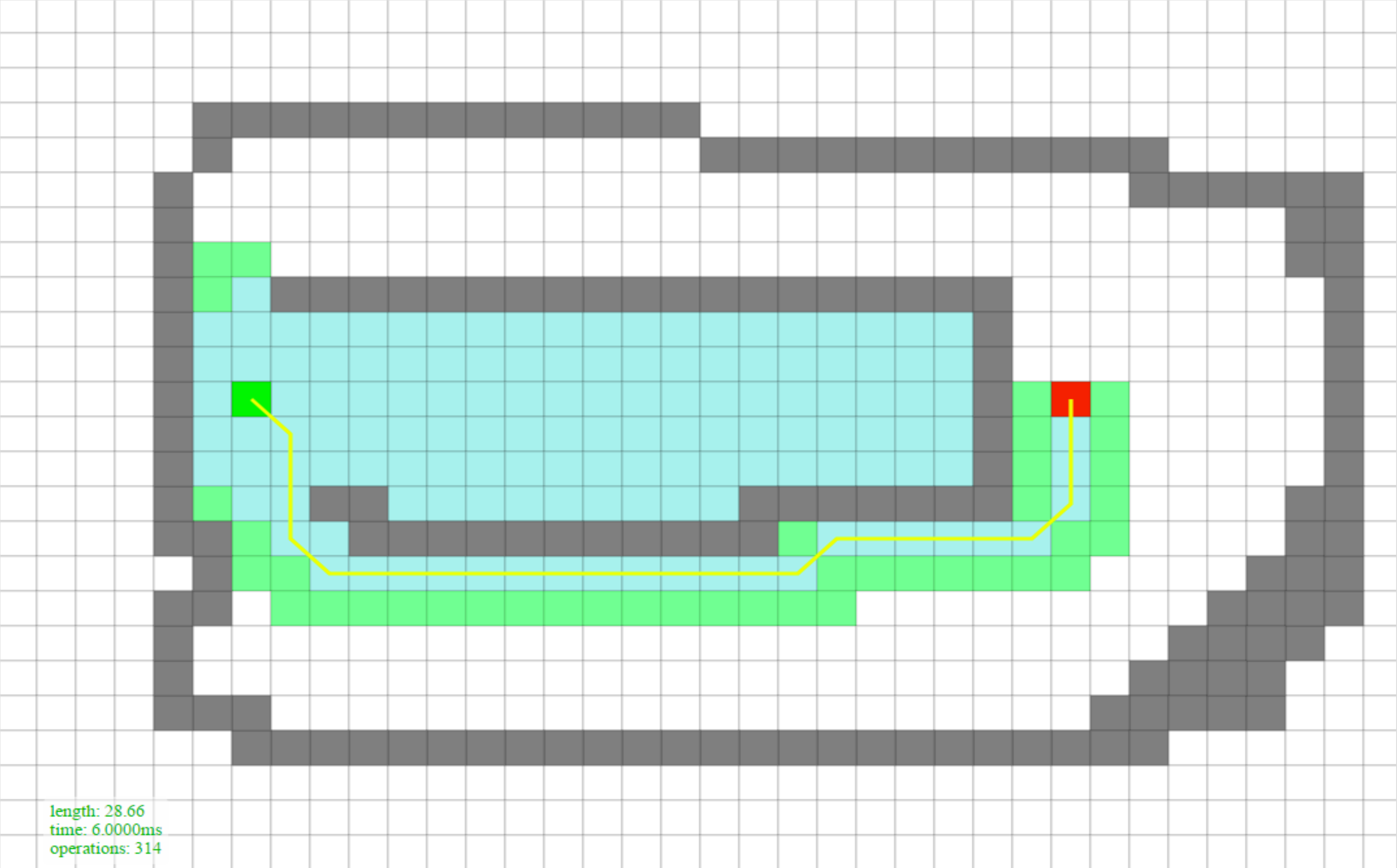
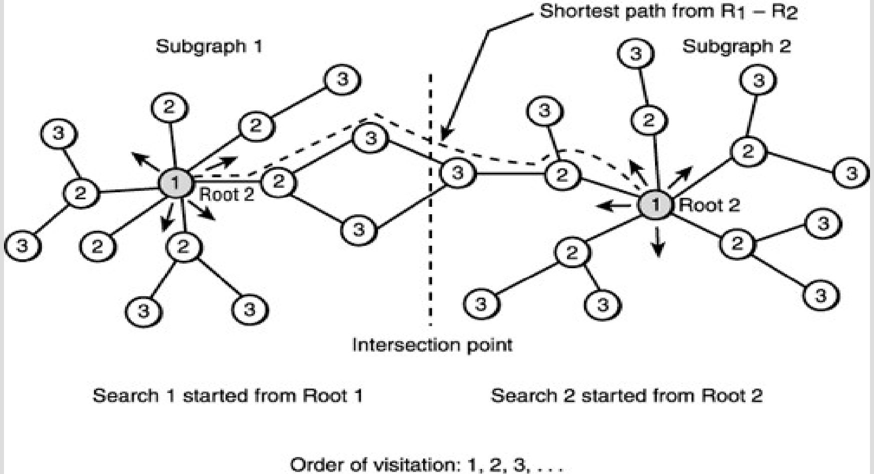
# Informed Search (Ch. 3.5-3.6)



# Bidirectional search

Bidirectional search starts from both the goal and start (using BFS) until the trees meet

This is better as $2*(b^{d/2}) < b^d$
(the space is much worse than IDS, so only applicable to small problems)



# Summary of algorithms Fig. 3.21, p. 91

| Criterion | Breadth-First | Uniform-Cost | Depth-First | Depth-Limited | Iterative Deepening DLS | Bidirectional (if applicable) |
|---|---|---|---|---|---|---|
| Complete? | Yes[a] | Yes[a,b] | No | No | Yes[a] | Yes[a,d] |
| Time | $O(b^d)$ | $O(b^{\lfloor 1+C^*/\varepsilon \rfloor})$ | $O(b^m)$ | $O(b^l)$ | $O(b^d)$ | $O(b^{d/2})$ |
| Space | $O(b^d)$ | $O(b^{\lfloor 1+C^*/\varepsilon \rfloor})$ | $O(bm)$ | $O(bl)$ | $O(bd)$ | $O(b^{d/2})$ |
| Optimal? | Yes[c] | Yes | No | No | Yes[c] | Yes[c,d] |

Generally the preferred uninformed search strategy

There are a number of footnotes, caveats, and assumptions.
See Fig. 3.21, p. 91.

[a] complete if b is finite

[b] complete if step costs $\geq \varepsilon > 0$

[c] optimal if step costs are all identical
(also if path cost non-decreasing function of depth only)

[d] if both directions use breadth-first search
(also if both directions use uniform-cost search with step costs $\geq \varepsilon > 0$)

# Informed search

In uninformed search, we only had the node information (parent, children, cost of actions)

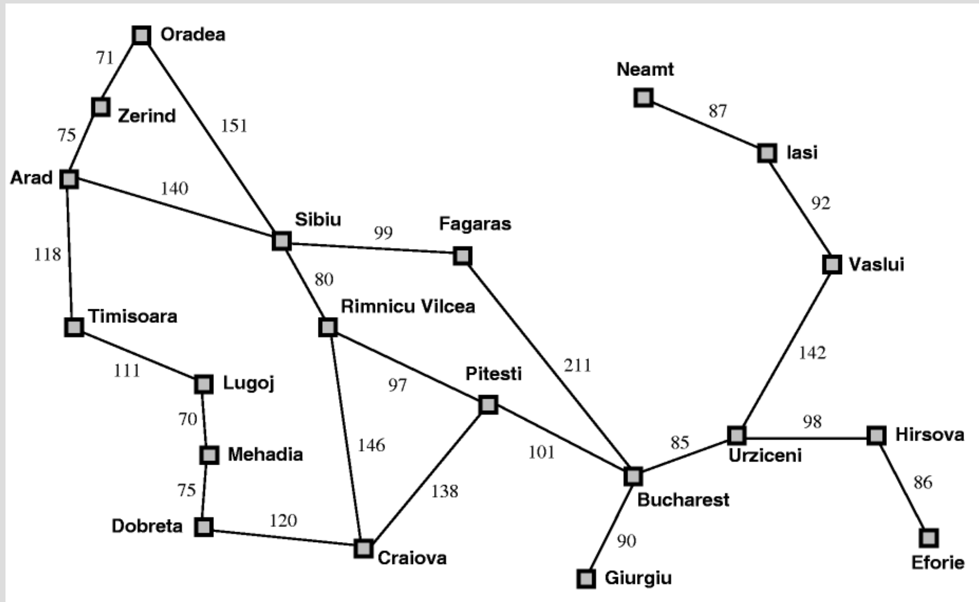
Now we will assume there is some additional information, we will call a <u>heuristic</u> that estimates the distance to the goal

Previously, we had no idea how close we were to goal, simply how far we had gone already

# Greedy best-first search

To introduce heuristics, let us look at the tree version of <u>greedy best-first search</u>

This search will simply repeatedly select the child with the lowest heuristic(cost to goal est.)




Straight−line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Greedy best-first search

This finds the path: Arad -> Sibiu -> Fagaras -> Bucharest

However, this greedy approach is not optimal, as that is the path: Arad -> Sibiu -> Rimmicu Vilcea -> Pitesti -> Bucharest

In fact, it is not guaranteed to converge (if a path reaches a dead-end, it will loop infinitely)

# A*

We can combine the distance traveled and the estimate to the goal, which is called A* (a star)

The method goes: (red is for "graphs")

initialize explored={}, fringe={[start,f(start)]}

1. Choose C = argmin(f-cost) in fringe
2. Add or update C's children to fringe, with associated f-value, remove C from fringe
3. Add C to explored
4. Repeat 1. until C == goal or fringe empty

# A*

f(node) = g(node) + h(node) heuristic

(estimate to-goal distance)

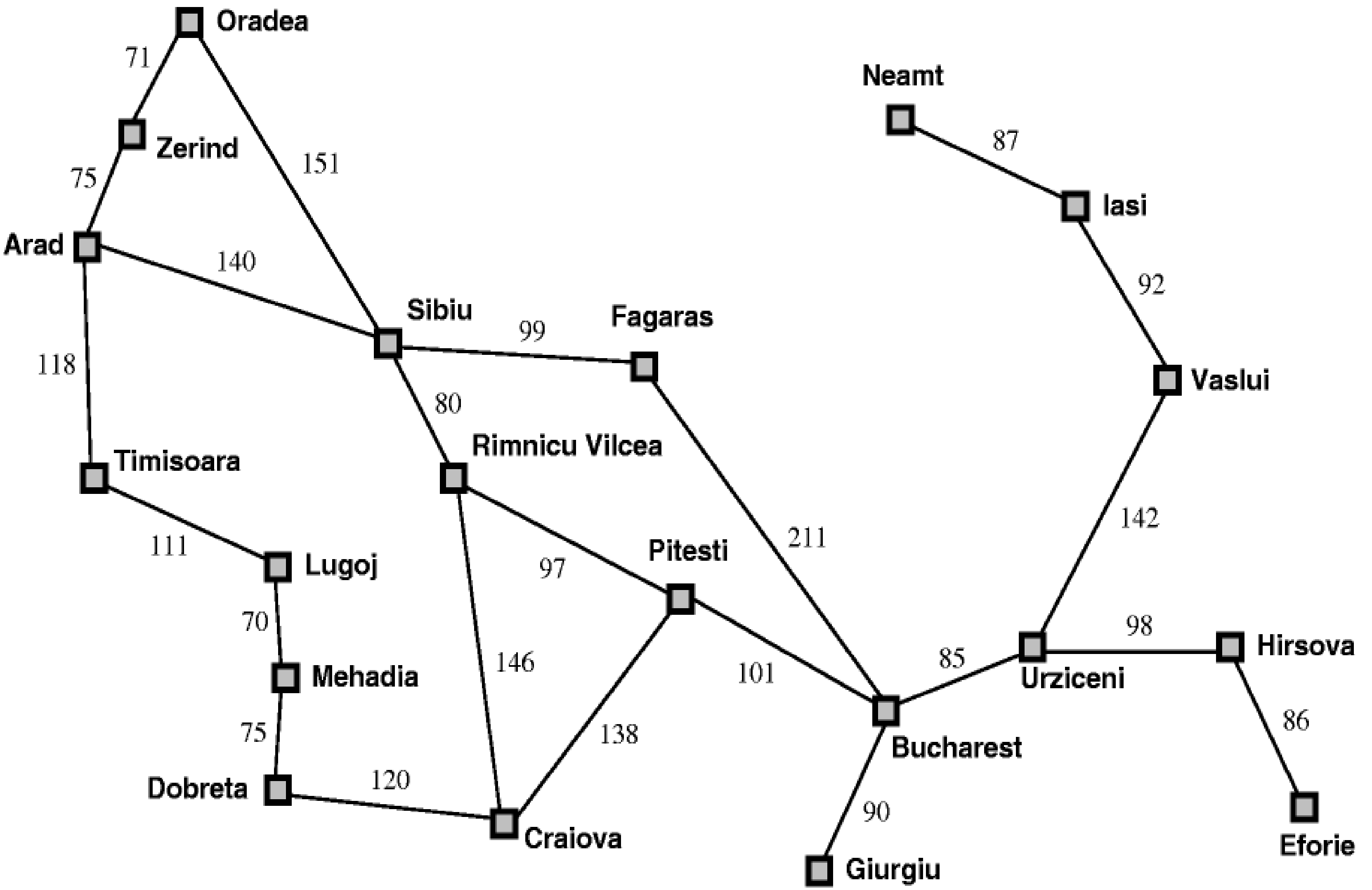
distance gone (traveled) so far

total cost estimate

We will talk more about what heuristics are good or should be used later

Priority queues can be used to efficiently store and insert states and their f-values into the fringe

# A*




Straight–line distance to Bucharest

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# A*

Step: Fringe (argmin)

0: [Arad, 366]

1: [Zerind, 75+374],[Sibu, 140+253],[Timisoara, 118+329]

1: [Zerind, 449], [Sibu, 393], [Timisoara, 447]

2: [Fagaras, 140+99+176], [Rimmicu Vilcea, 140+80+193], [Zerind, 449], [Timisoara, 447]

2: [Fagaras, 415], [Rimmicu Vilcea, 413], [Zerind, 449], [Timisoara, 447]

3: [Craiova, 140+80+146+160], [Pitesti, 140+80+97+100], [Fagaras, 415], [Zerind, 449], [Timisoara, 447]

3: [Craiova, 526], [Pitesti, 417], [Fagaras, 415], [Zerind, 449], [Timisoara, 447]

4: ... on next slide

# A*

4: [Bucharest, 140+99+211+0], [Craiova, 526], [Pitesti, 417], [Zerind, 449], [Timisoara, 447]

4: [Bucharest, 450], [Craiova, 526], [Pitesti, 417], [Zerind, 449], [Timisoara, 447]

5: [Craiova from Pitesti, 140+80+97+138+160], [Bucharest from Pitesti, 140+80+97+101+0], [Bucharest from Fagaras, 450], [Timisoara, 447], [Craiova from Rimmicu Vilcea, 526], [Zerind, 449]

5: [Craiova from Pitesti, 615], [Bucharest from Pitesti, 418], [Bucharest from Fagaras, 450], [Timisoara, 447], [Craiova from Rimmicu Vilcea, 526], [Zerind, 449]

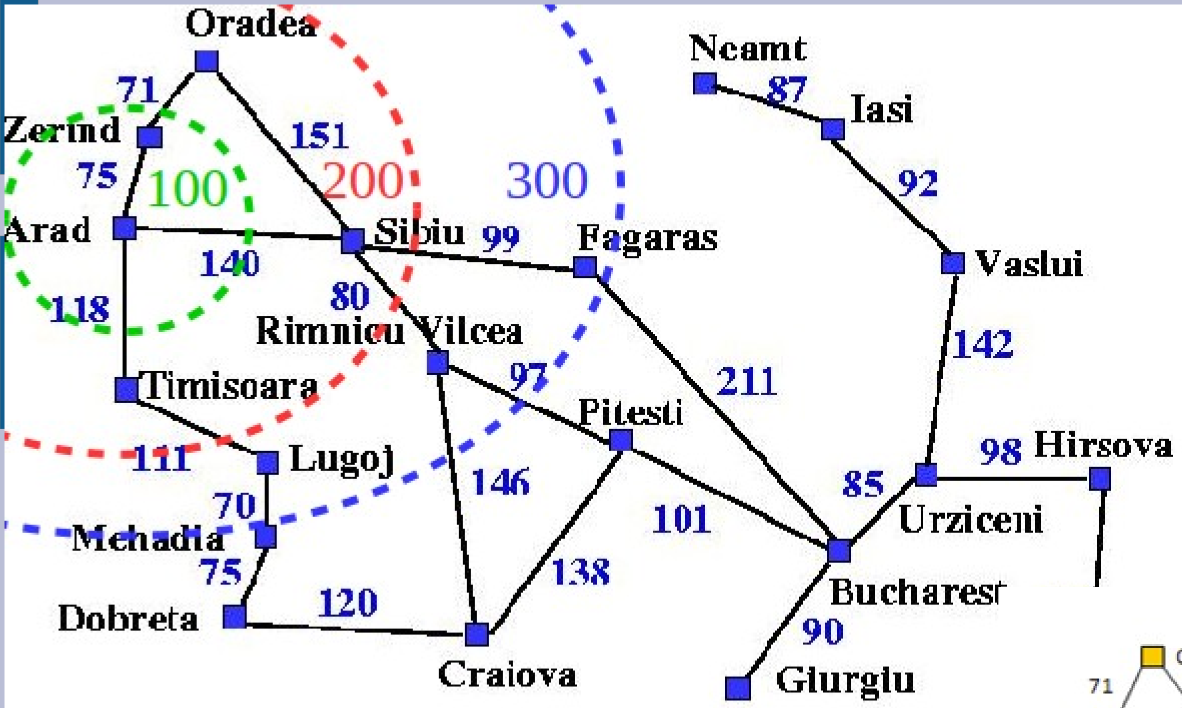
# A*

You can choose multiple heuristics (more later) but good ones skew the search to the goal

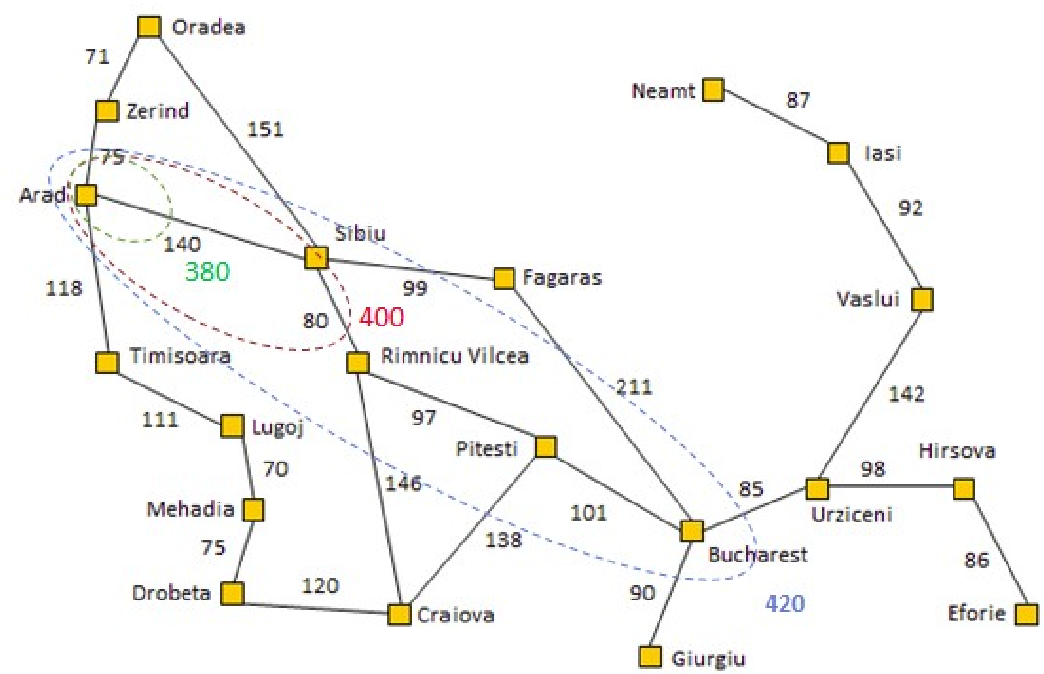
You can think circles based on f-cost:
-if h(node) = 0, f-cost are circles
-if h(node) = very good, f-cost long and thin ellipse

This can also be though of as topographical maps (in a sense)

# A*




h(node) = 0 (bad heuristic, no goal guidance)




h(node) = straight line distance (good heuristic)

# A*

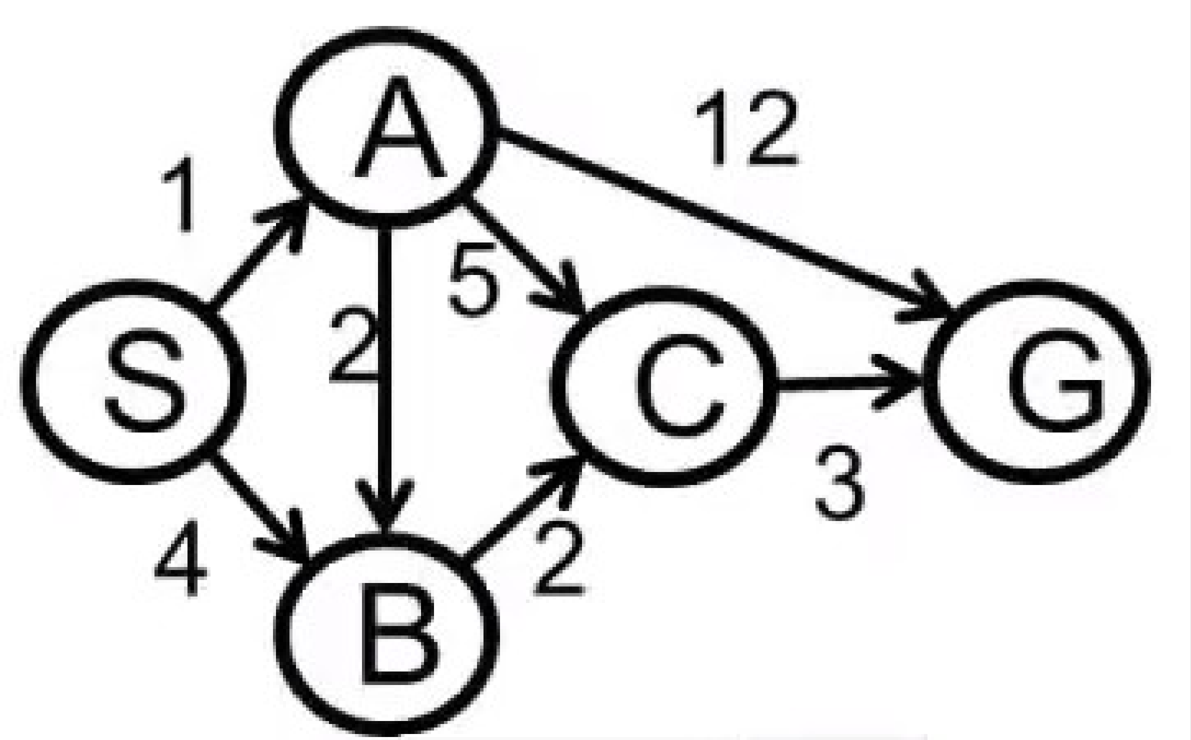
Good heuristics can remove "bad" sections of the search space that will not be on any optimal solution (called pruning)

A* is optimal and in fact, no optimal algorithm could expand less nodes (optimally efficient)

However, the time and memory cost is still exponential (memory tighter constraint)

# A*

You do it!




| State | H |
| --- | --- |
| S | 7 |
| A | 6 |
| B | 2 |
| C | 1 |
| G | 0 |

Arrows show children (easier for you)

(see: https://www.youtube.com/watch?v=sAoBeujec74)

# Iterative deepening A*

You can combine iterative deepening with A*

Idea:

1. Run DFS in IDS, but instead of using depth as cutoff, use f-cost
2. If search fails to find goal, increase f-cost to next smallest seen value (above old cost)

Pros: Efficient on memory

Cons: Large (LARGE) amount of re-searching

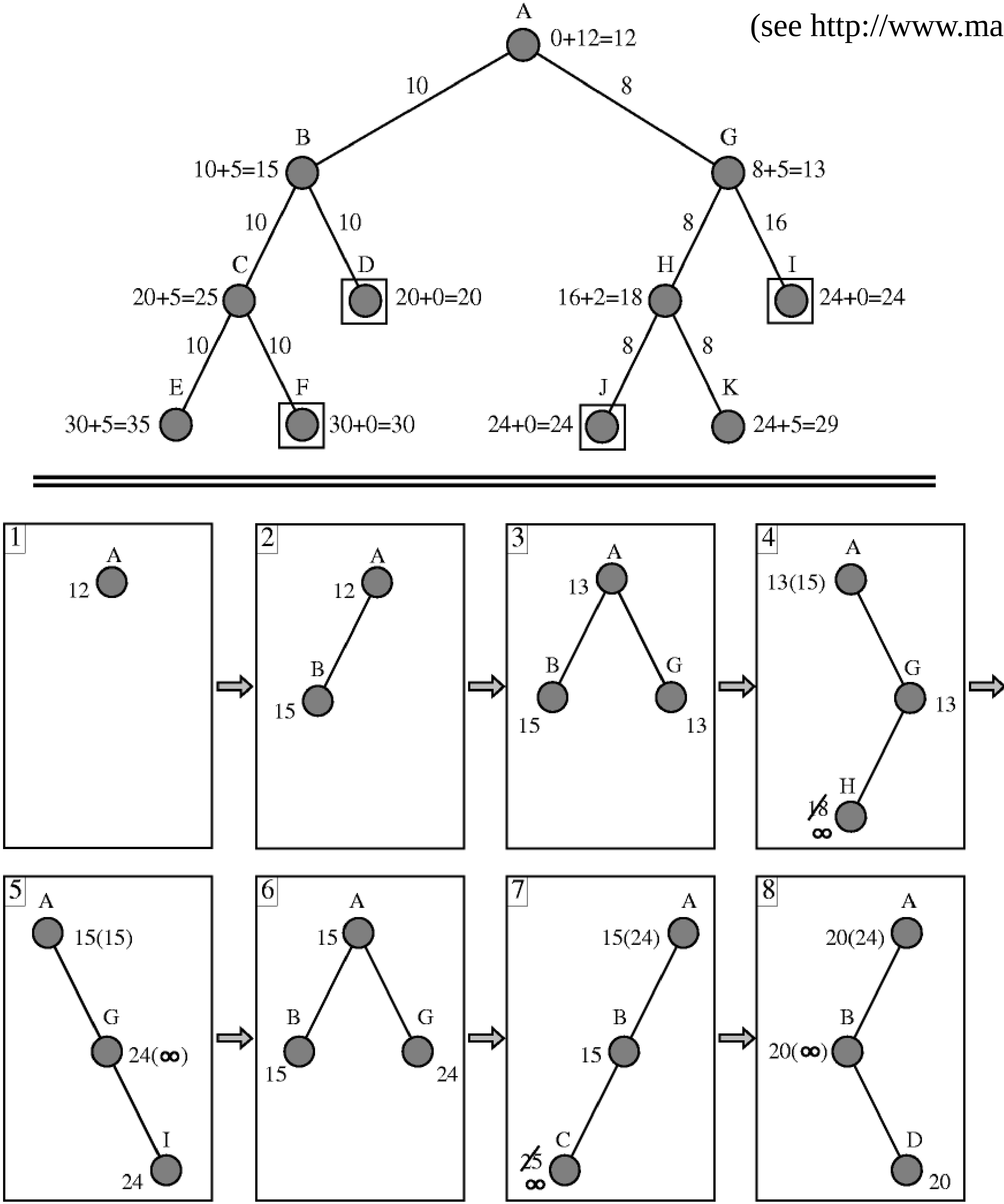
# SMA*

One fairly straight-forward modification to A* is <u>simplified memory-bounded A*</u> (SMA*)

Idea:

1. Run A* normally until out of memory
2. Let C = argmax(f-cost) in the leaves
3. Remove C but store its value in the parent (for re-searching)
4. Goto 1

(see http://www.massey.ac.nz/~mjjohnso/notes/59302/l04.html)




Here assume you can only hold at most 3 nodes in memory

# SMA*

SMA* is nice as it (like A*) find the optimal solution while keeping re-searching low (given your memory size)

IDA* only keeps a single number in memory, and thus re-searches many times (inefficient use of memory)

Typically there is some time to memory trade-off

# Heuristics

However, for A* to be optimal the heuristic h(node) needs to be...

For trees: <u>admissible</u> which means:

  h(node) $\leq$ optimal path from h to goal

  (i.e. h(node) is an underestimate of cost)

For graphs: <u>consistent</u> which means:

  h(node) $\leq$ cost(node to child) + h(child)

  (i.e. triangle inequality holds true)

  (i.e. along any path, f-cost increases)

# Heuristics

Consistent heuristics are always admissible
-Requirement: h(goal) = 0

Admissible heuristics **might not** be consistent

A* is guaranteed to find optimal solution
if the heuristic is admissible for trees
(consistent for graphs)

# Heuristics

In our example, the h(node) was the straight line distance from node to goal

This is an underestimate as physical roads cannot be shorter than this
(it also satisfies the triangle inequality)

Thus this heuristic is admissible
(and consistent)

# Heuristics

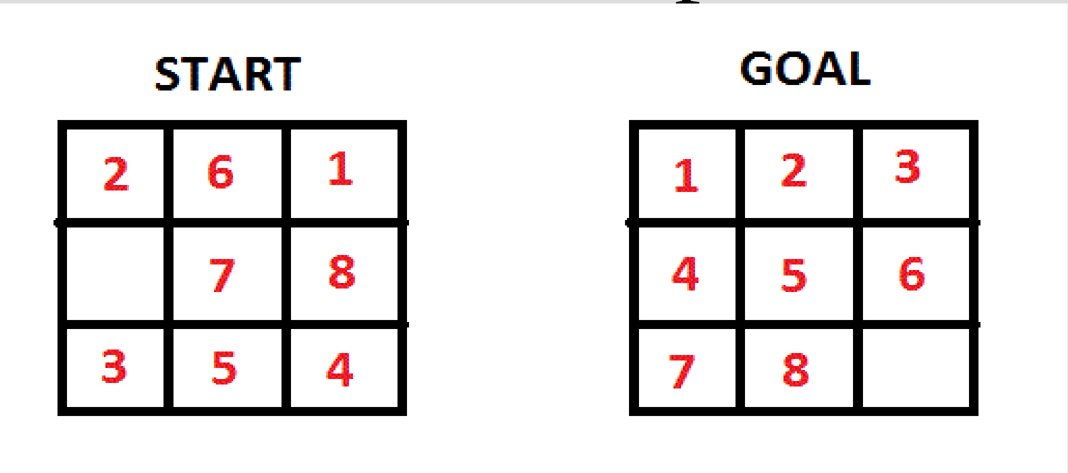
The straight line cost works for distances in the physical world, do any others exist?

One way to make heuristics is to relax the problem (i.e. simplify in a useful way)

The optimal path cost in the relaxed problem can be a heuristic for the original problem (i.e. if we were not constrained to driving on roads, we could take the straight line path)

# Heuristics

Let us look at 8-puzzle heuristics:




The rules of the game are:

- You can swap any square with the blank

Relaxed rules:

1. Teleport any square to any destination
2. Move any square 1 space (overlapping ok)

# Heuristics

1. Teleport any square to any destination
Optimal path cost is the number of mismatched squares (blank included)

2. Move any square 1 space (overlapping ok)
Optimal path cost is Manhattan distance for each square to goal summed up

Which ones is better? (Note: these optimal solutions in relaxed need to be computed fast)

# Heuristics

h1 = mismatch count
h2 = number to goal difference sum

| | Search Cost | | | Effective Branching Factor | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | $A^*(h_1)$ | $A^*(h_2)$ | IDS | $A^*(h_1)$ | $A^*(h_2)$ |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 364404 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | 3473941 | 539 | 113 | 2.83 | 1.44 | 1.23 |
| 16 | – | 1301 | 211 | – | 1.45 | 1.25 |
| 18 | – | 3056 | 363 | – | 1.46 | 1.26 |
| 20 | – | 7276 | 676 | – | 1.47 | 1.27 |
| 22 | – | 18094 | 1219 | – | 1.48 | 1.28 |
| 24 | – | 39135 | 1641 | – | 1.48 | 1.26 |

# Heuristics

The real branching factor in the 8-puzzle:

- 2 if in a corner
- 3 if on a side
- 4 if in the center
- (Thus larger “8-puzzles” tend to 4)

An effective branching factor finds the “average” branching factor of a tree (smaller branching = less searching)

# Heuristics

The effective branching factor is defined as:

$$N = b^* + (b^*)^2 + (b^*)^3 + ... + (b^*)^d$$

... where:

N = the number of nodes (i.e. size of fringe + size of explored if tree search)

$b^*$ = effective branching factor (to find)

d = depth of solution

No easy formula, but can approximate:

$$N^{1/(d+1)} \leq b^* \leq N^{1/d}$$

# Heuristics

h2 has a better branching factor than h1, and this is not a coincidence...

h2(node) $\geq$ h1(node) for all nodes, thus we say h2 dominates h1 (and will thus perform better)

If there are multiple non-dominating heuristics: h1, h2... Then h* = max(h1, h2, ...) will dominate h1, h2, ... and will also be admissible /consistent if h1, h2 ... are as well

# Heuristics

If larger is better, why do we not just set $h(node) = 9001$?

# Heuristics

If larger is better, why do we not just set h(node) = 9001?

This would (probably) not be admissible...

If h(node) = 0, then you are doing the uninformed uniform cost search

If h(node) = optimal_cost(node to goal) then will ONLY explore nodes on an optimal path

# Heuristics

You cannot add two heuristics ($h^* = h1 + h2$), unless there is no overlap (i.e. h1 cost is independent of h2 cost)

For example, in the 8-puzzles:

h3: number of 1, 2, 3, 4 that are misplaced

h4: number of 5, 6, 7, 8 that are misplaced

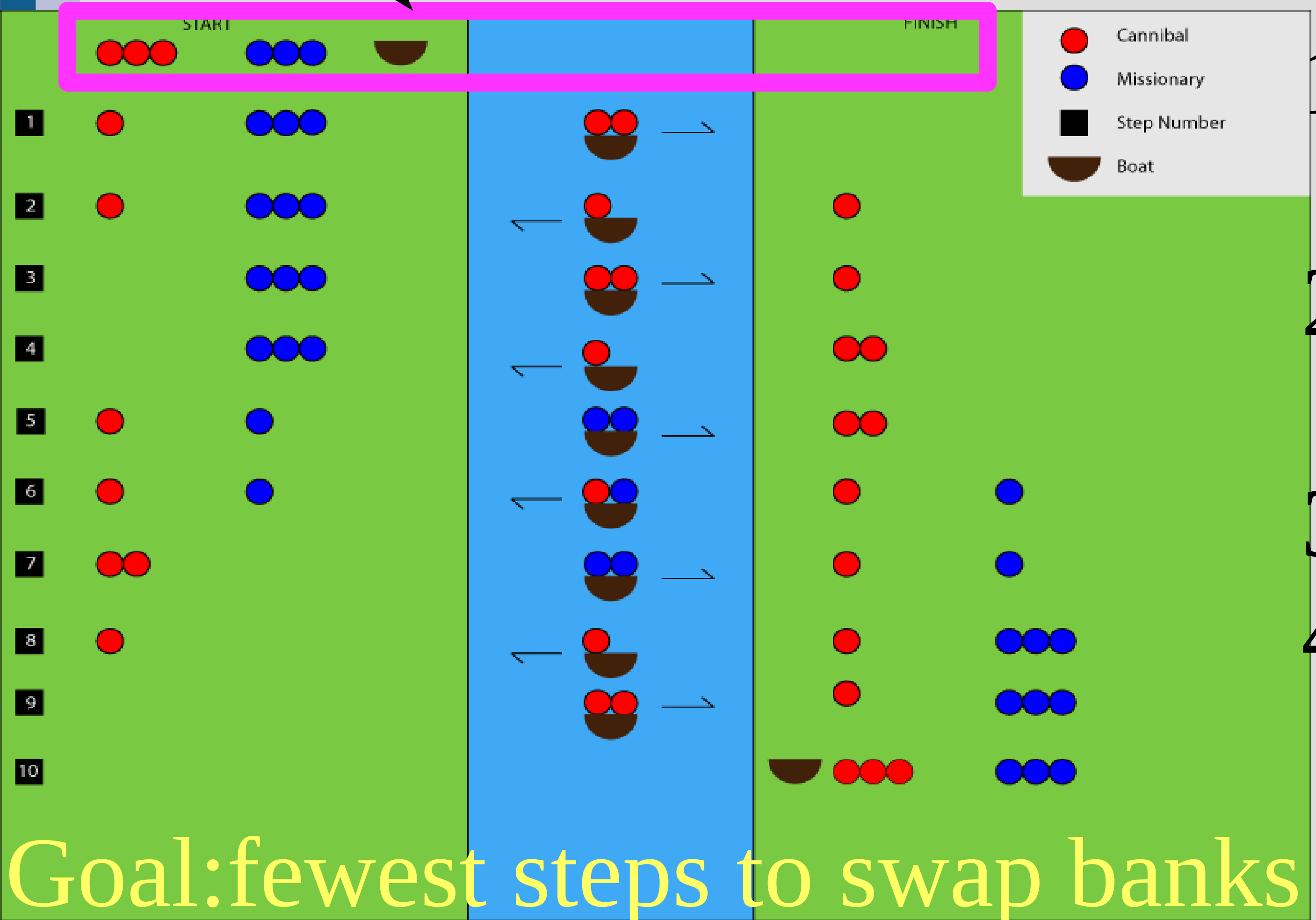
There is no overlap, and in fact:

$h3 + h4 = h1$ (as defined earlier)

# Heuristics

Cannibals & missionaries problem:

initial




Goal:fewest steps to swap banks

Rules:

1. Either bank: m>=c, if c>0
2. 2 ppl in boat max
3. Start: 3m & 3c
4. Need 1 in boat to move

# Heuristics

What relaxation did you use? (sample)

Make a heuristic for this problem

Is the heuristic admissible/consistent?

# Heuristics

What relaxation did you use? (sample)

Remove needing person in boat to move

Make a heuristic for this problem

h1 = [num people wrong bank]

as you can move 2 people across in 2 steps

Is the heuristic admissible/consistent?

YES! The point of relaxing guarantees admissibility!