Lecture 3 Sept 8, 2022

- Informed search

Chapter 3, Sec 5 of Russell and Norvig

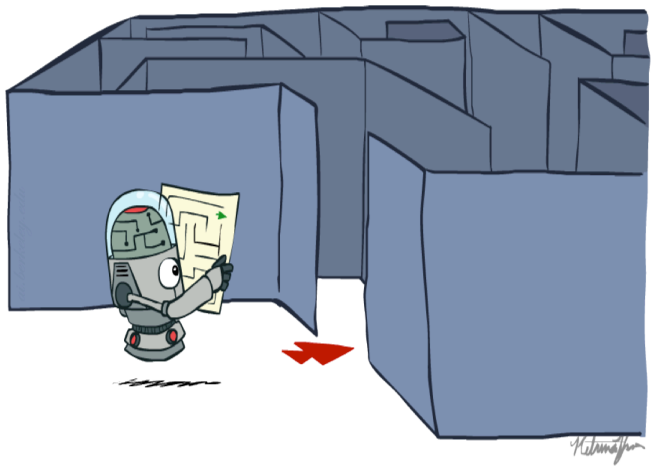
Chapter 2 of Edelkamp

Outline

- Greedy search
  - greedy depth-first search
  - best-first search
- A* search
- Properties of Heuristics
  - Admissibility, monotonicity, etc.
- Memoized search (a.k.a. dynamic programming)
- Automatic generation of heuristics

Recap: Search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test



- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)

- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

Tree Search vs. graph search

function TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  loop do
    if the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

---

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

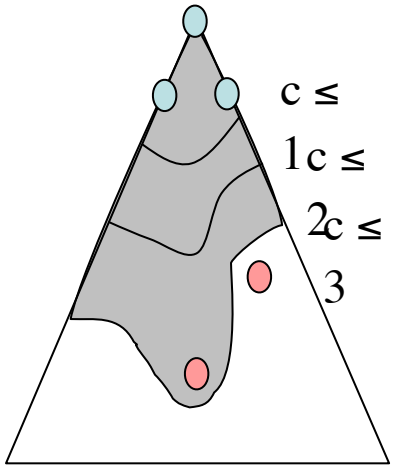
General graph search

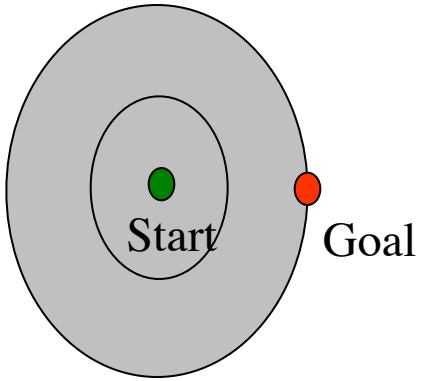
```
function GRAPH-SEARCH(problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?( fringe) then return failure
        node ← REMOVE-FIRST( 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 ← INSERT-ALL(EXPAND(node, problem), fringe)
```

This is actually a collection of algorithms depending on how REMOVE-FIRST is performed on from fringe.

Uniform Cost Search

- This is an extension of BFS when applied to weighted graphs.
- Keep the total weight of the distance from start to current node associated with an open node n and expand the one with the smallest weight.





Priority queue implementation

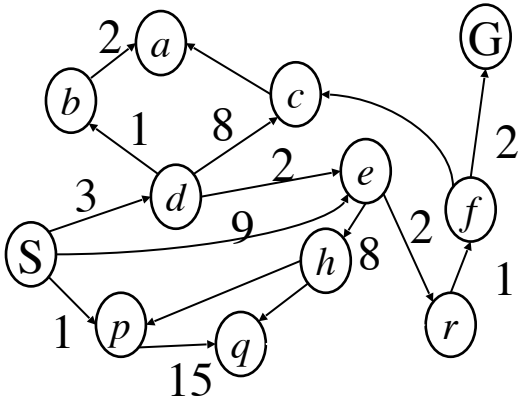
- Priority queue:
  - Insert
  - Delete-min

- Data structures:
  - Binary heap
  - Binomial heap, pairing heap, …

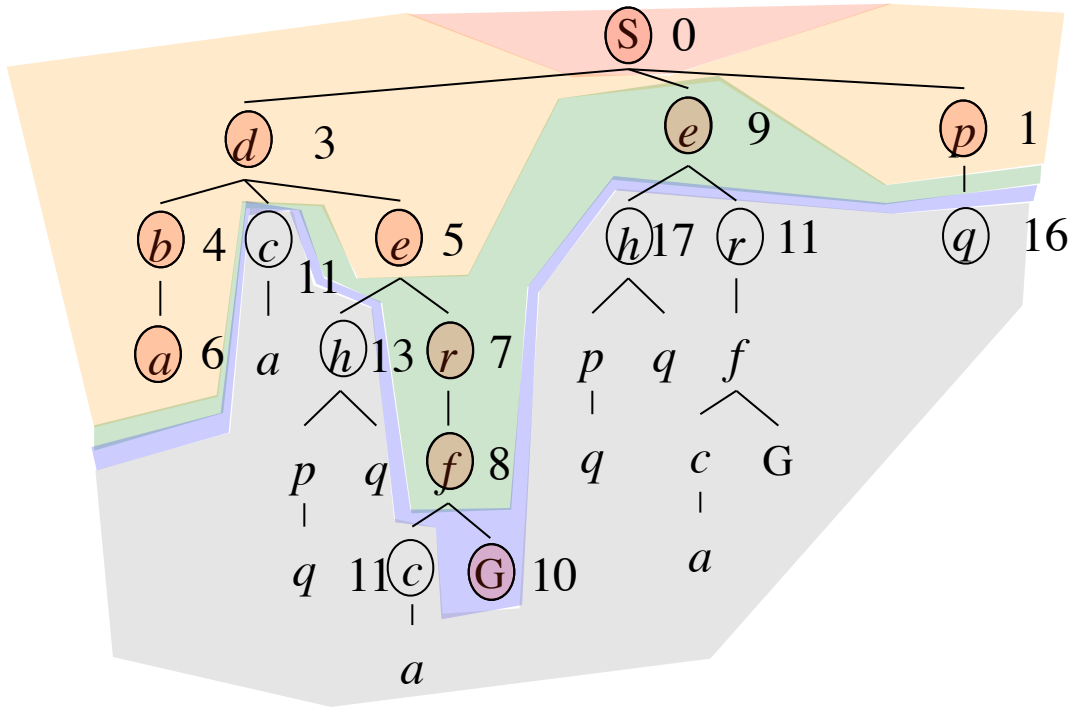
Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)



Cost contours



Uniform Cost Search (UCS) Properties

- What nodes does UCS expand?
  - Processes all nodes with cost less than cheapest solution!
  - If that solution costs C^* and arcs cost at least ε , then the "effective depth" is roughly C^*/ε
  - Takes time $O(b^{C^*/\varepsilon})$ (exponential in effective depth)

- How much space does the fringe take?
  - Has roughly the last tier, so $O(b^{C^*/\varepsilon})$

- Is it complete?
  - Assuming best solution has a finite cost and minimum arc cost is positive, yes!

- Is it optimal?
  - Yes!

Heuristic - estimate of distance to goal

Uninformed search - children (successors) of a node were considered in arbitrary order.

Heuristic search – successors are compared to determine which one is more likely to lead to a solution.

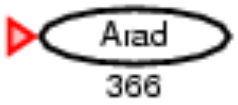
$H(n)$ = estimated distance from current node n to a goal node.

Search cost – if the edges are weighted, moving from p to q incurs a cost of $w(p, q)$.

Greedy search

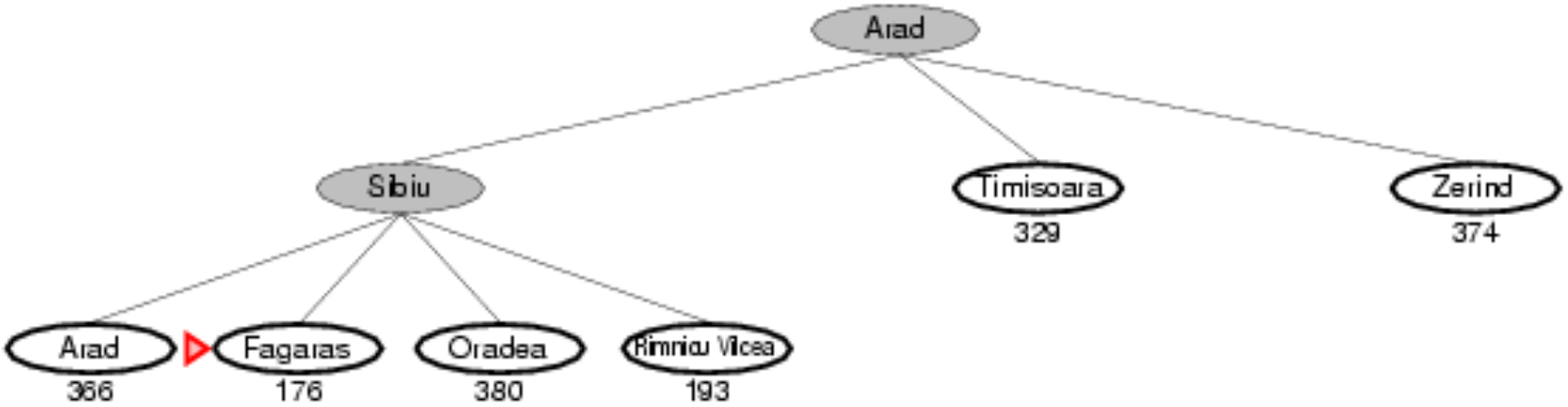
- Idea: use an evaluation function $h(n)$ for each node n. Remove from fringe a node with the smallest h value and add its children to the fringe.
- Repeat until the goal node leaves the fringe.

Greedy search example

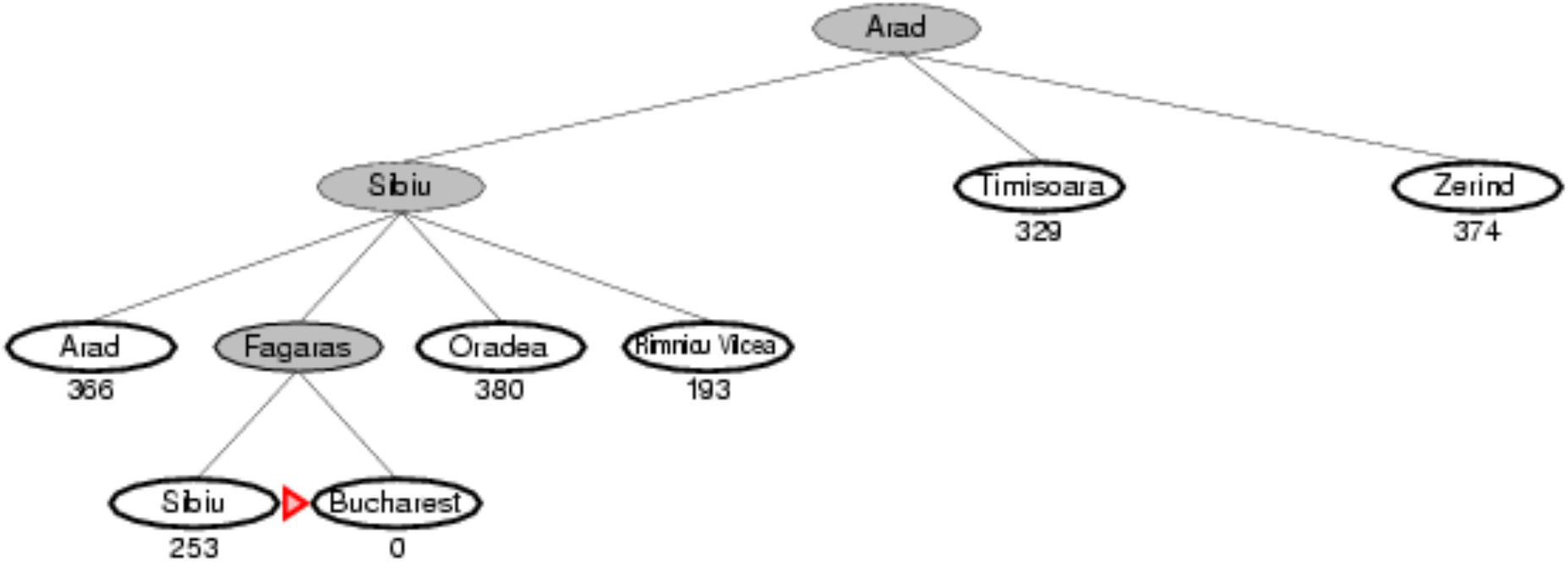


Greedy search example

Greedy search example

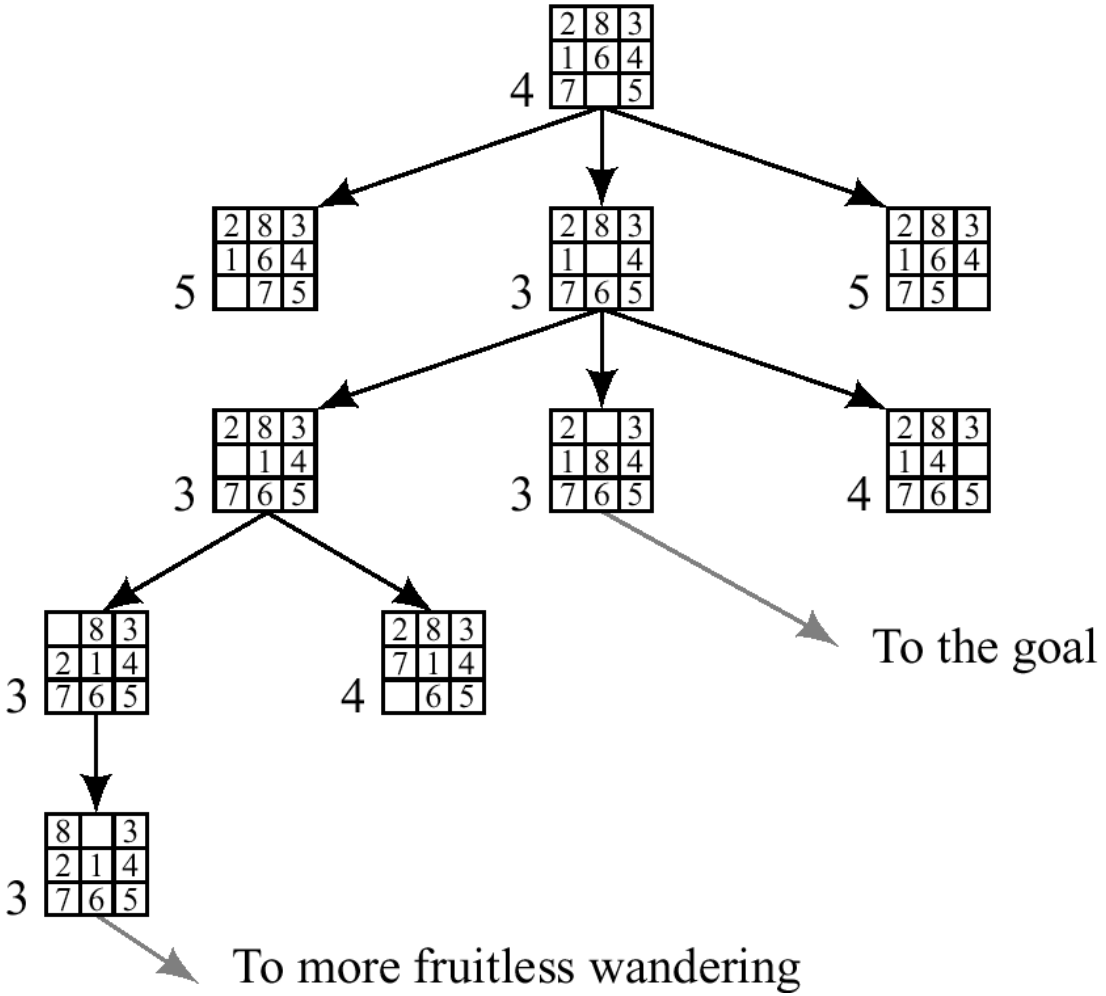


Greedy search example



Greedy search

$h(n)$ = number of misplaced tiles



Problems with Greedy Search

- Not complete

- Not optimal
- Irrevocable (this can also be its strength since it can find a solution fast.)

best-first search

- Idea: avoid expanding paths that are already expensive even if the estimated distance to the goal is small.

- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$ = cost so far to reach n
  - $h(n)$ = estimated cost from n to goal
  - $f(n)$ = estimated total cost of path through n to goal

Admissible heuristics

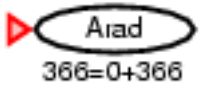
- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic.

Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

When best-first search is implemented with an admissible heuristic function $h(n)$, it is called A* algorithm.

A* search: Example

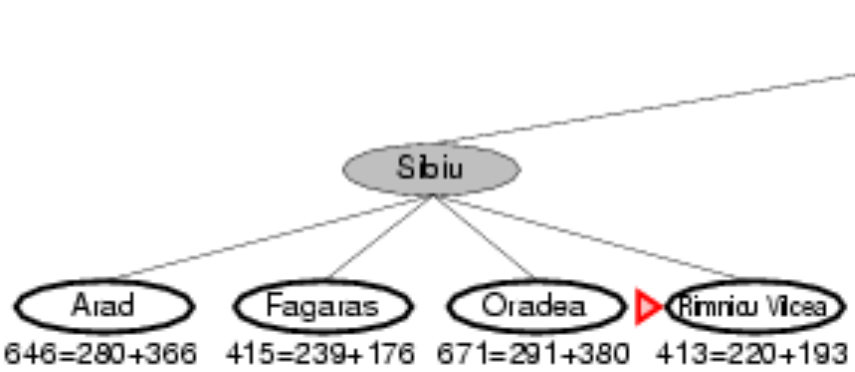


Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

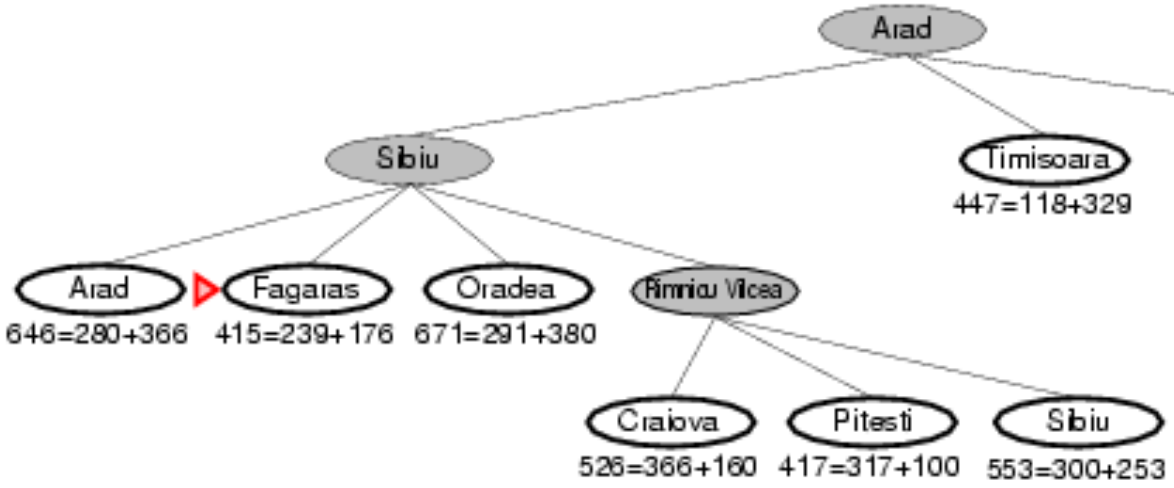
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search: 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

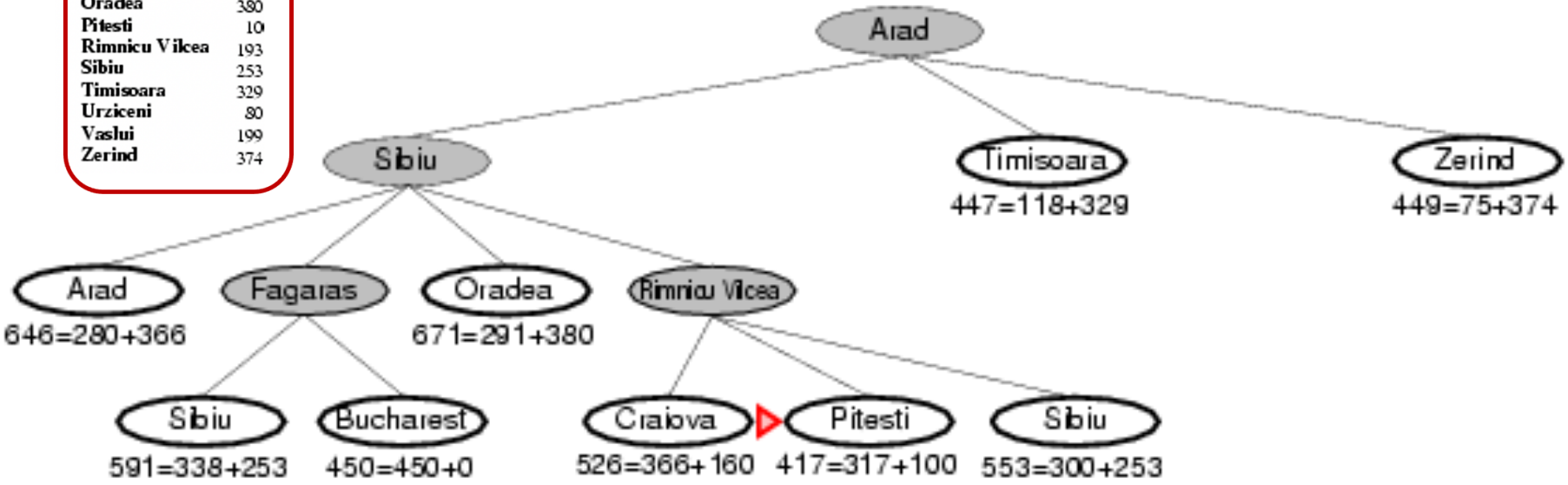
A* search example



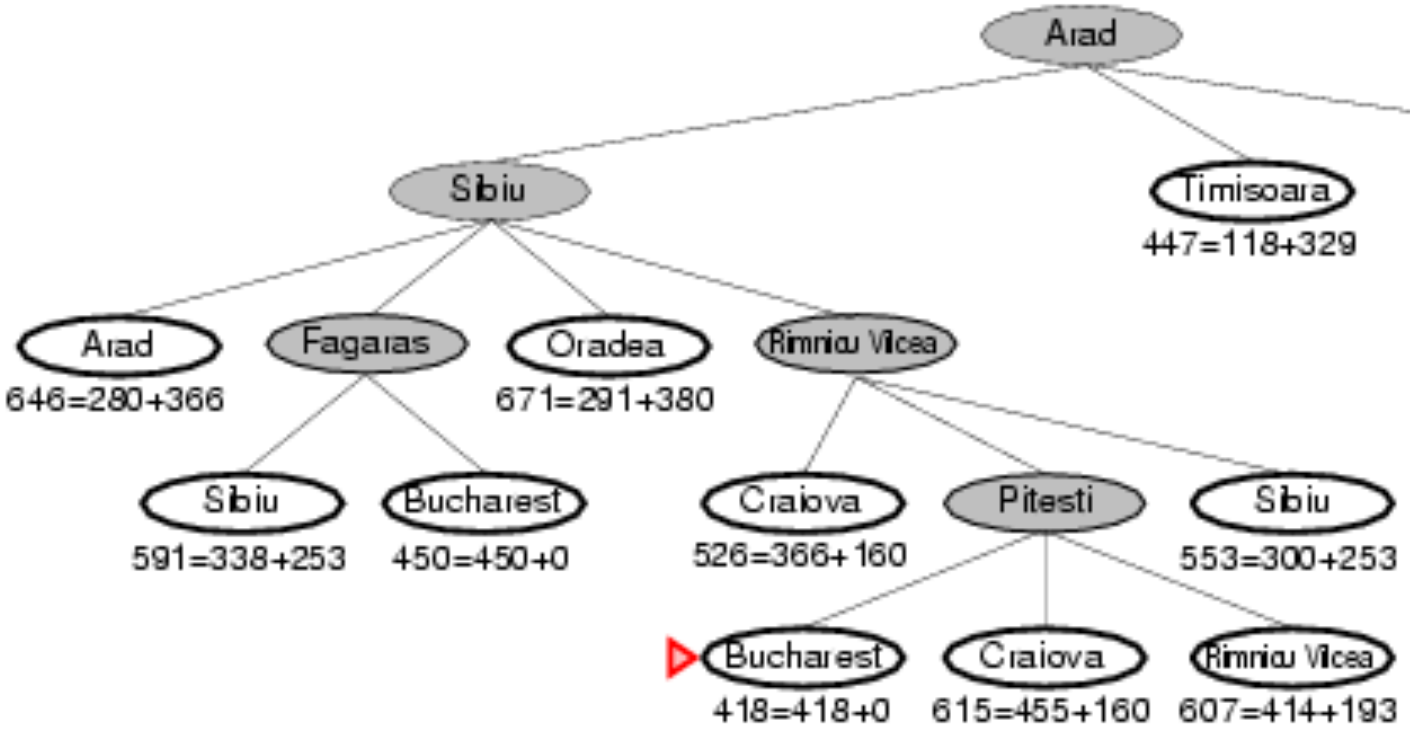
Straight-line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search 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* search 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* tree search version

```
PriorityQueue q;
q.insert(initialState, h(initialState));
while (!q.isEmpty()){
  node = q.remove();
  if (goalTest(node)) return node;
  foreach (n in successors(node, operators))
    q.insert(n, g(n) + h(n));
}
return FAIL;
```

A* graph search version

Procedure A*
Input: Implicit problem graph with start node s, weight function w, heuristic h, successor generation function *Expand*, and predicate *Goal*
Output: Cost-optimal path from s to $t \in T$, or $\emptyset$ if no such path exists

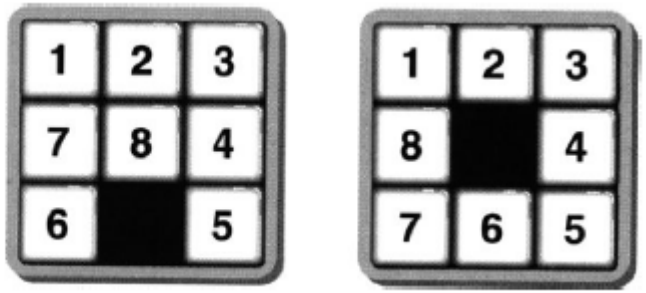
```
Closed ← ∅                                   ;; Initialize structures
Open ← {s}                                   ;; Insert s into search frontier
f(s) ← h(s)                                  ;; Initialize estimate
while (Open ≠ ∅)                             ;; As long as there are frontier nodes
   Remove u from Open with minimum f(u)      ;; Select node for expansion
   Insert u into Closed                      ;; Update list of expanded nodes
   if (Goal(u)) return Path(u)               ;; Goal found, return solution
   else Succ(u) ← Expand(u)                  ;; Expansion yields successor set
      for each v in Succ(u)                  ;; For all successors v of u
         Improve(u,v)                        ;; Call relaxation subroutine
return ∅                                     ;; No solution exists
```

Procedure Improve
Input: Nodes u and v, v successor of u
Side effects: Update parent of v, $f(v)$, *Open*, and *Closed*

```
if v in Open                                   ;; Node already generated but not expanded
   if (g(u) + w(u,v) < g(v))                   ;; New path is cheaper
      parent(v) ← u                            ;; Set predecessor pointer
      f(v) ← g(u) + w(u,v) + h(v)              ;; DecreaseKey operation
else if v in Closed                            ;; Node v already expanded
   if (g(u) + w(u,v) < g(v))                   ;; New path cheaper
      parent(v) ← u                            ;; Set predecessor pointer
      f(v) ← g(u) + w(u,v) + h(v)              ;; Update estimate
      Remove v from Closed                     ;; Reopening of v
      Insert v into Open with f(v)             ;; Reopening of node
else                                           ;; Node not seen before
   parent(v) ← u                               ;; Set predecessor pointer
   Initialize f(v) ← g(u) + w(u,v) + h(v)      ;; First estimate
   Insert v into Open with f(v)                ;; Add v to search frontier
```

Sliding piece puzzle (8-puzzle)

FIGURE 4.1
An instance of the 8-puzzle



Cost of each sliding move = 1

Heuristic: number of misplaced tiles

Blank Moves	Distance
left	2
right	4
up	3

Our search heuristic moves as quickly toward the goal as possible.

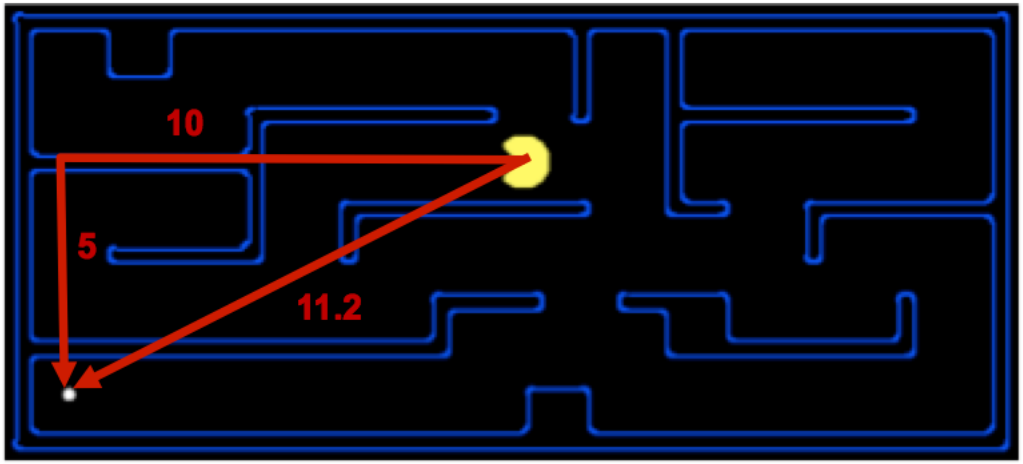
We will select the move of moving the blank to the left in Figure above.

Extending this analysis will lead us to move the blank up next and then to the right, arriving at the goal configuration after three moves.

Maze search problem

Goal: Find a path from between two specific nodes.

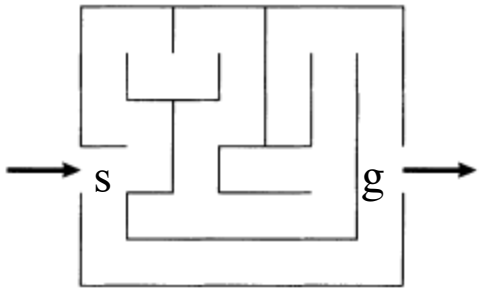
Heuristic: distance from node n to g disregarding the barriers.
(either Euclidean or Manhattan)



Maze search problem

Goal: Find a path from s to g.

FIGURE 4.2
A maze with two solutions



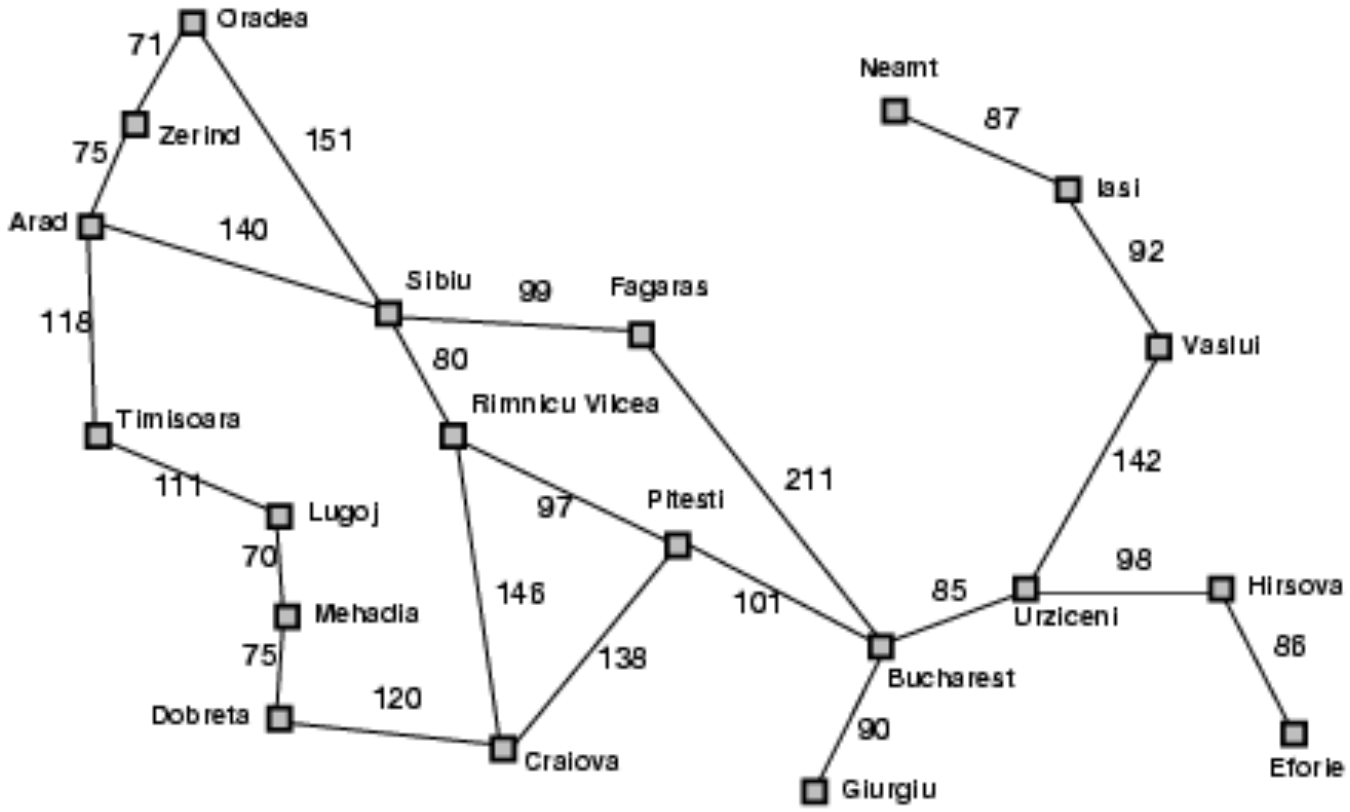
Heuristic: distance from node n to g disregarding the barriers.
(either Euclidean or Manhattan)

Does not work well in this example.

Common feature of all search

- All these search algorithms are the same except for fringe strategies
  - Conceptually, all fringes are priority queues (i.e. collections of nodes with attached priorities)
  - Practically, for DFS and BFS, you can avoid the log(n) overhead from an actual priority queue, by using stacks and queues
  - Can even code one implementation that takes a variable queuing object

Romania with step costs in km



h(n)

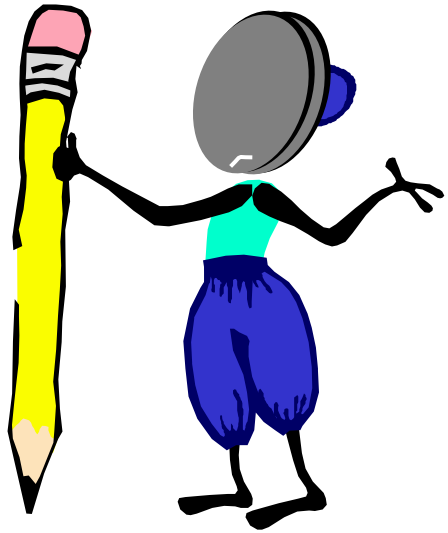
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Pancake sorting problem

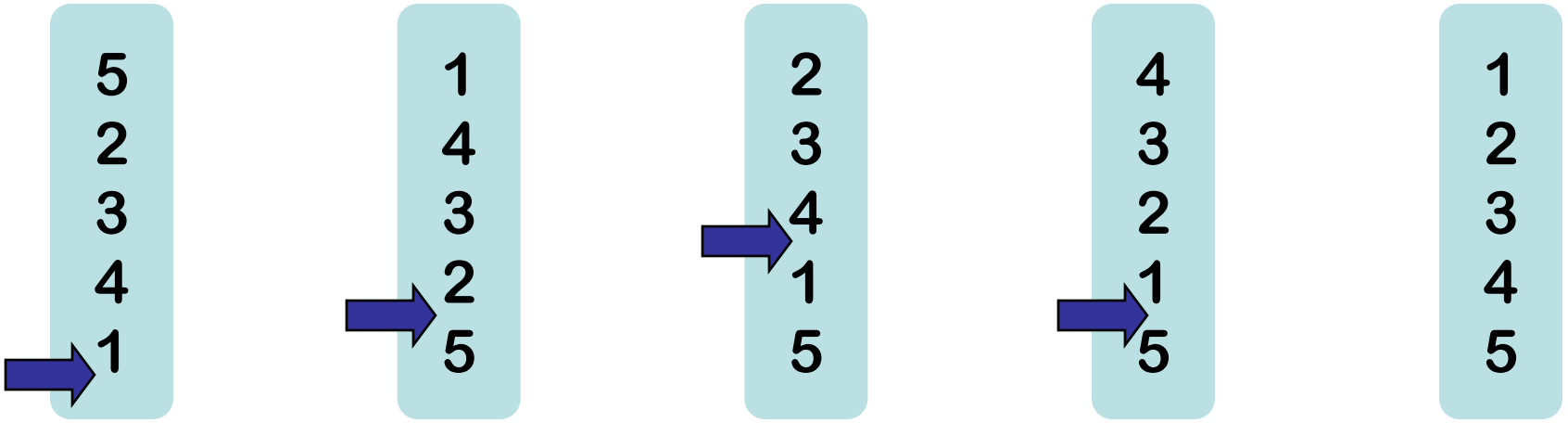
- Several pancakes are on a plate.
- Goal is to sort them (so that the largest is at the bottom).
- Operation: pick the top k pancakes (using a spatula) and flip them over.
- Cost: k units for flipping k pancakes at a time.

What is the cost to sort this stack?

5
2
3
4
1



One way to sort pancakes

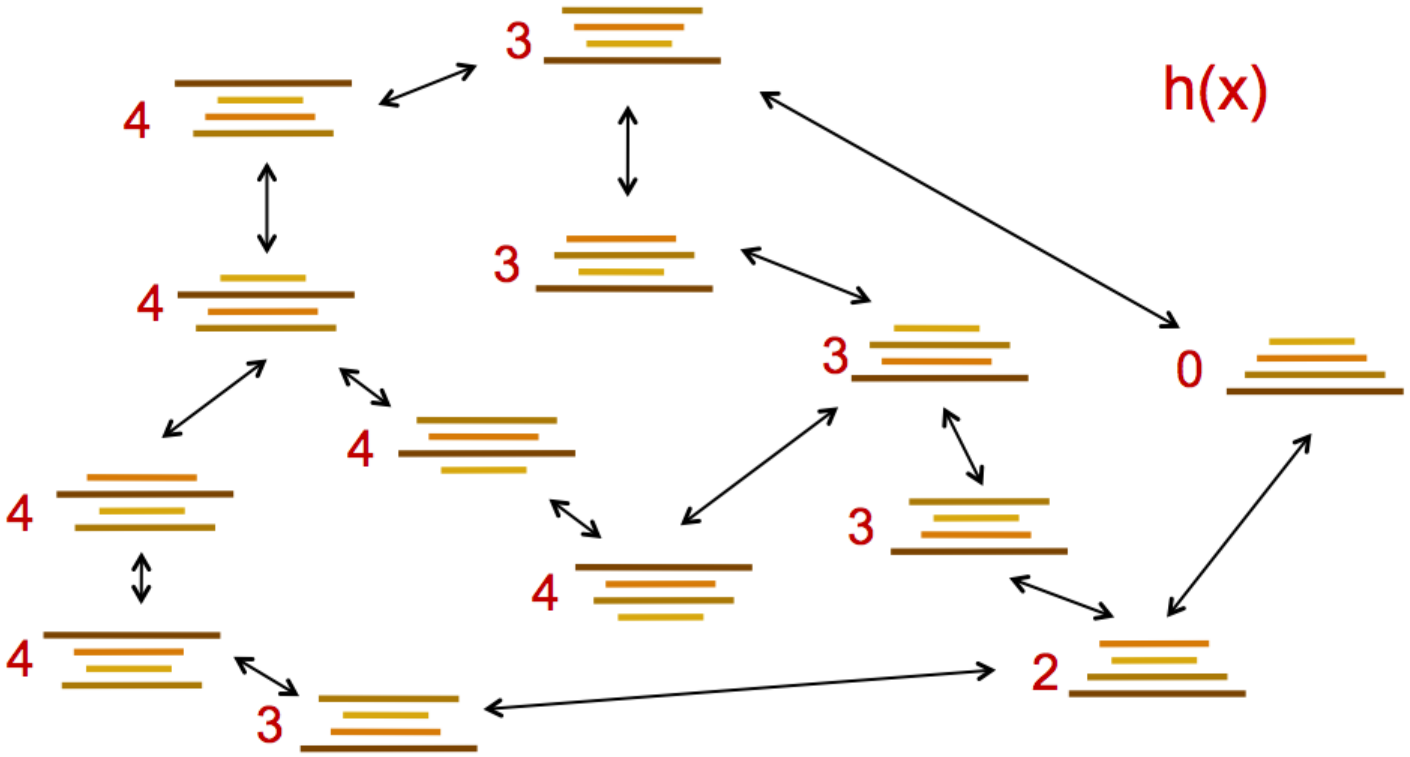


Cost associated with the path: 5 + 4 + 3 + 4

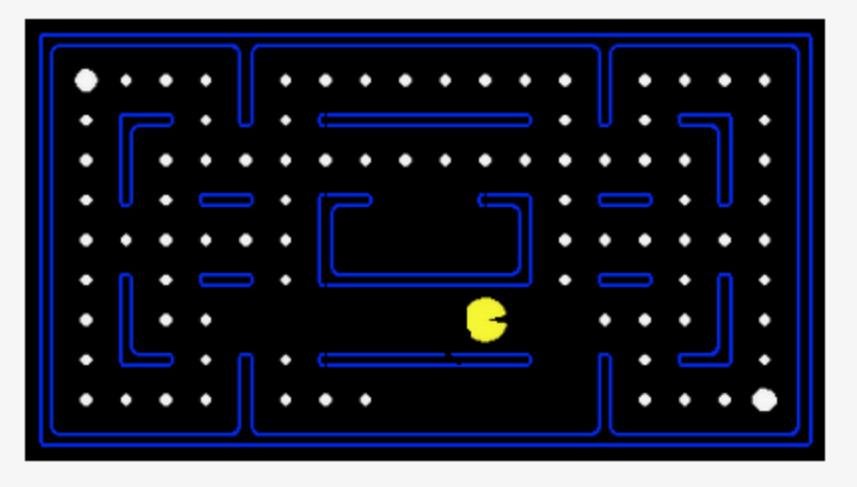
$H(x)$ = the number of the largest pancake that is not in place.

Graph of pancake sorting

Heuristic: the number of the largest pancake that is still out of place



Pacman dot eating problem



- As the pacman passes through a square with a dot, it gets eaten. Goal is to eat all the dots.

- Minimize: total distance traveled by pacman. (each move counts as 1 unit.)

- Is “h(n) = Farthest Manhattan distance between two dots” admissible?

H(n) for some problems

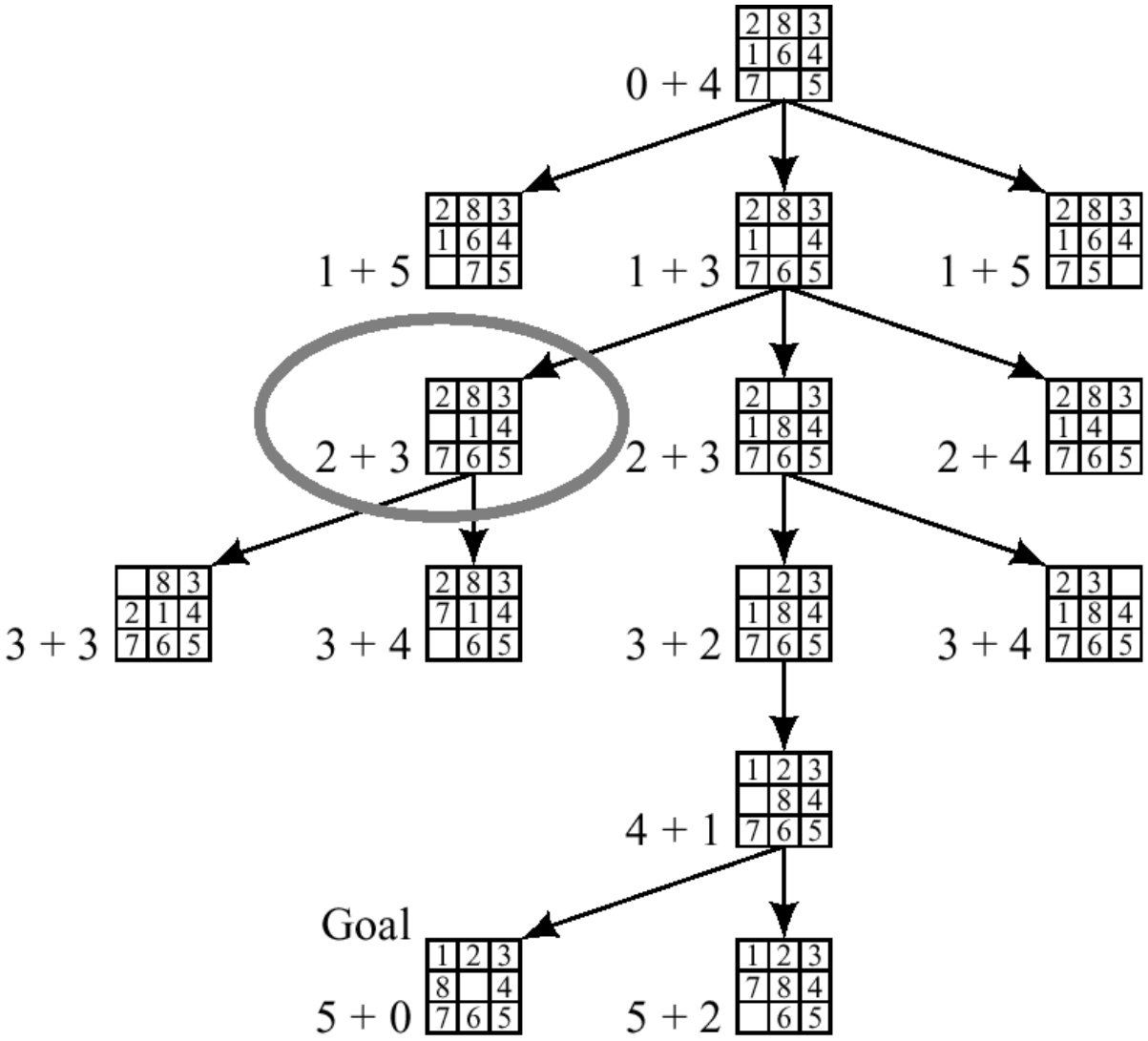
- 8-puzzle (sliding piece puzzle)
  - W(n): number of misplaced tiles
  - Manhatten distance
  - More elaborate ones based on pattern data bases

- for sorting by reversal problem what is a good admissible *h function*? (for two costs: cost of each reversal is 1 or k if the length of the reversed block is k.)

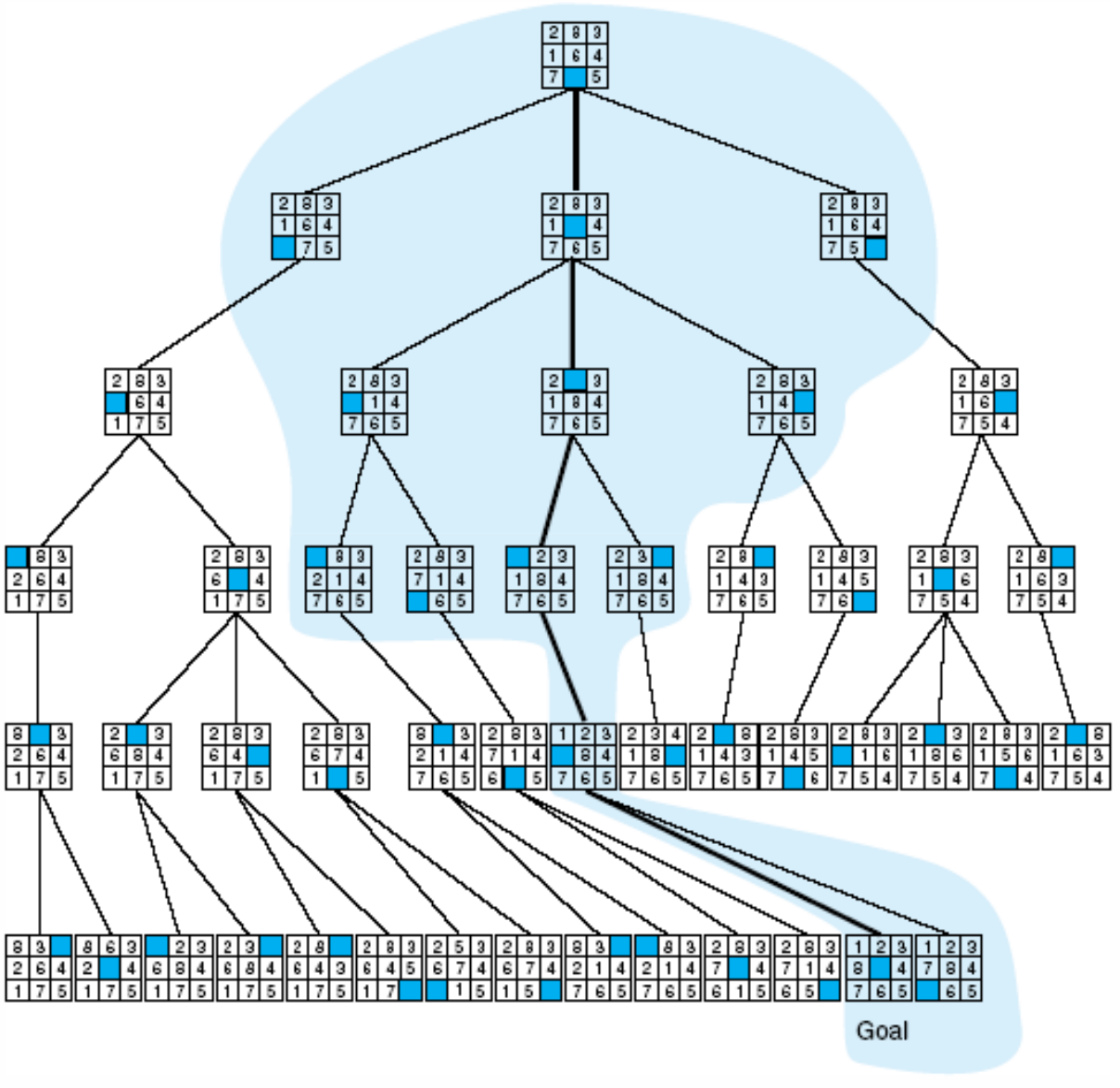
- For path problem, geometric distance (in a plane, it is the straight-line distance).

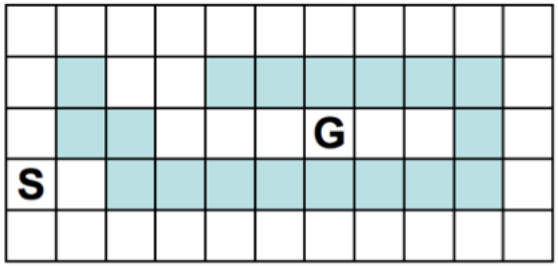
- Pacman dot eating problem – we will discuss some possible h functions in HW, quiz etc.

Best-first search for 8-puzzle using number of misplaced tiles as h

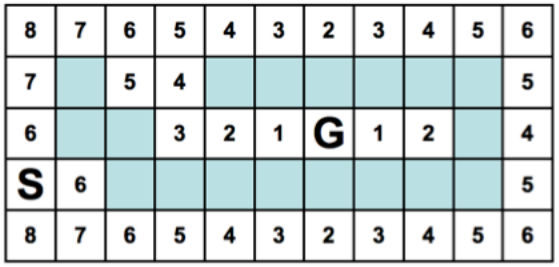


A* with h(n) = number of tiles out of place

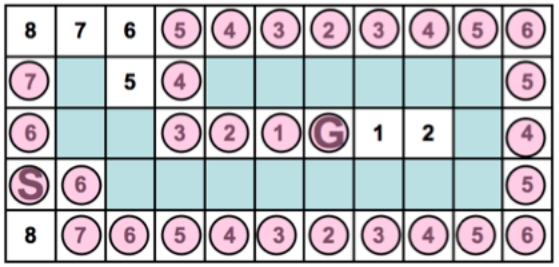




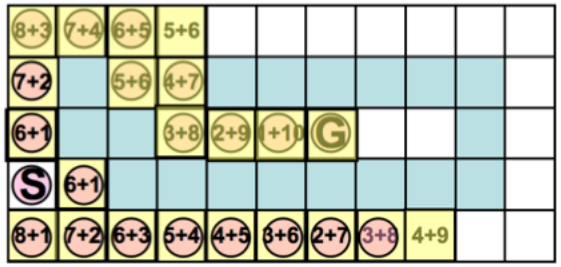
(a)



(b)



(c)



(d)

An example of a search problem where the goal is to get from S to G. (b) The search space, labeled with the Manhattan distance heuristic. (c) The set of nodes expanded by greedy search. The solution is clearly suboptimal. (d) The set of nodes expanded by A∗ search. (Note: the pair of numbers in each square represent the heuristic value h(n) and the distance g(n) from the start

Optimality of A*

Recall:

- A heuristic $h(n)$ is admissible if for every node n, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.

- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic

- All the h functions we presented for various problems are admissible. (e.g. Manhattan distance in 15-puzzle, largest pancake not in place for pancake sorting, Euclidean distance in the case of road maps.)

Key lemma to show optimality of A*

Lemma 9.1

At every step before termination of A*, *there is always a node, say,* n^*, *on* OPEN *with the following properties:*

1. n^* *is on an optimal path to a goal.*
2. A* *has found an optimal path to* n^*.
3. $\hat{f}(n^*) \leq f(n_0)$.

Proof:

$\hat{f}(n^*) = \hat{g}(n^*) + \hat{h}(n^*)$ by definition

$\leq g(n^*) + h(n^*)$ because $\hat{g}(n^*) = g(n^*)$ and $\hat{h}(n^*) \leq h(n^*)$

$\leq f(n^*)$ since $g(n^*) + h(n^*) = f(n^*)$ by definition

$\leq f(n_0)$ because $f(n^*) = f(n_0)$ since n^* is on an optimal path

Optimality of A*

Theorem: Tree search version of A*, when used with admissible h(n), is optimal (under some technical conditions).

s	a state in search space
n	a node in the search tree
n_g	A goal node
$g(n)$	The cost of the path to node n
$h(n)$	The heuristic function evaluated at n
$h^*(n)$	The actual cost of the least-cost path from n to a goal state
$f(n)$	The estimated cost of the least-cost path that goes from the root node through n to a goal, $f(n) = g(n) + h(n)$
$f^*(n)$	The actual cost of the least-cost path that goes from the root node through n to a goal, $f^*(n) = g(n) + h^*(n)$
$Pa(n)$	The parent of node n in the search tree.

Proof:

Suppose h is an admissible heuristic, and n_g is the *first* goal node to be expanded. Since n_g is a goal and h is an admissible heuristic, we have

$$\begin{aligned} 0 &\leq h(n_g) && \text{since heuristic functions are nonnegative} \\ &\leq h^*(n_g) && \text{since } h \text{ is admissible} \\ &= 0 && \text{since } n_g \text{ is a goal} \end{aligned}$$

Hence, $h(n_g) = 0$, and therefore

$$f(n_g) = g(n_g) + h(n_g) = g(n_g). \tag{1}$$

We need to guarantee that the path to n_g is no longer than any potential path through some other unexpanded node. Suppose that (by expanding more nodes) it is possible to reach some other goal node n'_g other than the one n_g chosen by A^*. We will prove that $g(n'_g) \geq g(n_g)$.

There is some unique path from the root to n'_g. Let n be the *first* node on this path that has not yet been expanded. Note therefore that n's parent must have been expanded previously, and therefore n is on the fringe of our search tree at the instant that A^* expanded n_g.

Let $f^*(n)$ denote the minimum cost path to a goal that passes through n. Since n'_g is a goal node, and the path to n'_g passes through n, we must have

$$f^*(n) \leq g(n'_g).$$

So, by admissibility, we have

$$f(n) = g(n) + h(n) \leq g(n) + h^*(n) = f^*(n) \leq g(n'_g). \tag{2}$$

Furthermore, n is on the fringe, and was therefore on the priority queue (along with n_g) at the instant A^* expanded the goal node. But A^* chose to expand n_g rather than n. Since A^* chooses nodes to expand in order of priority, this implies that

$$f(n_g) \leq f(n). \tag{3}$$

Putting together equations (1-3), we get

$$\begin{aligned} g(n_g) &= f(n_g) && \text{by Equation (1)} \\ &\leq f(n) && \text{by Equation (3)} \\ &\leq g(n'_g) && \text{by Equation (2)} \end{aligned}$$

This proves that $g(n'_g) \geq g(n_g)$ for any goal node n'_g other than the one chosen by A^*. This therefore shows that A^* finds a minimum cost path to the goal.

Maze search with different search techniques

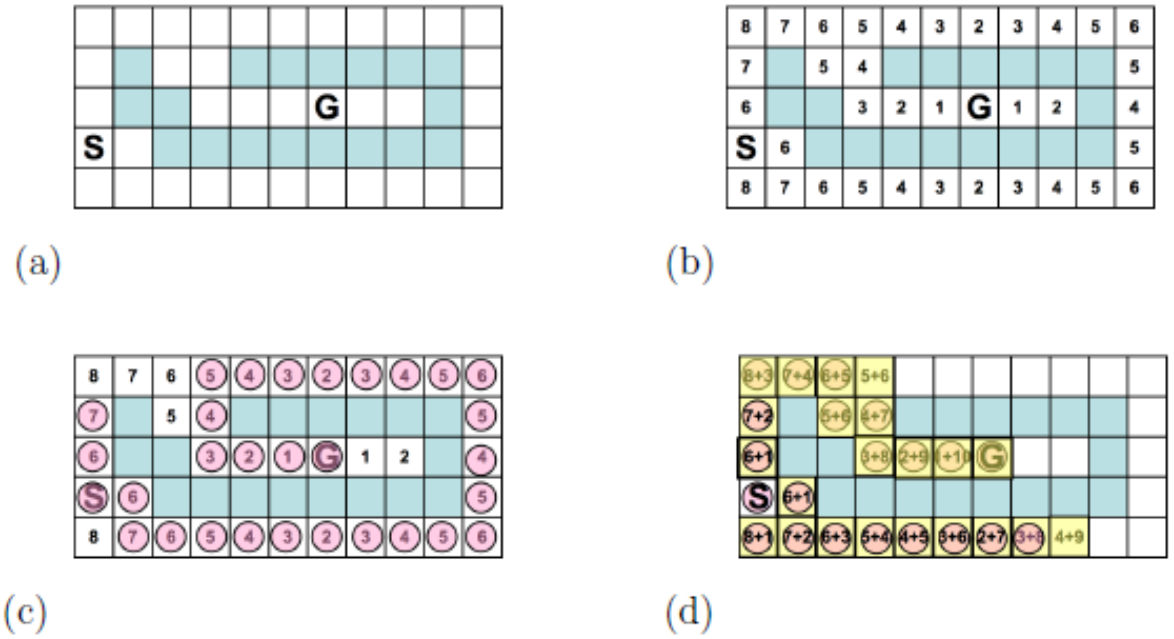


Figure 4: (a) An example of a search problem where the goal is to get from S to G. (b) The search space, labeled with the Manhattan distance heuristic. (c) The set of nodes expanded by best-first search. The solution is clearly suboptimal. (d) The set of nodes expanded by A^* search. (Note: the pair of numbers in each square represent the heuristic value $h(n)$ and the distance already traveled $g(n)$.)

(c) Uses greedy best-first search

DEFINITION

MONOTONICITY

A heuristic function h is monotone if

1. For all states n_i and n_j, where n_j is a descendant of n_i,

   $h(n_i) - h(n_j) \leq cost(n_i,n_j)$,

   where $cost(n_i,n_j)$ is the actual cost (in number of moves) of going from state n_i to n_j.

2. The heuristic evaluation of the goal state is zero, or h(Goal) = 0.

Monotonic h function is also called consistent function.

Monotonic vs. admissible

Fact: A monotonic heuristic is always admissible, but the converse is not always true.

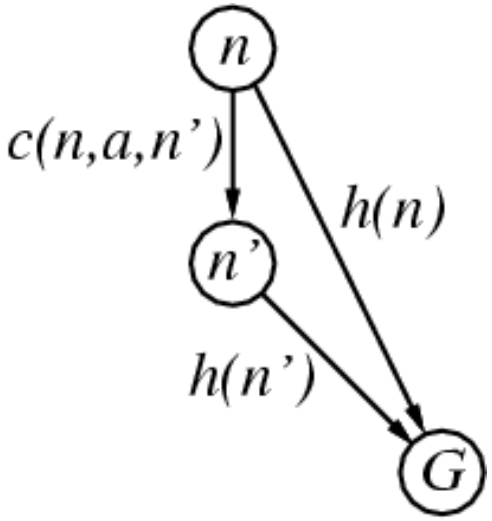
We will first show that the converse is not true.

(A counter-example with only 3 states can be constructed.)

monotonic heuristic - properties

- If *h* is monotonic, we have

$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$



i.e., *f(n)* is non-decreasing along any path.

Call this condition (a). Let condition (b) be the requirement that h(n) = 0 for any goal node n.

Monotonic vs. admissible

Theorem : *If a heuristic function h is monotonic, then it is admissible.*

Proof: Let n_g be an arbitrary goal node, and let n be a node on the path to n_g. Then we have

$$\begin{aligned} g(n) + h(n) &= f(n) \\ &\leq f(n_g) \quad \text{by condition (a)} \\ &= g(n_g) \quad \text{by condition (b).} \end{aligned}$$

By subtracting $g(n)$ from both sides, we get

$$h(n) \leq g(n_g) - g(n).$$

This must hold true for any n_g which is a descendant of n, and in particular the shortest one. Hence, $h(n) \leq f^*(n) - g(n) = h^*(n)$.

Monotonic heuristic – another simple fact

Question: Suppose we choose h(n) = 0 for all n. For what search problems is this a monotonic heuristic?

Answer: h(n) <= h(m) + w(n, m) for all n, m where the edge weight from n to m is w(n, m).

→ w(n, m) >= 0, i.e., all edge weights must be positive.

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)

  then h_2 dominates h_1

  h_2 is better for search

Claim: If h_2 dominates h_1, then every node expanded by h_2 is also expanded by h_1

Effectiveness of A* Search Algorithm

Average number of nodes expanded

d	IDS	A*(h1)	A*(h2)
2	10	6	6
4	112	13	12
8	6384	39	25
12	364404	227	73
14	3473941	539	113
20	----------	7276	676

Study by Rina Dechter of UCI.

Average over 100 randomly generated 8-puzzle problems

h1 = number of tiles in the wrong position

h2 = sum of Manhattan distances

Relaxed problems

- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution

Summary

- In practice we often want the goal with the minimum cost path.
- Exhaustive search is impractical except on small problems.
- Heuristic estimates of the path cost from a node to the goal can be efficient in reducing the search space.
- The A* algorithm combines all of these ideas with admissible heuristics (which underestimate), guaranteeing optimality.
- Properties of heuristics:
  - admissibility, monotonicity, dominance, accuracy