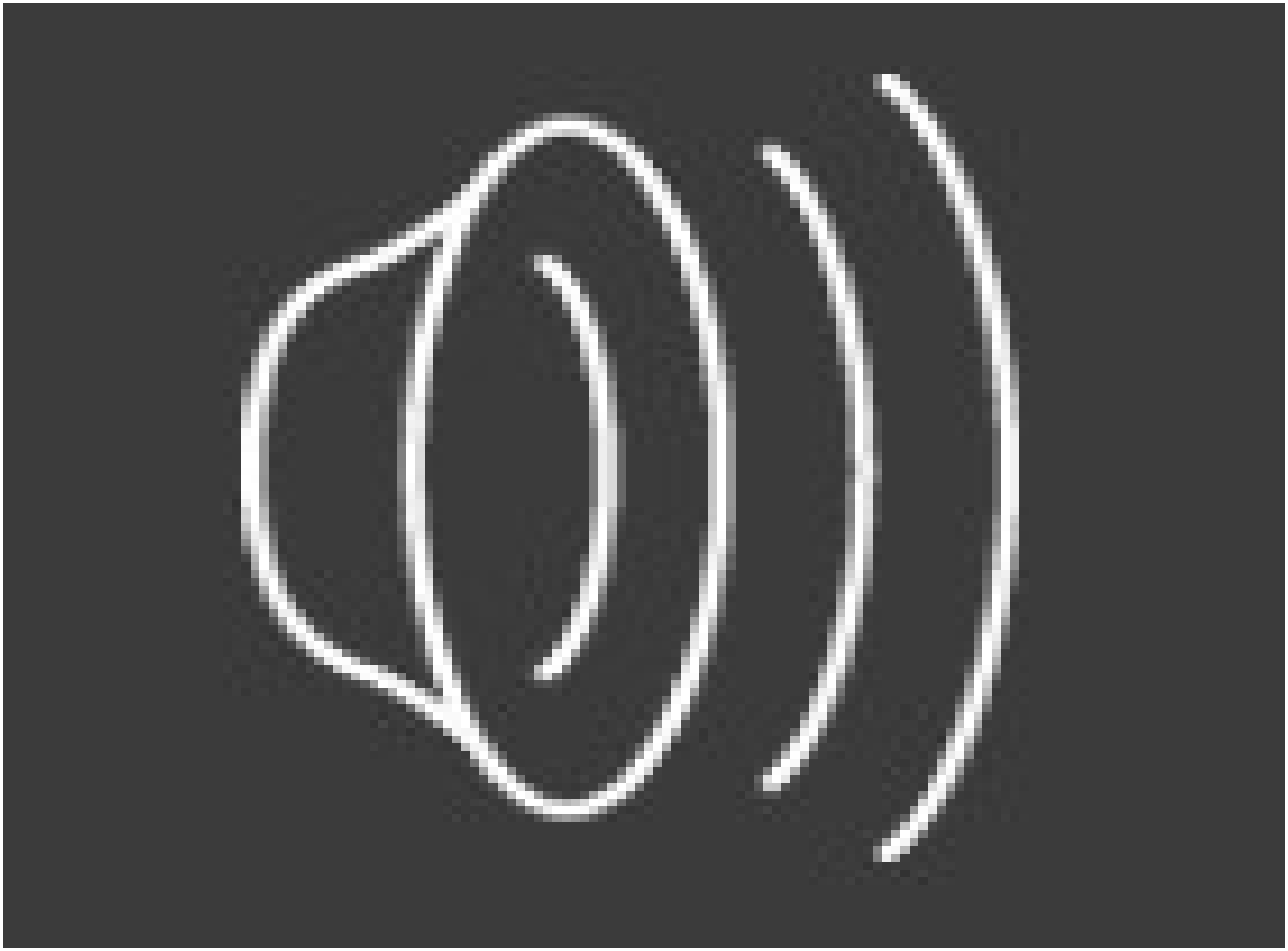
DFS vs BFS: which one is this?



BFS/UCS: which is this?



BFS/UCS: which is this?



DFS Properties: Graph search version

This is the "graph search" version of the algorithm

Is DFS <u>complete</u>?
– only if you track the explored set in memory

What is the <u>time complexity</u> of DFS (graph version)?
– how many states are expanded before finding a solution?
    – complexity = number of states in the graph

What is the <u>space complexity</u> of DFS (graph version)?
– how much memory is required?
    – complexity = number of states in the graph

Is DFS optimal?
– is it guaranteed to find the best solution (shortest path)?

DFS Properties: Graph search version

This is the “graph search” version of the algorithm

Is DFS complete?
– only if you track the explored set in memory

What is the time complexity of DFS (graph version)?
– how many states are expanded before finding a solution?
  – complexity = number of states in the graph

What is the space complexity of DFS (graph version)?
– how much memory is required?
  – complexity = number of states in the graph

Is DFS optimal?
– is it guaranteed to find the best solution (shortest path)?

So why would we ever use this algorithm?

DFS: Tree search version

This is the “tree search” version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

DFS: Tree search version

This is the “tree search” version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

What is the space complexity of DFS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $O(bm)$

DFS: Tree search version

This is the "tree search" version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

What is the space complexity of DFS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $\boxed{O(bm)}$

This is why we might want to use DFS

DFS: Tree search version

This is the "tree search" version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

What is the space complexity of DFS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $O(bm)$

What is the time complexity of DFS (tree version)?
– how many states are expanded before finding a solution? $O(b^m)$
  – complexity =

DFS: Tree search version

This is the "tree search" version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

What is the space complexity of DFS (tree version)?
– how much memory is required?
- – b: branching factor
- – m: maximum depth of any node
- – complexity = $O(bm)$

What is the time complexity of DFS (tree version)?
– how many states are expanded before finding a solution?
- – complexity = $O(b^m)$

Is it complete?

DFS: Tree search version

This is the "tree search" version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

What is the space complexity of DFS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $O(bm)$

What is the time complexity of DFS (tree version)?
– how many states are expanded before finding a solution? $O(b^m)$
  – complexity

Is it complete? NO!

DFS: Tree search version

This is the "tree search" version of the algorithm

Suppose you don't track the explored set.
– why wouldn't you want to do that?

What is the space complexity of DFS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $O(bm)$

What is the time complexity of DFS (tree version)?
– how many states are expanded before finding a solution?
  – complexity = $O(b^m)$

Is it complete?

NO!
What do we do???

IDS: Iterative deepening search

What is IDS?
– do depth-limited DFS in stages, increasing the maximum depth at each stage

IDS: Iterative deepening search

What is IDS?
– do depth-limited DFS in stages, increasing the maximum depth at each stage

What is depth limited search?
– any guesses?

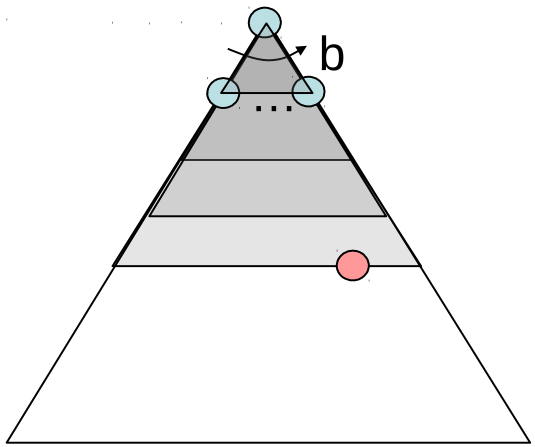
IDS: Iterative deepening search

What is IDS?
– do depth-limited DFS in stages, increasing the maximum depth at each stage

What is depth limited search?
– do DFS up to a certain pre-specified depth

IDS: Iterative deepening search

- Idea: get DFS's space advantage with BFS's time / shallow-solution advantages
  - Run a DFS with depth limit 1. If no solution…
  - Run a DFS with depth limit 2. If no solution…
  - Run a DFS with depth limit 3. …..



- Isn't that wastefully redundant?
  - Generally most work happens in the lowest level searched, so not so bad!

IDS

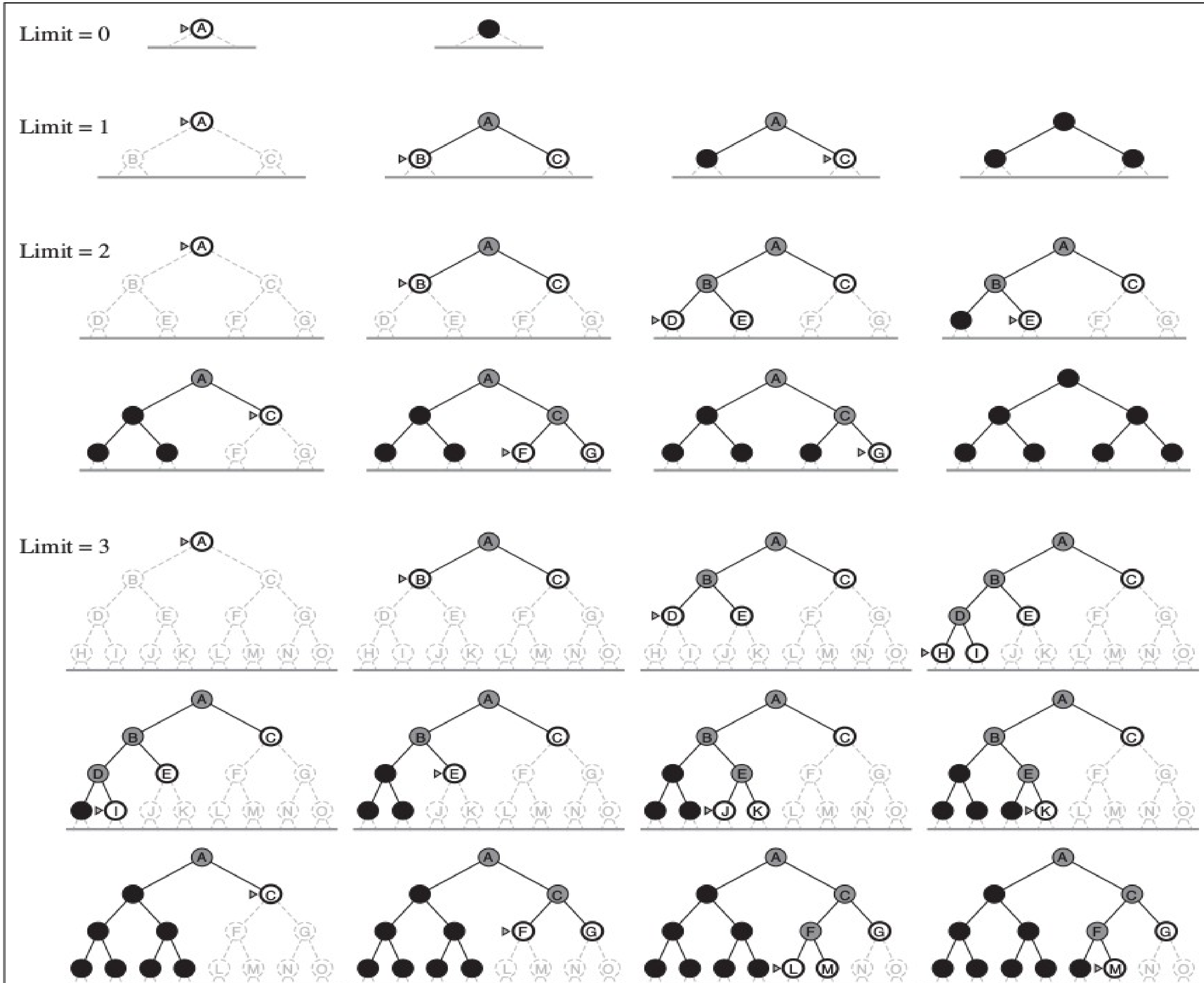


Figure 3.19 Four iterations of iterative deepening search on a binary tree.

IDS

What is the <u>space complexity</u> of IDS (tree version)?
– how much memory is required?

- b: branching factor
- m: maximum depth of any node
- complexity = $O(bm)$

What is the <u>time complexity</u> of DFS (tree version)?
– how many states are expanded before finding a solution? $O(b^m)$

- complexity =

Is it complete?

IDS

What is the <u>space complexity</u> of IDS (tree version)?
– how much memory is required?
  – b: branching factor
  – m: maximum depth of any node
  – complexity = $O(bm)$

What is the <u>time complexity</u> of DFS (tree version)?
– how many states are expanded before finding a solution? $O(b^m)$
  – complexity =

Is it complete? YES!!!

Is it optimal?

IDS

What is the <u>space complexity</u> of IDS (tree version)?
– how much memory is required?

- b: branching factor
- m: maximum depth of any node
- complexity = $O(bm)$

What is the <u>time complexity</u> of DFS (tree version)?
– how many states are expanded before finding a solution?

- complexity = $O(b^m)$

Is it complete? YES!!!

Is it optimal? YES!!!

General thoughts about search

If your model is wrong, then your solution will be wrong.

- In November 2010, Nicaraguan troops unknowingly crossed the border to Costa Rica, removed that country's flag and replaced it with their own. The reason: Google Maps told the troops' commander the territory belonged to Nicaragua.