Applied AI

Lecture 3

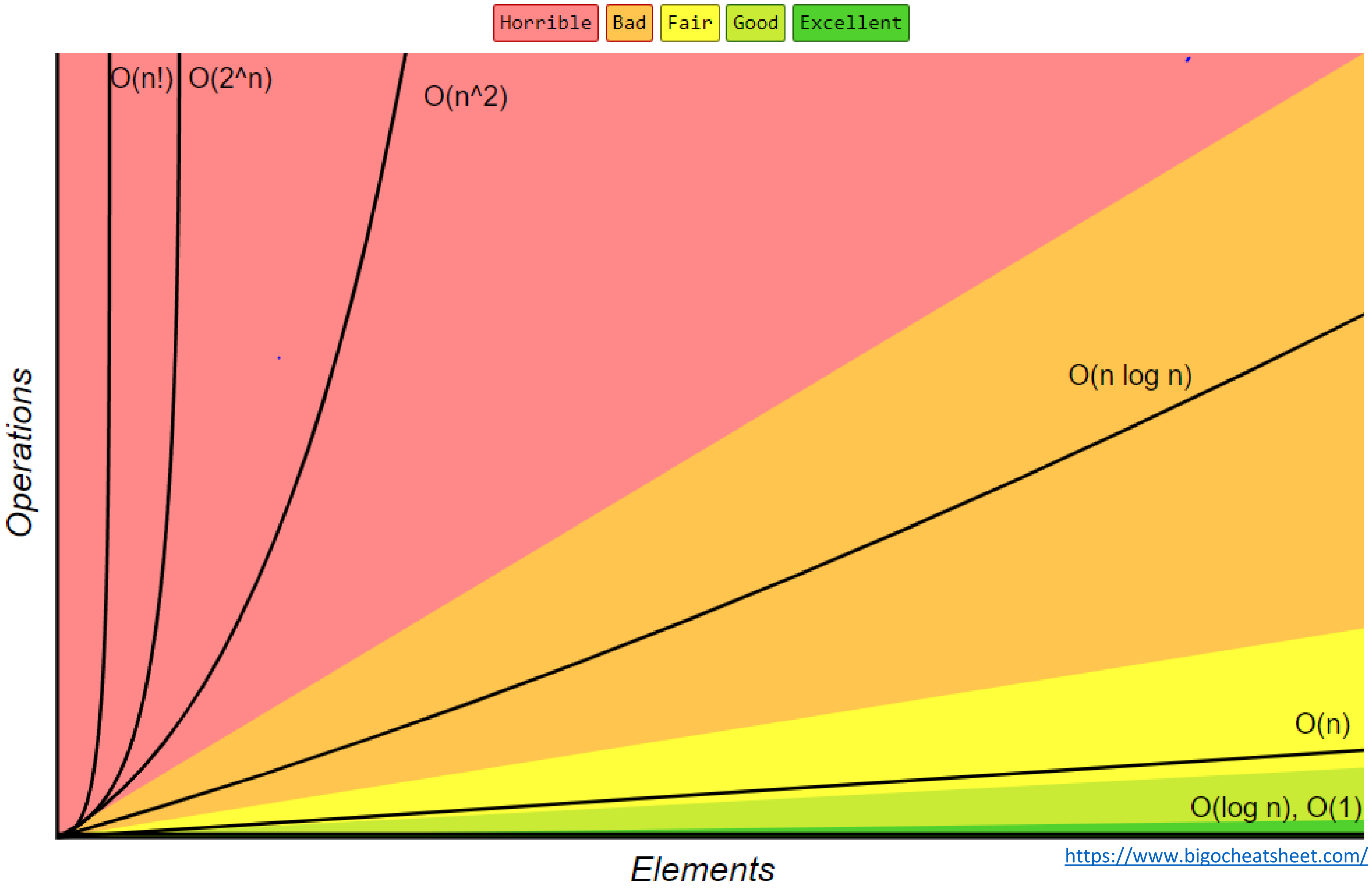
Dr Artie Basukoski

[slides adapted from Artificial Intelligence: A Modern Approach, Russel and Norvig]

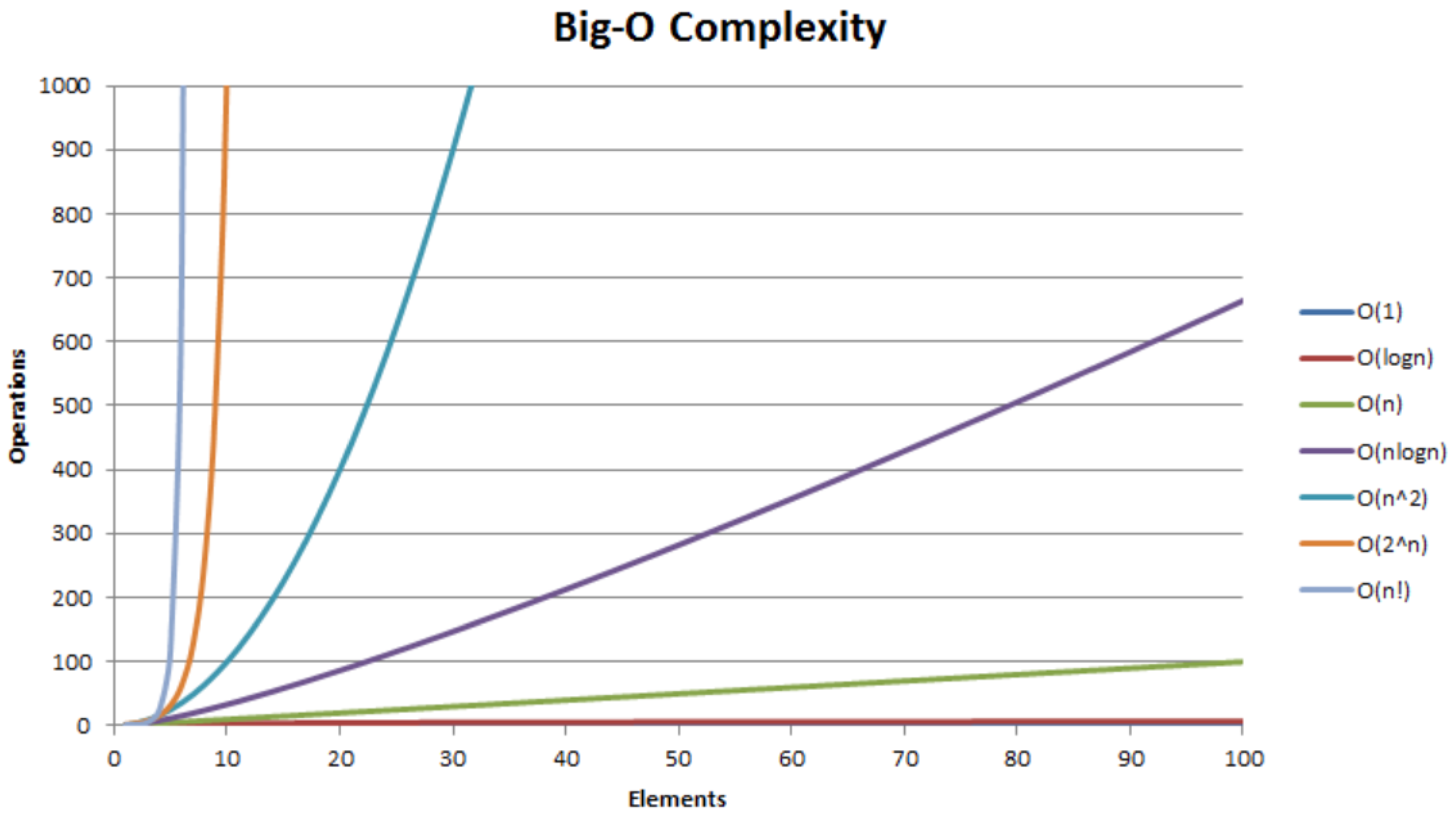
Agenda

- Search continued
- Big Oh Notation
- Comparison of uninformed search algorithms
- Inform the search algorithms
- Comparison informed search algorithms

Big-O Complexity Chart



https://www.bigocheatsheet.com/


Big-O Complexity
Operations
1000
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
Elements
O(1)
O(logn)
O(n)
O(nlogn)
O(n^2)
O(2^n)
O(n!)

Big-O for some common algorithms

Data Structure Operation	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	Θ(1)	Θ(n)	Θ(n)	Θ(n)	Θ(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)	Θ(n)	Θ(1)	Θ(1)	Θ(n)

Array Sorting Algorithms	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n log(n))
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	Θ(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n log(n))
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	Θ(n)

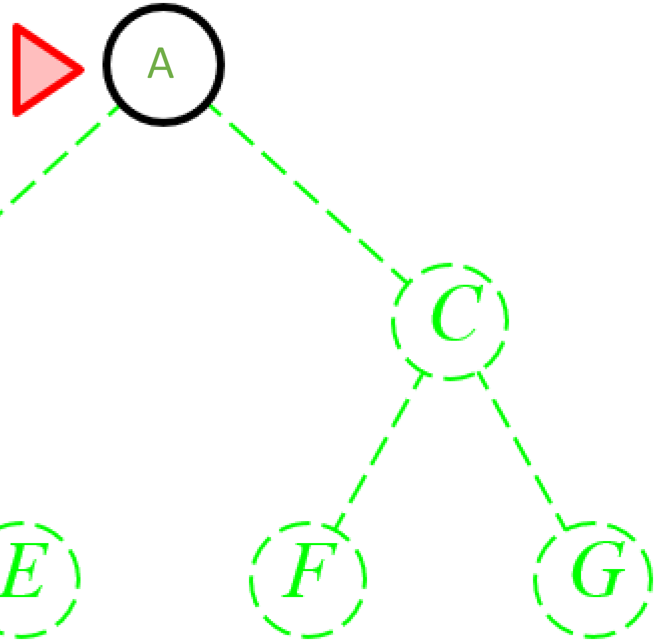
https://www.bigocheatsheet.com/

Breadth first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



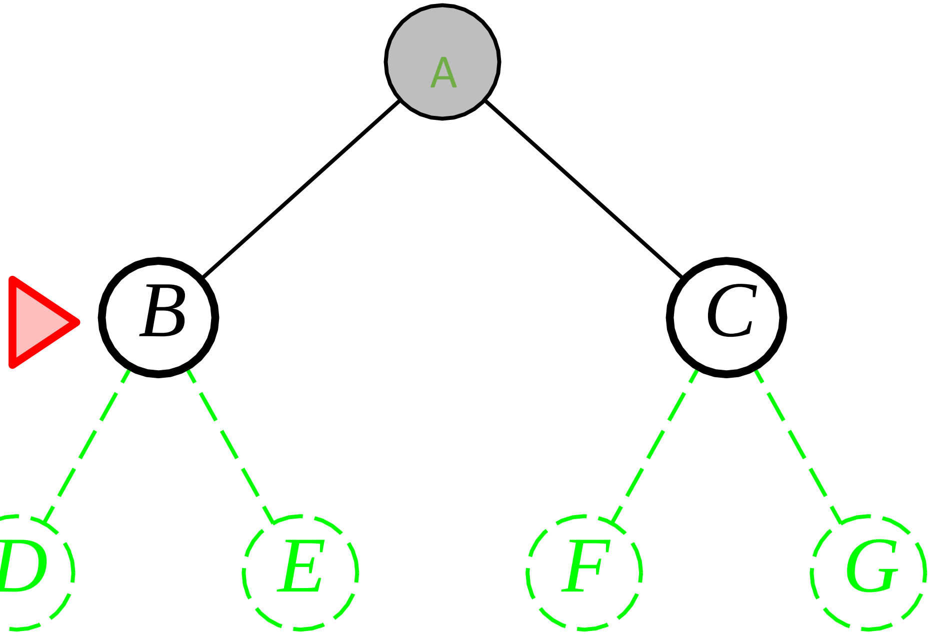
fringe = [A]

Breadth first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



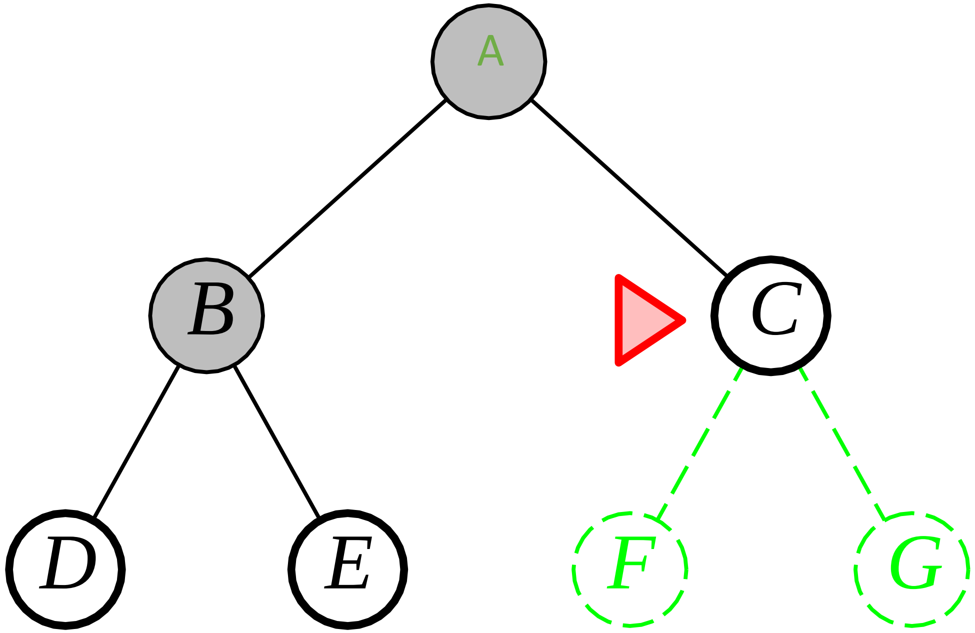
fringe = [B,C]

Breadth first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



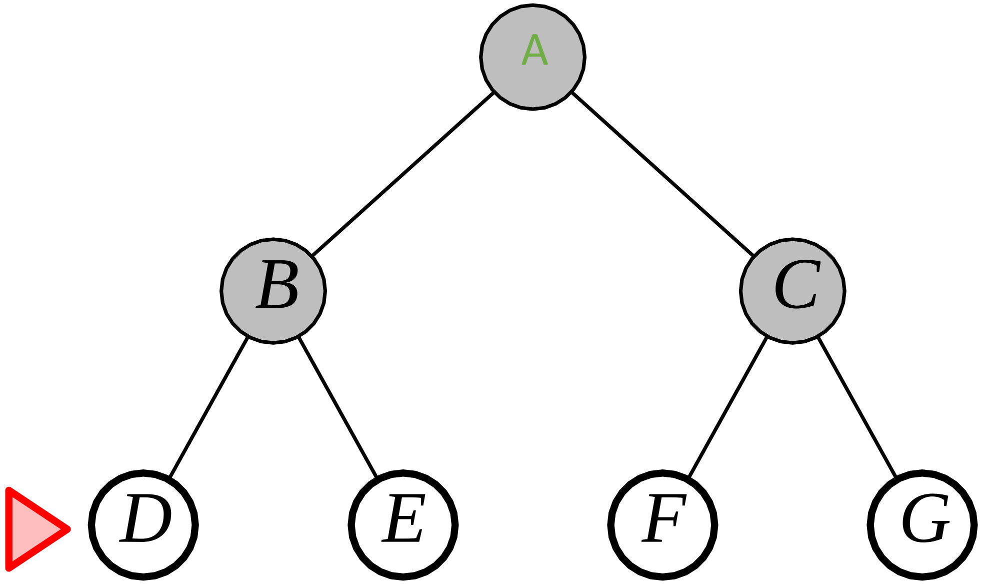
fringe = [C,D,E]

Breadth first search

Expand shallowest unexpanded node

Implementation:

fringe is a FIFO queue, i.e., new successors go at end



fringe = [D,E,F,G]

Some Definitions to analyse search

- Complete: Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
- Time (complexity): How long does it take to find a solution?
- Space (complexity): How much memory is needed to perform the search?
- Optimal: Does it provide the lowest path cost among all solutions.?
- *b – branching factor*
- *d – depth of shallowest solution*
- *m – maximum depth or when there is no solution*
- *l – depth limit*
- *C* – cost of optimal path*

Properties of breadth-first search

Complete: Yes (if b is finite)

Time: $1 + b + b^2 + b^3 + \ldots + b^d + b(b^d - 1) = O(b^{d+1})$, i.e., exp. in d

Space: $O(b^{d+1})$ (keeps every node in memory)

Optimal: Yes (if cost = 1 per step); not optimal in general

Space is the big problem; can easily generate nodes at 100MB/sec so 24hrs = 8640GB.

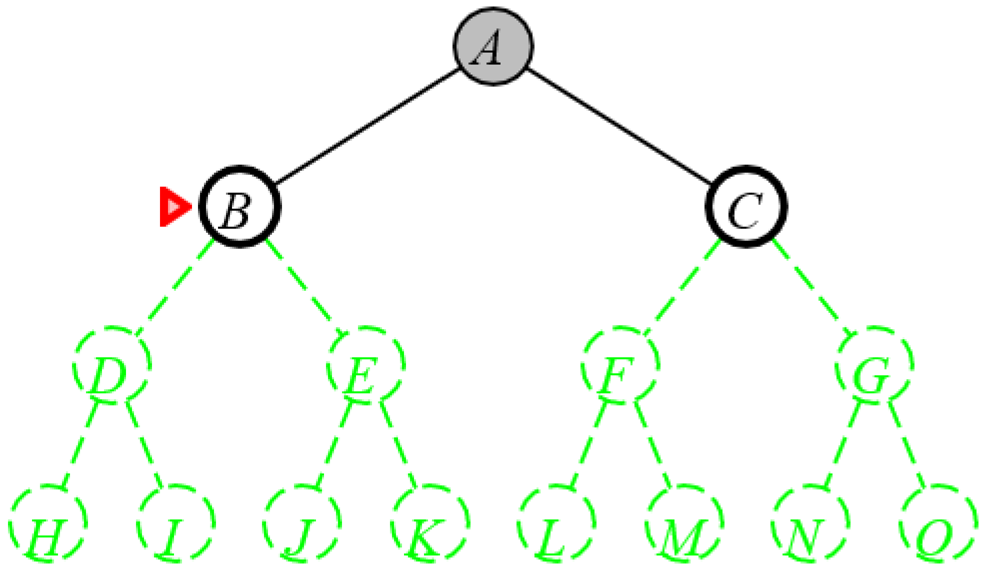
Depth first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

fringe = [B,C]



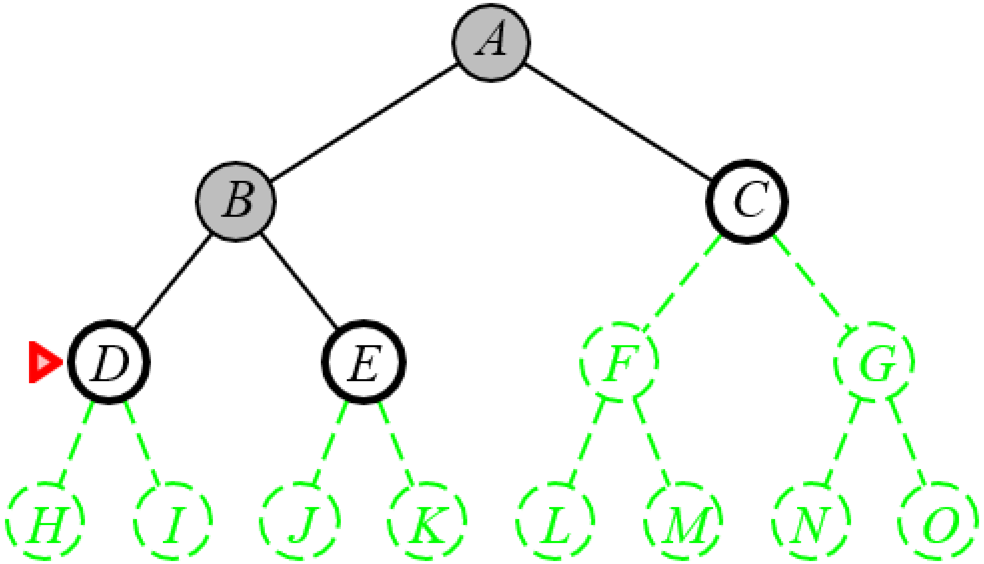
Depth first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

fringe = [D,E,C]



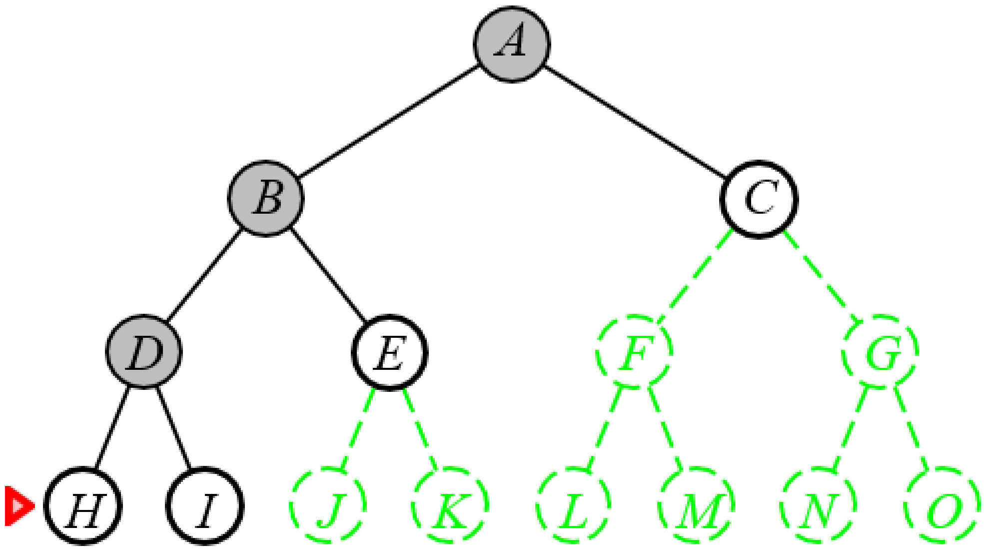
Depth first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

fringe = [H,I,E,C]



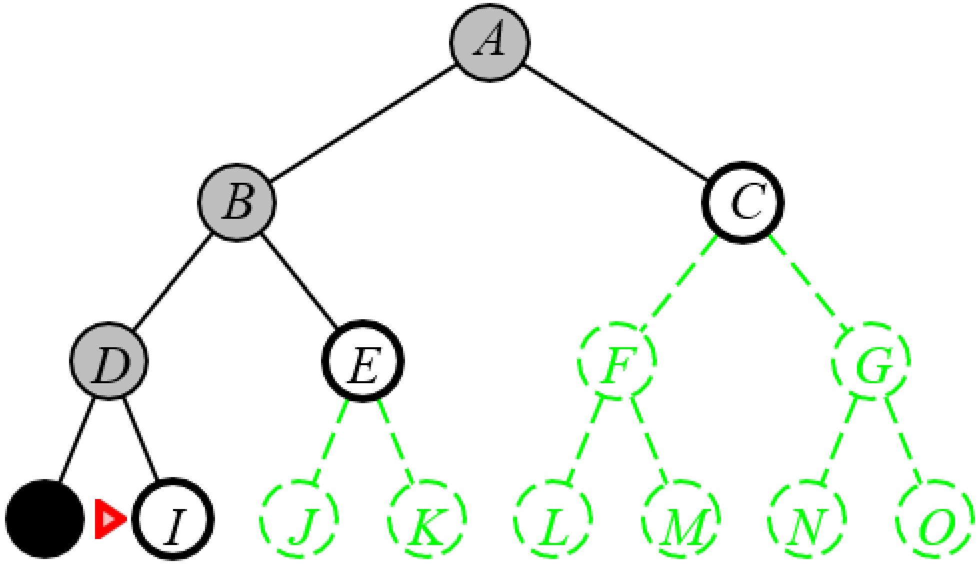
Depth first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

fringe = [I,E,C]

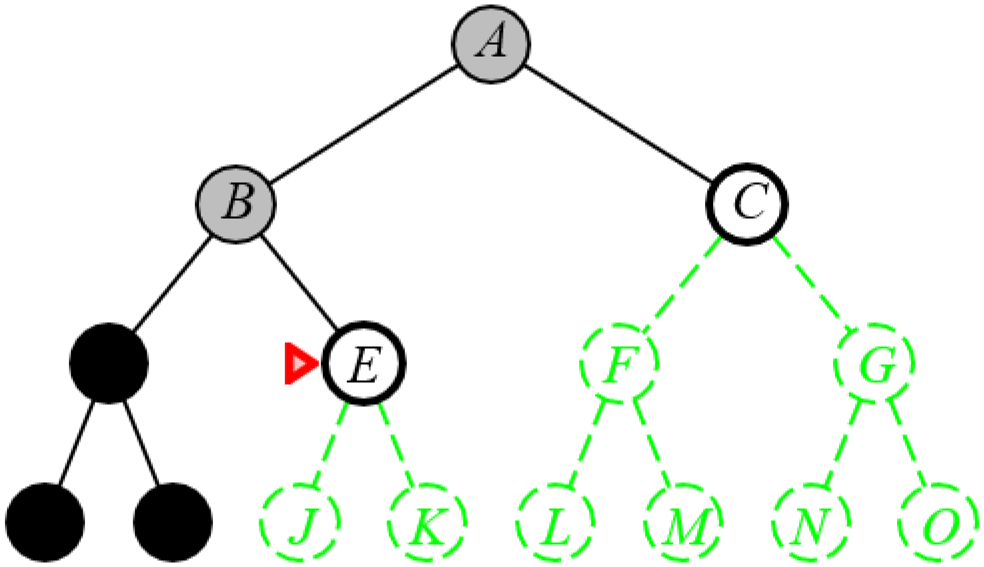


Depth first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front



fringe = [E,C]

What is next??

Properties of depth-first search

Complete: No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path
⇒ complete in finite spaces

Time: $O(b^m)$: terrible if m is much larger than d
but if solutions are dense, may be much faster than breadth-first

Space: $O(bm)$, i.e., linear space!

Optimal: No

How do we deal with these problems?

- Limit depth.
- Iterative deepening.
- Think about using heuristics.
- What are heuristics – rules-of-thumb that can be applied to guide decision-making based on a more limited subset of the available information.

“Uses domain-specific hints about the location of goals”

HAL in 2001:A Space Odyssey stood for “Heuristic Algorithmic”

Depth-limited research

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors

Recursive implementation:

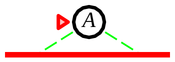
```
function Depth-Limited-Search( problem,limit) returns soln/fail/cutoff
    Recursive-DLS(Make-Node(Initial-State[problem]),problem,limit)

 function Recursive-DLS(node,problem,limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if Goal-Test(problem,State[node]) then return node
    else if Depth[node] = limit then return cutoff
    else for each successor in Expand(node,problem) do
        result ← Recursive-DLS(successor,problem,limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result /= failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Iterative deepening search

```
function Iterative-Deepening-Search( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← Depth-Limited-Search( problem, depth)
        if result /= cutoff then return result
    end
```

Iterative deepening search $l = 0$

Limit = 0

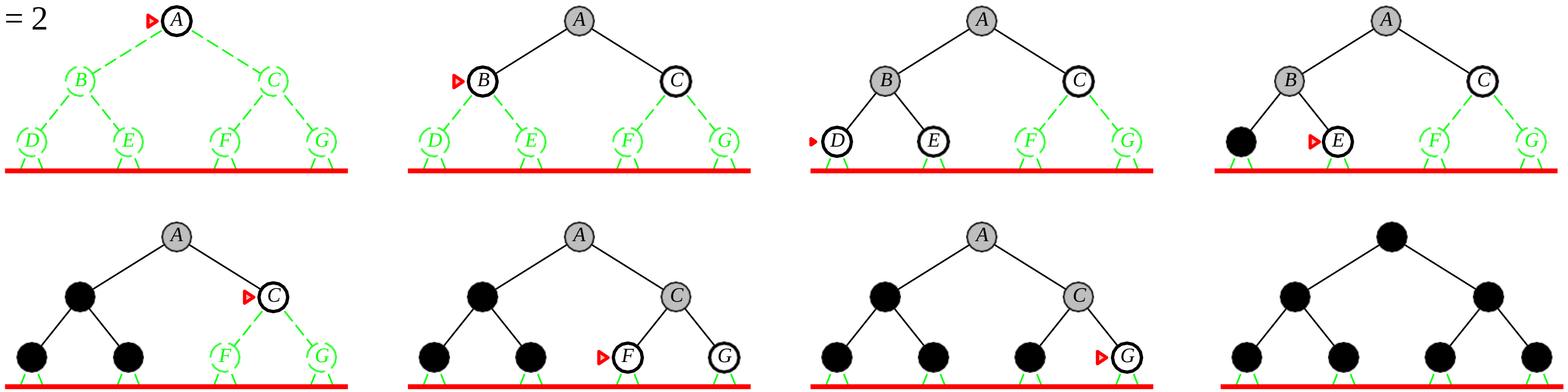


Iterative deepening search $l = 1$

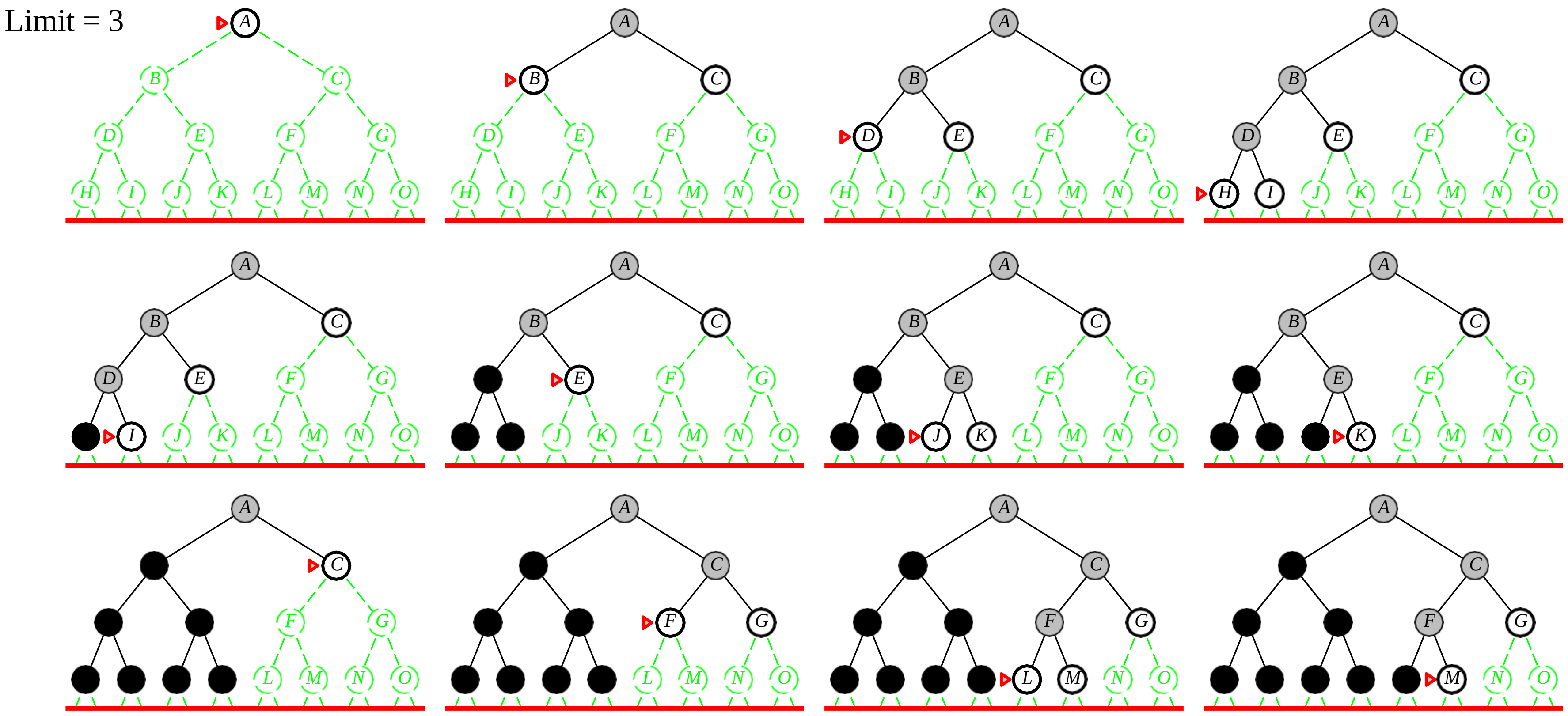
Limit = 1

Iterative deepening search $l = 2$

Limit = 2



Iterative deepening search $l = 3$

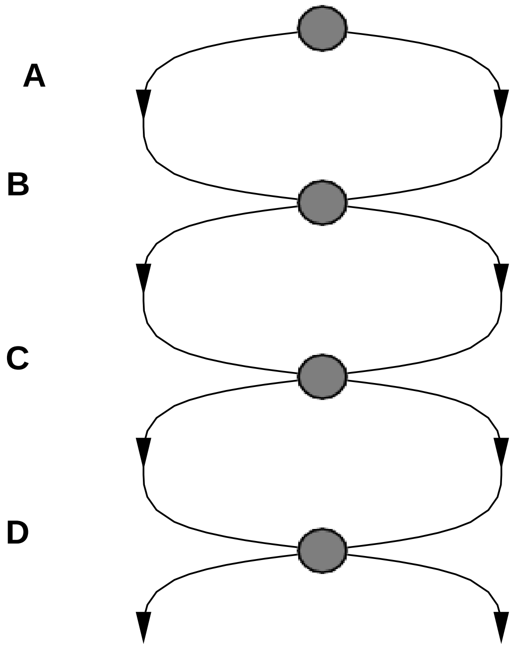


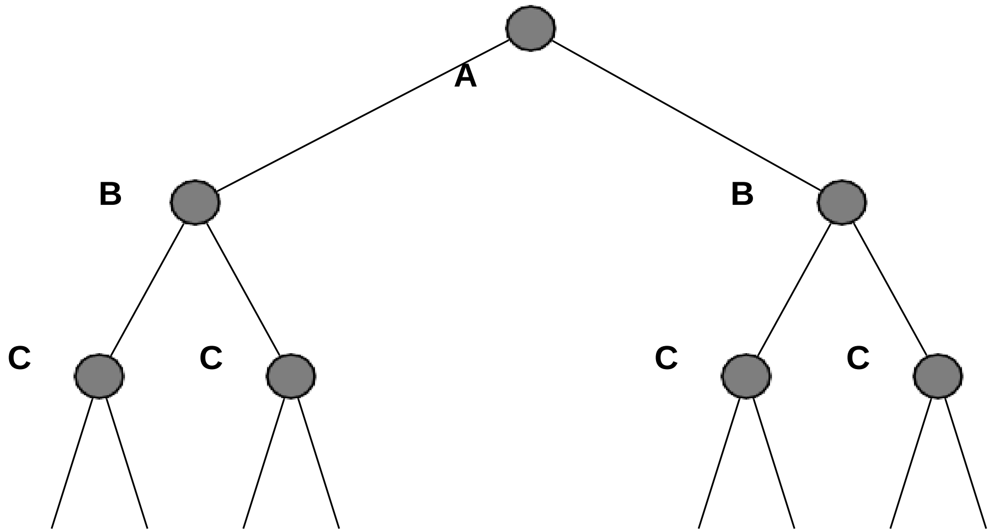
Comparison of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes$*$	Yes$*$	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	b^{l}	b^{m}	b^{l}	b^{d}
Space	b^{d+1}	b^{l}	bm	bl	bd
Optimal?	Yes$*$	Yes	No	No	Yes$*$

Repeated states

Failure to detect repeated states can turn a linear problem into an exponential one!





Graph search – detect loops

```
function Graph-Search(problem, fringe) returns a solution, or failure
   closed ← an empty set
   fringe ← Insert(Make-Node(Initial-State[problem]),fringe)
   loop do
       if fringe is empty then return failure
       node ← Remove-Front(fringe)
       if Goal-Test(problem,State[node]) then return node
       if State[node] is not in CLOSED then
           add State[node] to CLOSED
           fringe ← InsertAll(Expand(node,problem),fringe)
   end
```

Uninformed Search Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies – BFS, DFS, Iterative Deepening
- Iterative deepening uses only linear space and not much more time than other uninformed algorithms
- Graph search can be exponentially more efficient than tree search

Informed search algorithms

Best-first search

Idea: use an evaluation function for each node
– estimate of "desirability"

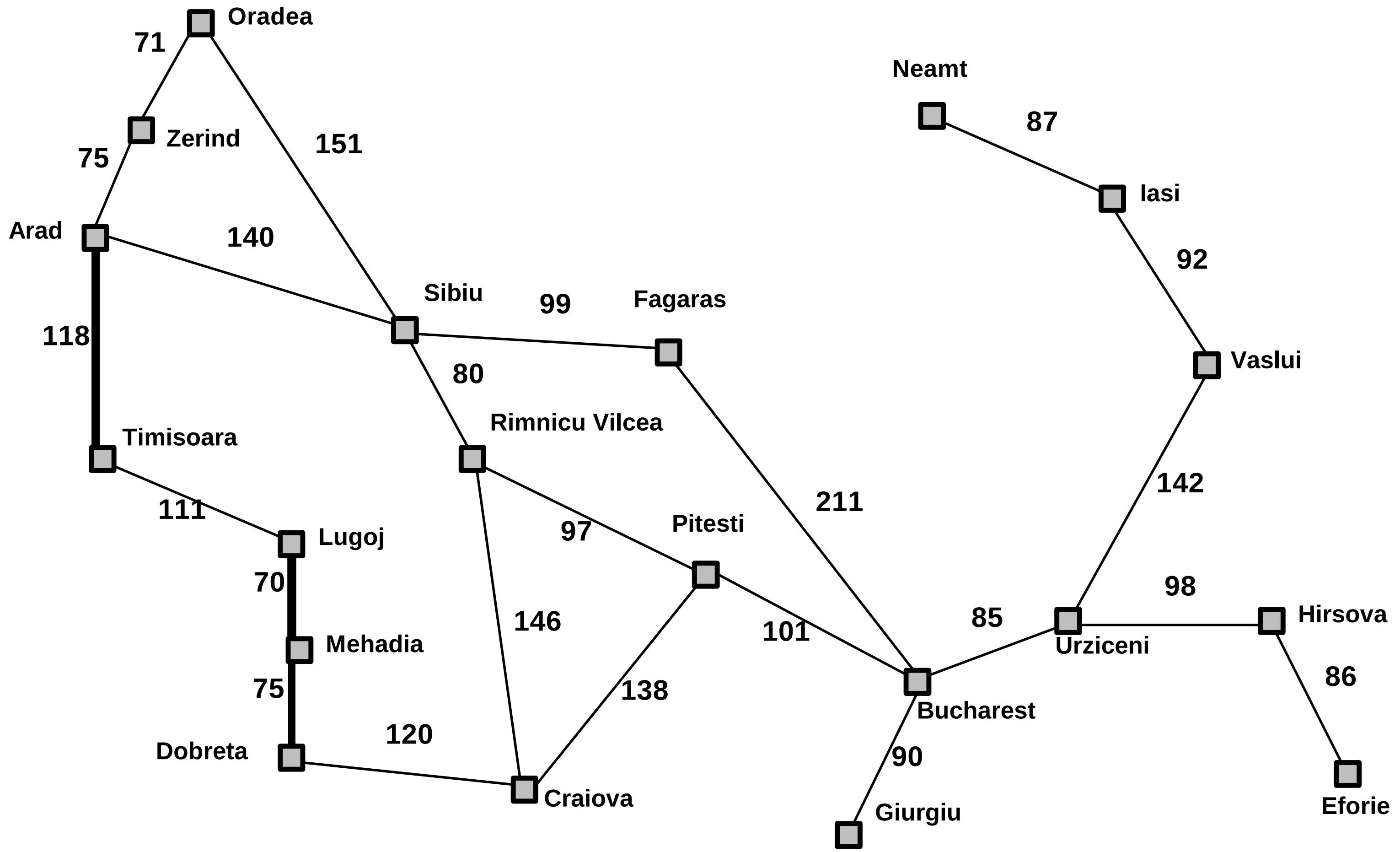
⇒ Expand most desirable unexpanded node

Implementation:

fringe is a queue sorted in decreasing order of desirability

Special cases:
greedy search
A* search

Romania graph with step costs in kilometres



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

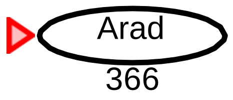
Greedy search

Evaluation function $h(n)$ (heuristic)
    = estimate of cost from n to the closest goal

E.g., $h_{\mathrm{SLD}}(n)$ = straight-line distance from n to Bucharest

Greedy search expands the node that appears to be closest to goal

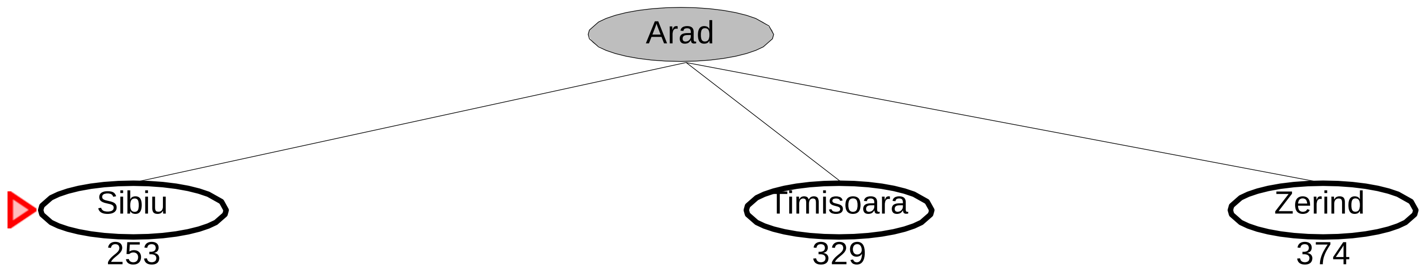
Greedy search example



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

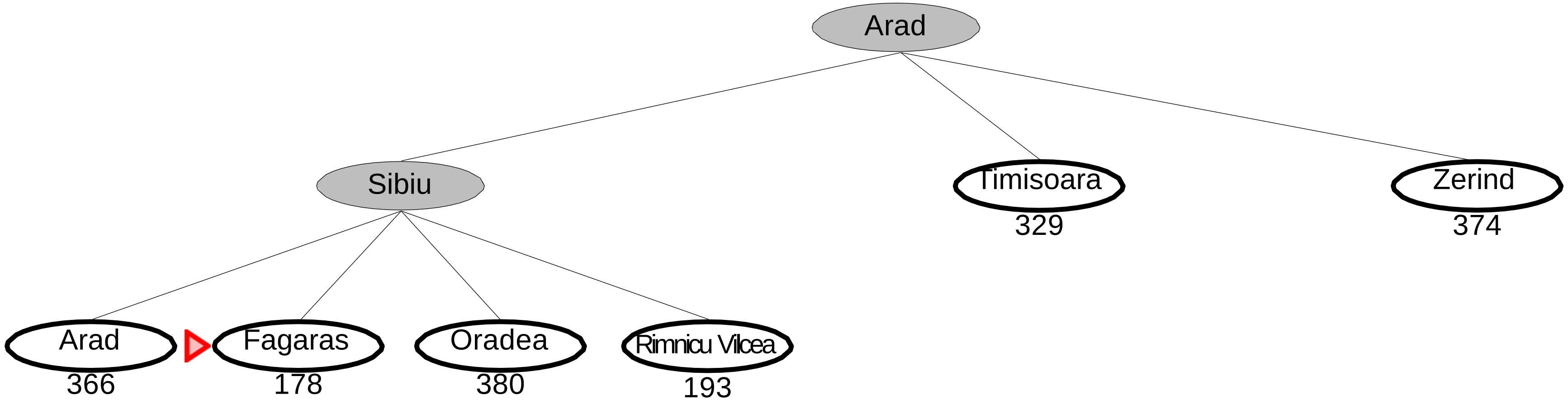
Greedy search example



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

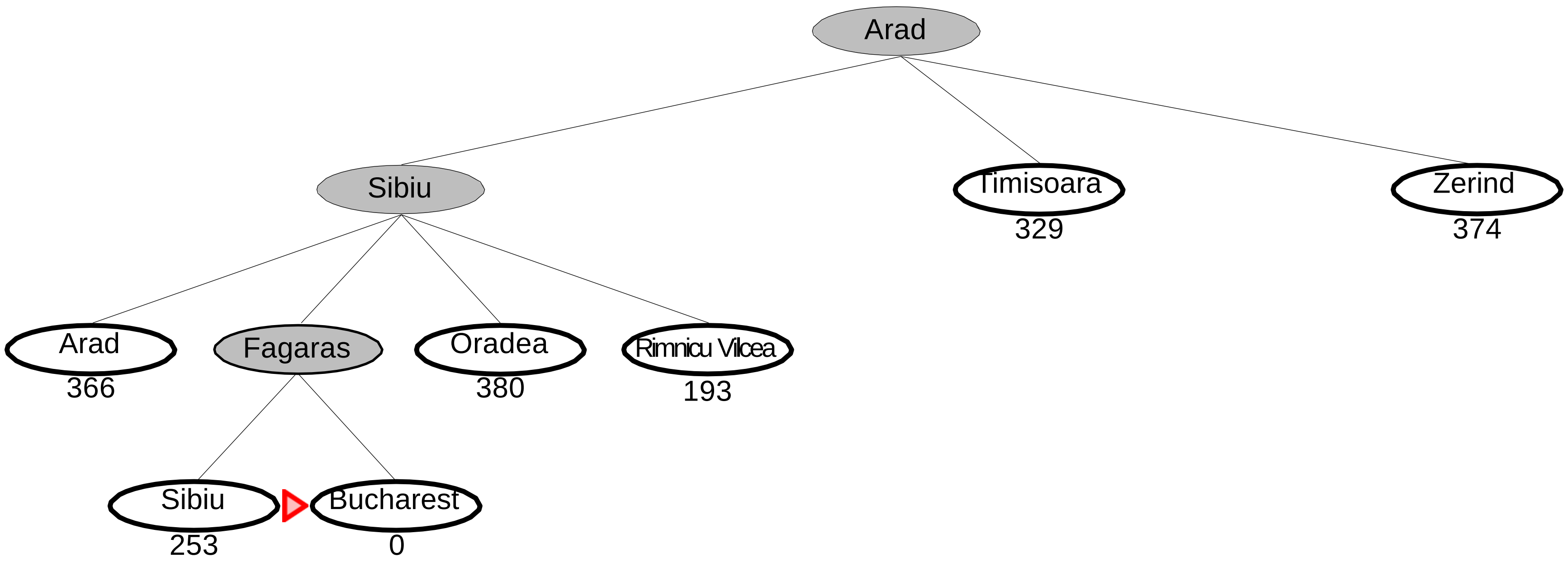
Greedy search example



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Greedy search example



Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Properties of greedy search

Complete: No–can get stuck in loops, e.g.,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

Time: $O(b^m)$, but a good heuristic can give dramatic improvement

Space: $O(b^m)$—keeps all nodes in memory

Optimal: No

A* search

Idea: avoid expanding paths that are already expensive

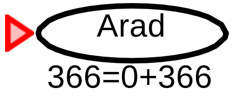
Evaluation function $f(n) = g(n) + h(n)$

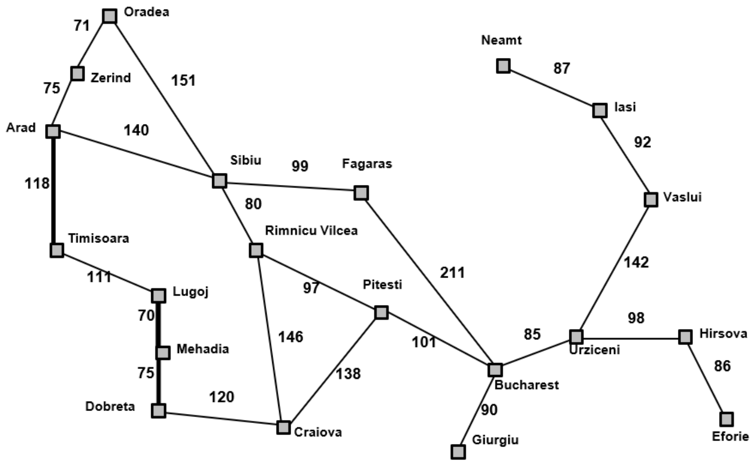
$g(n)$ = cost so far to reach n

$h(n)$ = estimated cost to goal from n

$f(n)$ = estimated total cost of path through n to goal

A* search example

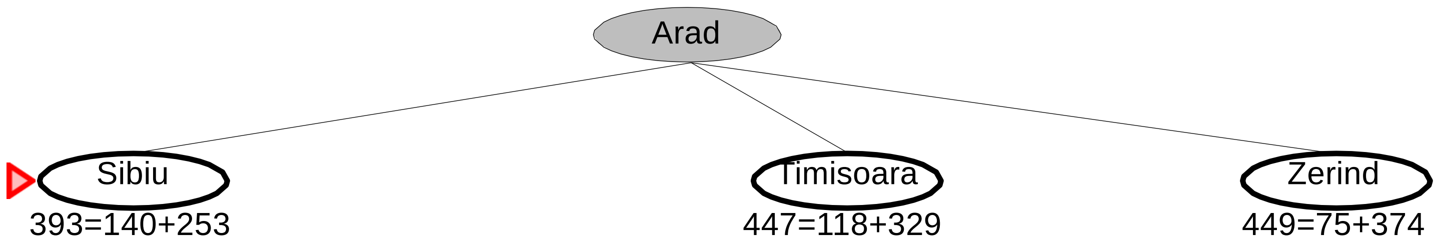




Straight−line distance
to Bucharest

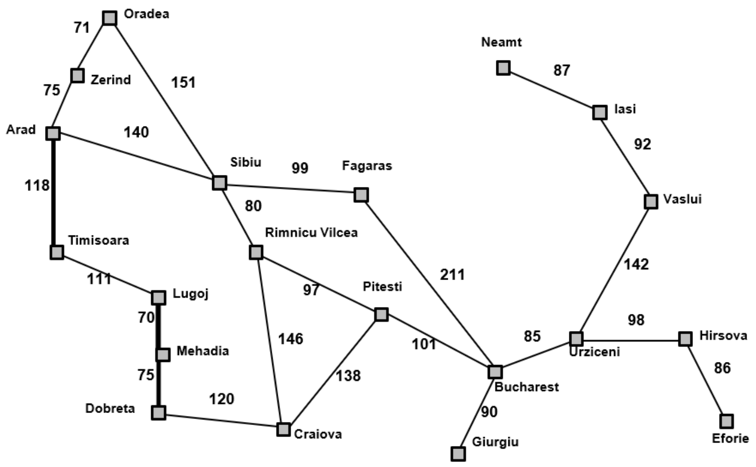
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

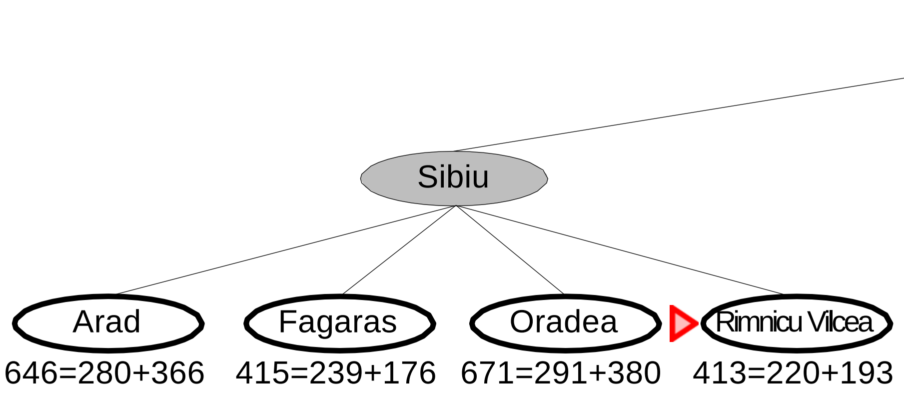


Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

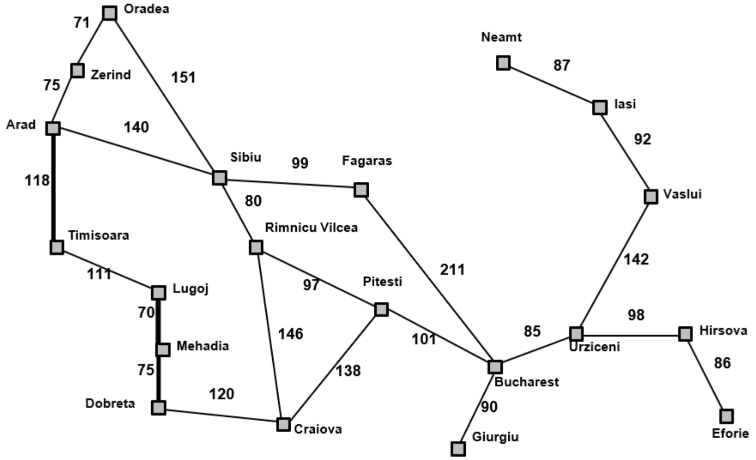


A* search example

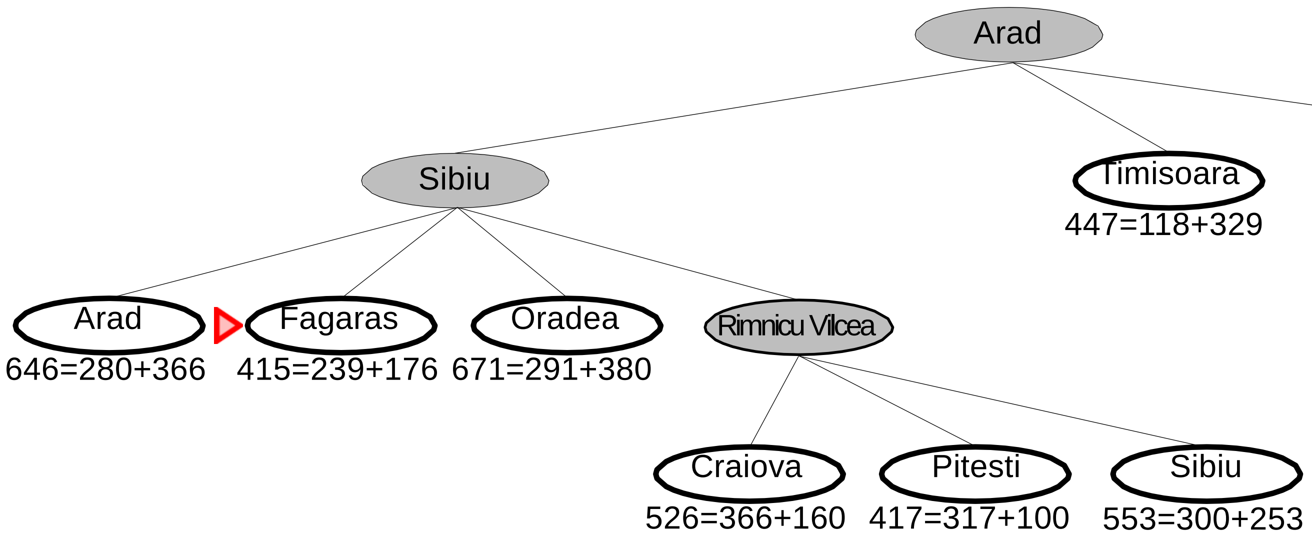


Straight−line distance to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

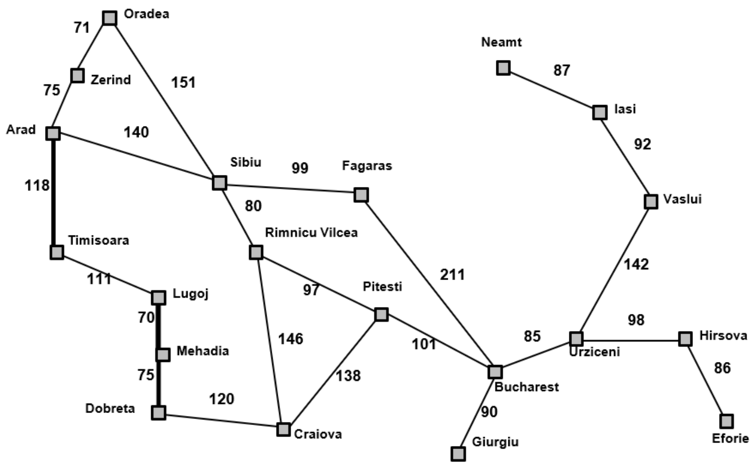


A* search example

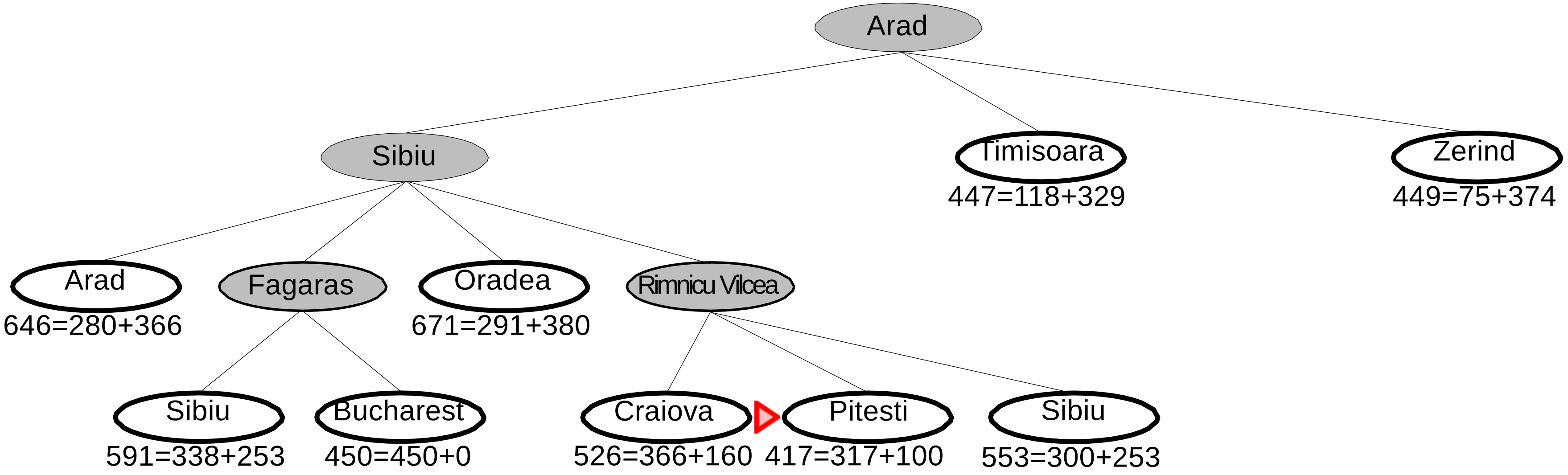


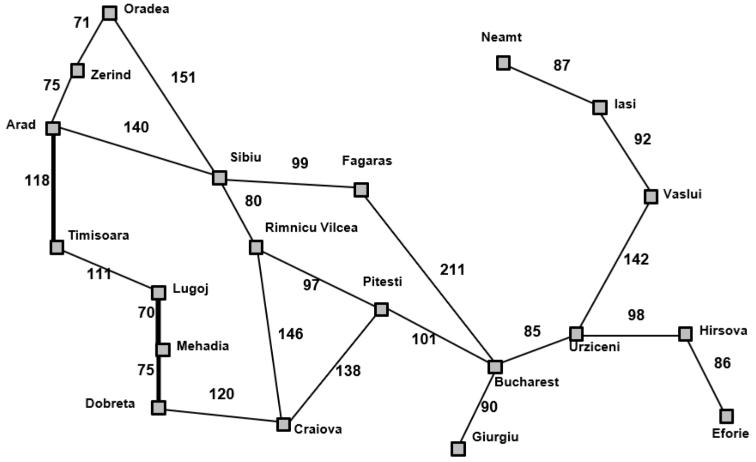
Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



A* search example

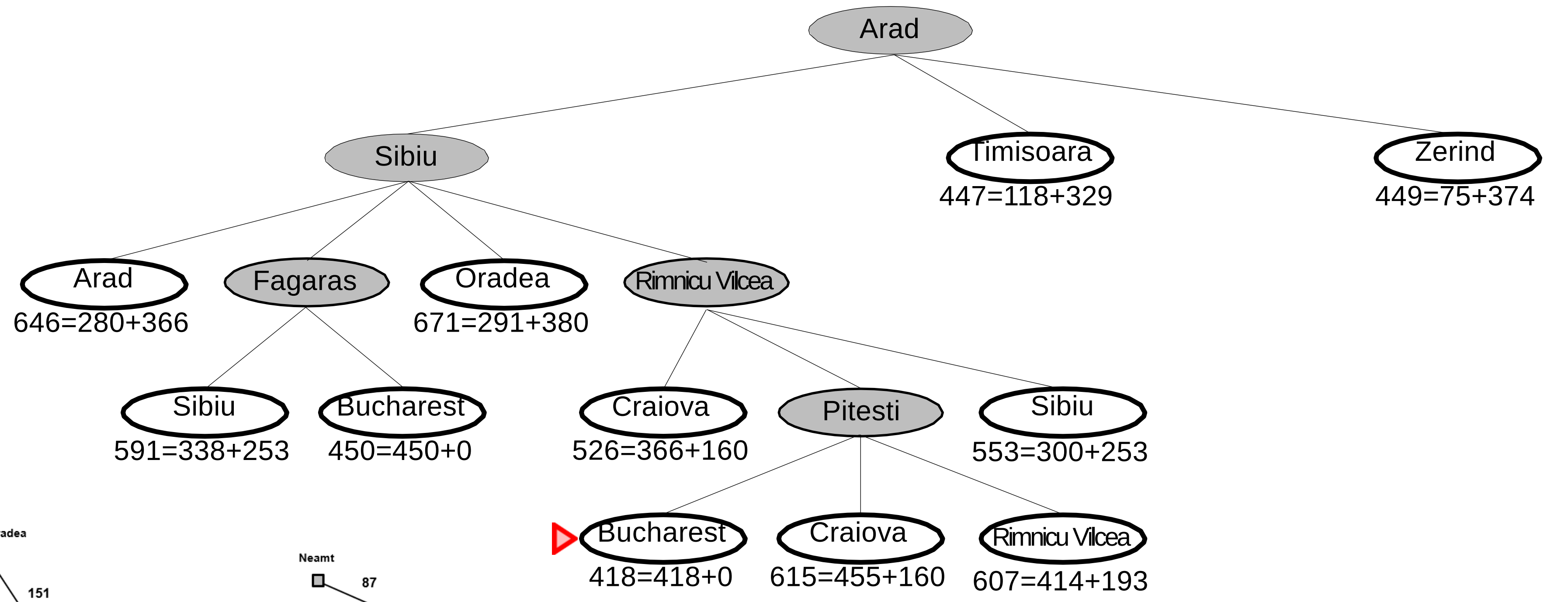


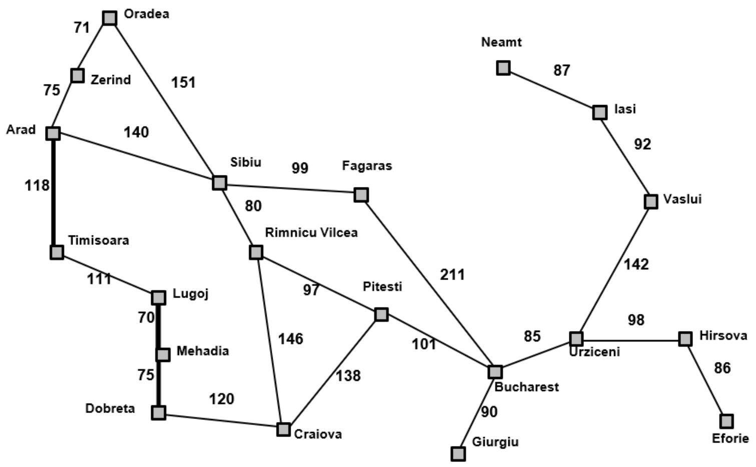


Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example





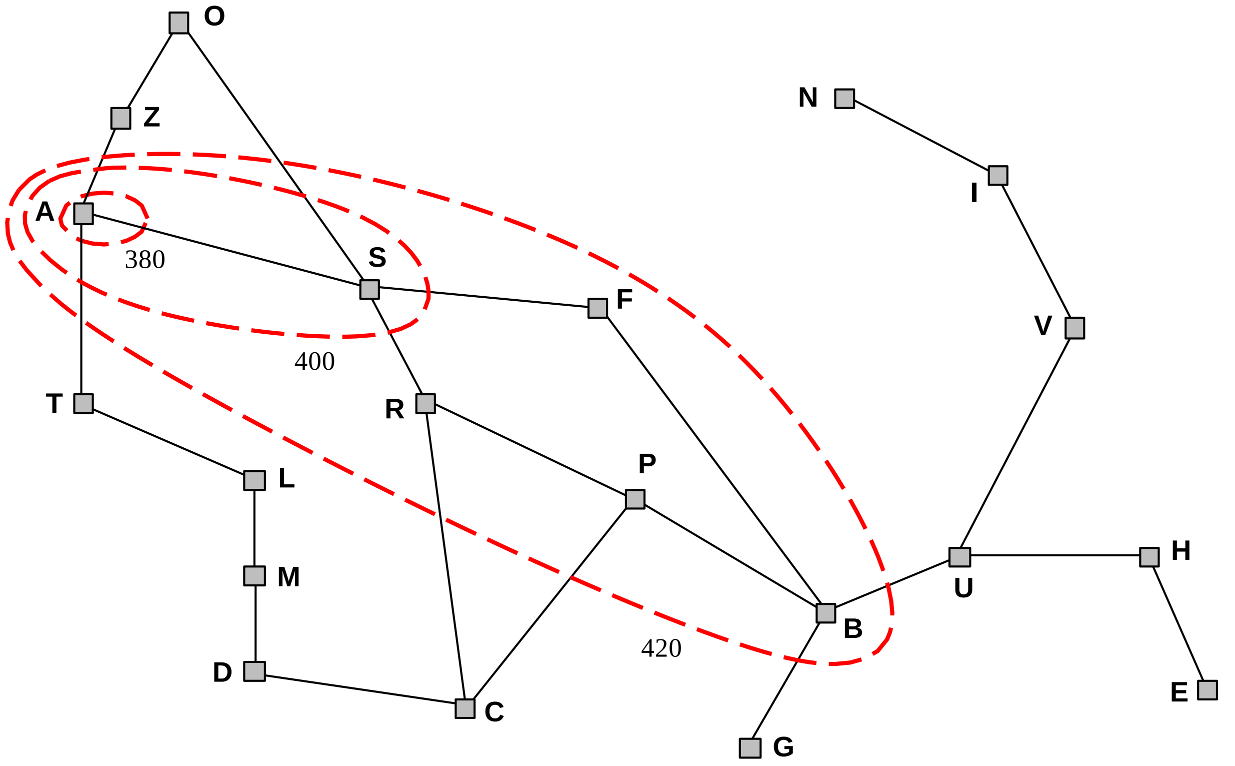
Straight−line distance
to Bucharest

Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Optimality of A*

A* expands nodes in order of increasing f value

Gradually adds "f-contours" of nodes (breadth-first adds layers)
Contour i has all nodes where $f_i < f_{i+1}$



Admissible heuristic

- A$*$ search uses an admissible heuristic, one that never overestimates the cost to reach goal.

- i.e., $h(n) \leq h^*(n)$ where $h^*(n)$ is the $\texttt{true}$ cost from n.
- (Also require $h(n) \geq 0$, so $h(G) = 0$ for any goal G.)

- E.g., $h_{\text{SLD}}(n)$ never overestimates the actual road distance

- Theorem: A$*$ search is optimal

Properties of A* algorithm

Complete: Yes, unless there are infinitely many nodes with $f \leq f(G)$

Time: Exponential in [relative error in $h \times$ length of soln.]

Space: Keeps all nodes in memory

Optimal: Yes—cannot expand f_{i+1} until f_i is finished A$*$

expands all nodes with $f(n) < C*$
A$*$ expands some nodes with $f(n) = C*$
A$*$ expands no nodes with $f(n) > C*$

Summary of informed search algorithms

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest h

– incomplete and not always optimal

A*search expands lowest $g + h$

– complete and optimal

– also optimally efficient (up to tie-breaks, for forward search)

A* is widely used

End

- Any questions?