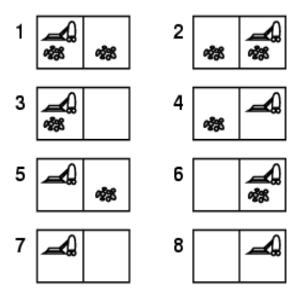
INFORMED SEARCH AND HEURISTICS

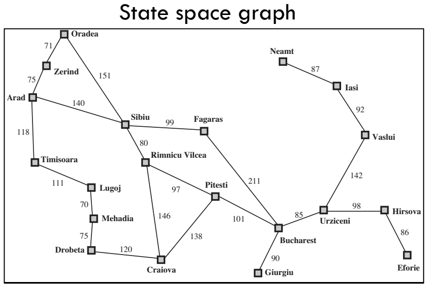
Recap: Uninformed Search

- We have an rational agent. But how does the agent actually achieve its goal?
- Search for a solution - a sequence of actions that leads from the initial state to the goal state
- Uninformed search algorithms
  - Uses no information beyond problem
  - Discrete environment
  - Offline exploration

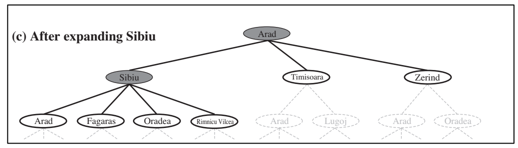


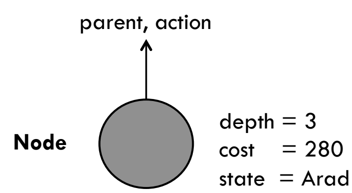
Recap: State space graph vs. Search tree

State space graph



Search tree





Recap: Tree Search

function TREE-SEARCH(*problem*, *strategy*) **returns** a solution or failure

  initialize the frontier using the initial state of *problem*

  loop do

    if the frontier is empty **return** failure

    choose leaf node according to *strategy* and remove from frontier

    if node contains goal state **return** solution

    expand chosen node and add resulting nodes to frontier

Recap: Graph-search

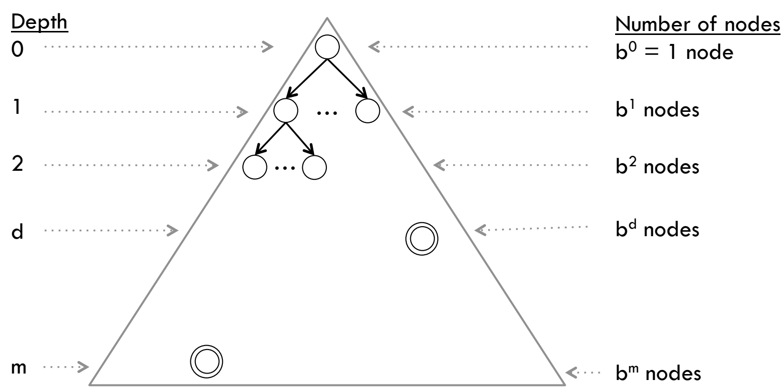
```
function GRAPH-SEARCH(problem, strategy) returns a solution or failure
    initialize the frontier using the initial state of problem
    initialize explored set to empty
    loop do
        if the frontier is empty return failure
        choose leaf node according to strategy and remove from frontier
        if node contains goal state return solution
        add node to explored set
        expand chosen node and add resulting nodes to frontier
        only if not in frontier or explored set
```

Recap: analyzing search algorithms

- Time (Big-O)
  - approximately the number of nodes processed (frontier plus explored lists)
- Space (Big-O)
  - the max # of nodes stored in memory at any time
- Complete (yes/no)
  - If a solution exists, will we find it?
- Optimal (yes/no)
  - If we return a solution, will it be the best/optimal solution, i.e. solution with lowest path cost

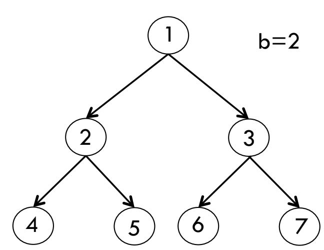
Recap: analyzing search algorithms

- When analyzing time and space, it is useful to define some notation:
  - b – branching factor, i.e. max number of successors of any node
  - d – depth of the shallowest goal node
  - m – maximum possible depth of search tree



Recap: Breadth-first search

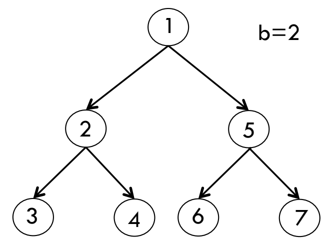
- Strategy: choose shallowest unexpanded node
- Implementation: FIFO queue
- Finds the shallowest goal node



- Time
  - Processes $1 + b + b^2 + \ldots + b^d = O(b^d)$ nodes
- Space
  - In the worst case, the goal node is the last node at depth d $= O(b^d)$
- Complete?
  - Yes, if b is finite
- Optimal?
  - Yes, if the path cost to a node is a non-decreasing function of the depth of the node
  - rule-of-thumb: use if step costs are constant

Recap: Depth-first search

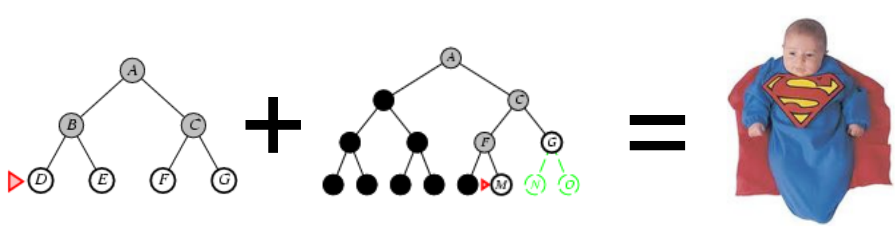
- Strategy: choose deepest unexpanded node
- Implementation: LIFO queue (a.k.a. stack)
- Finds the leftmost goal node



- Time
  - Processes $1 + b + b^2 + \ldots + b^m = O(b^m)$ nodes
- Space
  - Keep track of only O(bm) nodes
- Complete?
  - Yes, if the state space is finite and no loops
- Optimal?
  - No, finds the leftmost goal node

Improvements?

- Can we combined the optimality and completeness of BFS with the memory of DFS?



Recap: Depth limited DFS

- DFS, but with a depth limit **L** specified
  - Nodes at depth **L** are treated as if they have no successors
  - We only search down to depth **L**
- Time?
  - $O(b^L)$
- Space?
  - O(bL)
- Complete?
  - No, if solution is deeper than L
- Optimal
  - No, for same reasons DFS isn't

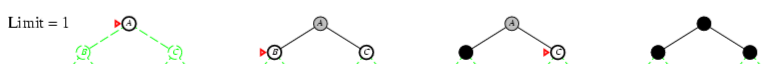
Recap: Iterative deepening search (IDS)

```
for L=0, 1, 2, …
    run depth-limited DFS
    if solution found return result
```

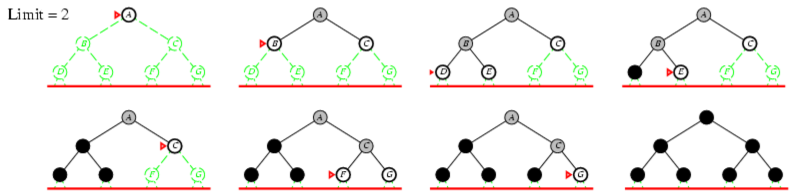
- Blends the benefits of BFS and DFS
  - similar to BFS, all nodes at depth L searched before L+1
  - but has the memory requirements of DFS
- Will find the solution when **L** is the depth of the shallowest goal

Iterative deepening search *L* =0

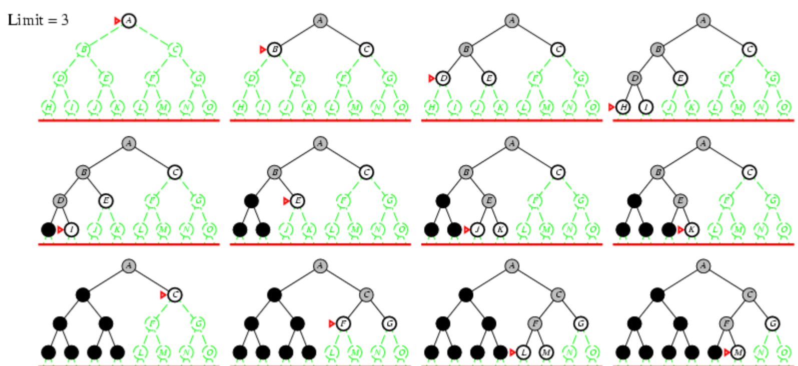
Iterative deepening search *L* =1



Iterative deepening search L =2



Iterative deepening search L =3



Time complexity for IDS

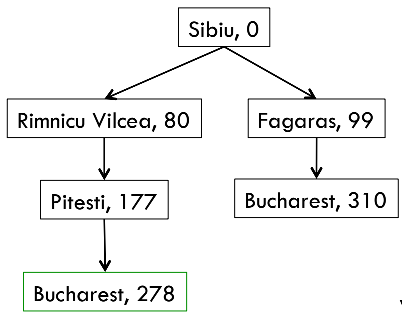
- $L = 0$: 1
- $L = 1$: $1 + b$
- $L = 2$: $1 + b + b^2$
- $L = 3$: $1 + b + b^2 + b^3$
- …
- $L = d$: $1 + b + b^2 + b^3 + \ldots + b^d$
- Overall:
  - $d(1) + (d-1)b + (d-2)b^2 + (d-3)b^3 + \ldots + b^d$
  - $O(b^d)$
  - the cost of the repeat of the lower levels is subsumed by the cost at the highest level

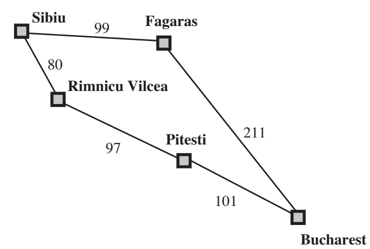
Analysis of IDS

- Time
  - $O(b^d)$
- Space
  - $O(bd)$
- Complete?
  - Yes
- Optimal?
  - Yes

Recap: Uniform-cost search

- Strategy: choose node with lowest path cost
- Implementation: priority queue





What solution would BFS have returned?

Summary of algorithms

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal?	Yes	Yes	No	No	Yes

C^* is the path cost of the optimal solution

ϵ is the minimum step cost

C^*/ϵ then is the average depth of the solution

Today

- Reading
  - AIMA Chapter 3
- Goals
  - Informed search algorithms
    - Greedy best-first search
    - A-star search
  - Characteristics of heuristics
  - Creating heuristics for problems

Informed search

- Use information beyond the problem to guide the search process to promising regions
- Define an evaluation function f(n) for each node n
  - estimates "desirability" of node
  - choose most desirable node from frontier (priority queue)
- Choices for f(n)
  - g(n) = distance from start node
  - h(n) = *estimate* of distance to goal node (heuristic function)

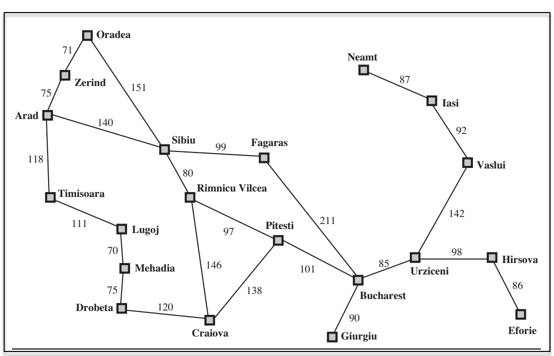
Informed search

- Recall for uninformed search
  - FIFO queue – BFS
  - LIFO queue – DFS
  - priority queue with g(n) (distance from start state) – UCS
- For informed search
  - priority queue with f(n) = g(n) – UCS
  - priority queue with f(n) = h(n) – Greedy best-first search
  - priority queue with f(n) = g(n) + h(n) – A* (A-star) search

g(n) = distance from start   h(n) = estimate to goal

Heuristic functions

- An heuristic function is an *estimate* of cost from n to the goal
- Example: straight-line distance



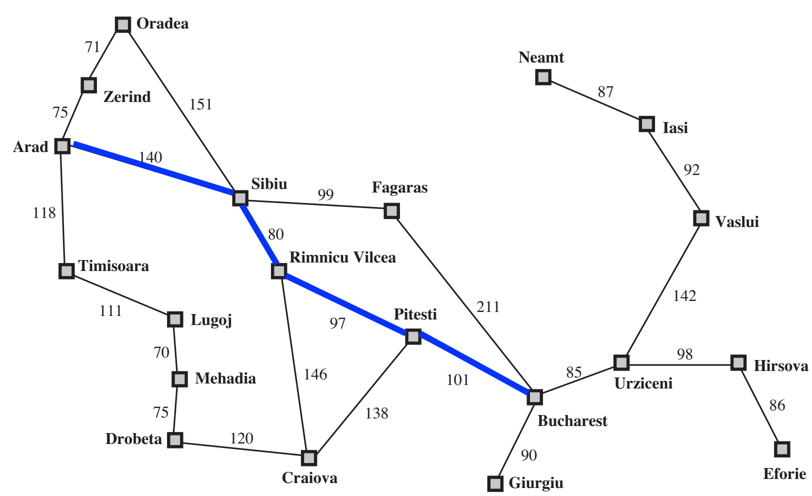
h(n) = straight-line distance

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 best-first search

- Define f(n) = h(n)
- Expand the node that seems closest to the goal
- What could possibly go wrong?

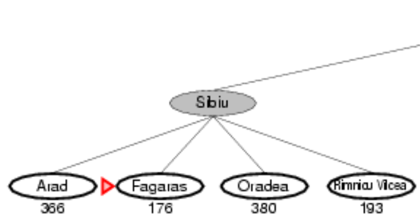
Path to Bucharest



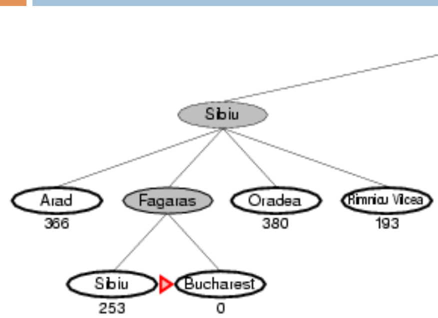
Greedy best-first search example

Greedy best-first search example

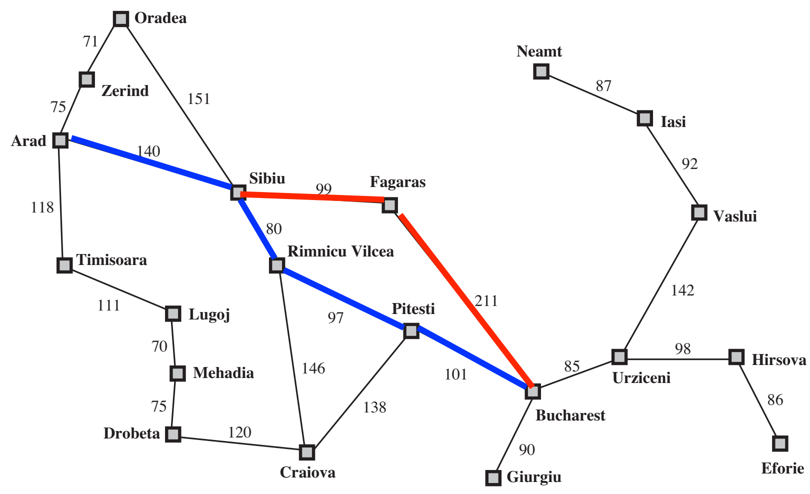
Greedy best-first search example



Greedy best-first search example



Path to Bucharest

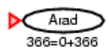


A* search

- Greedy best-first search considers only cost to goal
  - Leads confidently to the (wrong) solution
- Idea: avoid expanding paths that are *already* expensive
- Evaluation function $f(n) = g(n) + h(n)$

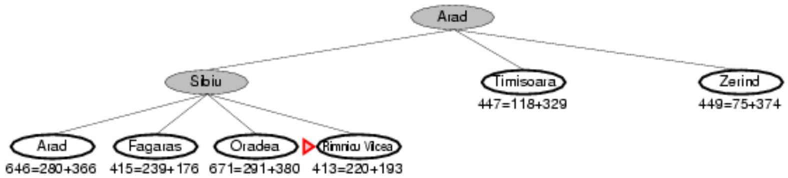
g(n) = distance from start   h(n) = estimate to goal

A* search example

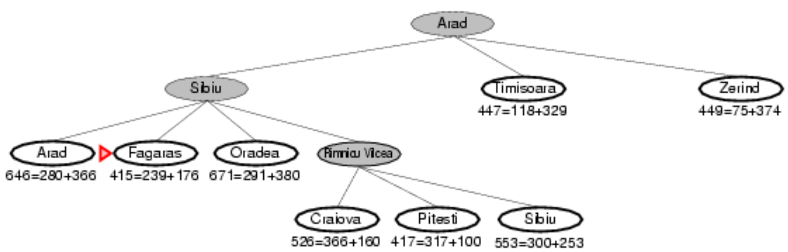


A* search example

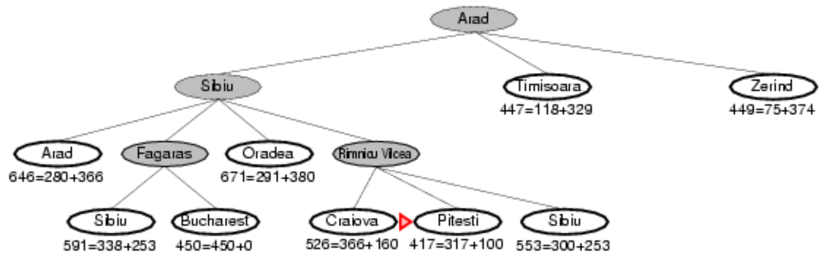
A* search example



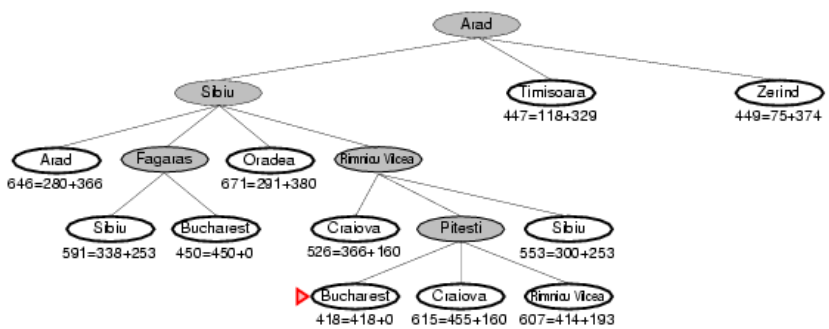
A* search example



A* search example



A* search example



A^* search: conditions for optimality

- A heuristic h(n) is admissable if it never *overestimates* the cost to the goal

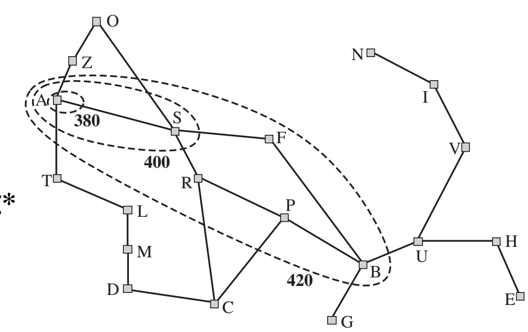
  $0 \leq h(n) \leq h^*(n)$ $h^*(n)$ is true cost to goal

- A heuristic h(n) is consistent if

  $h(n) \leq c(n, a, n') + h(n')$ n' is a successor

- Tree-search version of A^* is optimal if h(n) is admissable
- Graph-search version of A^* is optimal if h(n) is consistent

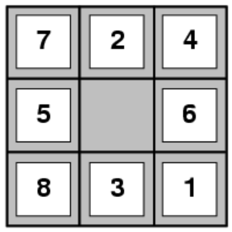
Properties of A^* search



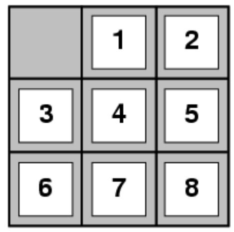
- A* expands
  - all nodes with f(n) < C*
  - some nodes with f(n) = C*
  - no nodes with f(n) > C*
- Optimally efficient
- Complete if finite number of nodes with f(n) ≤ C*

Creating admissable heuristic functions

- How do we construct a heuristic function that doesn't overestimate the cost to the goal?



Start State

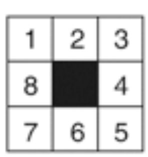


Goal State

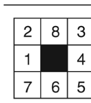
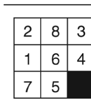
- What are some ideas for heuristic functions?

Creating admissable heuristic functions

- Two-well used heuristics:
  - h_1 = number of misplaced tiles
  - h_2 = sum of the distances of the tiles from goal positions (Manhattan distance)



Goal

	5	6
	3	4
	5	6
	Tiles out of place	Sum of distances out of place

Why are these admissable?

Creating admissable heuristic functions

- Often admissable heuristics are solutions to relaxed problems with fewer restrictions
- A tile can move from square A to square B if
  - A is horizontally or vertically adjacent to B
  - B is blank
- Pick up tiles and place in the correct spot
  - Induces h_1 heuristic, i.e. number tiles out of place

	Number nodes expanded for solution depth d		
	d = 4	d = 8	d= 12
IDS	112	6384	3.6 million
$A^*(h_1)$	13	39	227

Creating admissable heuristic functions

- Often admissable heuristics are solutions to relaxed problems with fewer restrictions
- A tile can move from square A to square B if
  - ~~A is horizontally or vertically adjacent to B~~
  - ~~B is blank~~

  ← relax the rules
- Induces h_1 heuristic, i.e. number of tiles out of place
  - Allows you to pick up the tiles and place in the correct spot

	Number nodes expanded for solution depth d		
	d = 4	d = 8	d= 12
IDS	112	6384	3.6 million
$A^*(h_1)$	13	39	227

Creating admissable heuristic functions

- Often admissable heuristics are solutions to relaxed problems with fewer restrictions
- A tile can move from square A to square B if
  - A is horizontally or vertically adjacent to B
  - ~~B is blank~~

relax the rules

- Induces h_2 heuristic, i.e. sum of distances to goal position
  - Allows you to move a tile to an adjacent square

	Number nodes expanded for solution depth d		
	d = 4	d = 8	d= 12
IDS	112	6384	3.6 million
$A^*(h_1)$	13	39	227
$A^*(h_2)$	12	25	73

Creating admissable heuristic functions

	Number nodes expanded for solution depth d		
	d = 4	d = 8	d= 12
IDS	112	6384	3.6 million
$A^*(h_1)$	13	39	227
$A^*(h_2)$	12	25	73

expands fewer nodes

Creating admissable heuristic functions

- Some heuristics are better than others
  - If $h_1(n) \leq h_2(n) \leq h^*(n)$ then h2 dominates h1
  - Manhattan distance dominates tiles out of place
  - A-star search using h_2 will never expand more nodes than A-star search using h_1
  - Can combine admissable heuristics using max

Informed search summary

- Uniform-cost search considers only the cost from the start node
- Greedy best-first search considers only the (estimate of the) cost to the goal node
  - Confidently heads straight to the (wrong) solution
- A^* search considers both cost from start and estimate to goal
  - A^* is optimal with admissable/consistent heuristic
- A good heuristic is the key
  - Consider solutions to relaxed problems