UNINFORMED AND INFORMED SEARCH

Step One: Formulate the search problem

A well-defined search problem includes:

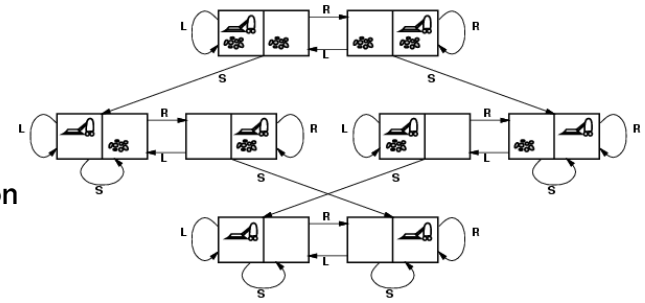
- states
- initial state
- actions
- successor function

Induce the state space graph

- goal test
- path cost (reflects performance measure)

Step One: Vacuum world

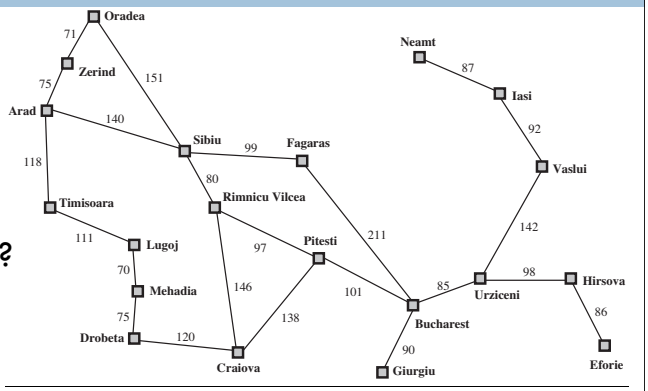
- states?
- initial state?
- actions?
- successor function
- goal test?
- path cost?



There are 8 states: all possible configuration of dirt and position of the vacuum. There are four actions: Left, Right, NoOp, and Suck. The successor function is just the resulting state after taking the action. The goal state is no dirt and the vacuum in either A or B. The path cost is 1 per action.

Step One: Path to Bucharest

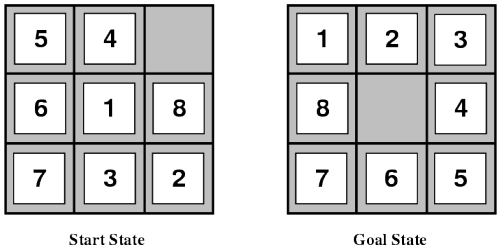
- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space graph look like?



states are the cities. Initial state is Arad. Actions are to take a road leading out of the current city. Successor function is the destination city. The goal test is Bucharest. The path cost might be 1 for each action or it could be the distance btw. two cities.

Step One: 8-puzzle

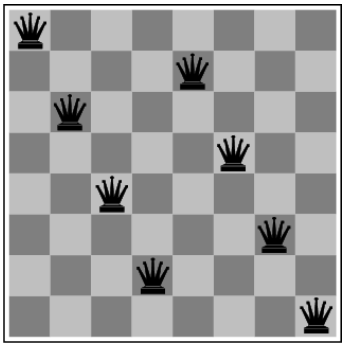
- states?
- initial state?
- actions?
- successor function?
- goal test?
- path cost?
- What does the state space look like?



states – all possible configurations of the 8 tiles and the blank space
actions – move the blank space UP, DOWN, LEFT, RIGHT
path cost – a cost of 1 per action

Step One: 8-queens puzzle

- states?
- initial state?
- actions?
- goal test?
- path cost?
- What does the state space look like?



incremental formulation: Initial state is a blank board. An action is to place a queen in the leftmost empty column (such that it is not in conflict with any previously placed queens)
Complete-state formulation: Initial state is 8 queens on the board. An action is to move a queen.
Note the path cost is irrelevant. We care only about the final configuration.

Step Two: Search

- Basic Idea
  - Pick a node
  - If not goal state
    - expand node by generating all its successors
    - mark node as explored
  - Repeat till goal found
- Necessary data structures
  - frontier - nodes that were generated but not yet expanded
  - (explored - nodes that have been expanded)

Graph-search

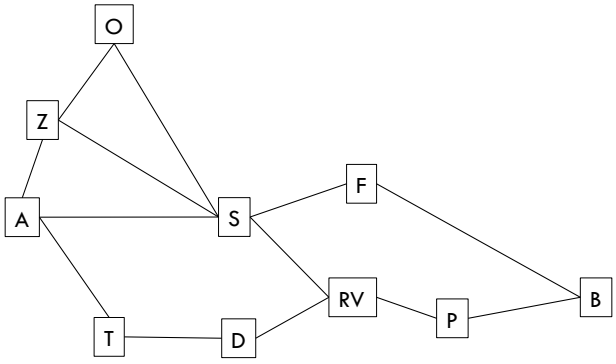
```
function GRAPH-SEARCH(problem, strategy) returns a solution or failure
        initialize the frontier using the initial state of problem
        initialize explored set to empty
        loop do
                if the frontier is empty return failure
                choose leaf node according to 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
```

Search Strategies

A search strategy *specifies the order in which nodes are selected from the frontier* to be expanded

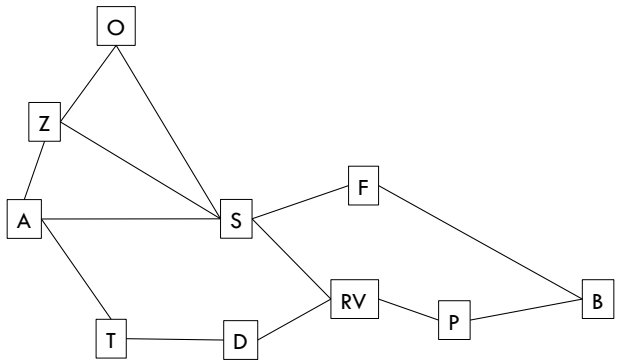
Breadth-first search (BFS)

- Expand shallowest unexpanded node
- Implementation: *frontier* is a FIFO queue



Depth-first search (DFS)

- Expand deepest unexpanded node
- Implementation: *frontier* is a LIFO queue (stack)



Evaluating search algorithm

- Time (Big-O)
  - approximately the number of nodes generated (frontier plus explored list)
- 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

Today

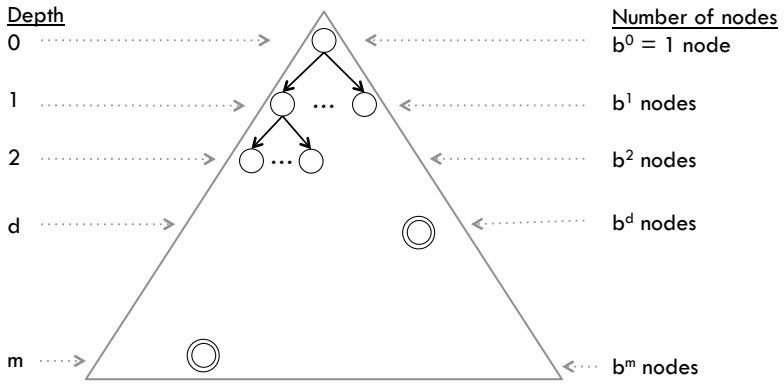
- Reading
  - AIMA Section 3.1-3.4 (uninformed search)
  - AIMA 3.5-3.6 (informed search)
- Objectives
  - Analyzing the complexity of search algorithms
  - UCS, DL-DFS, ID-DFS
  - Start on informed search algorithms

Evaluating search algorithm

- Time (Big-O)
  - approximately the number of nodes generated (frontier plus explored list)
- 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

Notation

- b – branching factor, i.e. max number of successors of any node
- d – depth of the shallowest goal node
- m – maximum length of any path in state space



Analyzing BFS

- Time: $O(b^d)$
- Space: $O(b^d)$
- Complete = YES if branching factor is finite
- Optimal = YES if path cost is non-decreasing function of depth of the node
- (Use when step costs are constant)

Analyzing DFS

- Time (for Tree-Search): $O(b^m)$
- Space (for Tree-Search): O(bm)
- Complete = YES, if space is finite (and no circular paths), NO otherwise
- Optimal = NO

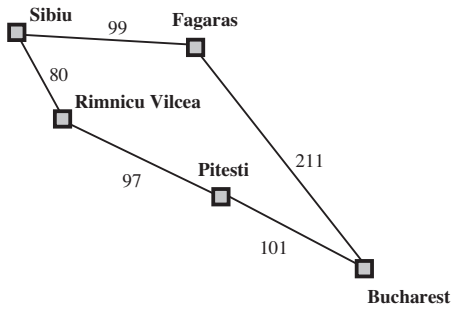
Time and memory requirements for BFS

Depth	Nodes	Time	Memory
2	1100	.11 sec	1 MB
4	111,100	11 sec	106 MB
6	10^7	19 min	10 GB
8	10^9	31 hours	1 terabyte
10	10^{11}	129 days	101 terabytes
12	10^{13}	35 years	10 petabytes
14	10^{15}	3,523 years	1 exabyte

BFS with b=10; 10,000 nodes/sec; 10 bytes/node

Uniform-cost search

- Expand node with lowest path cost
- Implementation:
  - *frontier* is a priority queue ordered by path cost



Analyzing Uniform-cost search

- Let C* be the cost of the optimal solution and ε be the minimum step cost
- Time: $O(b^{C^*/\varepsilon})$
- Space: $O(b^{C^*/\varepsilon})$
- Complete = YES if step cost exceeds epsilon
- Optimal = YES

Depth limited DFS

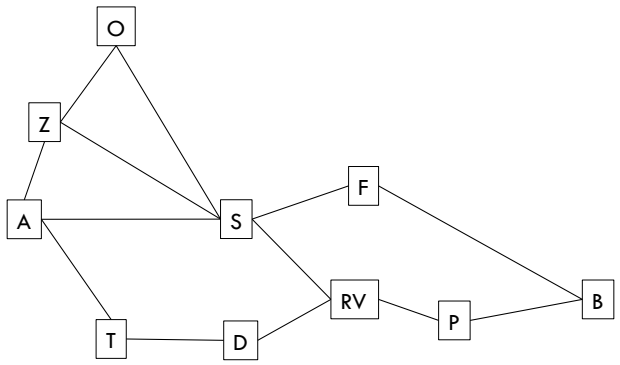
- DFS, but with a depth limit **L** specified
  - Nodes at depth **L** are treated as if they have no successors
  - We only search down to depth **L**
- Time?
  - $O(b^L)$
- Space?
  - O(bL)
- Complete?
  - No, if solution is longer than L
- Optimal
  - No, for same reasons DFS isn't

Iterative deepening search (IDS)

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

- Blends the benefits of BFS and DFS
  - searches in a similar order to BFS
  - but has the memory requirements of DFS
- Will find the solution when **L** is the depth of the shallowest goal

Iterative deepening search (IDS)



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

Analysis of IDS

- Space
  - O(bd)
- Complete?
  - Yes
- Optimal?
  - Yes

Summary of Uninformed Search

- Step One: Formulate the search problem
- Step Two: Search
  - Breadth-first search (queue)
  - Depth-first search (stack)
  - Uniform cost search (priority queue)
  - Iterative-deepening DFS (ID-DFS)
- Analyze search algorithms
  - Time, Space, Completeness, Optimality

Informed search (best-first search)

- Intuition: 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 to node n
  - h(n) = *estimate* of distance to goal node (heuristic function)

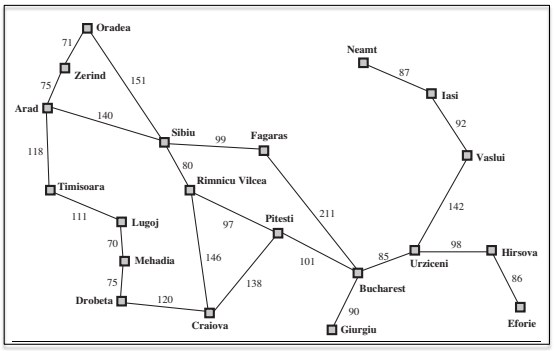
Informed search (best-first search)

- Uninformed search
  - BFS: FIFO queue
  - DFS: LIFO queue
  - UCS: priority queue with f(n) = g(n)
- Informed search
  - Greedy Best-First: priority queue with f(n) = h(n)
  - A*: priority queue with f(n) = g(n) + h(n)

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

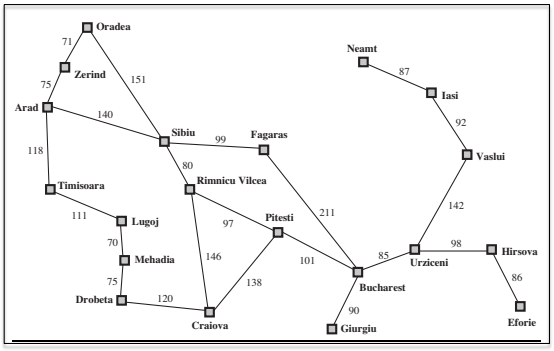
Heuristic functions

- An heuristic function h(n) estimates the cost from n to the goal



Heuristic functions

- An heuristic function h(n) estimates the cost from n to the goal
  - Example: straight-line distance



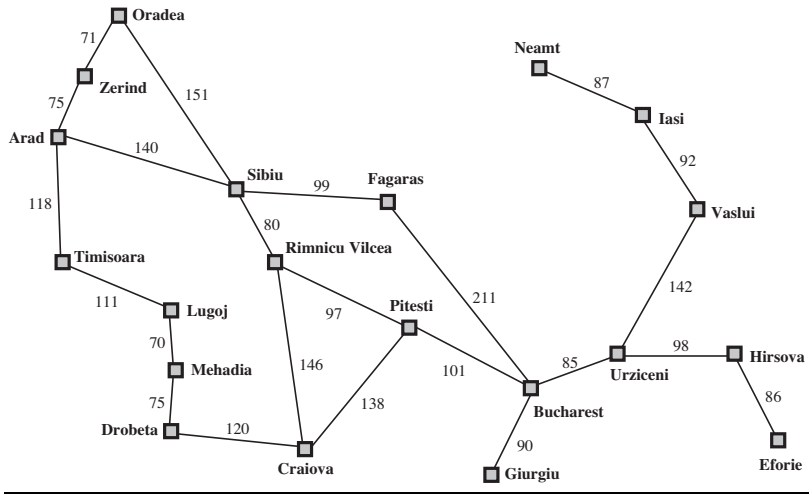
h(n) = straight-line distance

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

A* search

- Developed in 1968 by Hart et al. as a principled framework for using heuristic information to find minimum cost paths
- Evaluation function f(n) is the distance from the start g(n) plus the estimate to goal h(n)

A* search example



A* search: conditions for optimality

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

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

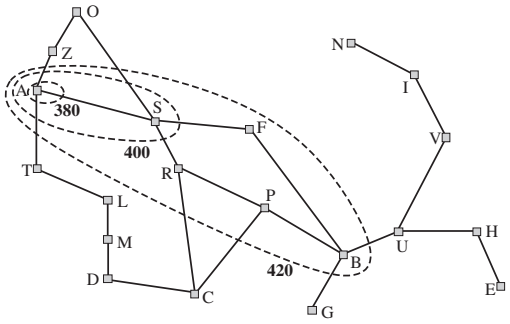
- A heuristic h(n) is consistent if

  $h(n) \le c(n, a, n') + h(n')$ 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

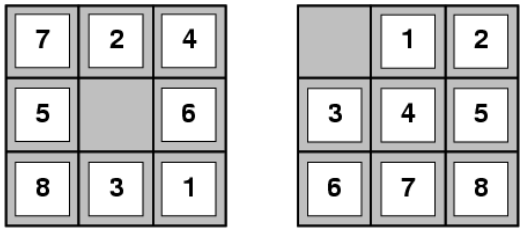
Properties of A* search

- 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

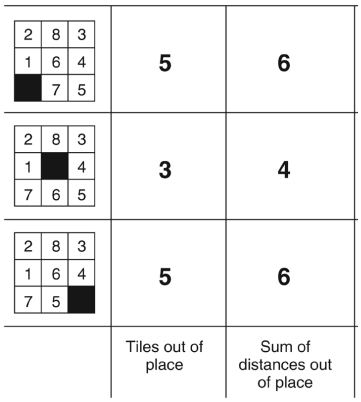
- How do we construct a heuristic function that doesn't overestimate the cost to the goal?



- 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)



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

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

expands fewer nodes

	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

- 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

- Uses additional information to guide search process
  - UCS – cost from start node
  - Greedy best-first – estimate of cost to goal
  - A^* uses both cost from start + estimate to goal
  - A^* is optimal with admissable/consistent heuristic
- A good heuristic is the key!
  - Consider solutions to relaxed problems