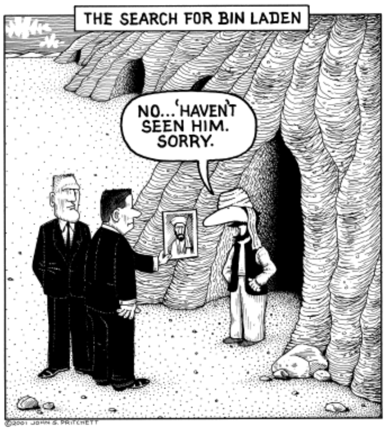
CSE 473

Lecture 4

Informed Search



Last Time

Blind (Uninformed) Search

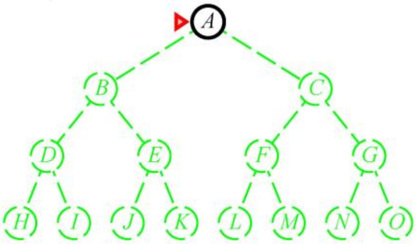
- Tree Search and Graph Search
- BFS
- UC-BFS
- DFS

Depth-first search

Expand deepest unexpanded node

Implementation:

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

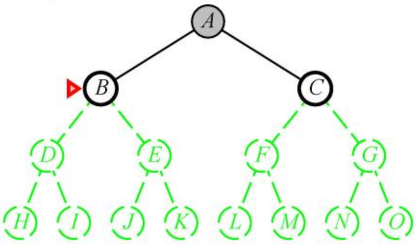


Depth-first search

Expand deepest unexpanded node

Implementation:

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

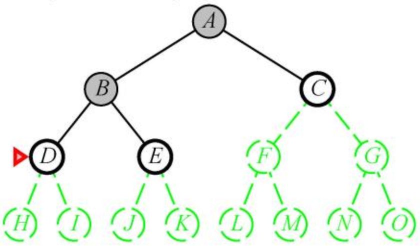


Depth-first search

Expand deepest unexpanded node

Implementation:

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

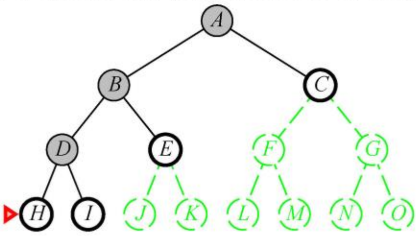


Depth-first search

Expand deepest unexpanded node

Implementation:

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

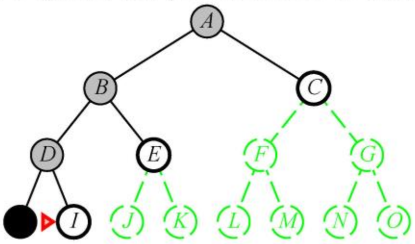


Depth-first search

Expand deepest unexpanded node

Implementation:

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

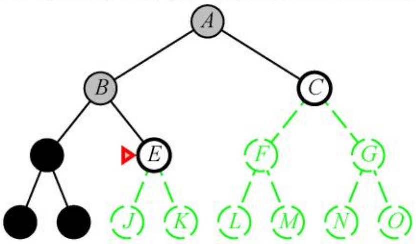


Depth-first search

Expand deepest unexpanded node

Implementation:

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

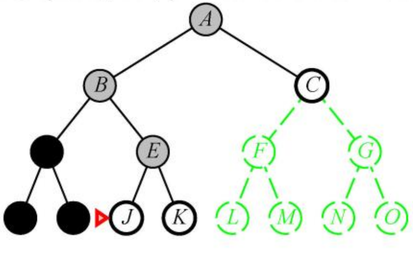


Depth-first search

Expand deepest unexpanded node

Implementation:

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

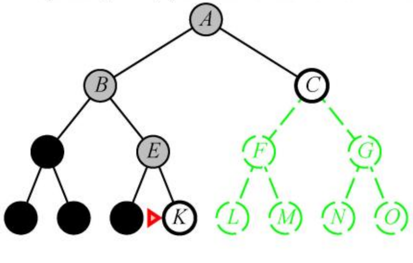


Depth-first search

Expand deepest unexpanded node

Implementation:

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

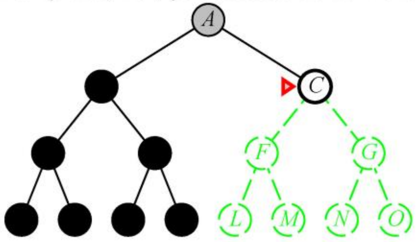


Depth-first search

Expand deepest unexpanded node

Implementation:

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

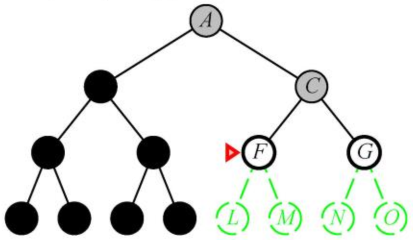


Depth-first search

Expand deepest unexpanded node

Implementation:

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

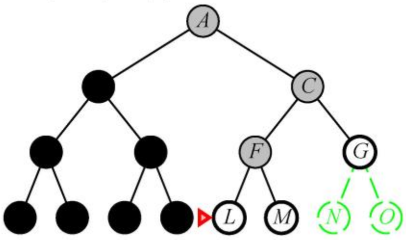


Depth-first search

Expand deepest unexpanded node

Implementation:

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

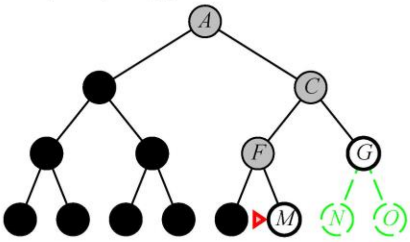


Depth-first search

Expand deepest unexpanded node

Implementation:

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



Properties of depth-first search

Complete??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path (using “explored” set)
⇒ complete in finite spaces

Time??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path (using “explored” set)
$\Rightarrow$ 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??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path (using “explored” set)
$\Rightarrow$ 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??

Properties of depth-first search

Complete?? No: fails in infinite-depth spaces, spaces with loops
Modify to avoid repeated states along path (using “explored” set)
⇒ 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

Space cost is a big advantage of DFS over BFS.
Example: b = 10 with 1000 Bytes/node
d = 16 ➔ 156 KB instead of 10 EB (1 billion GB)

Depth-limited search

= depth-first search with depth limit l,
i.e., nodes at depth l have no successors (can handle infinite state spaces)

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

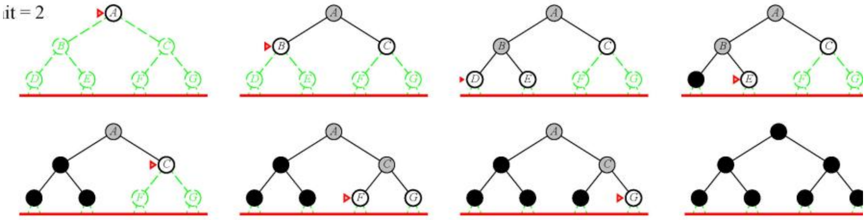
```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
    end
```

- **DFS with increasing depth limit**
- **Finds the best depth limit**
- **Combines the benefits of DFS and BFS**

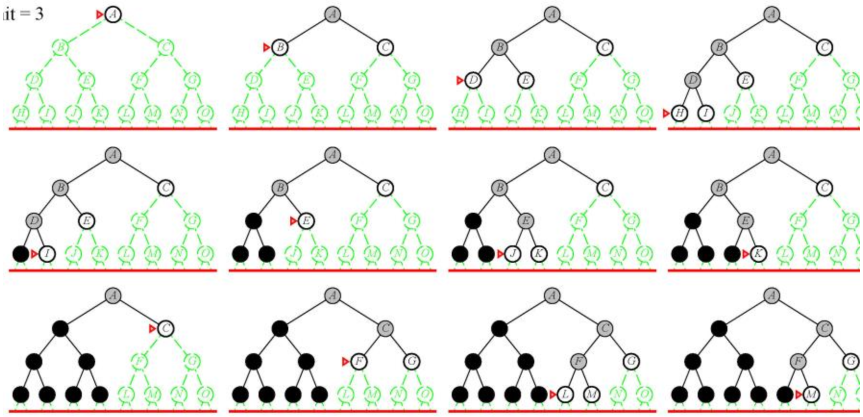
Iterative deepening search $l = 0$

Iterative deepening search $l = 1$

Iterative deepening search $l = 2$



Iterative deepening search $l = 3$



Properties of iterative deepening search

Complete??

Properties of iterative deepening search

Complete?? Yes

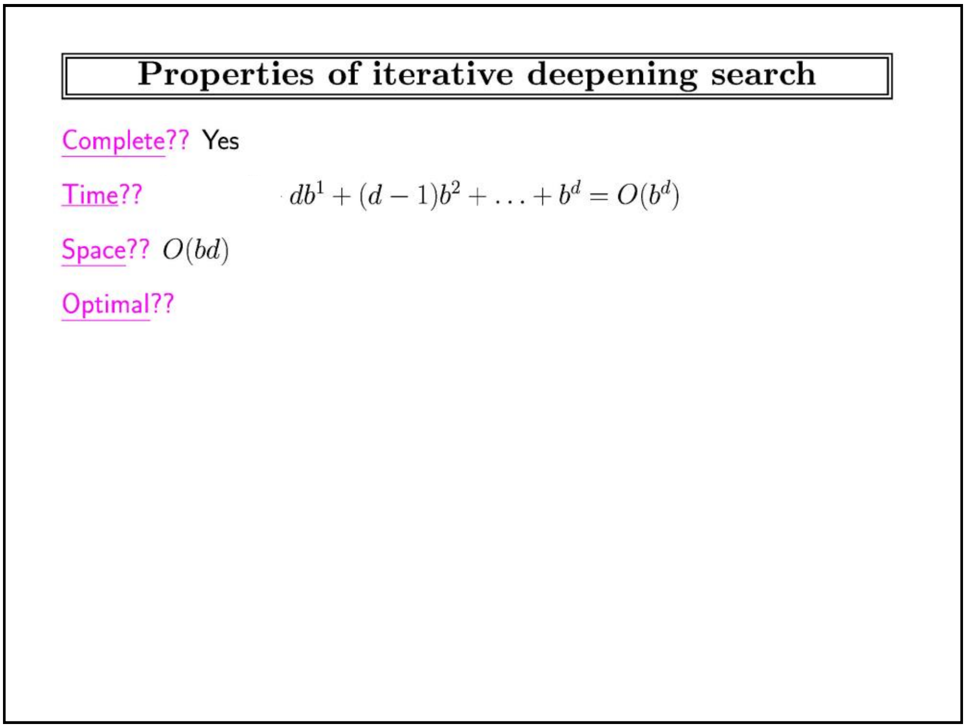
Time??

Properties of iterative deepening search

Complete?? Yes

Time?? $db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space??



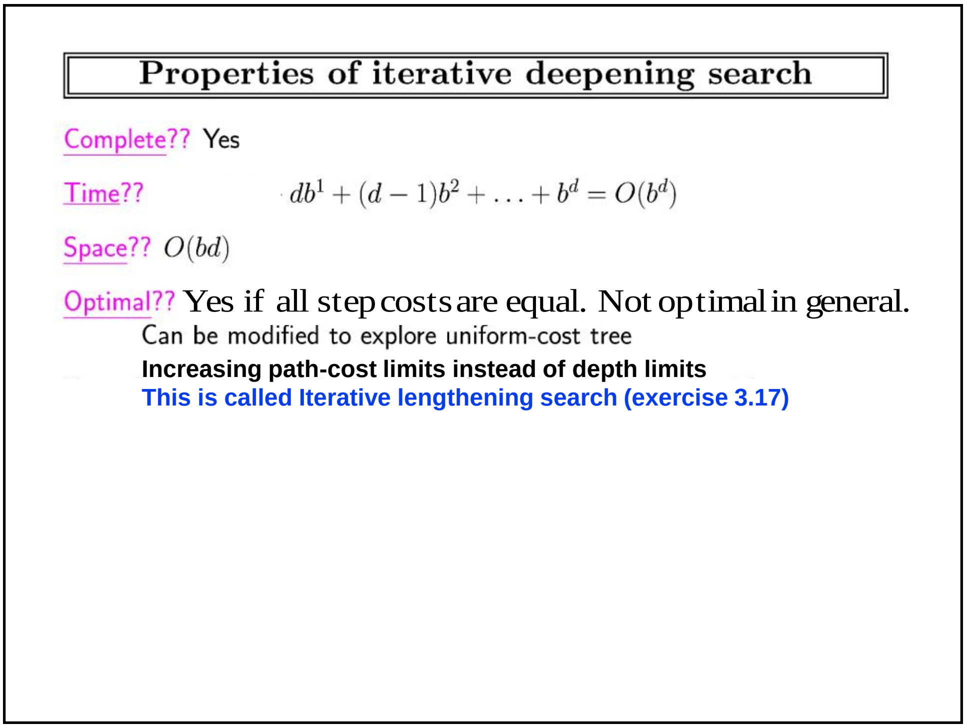
Properties of iterative deepening search

Complete?? Yes

Time?? $db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal??



Properties of iterative deepening search

Complete?? Yes

Time?? $db^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$

Space?? $O(bd)$

Optimal?? Yes if all step costs are equal. Not optimal in general.

Can be modified to explore uniform-cost tree

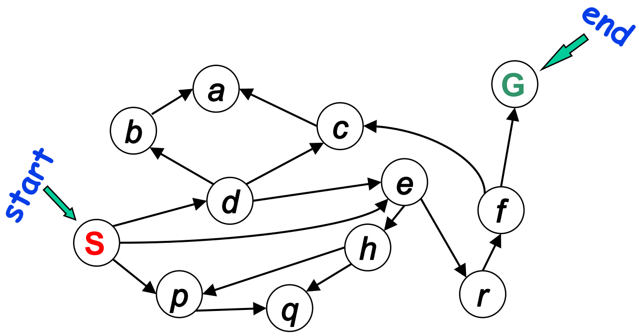
Increasing path-cost limits instead of depth limits

This is called Iterative lengthening search (exercise 3.17)

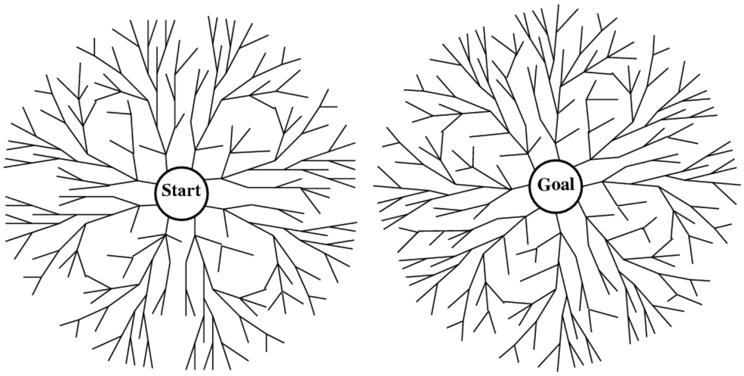
Summary of algorithms

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

Forwards *vs.* Backwards Search



Bidirectional Search



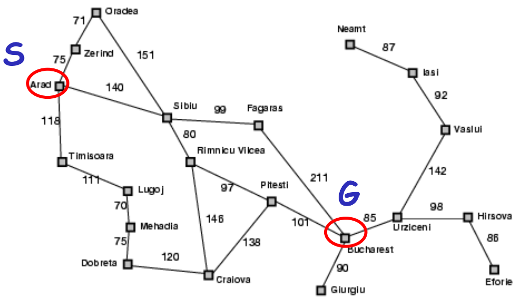
Motivation: Search time $b^{d/2} + b^{d/2} << b^d$
(E.g., $10^8+10^8 = 2\cdot 10^8 << 10^{16}$)
Can use breadth-first search or uniform-cost search
Hard for implicit goals e.g., goal = "checkmate" in chess

Can we do better?

Can we use problem-specific knowledge to speed up search and maintain optimality?

Informed Search

- General search problem: Actions have different costs



- Want to minimize *total cost* from start to goal

  Not just minimizing *path cost* like Uniform-cost search
- Idea: Use problem-specific knowledge to guide search by using “heuristic function”

Best-first Search

- Generalization of breadth first search
- Priority queue of nodes to be explored
- *Evaluation function f* (*n*) used for each node

```
Insert initial state into priority queue
While queue not empty
        Node = head(queue)
        If goal(node) then return node
        Insert children of node into pr. queue
```

Who's on (best) first?

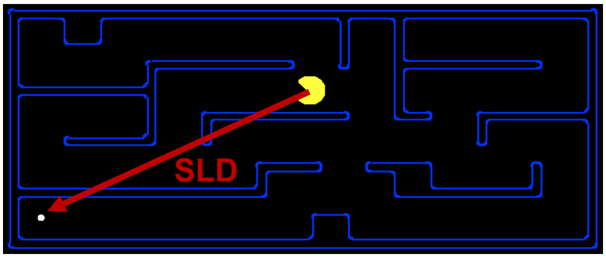
Examples of best-first search:

- Breadth-first search is best-first
  With f(n) = depth(n)

- Uniform-cost search is best-first
  With f(n) = g(n)
  where g(n) = path cost (sum of edge costs from start to n)

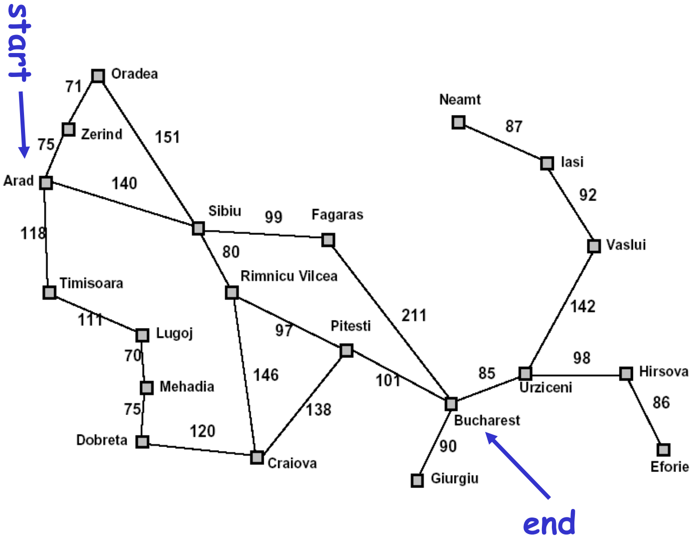
Greedy best-first search

- Use a *heuristic* evaluation function $f(n) = h(n)$ = estimate of cost from n to *goal*



- E.g., $h_{SLD}(n)$ = straight-line distance from n to destination
- Greedy best-first search expands the node that appears to be closest to goal

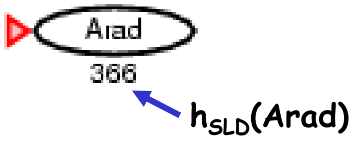
Example: Lost in Romania



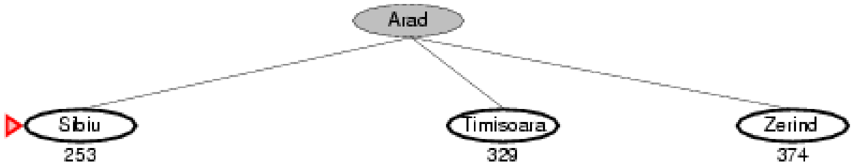
h(n)= SLD to Bucharest

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

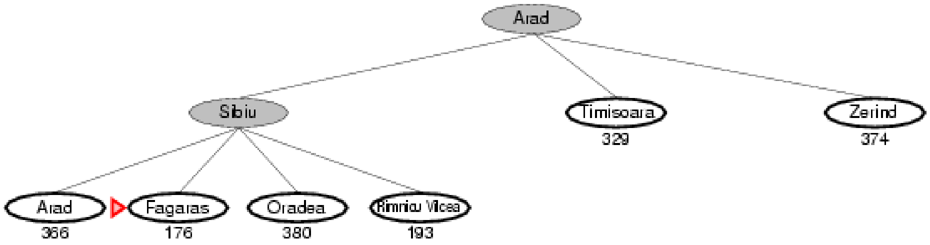
Example: Greedily Searching for Bucharest



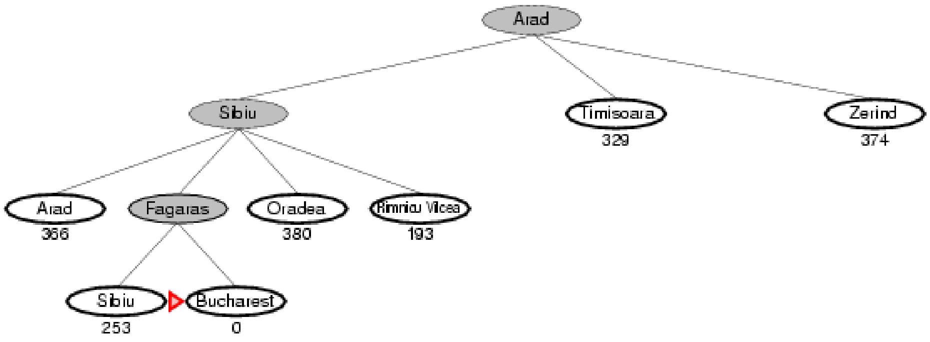
Example: Greedily Searching for Bucharest

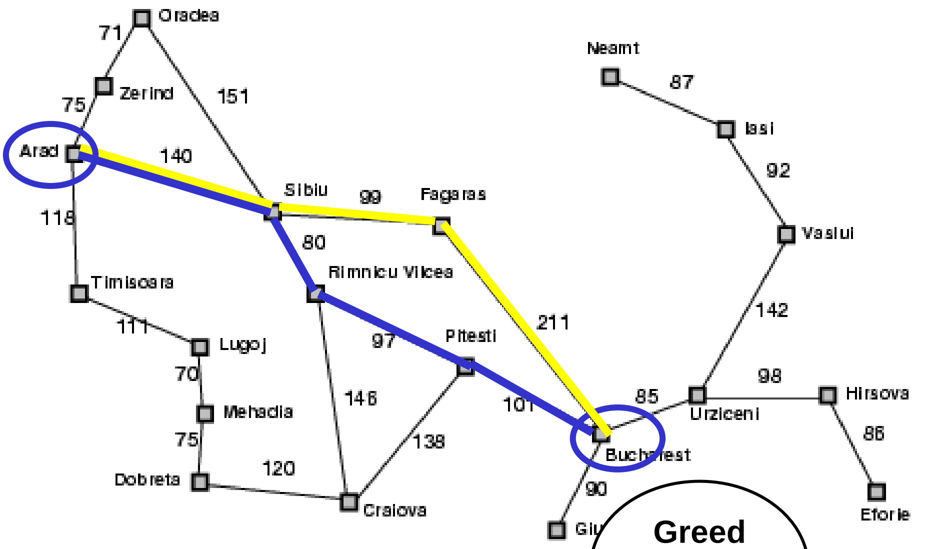


Example: Greedily Searching for Bucharest



Example: Greedily Searching for Bucharest





Yellow = greedy SLD-based search is NOT optimal!

Blue = optimal (418 versus 450)

Greed doesn’t pay!

Properties of Greedy Best-First Search

- Complete? No – can get stuck in loops (unless we keep an “explored” set)
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement
- Space? $O(b^m)$ (nodes in priority queue + explored set)
- Optimal? No, as our example illustrated

A* Search

(Hart, Nilsson & Rafael 1968)

Best first search with f(n) = g(n) + h(n)

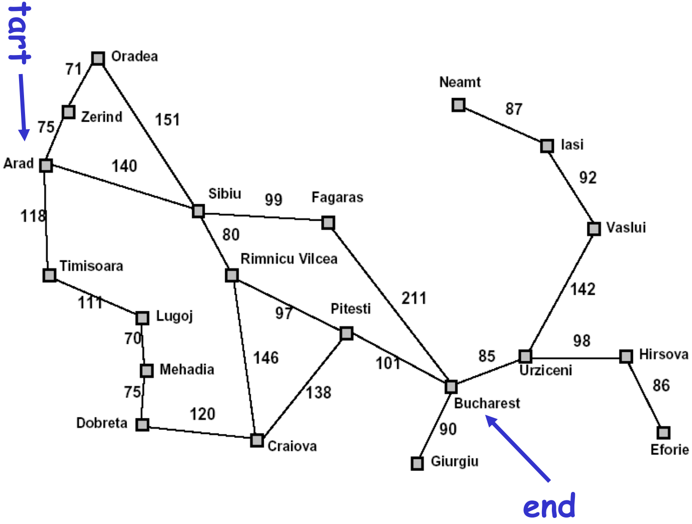
g(n) = sum of edge costs from start to n

heuristic function h(n) = estimate of lowest cost path from n to goal

If h(n) is “**admissible**” then tree-search will be optimal

Underestimates cost
of any solution which
can be reached from node
e.g., $h_{SLD}(n)$

Back in Romania Again



h(n)= SLD to Bucharest

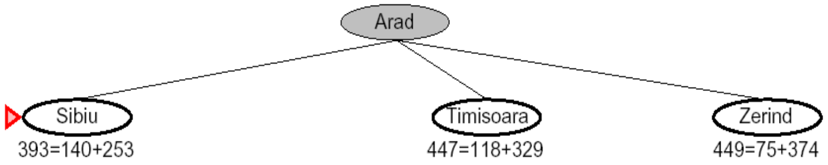
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	100
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Example

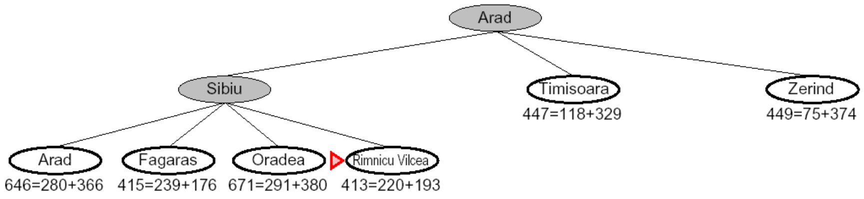
366=0+366

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

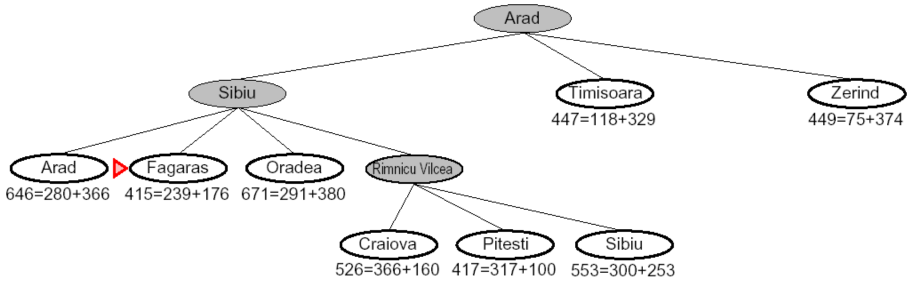
A* Example



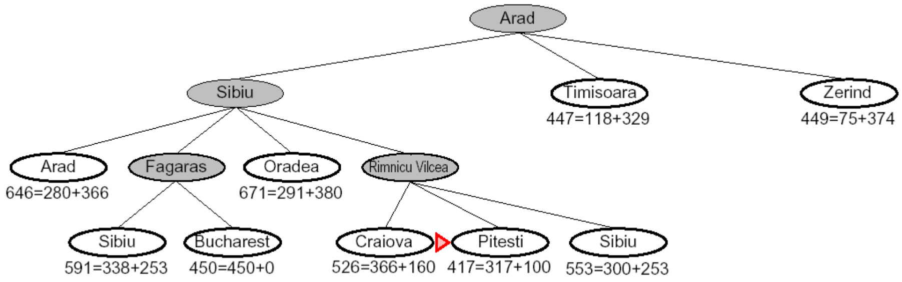
A* Example



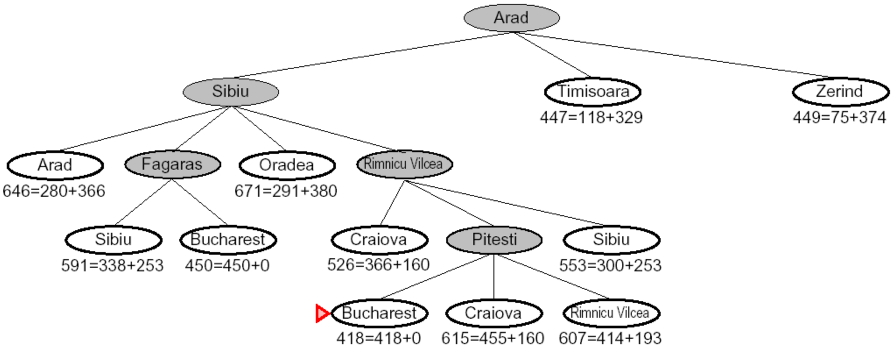
A* Example



A* Example



A* Example



Next Time

- More on A* and heuristic functions

To Do:

- Read Chapter 3
- Start Project #1