INFORMED SEARCH STRATEGIES

- The intelligence of a system cannot be measured in terms of search capacity, but in the ability to use knowledge about the problem to reduce/mitigate the combinatorial explosion.

- If the system had some control on the order in which possible solutions are generated, then it would be useful to use this order so that solutions have a high chance to appear before.

- Intelligence, for a system with limited processing capacity is the wise choice of what to do next ..... ".

INFORMED STRATEGIES

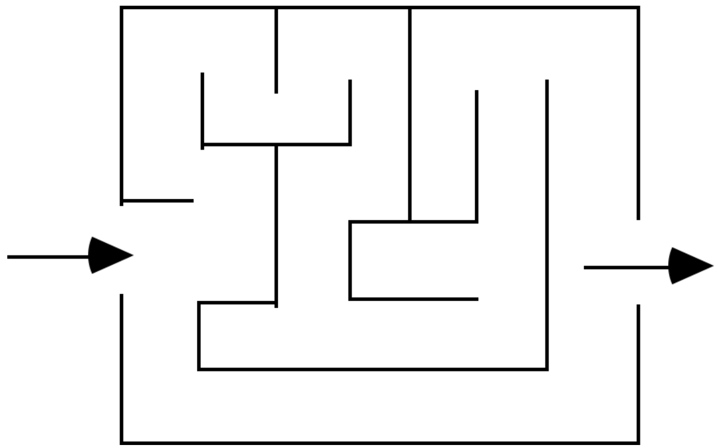
- **Newell-Simon**:
  - Uninformed search methods in a search space of depth d and branching factor b have space complexity proportional b^d to find a goal in one of the leaves.
  - This is unacceptable for certain complex problems. So we can resort to expanding the nodes using heuristic domain knowledge (evaluation functions) to decide which node to expand first.

INFORMED STRATEGIES

- Evaluation functions **give a computational estimate of the effort** to reach the final state.
- The time spent to evaluate a node by means of a heuristic function should correspond to a reduction in the size of the space explored.
  - Trade-off between the time it takes to solve the problem (baseline) and time spent in reasoning on top of it (meta-level).
  - Uninformed search strategies have no meta-level activities.

- PROBLEMS:
  - How to find the correct evaluation functions, ie how to select the most "promising nodes"?
  - It is difficult to numerically characterize the empirical knowledge about the problem.
  - Not always the obvious choice is the best.

EXAMPLE



- Heuristic Choice: always move to reduce the distance from the exit.
- In the case of this maze we would choose a longer way.

EXAMPLE: GAME OF 8

- The distance from the goal can be estimated based on the number of boxes out of place:

state

1	2	3
7	8	4
6		5

goal

1	2	3
8		4
7	6	5

  – Left: distance 2
  – Right: distance 4
  – High: distance 3

- It might be that the heuristics does not estimate the exact distance from the solution

1	2	3
7		4
6	8	5

Distance = 3 but 4 moves are needed to reach a solution

BEST FIRST SEARCH

Best first search uses ***evaluation functions*** that compute a number that represents the desirability of the node expansion.

Best-first means that the chosen node is the one considered as the most desirable.

QueuingFn = Insert successors in descending order of desirability.

Special cases:

- ***Greedy search or hill climbing***
- ***A* search***

BEST-FIRST SEARCH

- BEST-FIRST: Try to move to the maximum (resp. minimum) of a function that "estimates" the desirability (resp. cost) to reach the goal.

- Greedy:
  1. Let L be a list of the initial nodes of the problem, ranked according to their distance from the goal (ascending order);
  2. If L is empty fail; otherwise let n be the first node of L.
  3. If n is the goal return solution (the path to reach the goal)
  4. Otherwise remove n from L and add to L all the children of n. Then order the entire list L based on an estimate of the relative distance from the goal. Return to step 2

BEST-FIRST SEARCH

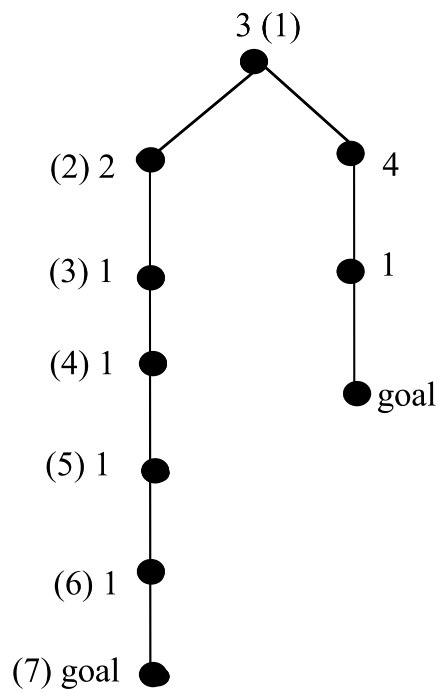
function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
  inputs: *problem*, a problem
    Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

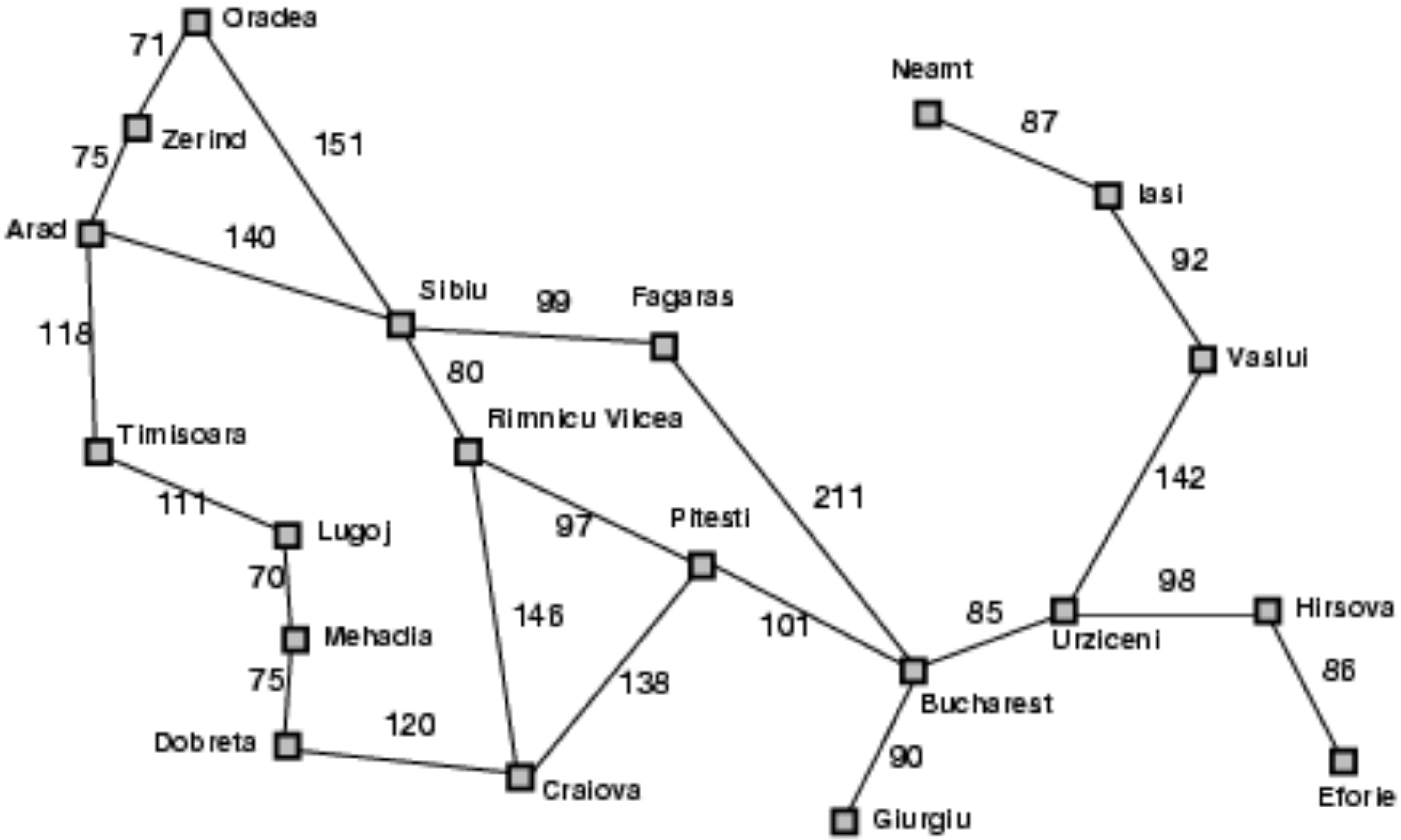
It uses the general search algorithm and **evaluation function *EvalFn***

BEST FIRST SEARCH

- The best-first search is not optimal in the sense that it is not guaranteed to find the best solution, or **the best path** toward a solution (remember the maze?).
- This is because the best-first technique tries to find as soon as possible a node with 0 distance from the goal and not the node with the lowest depth.



EXAMPLE



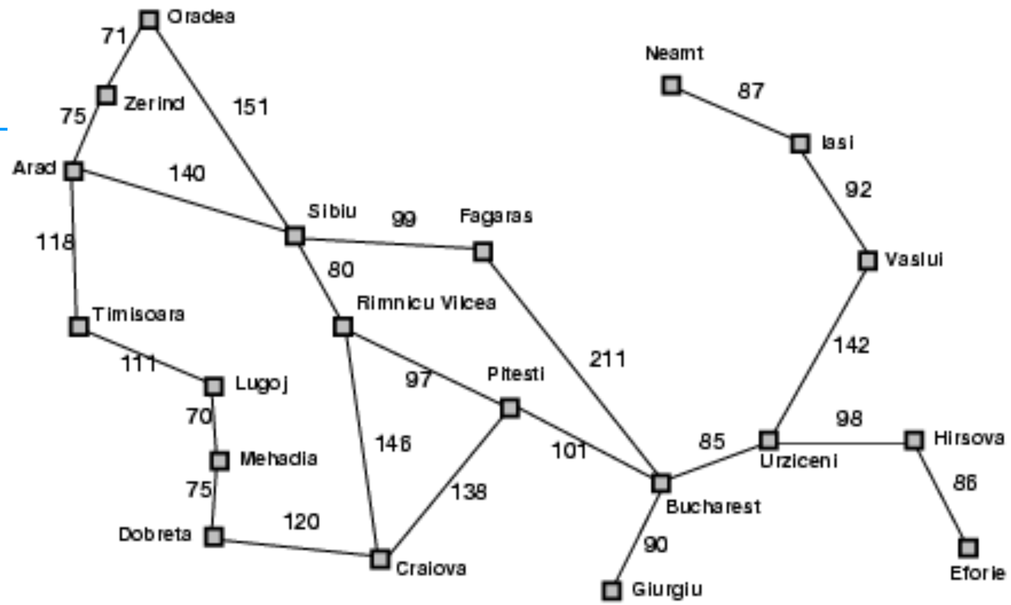
Straight–line distance
to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

BEST FIRST SEARCH

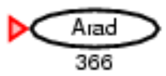
- Evaluation Function $f(n) = h(n)$ (heuristic)
- h(n) = Estimate of the cost from *n* to the *goal*
- eg, $h_{SLD}(N)$ = Straight-line distance between *n* and Bucharest
- The greedy best-first search expands the node that seems closer to the goal

EXAMPLE

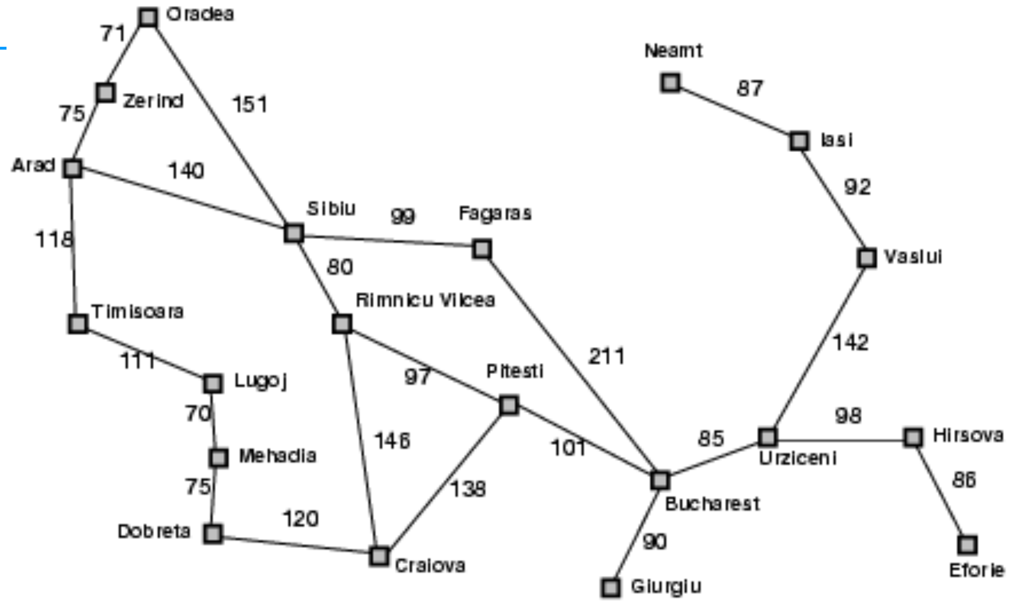


Straight–line distance to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



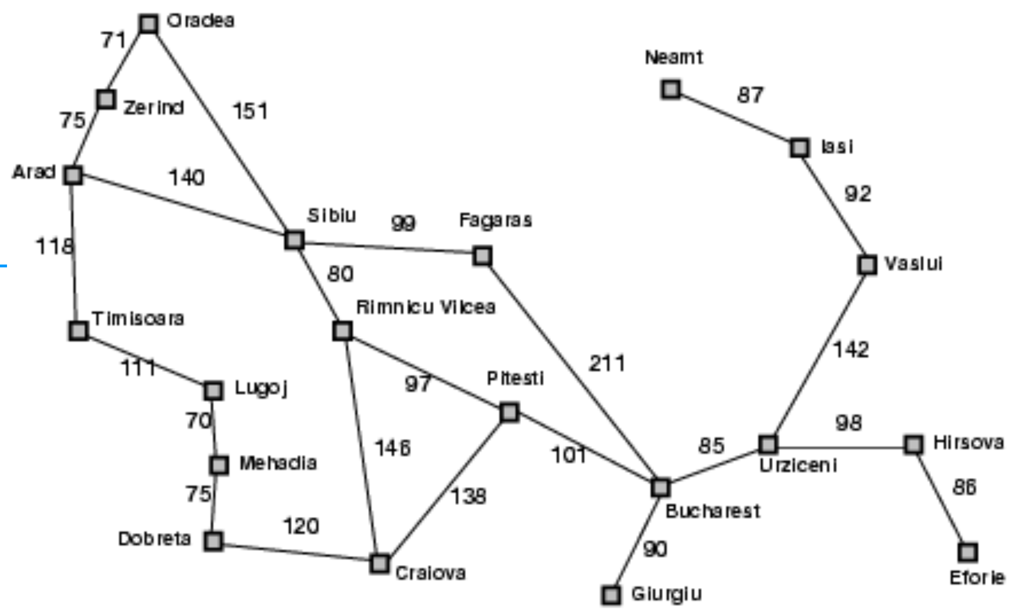
EXAMPLE



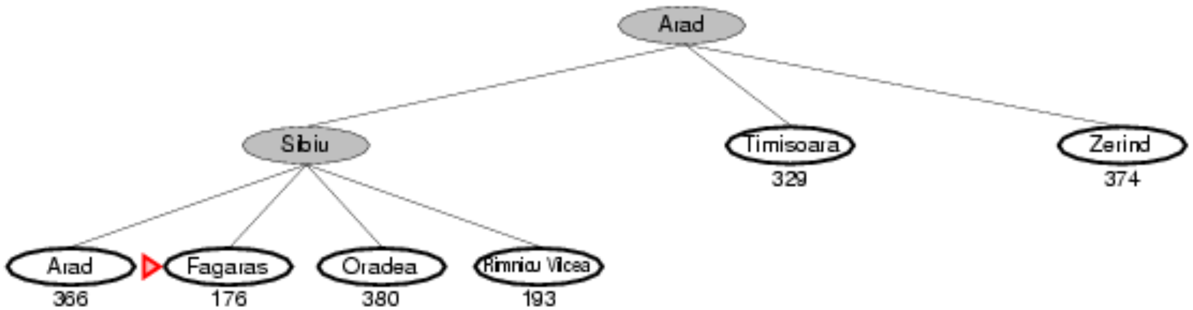
Straight–line distance
to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

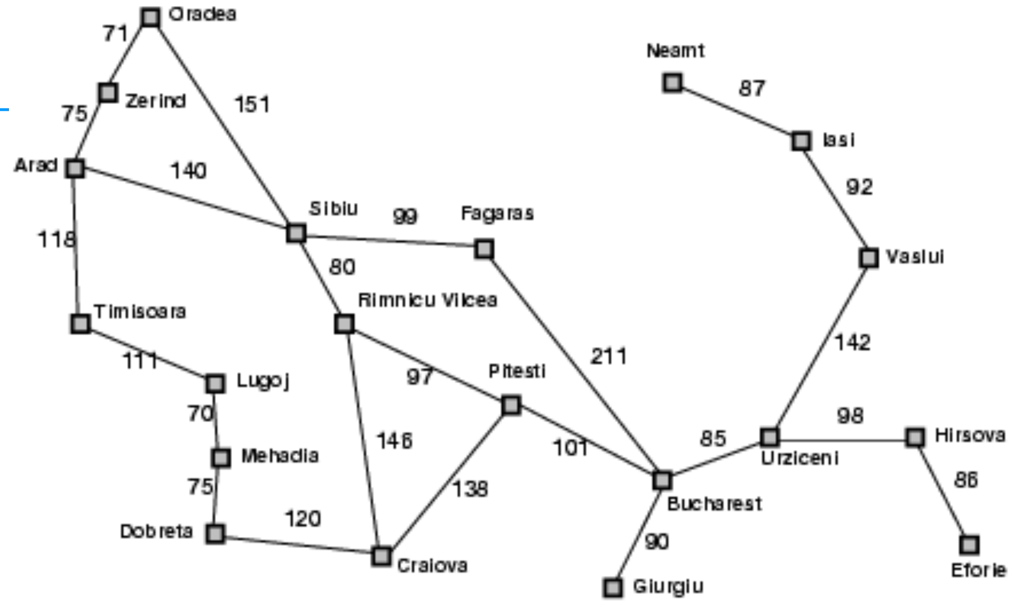
EXAMPLE



Straight–line distance 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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

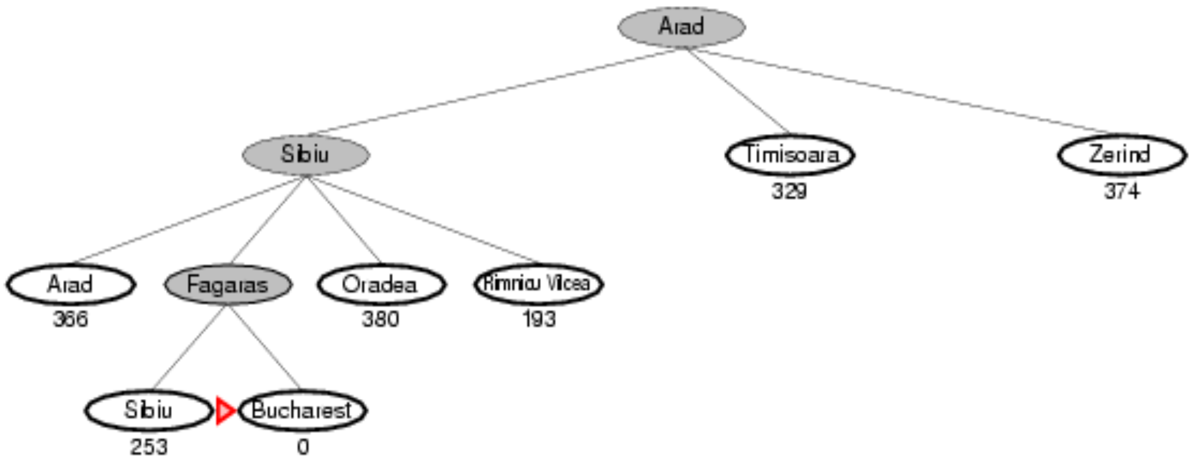


EXAMPLE



Straight–line distance
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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



BEST-FIRST SEARCH

- Best-first search is not optimal and may be incomplete (the same pittfall as the depth-first search).
- In the worst case, if the depth is d and the branching factor b the maximum number of expanded nodes will be b^d. (Time complexity).
- In addition, it keeps in memory all nodes, the spatial complexity coincides with the temporal one.
- With a good heuristic function, the spatial and temporal complexity can be reduced substantially.

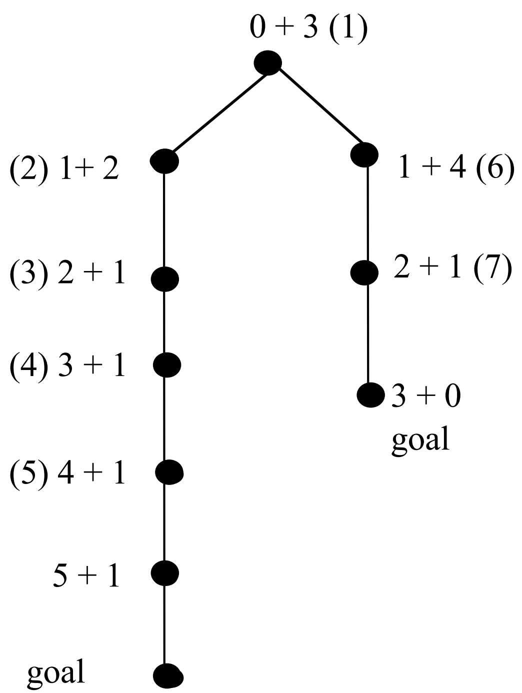
A* ALGORITHM

- Instead of only considering the distance to the goal, also consider the "cost" in reaching the node n from the root.
- We expand nodes for increasing values of f(n)

  $f(n) = g(n) + h'(n)$
- Where g (n) is the depth of the node, and h'(n) the estimated distance from the goal.
- We choose the node to expand as the one for which this sum is smaller.

A* ALGORITHM

- Basically we try to combine the benefits of depth-first search (efficiency) with the ones of uniform cost search (optimality and completeness).

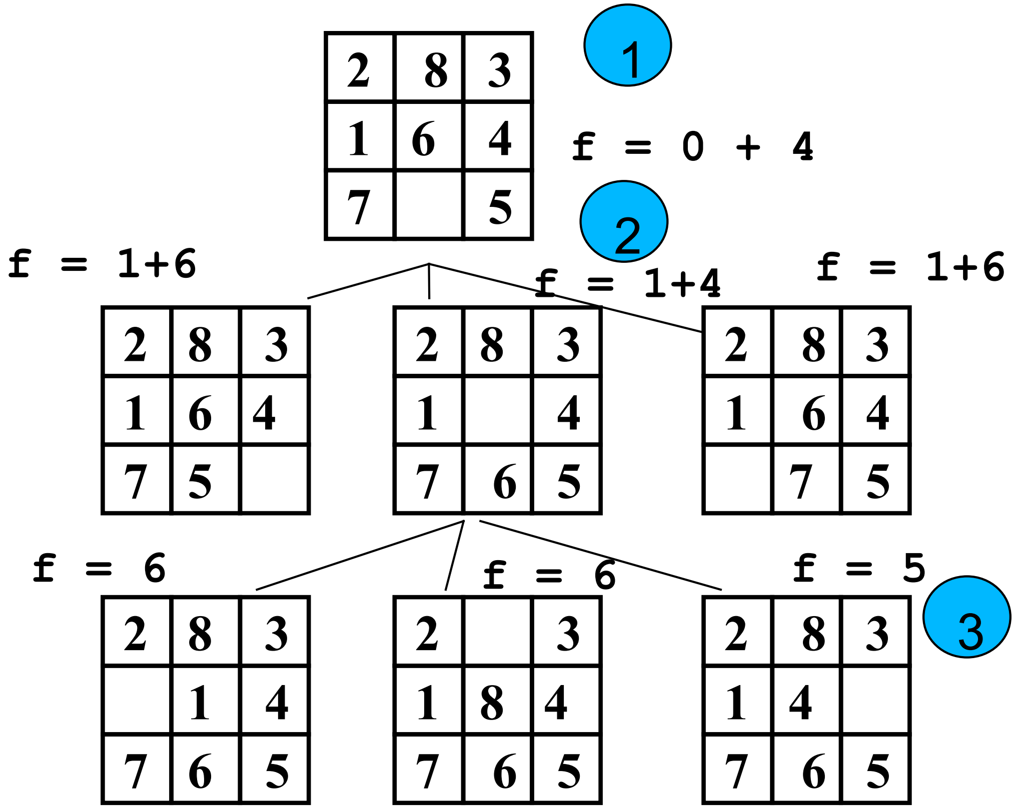


A* ALGORITHM

1. Let L be a list initial nodes of the problem.
2. If L is empty fail; otherwise et n be the node for which g (n) + h'(n) is minimal.
3. If n is the goal return solution (the path to reach it)
4. Otherwise remove n from L and add to L all the children of n labeled with g(n) + h’(n). Return to step 2

- Note:
  - the algorithm does not guarantee to find the optimal path. In the example, if the node with label 5 was the goal this would have been reached before the goal on the right (optimal).

EXAMPLE



$f(n) = g(n) + h'(n)$
where:

- $g(n)$ = depth of the node n;
- $h'(n)$ = number of tiles in the wrong place

etc….

Goal

1	2	3
8	4	
7	6	5

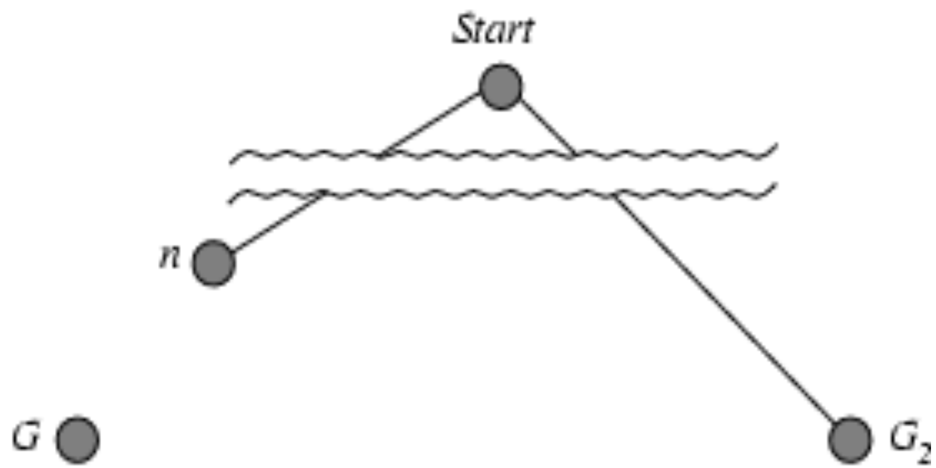
A* ALGORITHM

- A * does not guarantee to find the optimal solution (it depends on the heuristic function).
- Suppose you indicate with h(n) the true distance between the current node and the goal.
- The heuristic function h'(n) is **optimistic** that if we always have

  h '(n) <= h (n).
- This heuristic function is said to be **feasible**.
- Theorem:
  - If h '(n) <= h (n) for each node, then the A * algorithm always finds the optimal path to the goal
- Obviously the perfect heuristic h '= h is always feasible.
- If h '= 0 we always obtain a feasible heuristic function ⇒ breadth-first search

A* ALGORITHM OPTIMALITY PROOF

- Suppose you have created a sub-optimal goal G_2 *and* to have it in the queue. Let *n* be an unexpanded node in the queue such that *n* is on the shortest path to the optimal goal *G*.



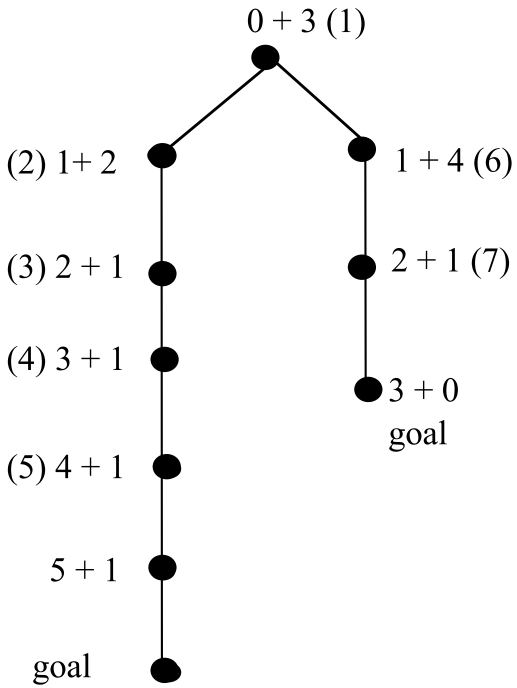
A* ALGORITHM OPTIMALITY PROOF

- $f\ (G_2) = g(G_2)$ as $h(G_2) = 0$
- $g(G_2) > g(G)$ as G_2 is sub-optimal
- $f\ (G) = g(G)$ as $h(G) = 0$
- $f\ (G_2) > F\ (G)$ from above

- $h\ '(n) \leq h\ (n)$ as h' is feasible permissible
- $g\ (n) + h\ '(n) \leq g\ (n) + h\ (n) = f\ (G)$ (n is on the path to G)
- $f\ (n) \leq f\ (G)$

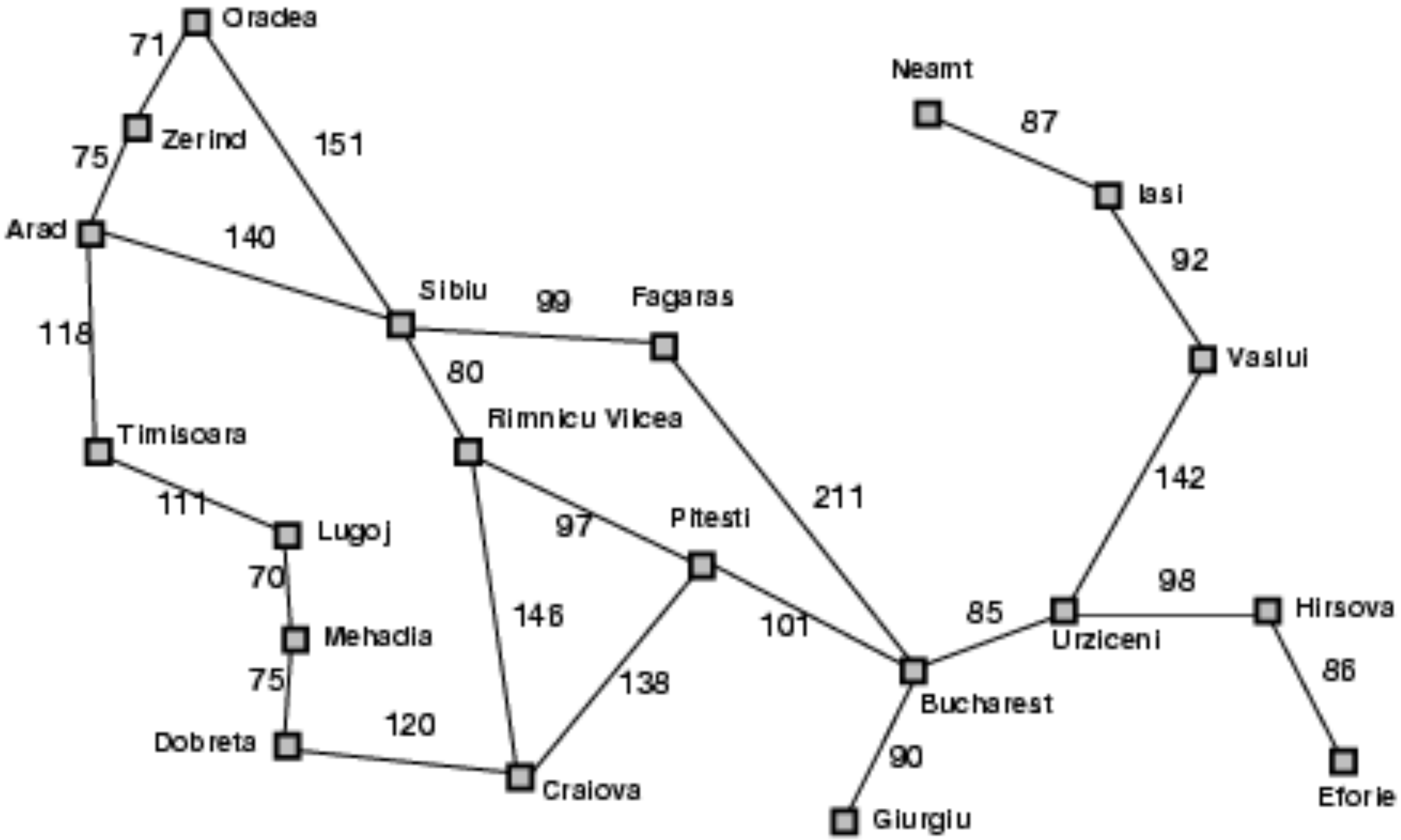
then $f\ (G_2) > f(n)$, and A* will never select G_2 for expansion.

EXAMPLE: SEARCH WITH FEASIBLE HEURISTIC FUNCTION



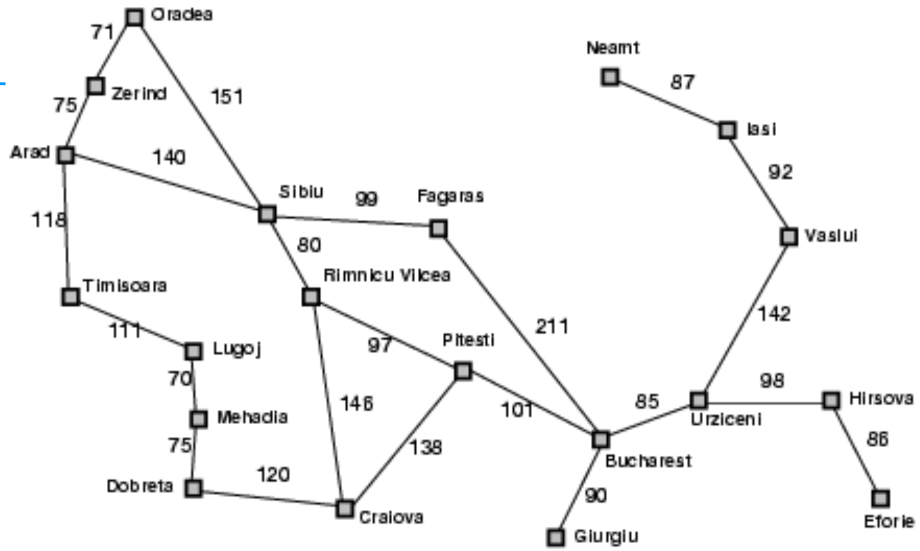
- The previous choice for the game of 8 (counting the tiles in the wrong position as h') is always feasible because every move can reduce the distance to at most one unit

EXAMPLE



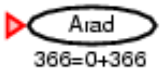
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* EXAMPLE

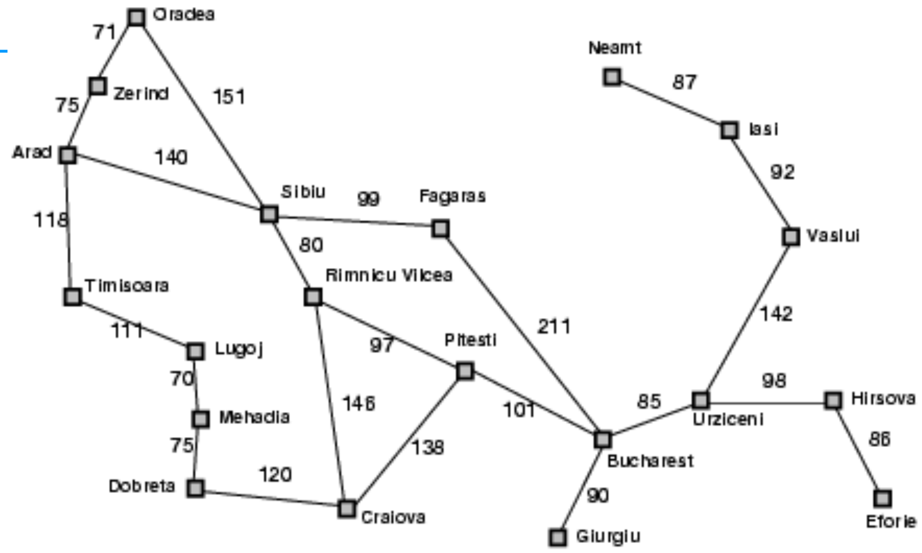


Straight–line distance to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



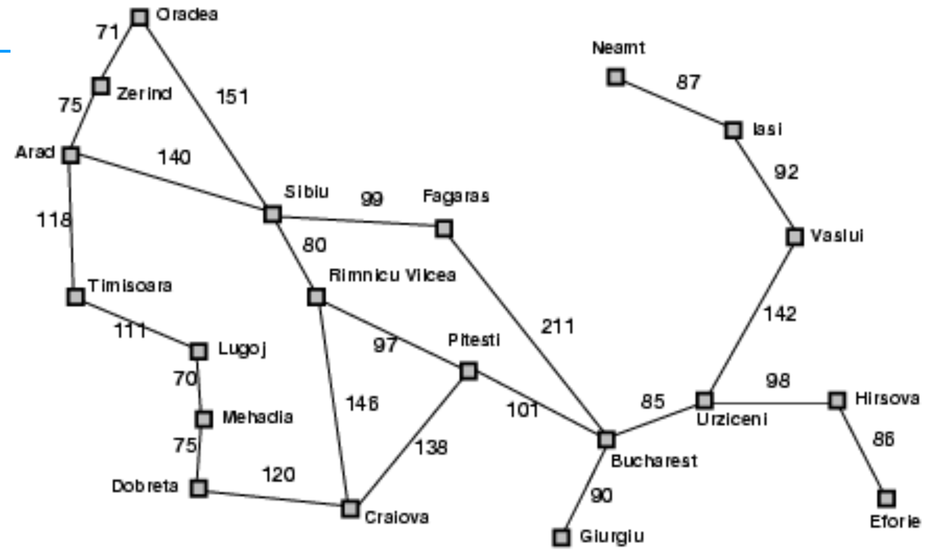
A* EXAMPLE



Straight–line distance
to Bucharest

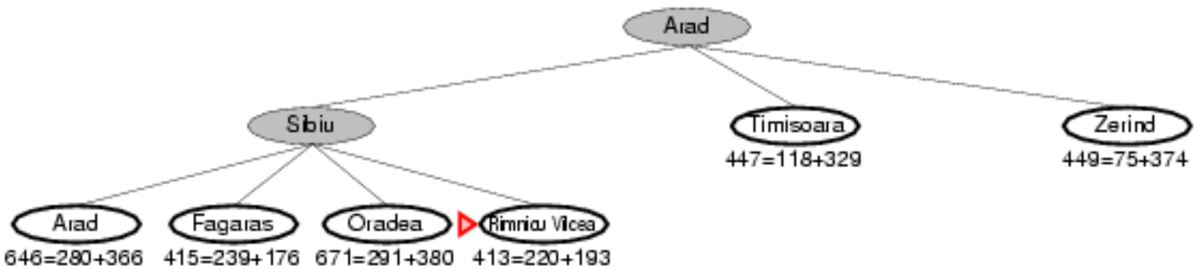
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* EXAMPLE

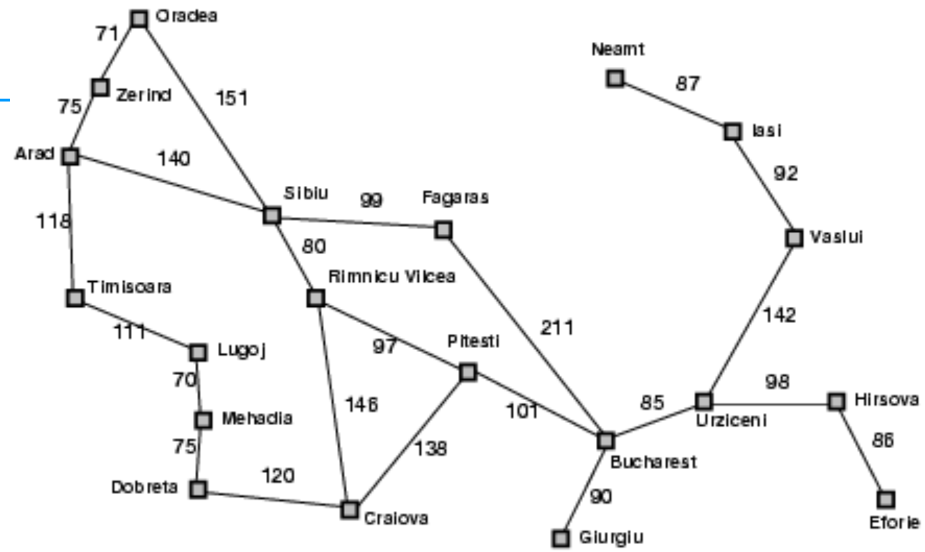


Straight–line distance to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

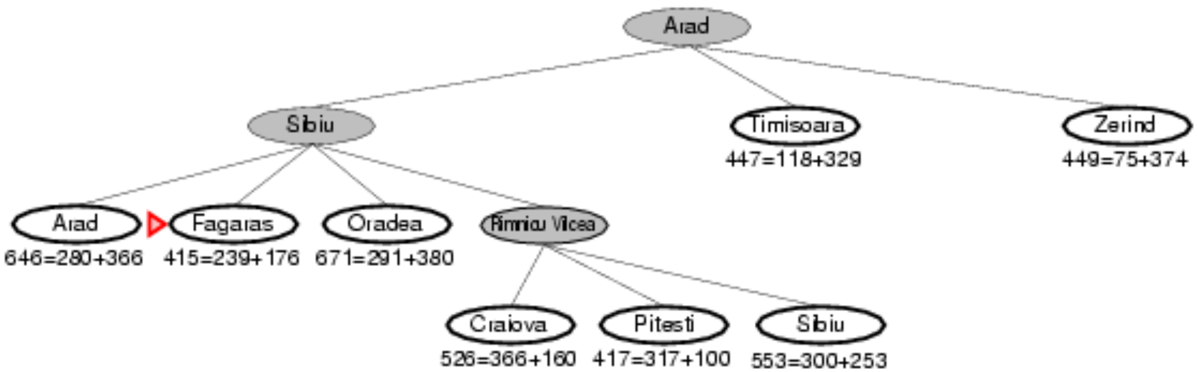


A* EXAMPLE

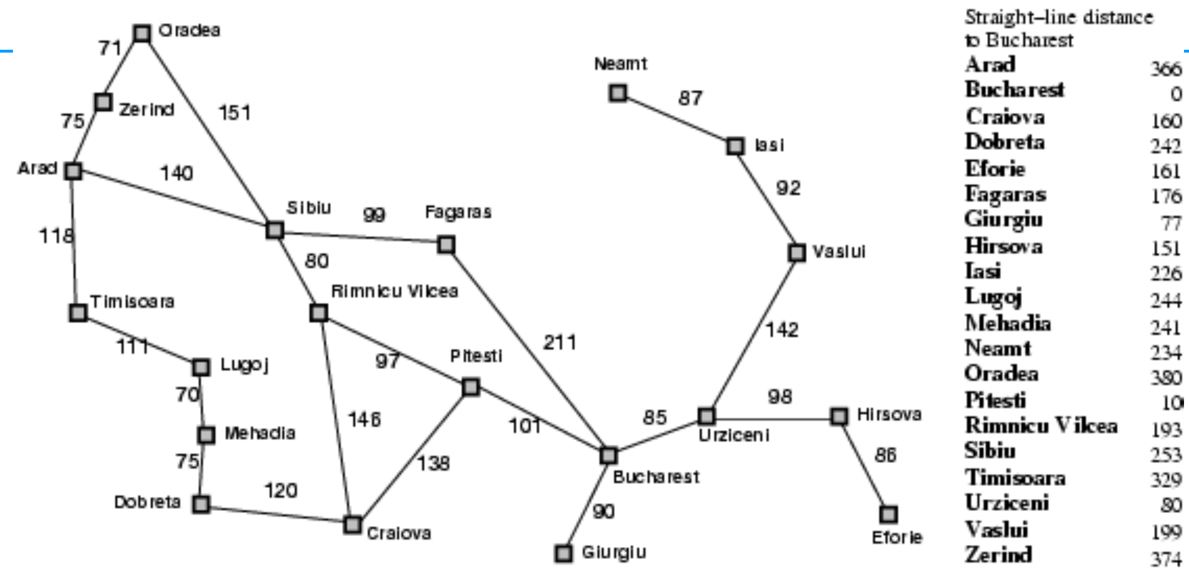


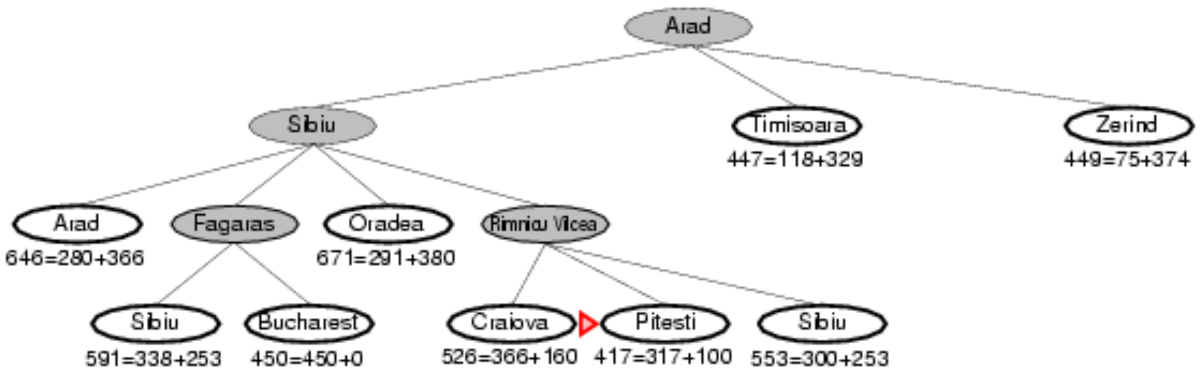
Straight–line distance
to Bucharest

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

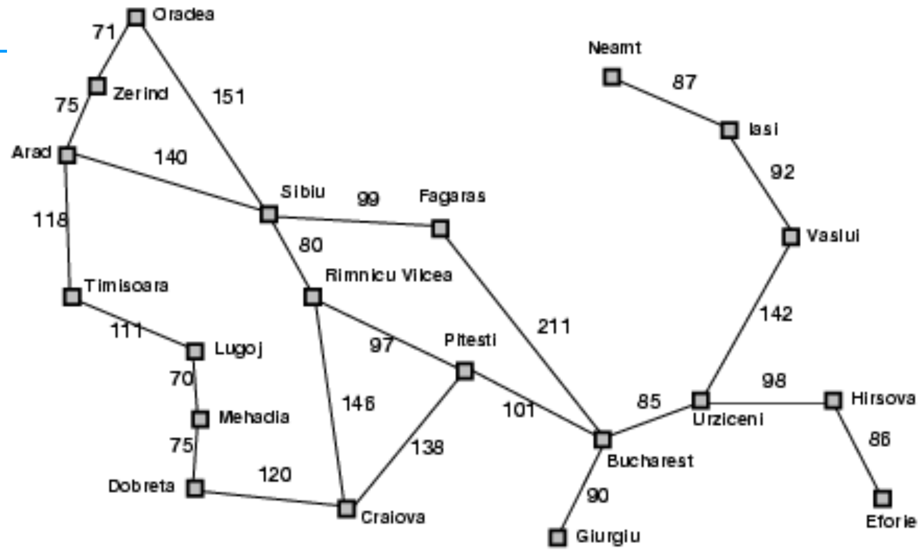


A* EXAMPLE


Oradea
71
Zerind
151
75
Arad
140
118
Sibiu
99
Fagaras
80
Rimnicu Vilcea
Timisoara
111
Lugoj
97
Pitesti
211
70
Mehadia
146
101
75
138
Dobreta
120
Craiova
Neamt
87
Iasi
92
Vaslui
142
98
85
Urziceni
Hirsova
86
Bucharest
90
Giurgiu
Eforie
Straight–line distance 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 10
Rimnicu Vilcea 193
Sibiu 253
Timisoara 329
Urziceni 80
Vaslui 199
Zerind 374

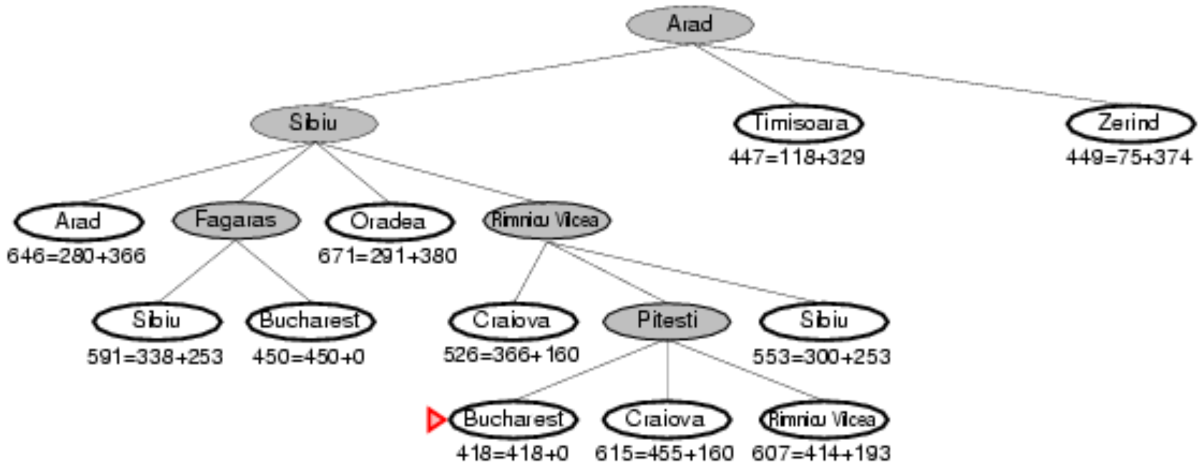

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

A* EXAMPLE

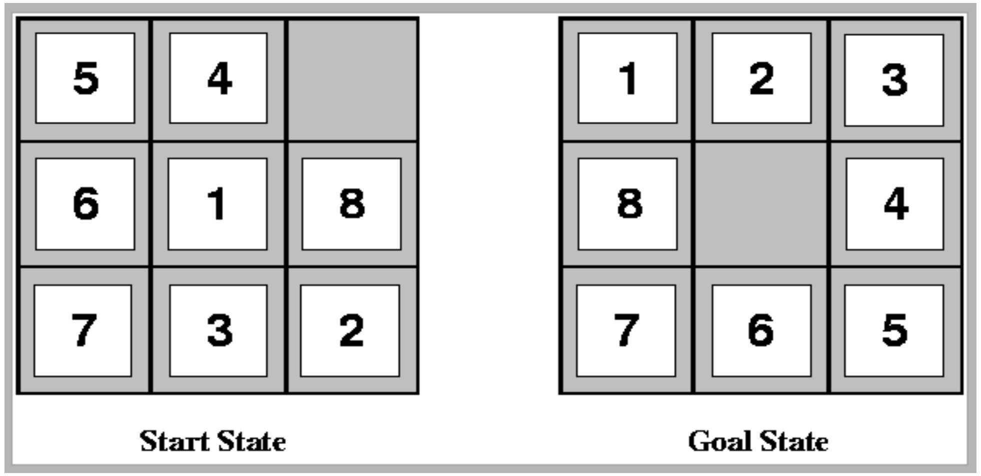


Straight–line distance to Bucharest

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	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



ELIGIBLE FUNCTIONS HEURISTICS



We can define different heuristics. For example:

h_1= Number of tiles that are out of place (h_1= 7)

h_2= The sum of the distances from the initial and final position of each tile. The distance is a sum of the horizontal and vertical distances (**Manhattan distance**). The tiles 1 to 8 in the initial state have a distance h_2=2+3+3+2+4+2+0+2=18

Both heuristics are eligible

ELIGIBLE FUNCTIONS HEURISTICS

- As $h_1' <= h_2'$ then h_2' is better
- It is better to use a heuristic function with higher values, provided it is optimistic.
- How to invent heuristics?
- Often the cost of an exact solution of a relaxed problem is a good heuristic for the original problem.
- If the problem definition is described in a formal language you can build relaxed problems automatically.
- Sometimes you can use the maximum among different heuristics.
- $h'(n) = \max(h_1(n), h_2(n) \ldots h_m(n))$.

EXAMPLE: GAME OF 8

- Description: a tile can move from square A to square B if A is adjacent to B and B is empty.

- Relaxed problems remove some conditions
  - A tile can move from square A to square B if A is adjacent to B. (manhattan distance)
  - A tile can move from square A to square B if B is empty.
  - A tile can move from square A to square B in one hop (tiles out of place).

FROM TREES TO GRAPHS

- We have assumed so far that the search space is a tree and not a graph. It is therefore not possible to achieve the same node from different paths.

- This assumption is of course simplistic: think of the game of 8, the missionaries and cannibals etc.

- How to extend the previous algorithms for dealing with graphs?

GRAPH-SEARCH

```
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 SOLUTION(node)
       if STATE[node] is not in closed then
           add STATE[node] to closed
           fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

SEARCH IN GRAPH WITH A*

- Two lists: **open and closed nodes**
  - **Closed nodes**: expanded nodes are removed from the list to avoid further examination;
  - **Open nodes**: Nodes still to be examined.

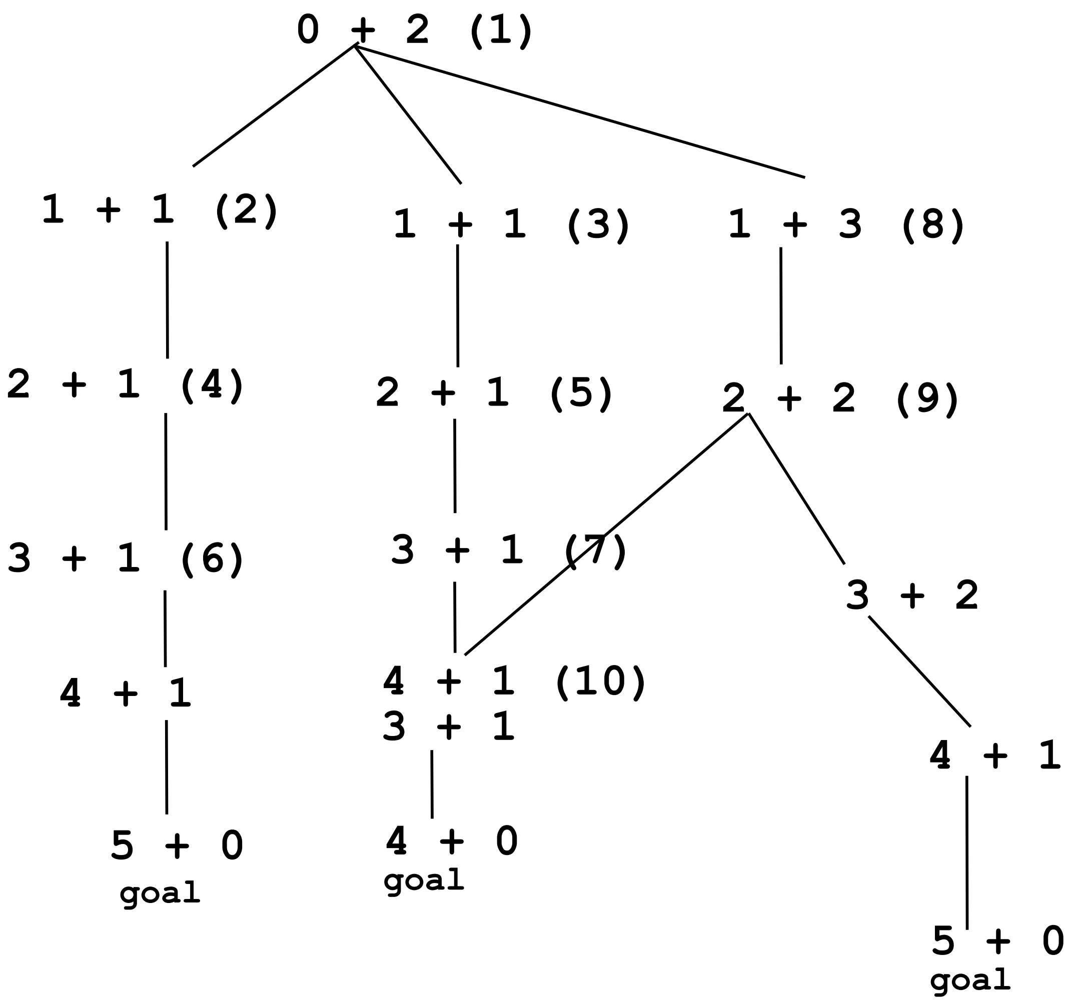
- A* searching in a graph instead of a tree.

- Changes:
  - The graph can become a tree with repeated nodes
  - Add the list of closed nodes and assume that g(n) evaluates the minimum distance of node n from the starting node.

A* ALGORITHM FOR GRAPH

- Let L_a be the open list of initial nodes of the problem.
- Let n be the node for which g (n) + h '(n) is minimal. If the list is empty, fail;
- If n is the goal then stop and return the path to reach it
- Otherwise remove n from L_a, enter it in the list of closed nodes L_c and add to L_a all the children of n, labelling them with the cost from the starting node to n.
- If a child node is already in L_a, do not add it, but update is label in case its cost (root to node) is smaller than the one it had.
- If a child node is already in L_c, do not add it to L_a, but if its cost is better, update its cost and the one of its children and descendant.
- Returns to step 2

EXAMPLE


0 + 2 (1)
1 + 1 (2)
1 + 1 (3)
1 + 3 (8)
2 + 1 (4)
2 + 1 (5)
2 + 2 (9)
3 + 1 (6)
3 + 1 (7)
3 + 2
4 + 1
4 + 1 (10)
3 + 1
4 + 1
5 + 0
goal
4 + 0
goal
5 + 0
goal

CONSISTENT HEURISTIC (MONOTONE)

- A further condition of h: consistency (or monotonicity)
- Definition: a heuristic is' consistent if for each node *n*, any successor *n'* of *n* generated by each action *a*,

  h (n) = 0 if the corresponding status is the goal

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

- With monotonicity, we are guaranteed to find the shortest path from the root to the goal

CONSISTENT HEURISTIC (MONOTONE)

- if *h* is consistent we have that:

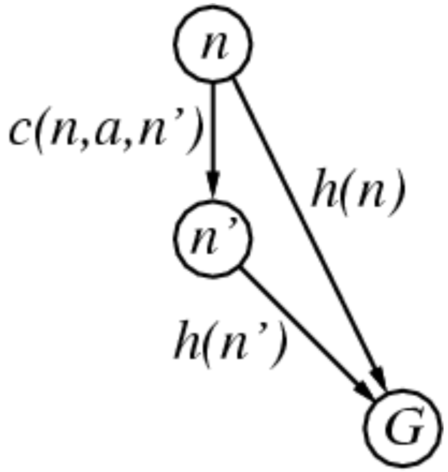
  f (n') = G (n') + h (n')

  = G (n) + c (n, a, n') + h (n')

  ≥ g (n) + h (n)

  = f(n)

- *f(n)* never decreases along a path.



- Theorem : if *h(n)* is consistent then A* using `GRAPH-SEARCH` is optimal