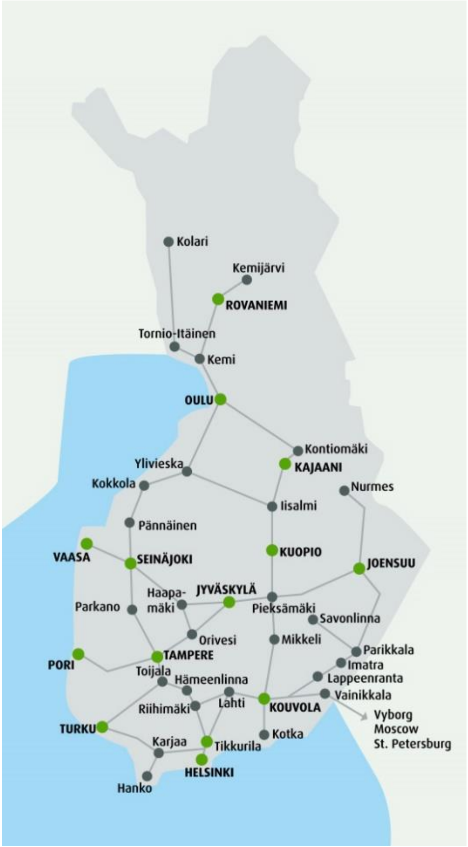
Tampere University

Solving Problems by Searching

CHAPTER 3 IN THE TEXTBOOK

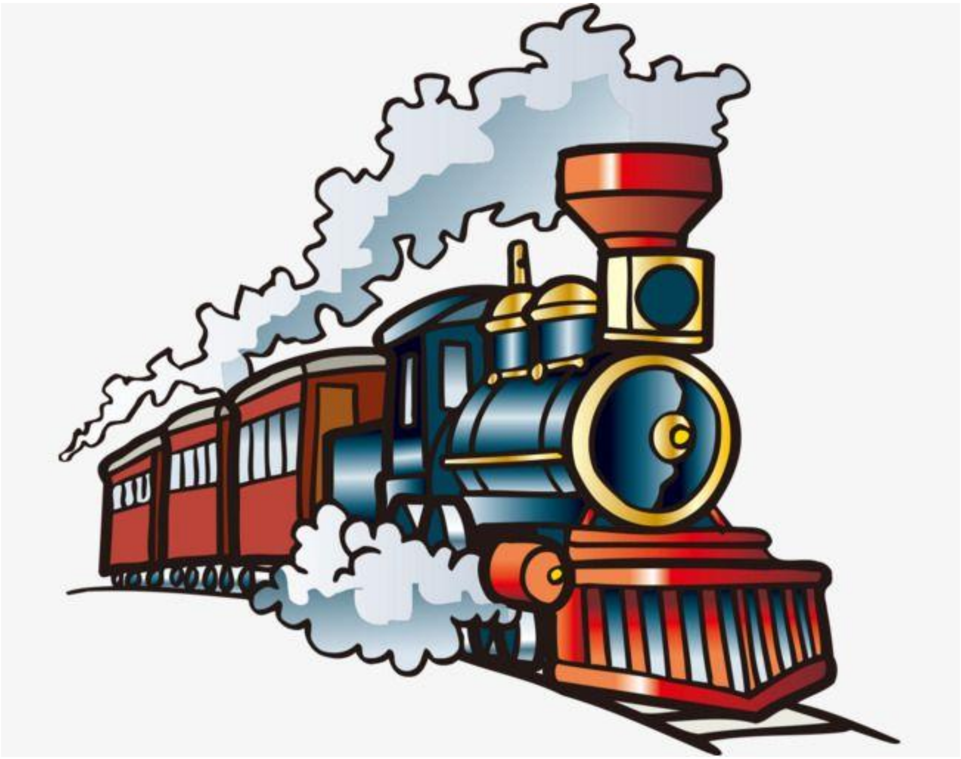
Graph searching



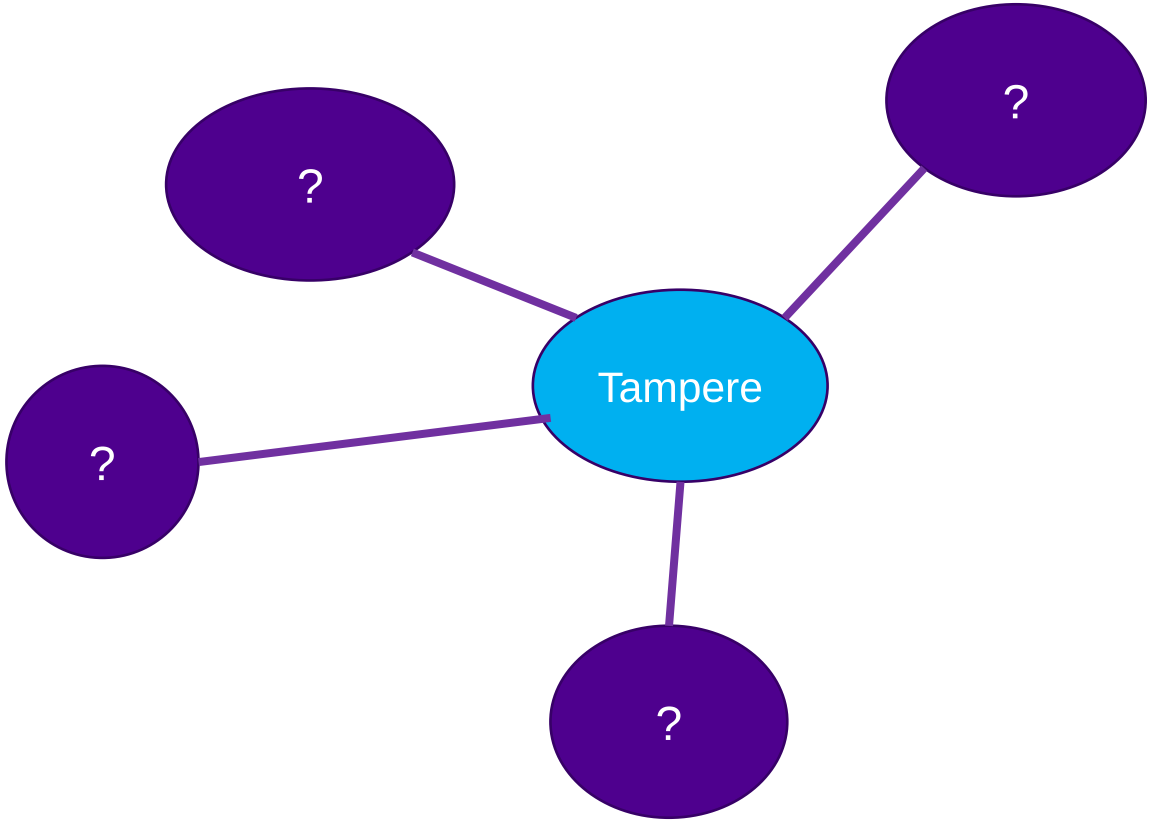
- You want to visit Kolari by train for a nice ski vacation
- Starting from Tampere is it possible at all?
- Which route is most economical in terms of distance, in time, money, …?
- Routes are represented as a graph of cities
- Estimated times (distances) between cities might be given

Tampere University

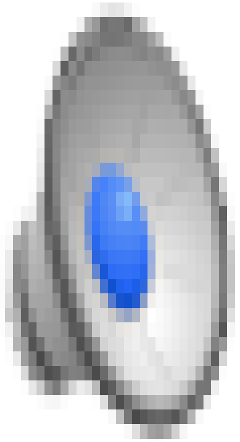
- **Start state** for our Kolari trip is Tampere train station
- The **goal state** is Kolari train station
- Primary interest is to find a way to reach Kolari
- Secondary interest is to find the most economical route to Kolari
- The **cost** of a path is the sum city costs en route
- Each station may have several **successors** (neighboring cities)
- Depending on our level of knowledge there are different search options



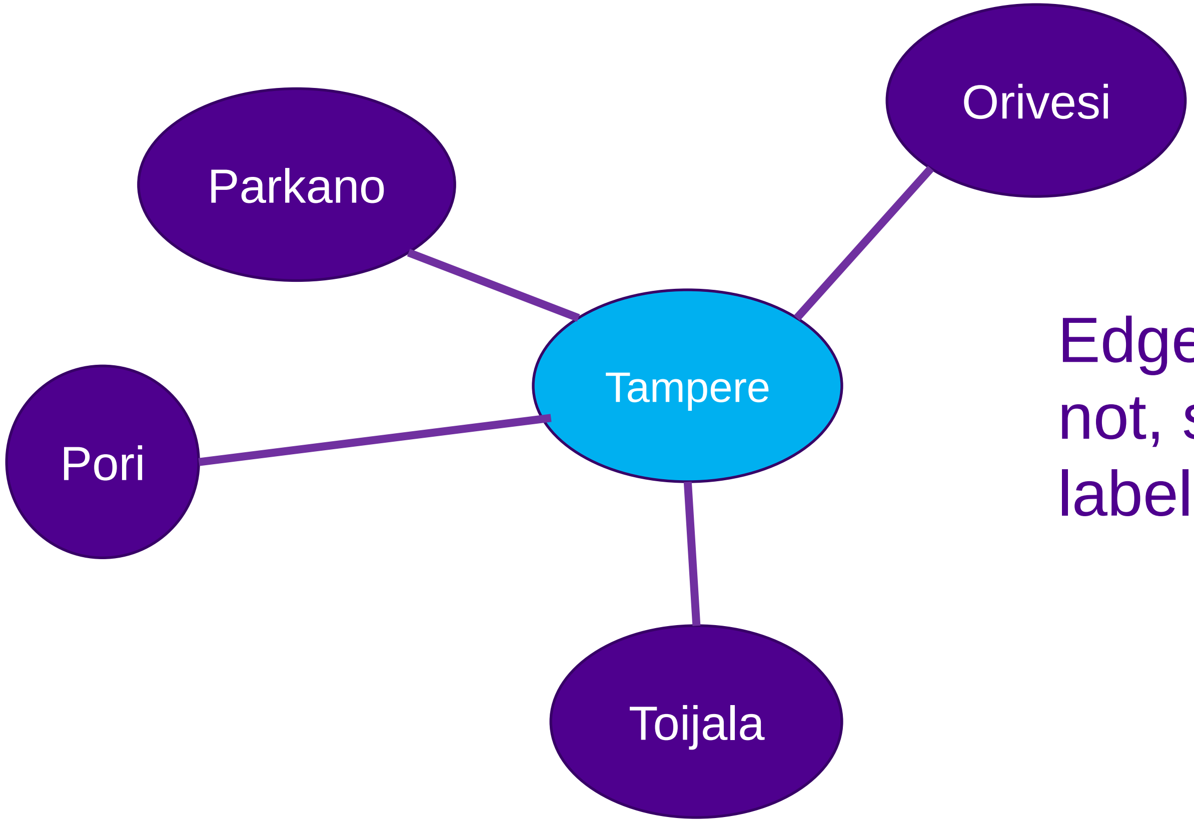
Unknown environment



Destination of trains not given

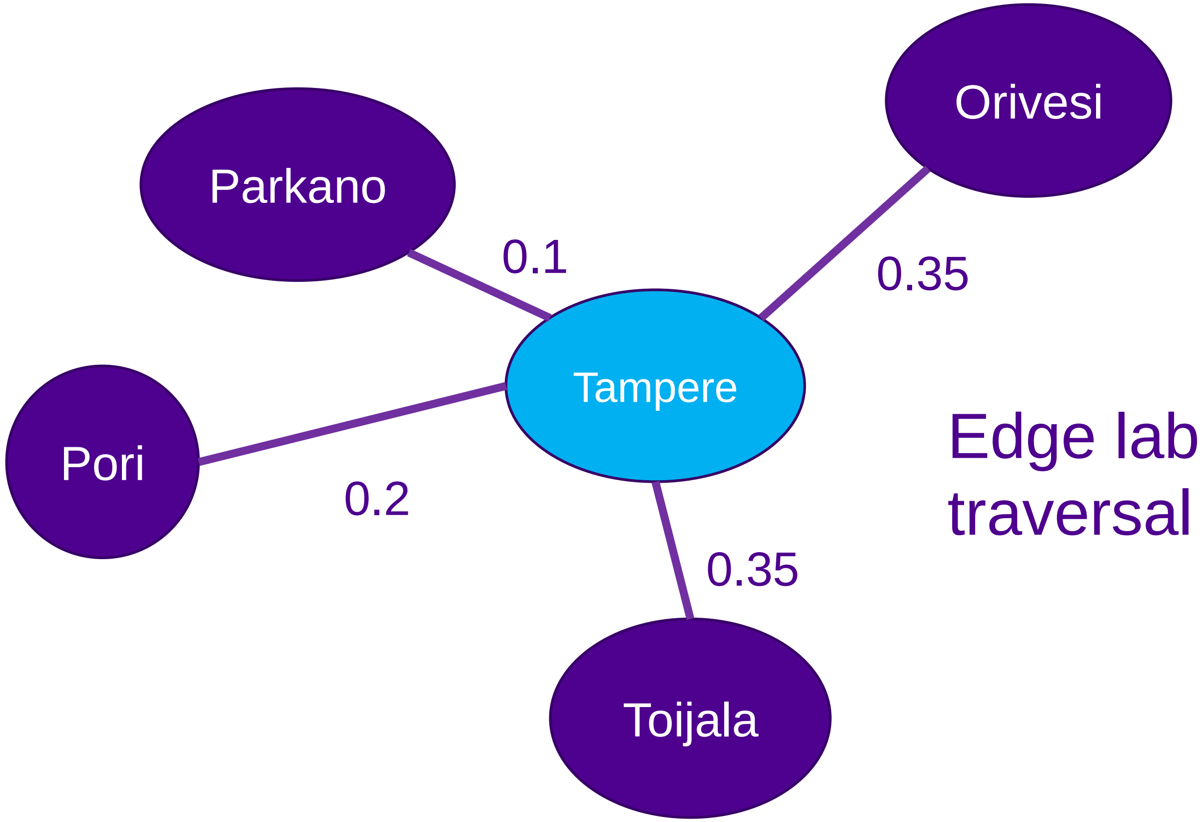


Known deterministic environment

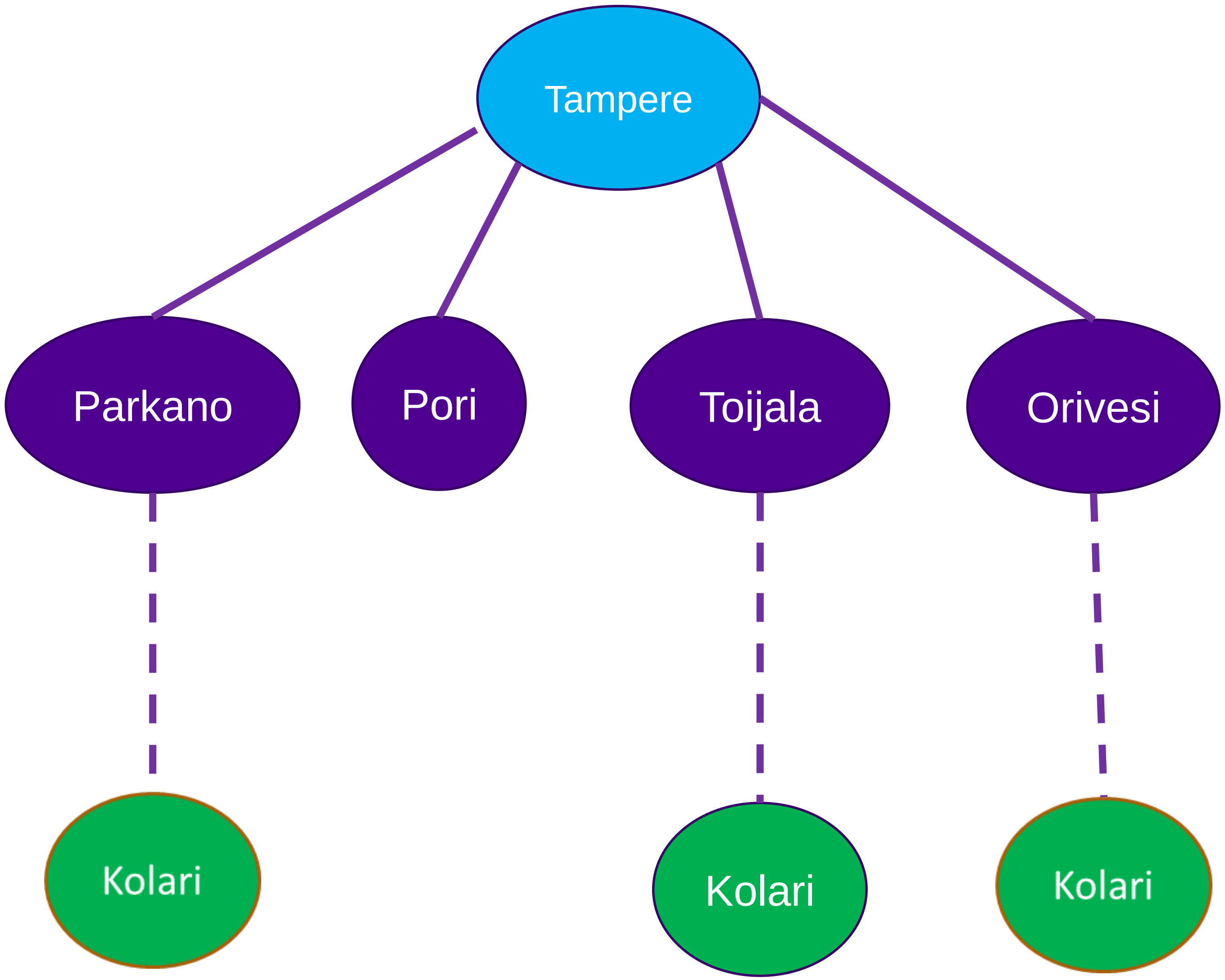


Edges labeled or not, semantics of labels

Nondeterministic environment



Edge labels are traversal probabilities


Tampere
Parkano
Pori
Toijala
Orivesi
Kolari
Kolari
Kolari

Exhaustive search — a.k.a. Brute-force

- Search spaces in general (real world) are huge
- The worst setting is that we have to examine all possible states of the search space (exhaust it)
- Systematically go through the possible states of the world
- Entails keeping track of visited states to avoid looping
- This is particularly true when we aim at optimizing some objective
- Test the satisfiability of a logical formula

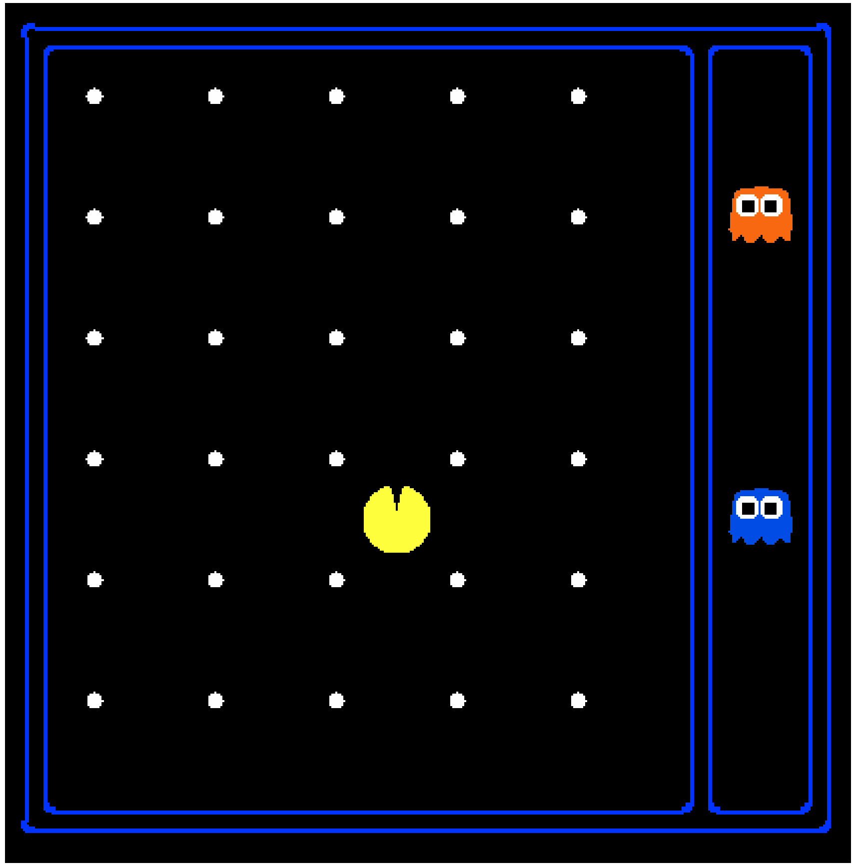
$$(x_1 \vee \neg x_2 \vee x_5) \wedge x_3 \vee \neg x_4$$
$$x_1 = T, x_2 = F, \ldots$$

- If the state space is large, exhaustive search is unavoidably slow (in asymptotic sense)

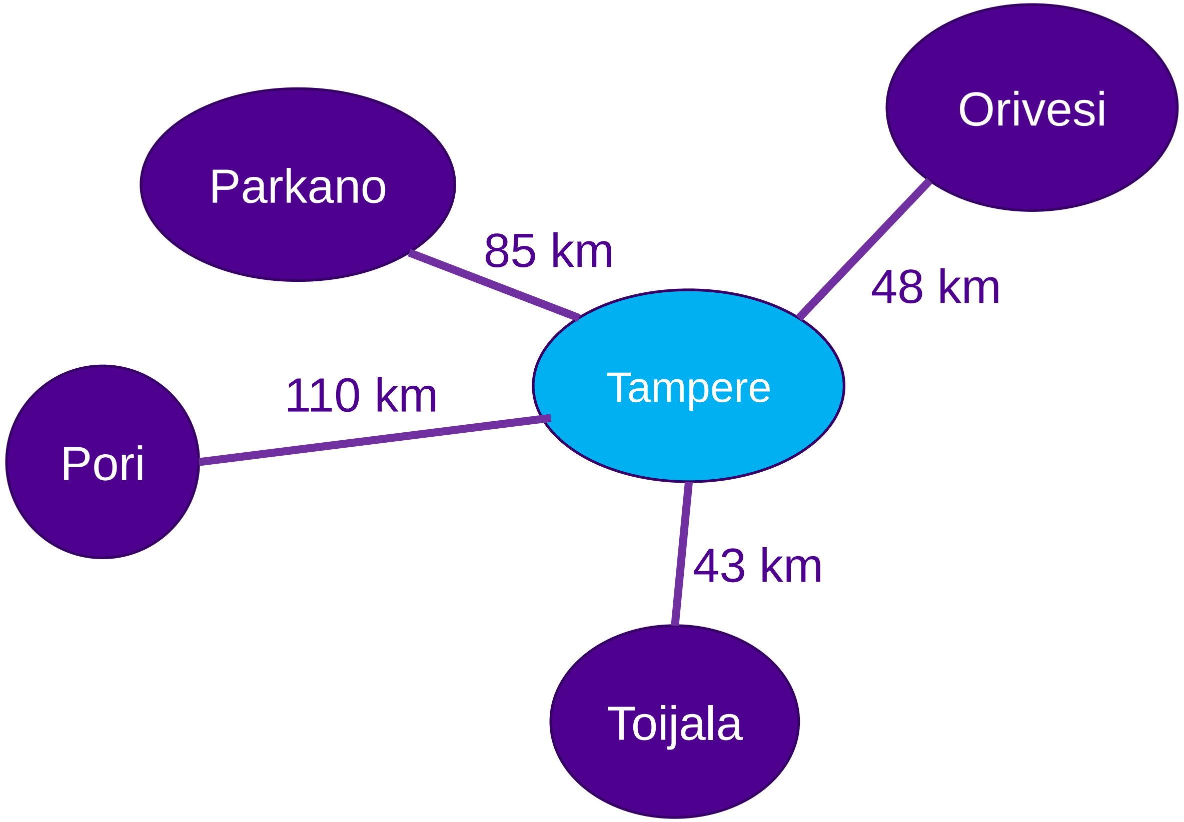
State Space Sizes

- World state:
  - Agent positions: 120
  - Food count: 30
  - Ghost positions: 12
  - Agent facing: NSEW

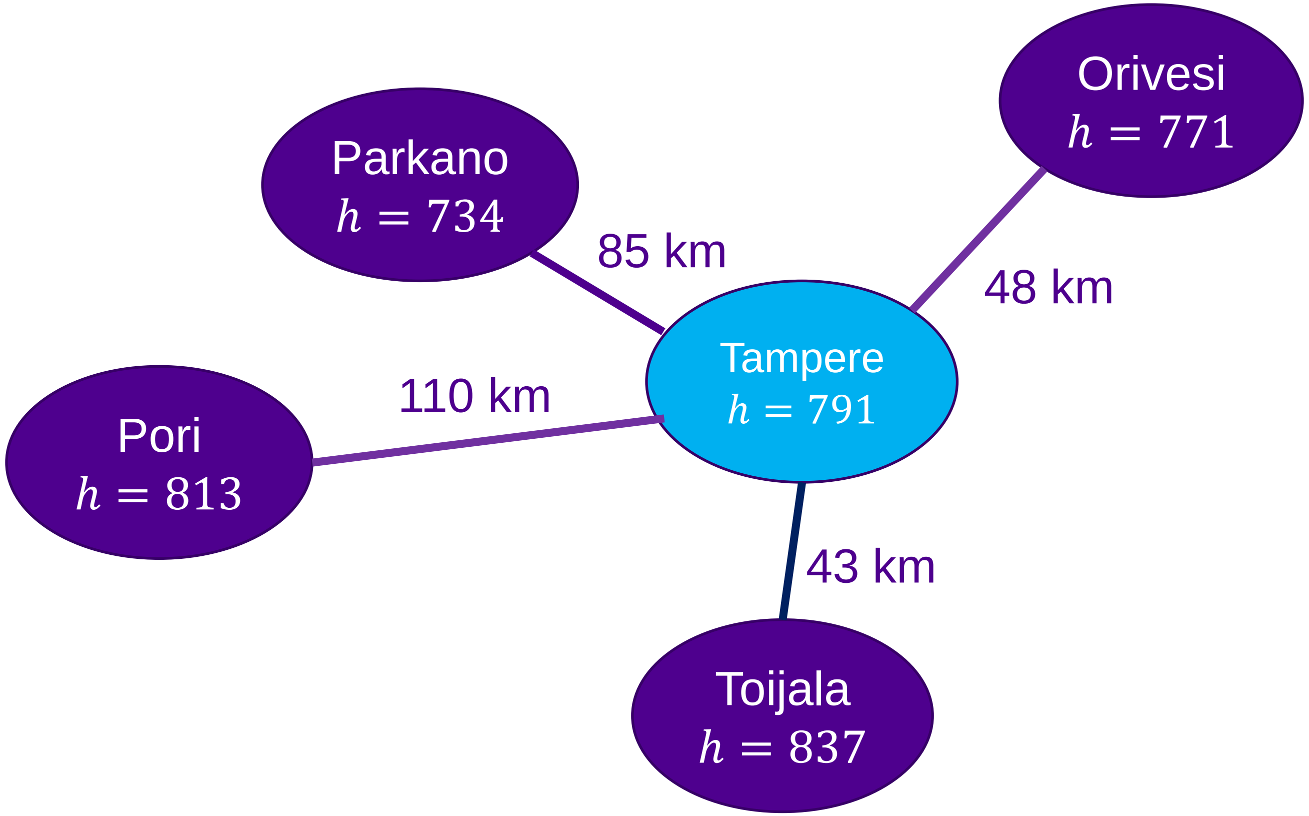
- How many
  - World states?
    $120 \times (2^{30}) \times (12^{2}) \times 4$
  - States for pathing?
    120
  - States for eat-all-dots?
    $120 \times (2^{30})$

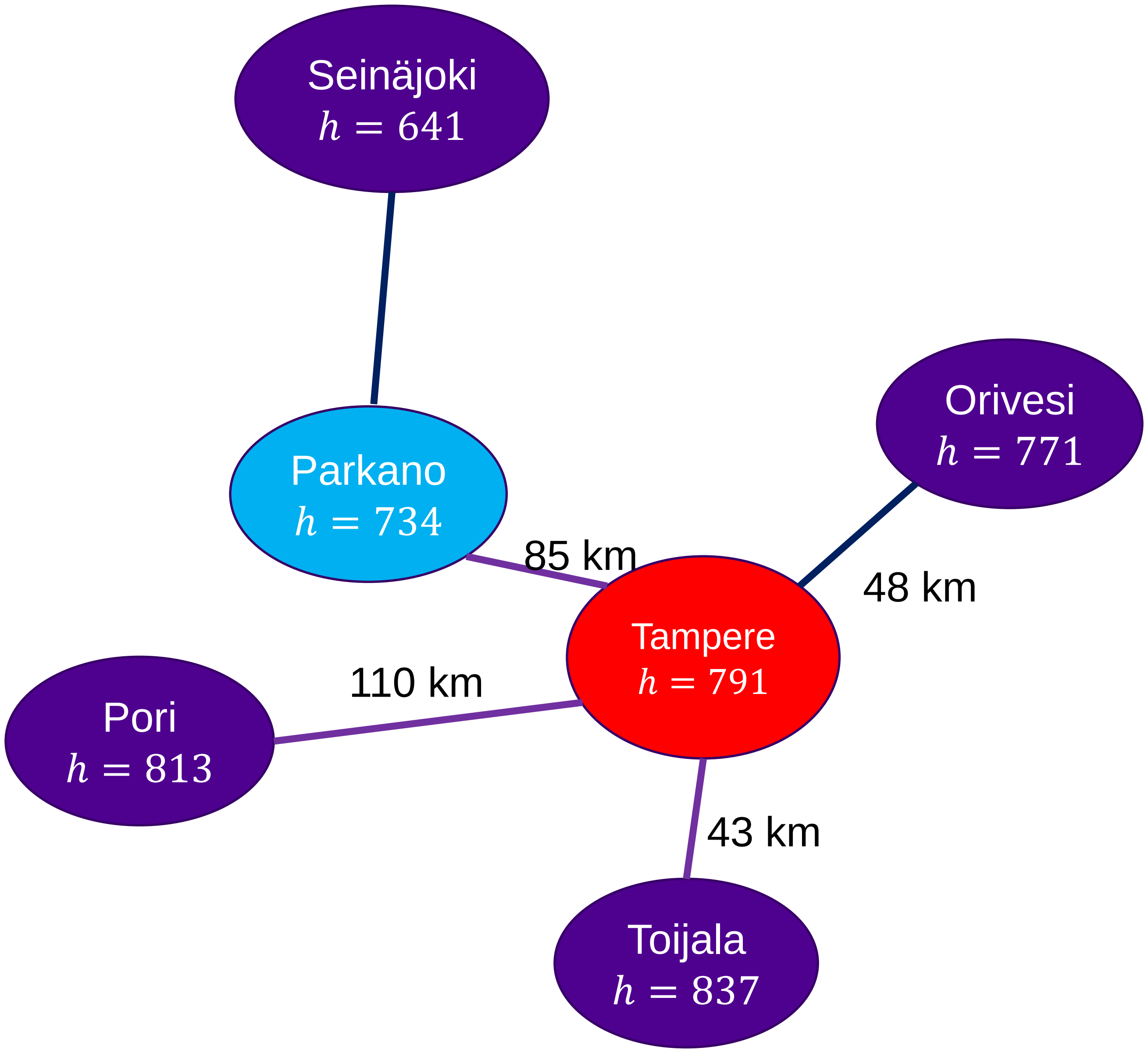


Informed search



Knowing heuristic info




Seinäjoki
$h = 641$
Parkano
$h = 734$
85 km
Orivesi
$h = 771$
48 km
Tampere
$h = 791$
110 km
Pori
$h = 813$
43 km
Toijala
$h = 837$

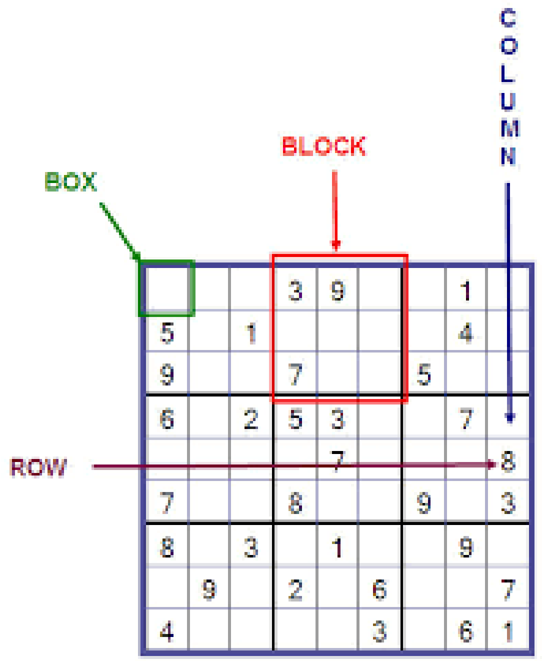
Sudoku solved?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Rules of Sudoku

- One of each number 1—9 on
  - Each block
  - Each row
  - Each column



Search strategy for Sudoku

- Start, e.g., from the left bottom corner of the matrix (if empty)
- Attempt numbers 1,2,…,9 in order until there is no illegal configuration according to the rules
- Continue from the next cell
- If at any point you run into an illegal configuration, back-up (recursively) to the last number selection and choose the next candidate.
- Eventually (after having examined enough value combinations) you'll find a solution as long as one exists

1. cell ← bottom left-most cell
2. while cell value is predetermined move to next cell
3. cell value ← 0
4. cell value++
5. if cell value ≤ 9 **and** legal configuration
6. move to next cell; **jump** to row 2
7. if cell value > 9 **then**
8. back-up to the previous not predetermined cell
9. jump to row 4

Mini Sudoku

	3		
		4	
			1
1	2		

MS2

	3		
		4	
			1
1	2	1	

MS3

	3		
		4	
			1
1	2	2	

MS4

	3		
		4	
			1
1	2	3	

MS5

	3		
		4	
			1
1	2	3	1

MS6

	3		
		4	
			1
1	2	3	2

MS7

	3		
		4	
			1
1	2	3	3

MS8

	3		
		4	
			1
1	2	3	4

MS9

	3		
		4	
1			1
1	2	3	4

Tampere University

MS10

	3		
		4	
2			1
1	2	3	4

MS11

	3		
		4	
3			**1**
1	**2**	3	4

MS12

	3		
		4	
3	1		1
1	2	3	4

MS13

	3		
		4	
3	2		1
1	2	3	4

MS14

	3		
		4	
3	3		1
1	2	3	4

MS15

	3		
		4	
3	4		**1**
1	**2**	3	4

MS16

	3		
		4	
3	4	1	**1**
1	**2**	3	4

MS17

	3		
		4	
3	4	2	**1**
1	**2**	3	4

MS18

	3		
1		4	
3	4	2	1
1	2	3	4

MS19

	3		
2		**4**	
3	4	2	**1**
1	**2**	3	4

MS20

	3		
2	1	**4**	
3	4	2	**1**
1	**2**	3	4

MS21

	3		
2	1	**4**	1
3	4	2	**1**
1	**2**	3	4

MS22

	3		
2	1	4	2
3	4	2	1
1	2	3	4

MS23

1	**3**		
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

MS24

2	**3**		
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

MS25

3	**3**		
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

MS26

4	**3**		
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

MS27

4	**3**	1	
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

MS28

4	**3**	1	1
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

MS29

4	**3**	1	2
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

How’d we do it

	3		
		4	
			1
1	2		

HW2

	3		
	1	4	
			1
1	2		

HW3

	3	1	
	1	4	
			1
1	2		

Tampere University

HW4

	3	1	
	1	**4**	
3			**1**
1	**2**		

HW5

	3	1	
	1	**4**	
3	4		**1**
1	**2**		

HW6

	3	1	
	1	**4**	
3	4	2	**1**
1	**2**		

HW7

	3	1	___
	1	**4**	
3	4	2	**1**
1	**2**	3	

HW8

	3	1	
	1	**4**	3
3	4	2	**1**
1	**2**	3	

HW9

	3	1	
	1	**4**	3
3	4	2	**1**
1	**2**	3	4

Tampere University

HW10

	3	1	2
	1	**4**	3
3	4	2	**1**
1	**2**	3	4

HW11

	3	1	2
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

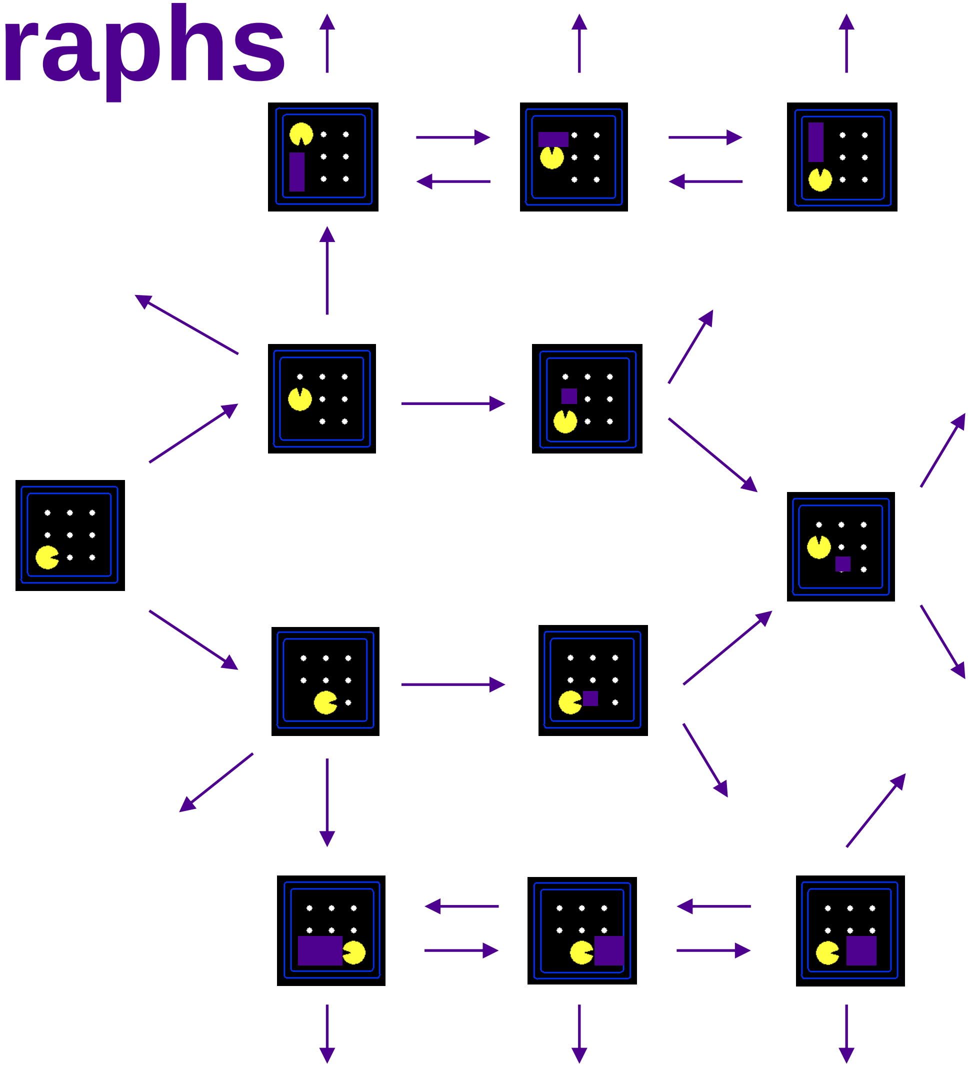
HW12

4	**3**	1	2
2	1	**4**	3
3	4	2	**1**
1	**2**	3	4

Search space

- In the worst case scenario, we have to look at all possible Sudoku configurations before finding the one and only solution
- Such a enumerative search is not usually feasible due to its inefficiency
- In two-player turn-taking board games the solution search happens in a game tree: White takes the first move in chess from among those available to it according to the rules of the game

State space graphs



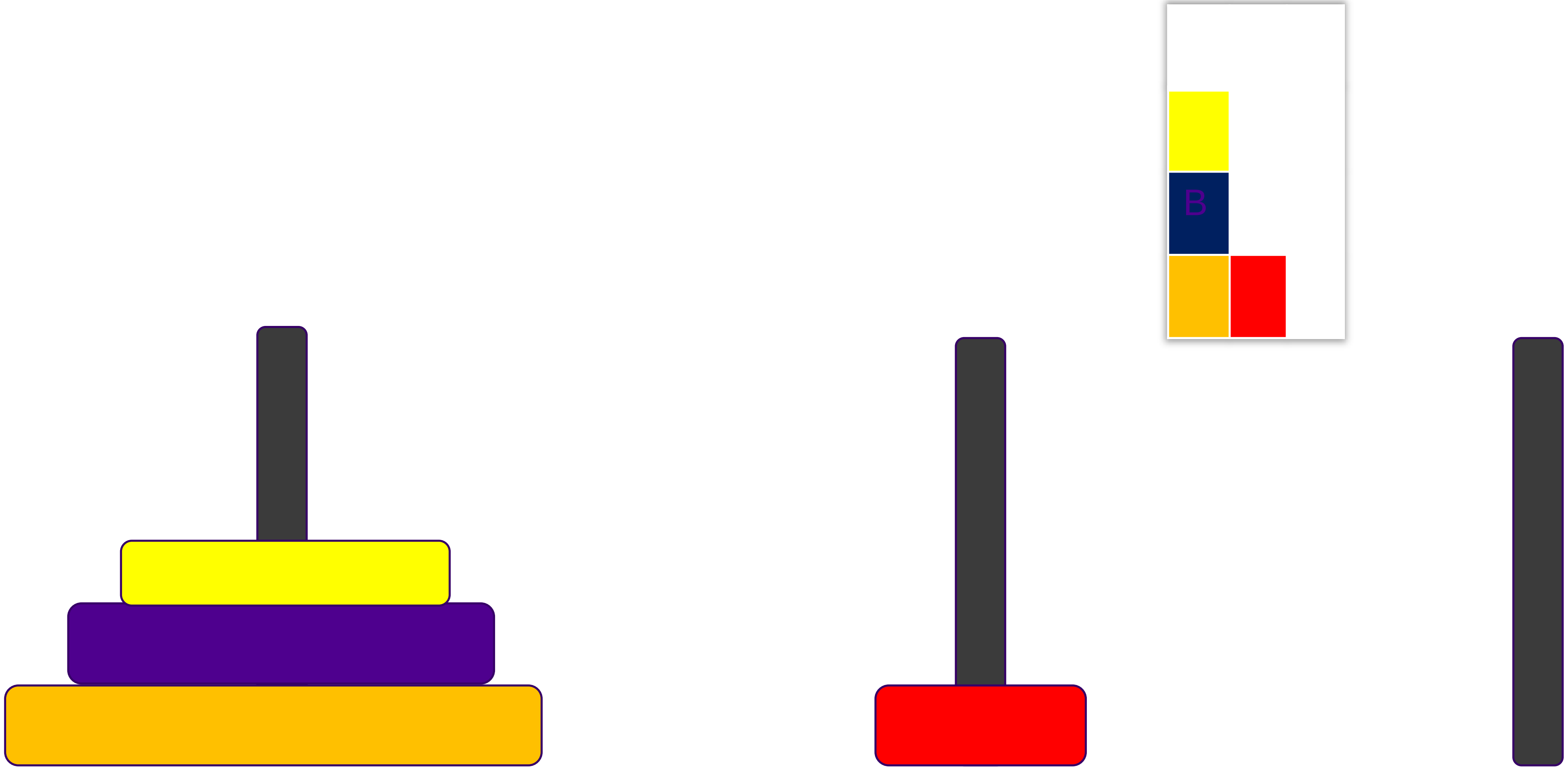
Tower(s) of Hanoi

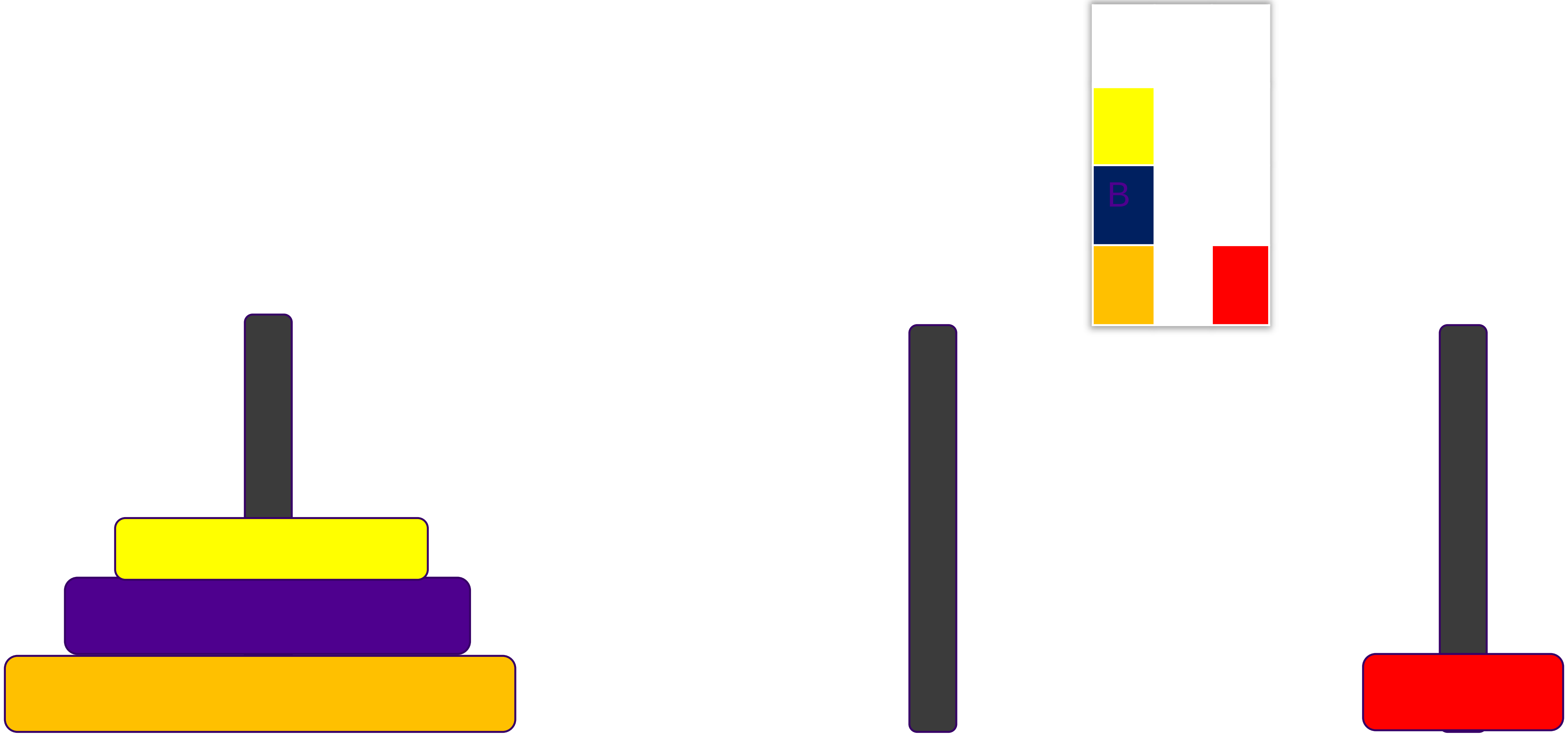
- The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:

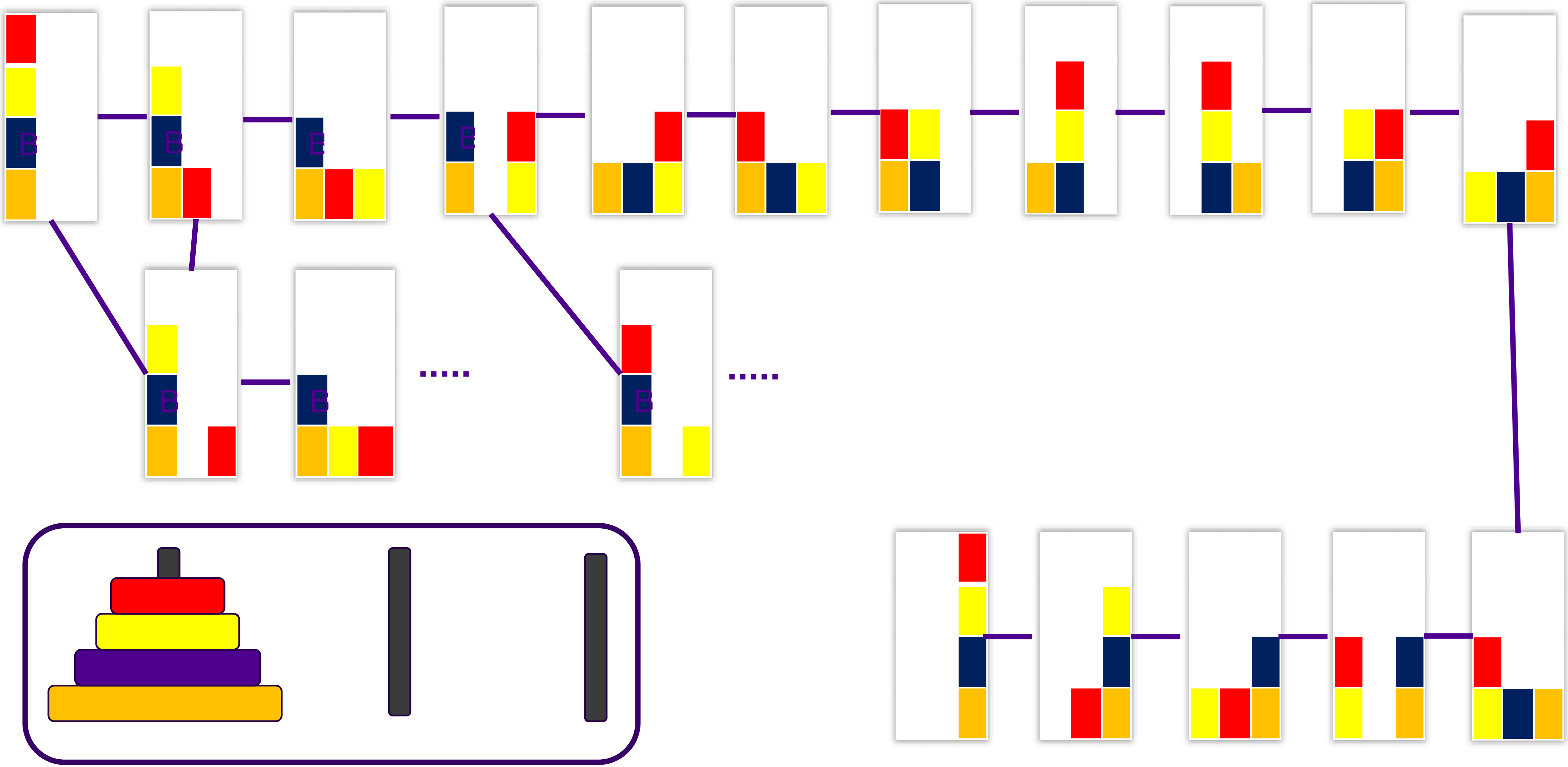
1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No larger disk may be placed on top of a smaller disk.

- With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks.

States & Legal Actions

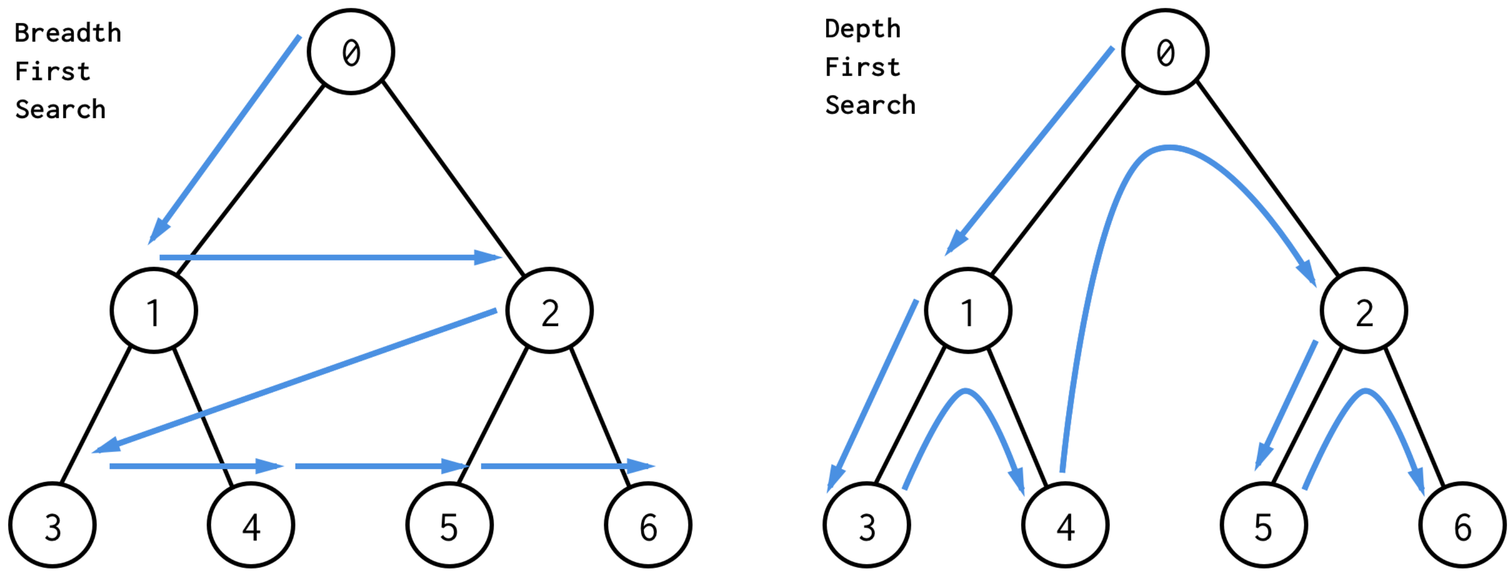

B


B


B
B
B
B
B
B
B
.....
.....

- Search may end up looping or oscillating in the search space – never proceed to goal
- Maintain visited state information to avoid repeating states
- Turn the search to a tree rather than a graph
- Need a systematic way of traversing the search space
- A **complete** search algorithm is guaranteed to find a solution, when one exists
- An **optimal** search strategy finds the optimal solution

Blind search strategies



BFS & DFS

- Blind since there is no indication where the goal state might reside
- Hence, in the worst case all nodes need to be examined
- The strategies differ in the number of nodes that they must retain

BFS

- Has to hold on to all nodes it has seen until the goal state is found
- The positive side is that it guarantees finding a goal if one exists: **complete**
- Also guarantees the **optimality** of the goal (if step costs are equal)

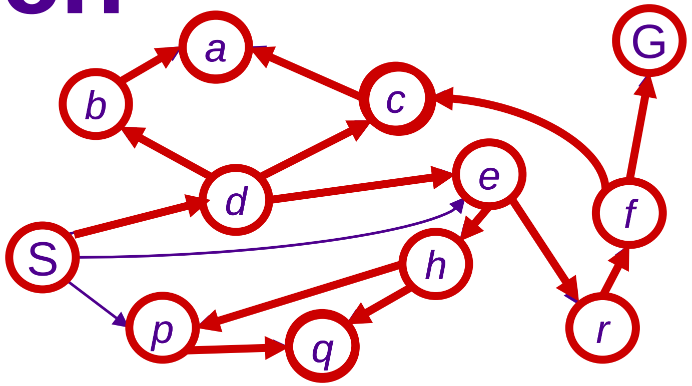
DFS

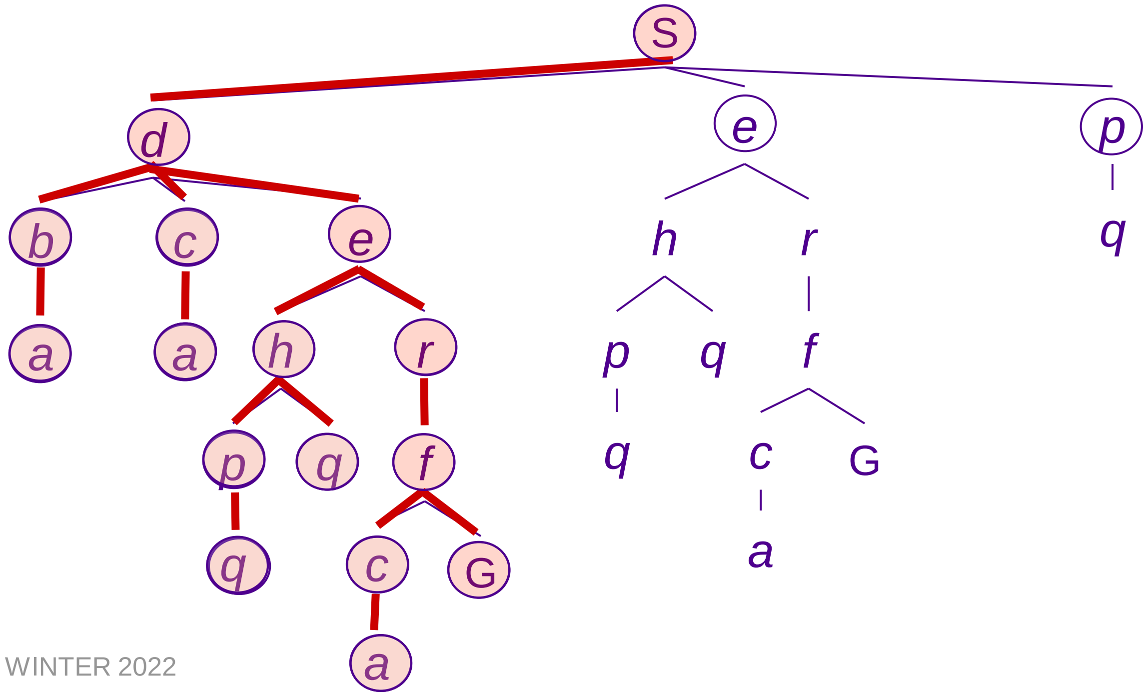
- May discard paths that have already been examined
- May lose itself to an infinite search path
- Might miss the shallowest goal state

Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack

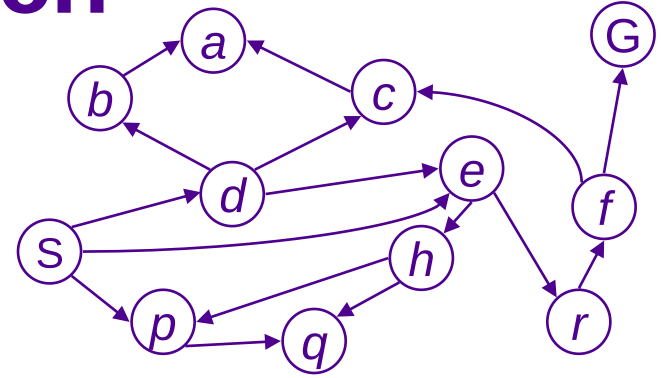


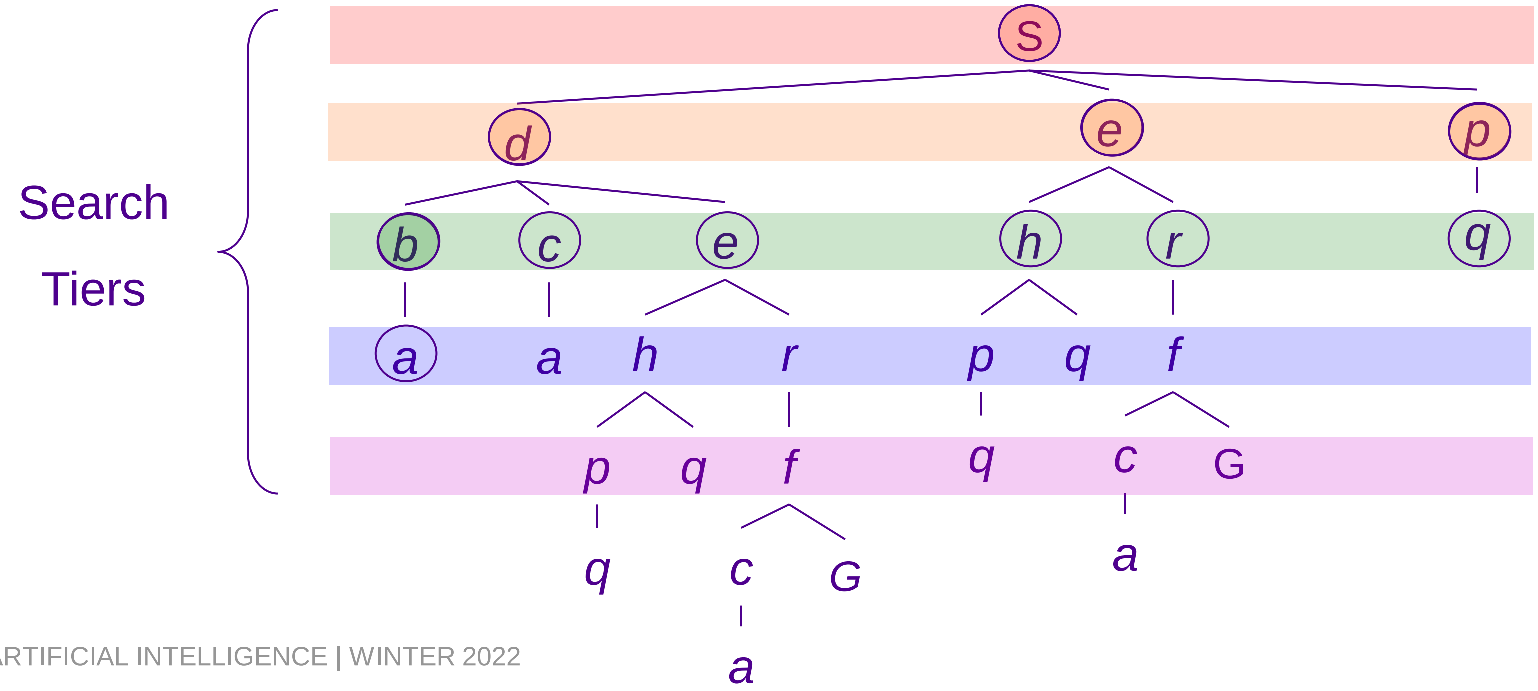


Breadth-First Search

Strategy: expand a shallowest node first

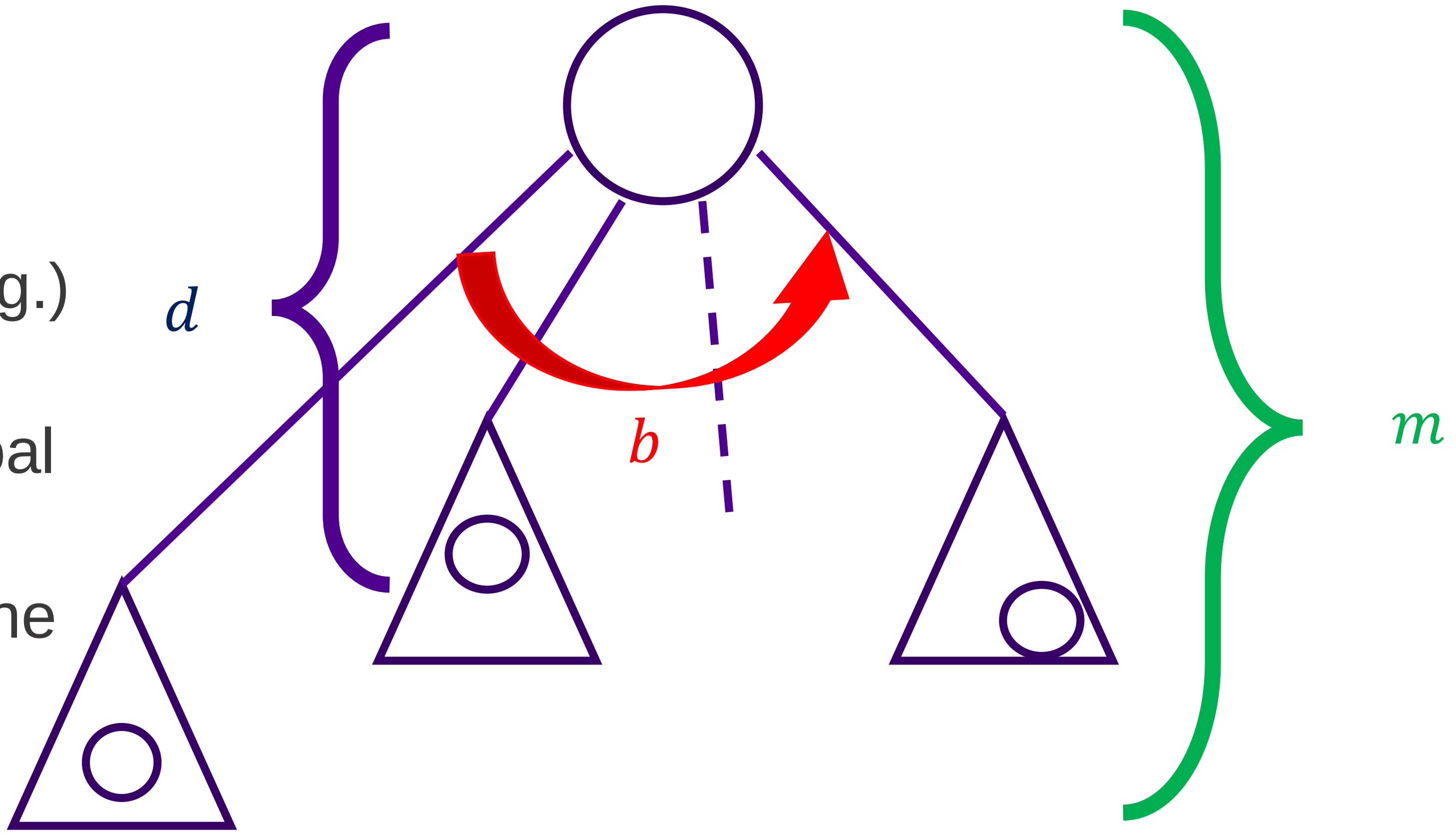
Implementation: Fringe is a FIFO queue





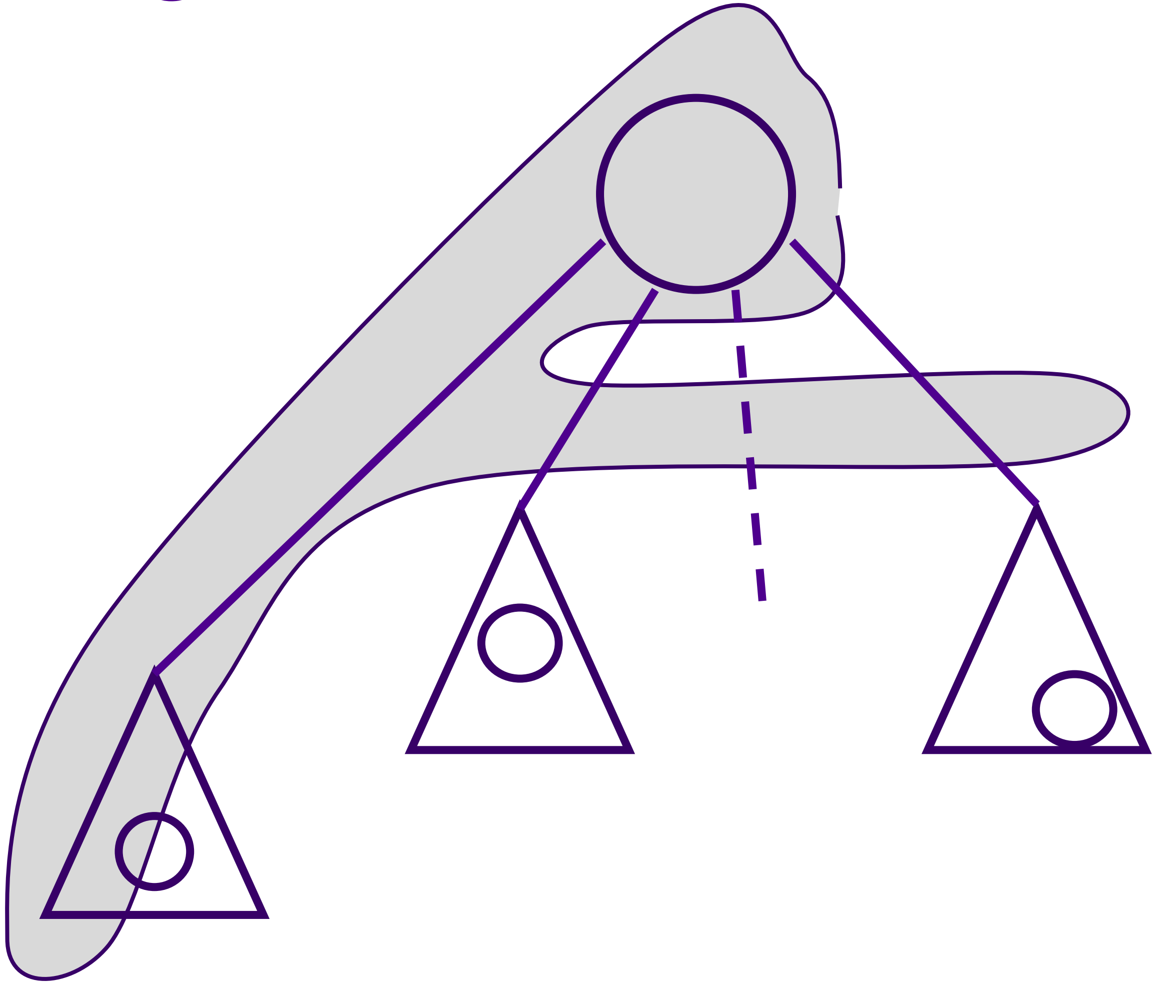
Search tree parameters

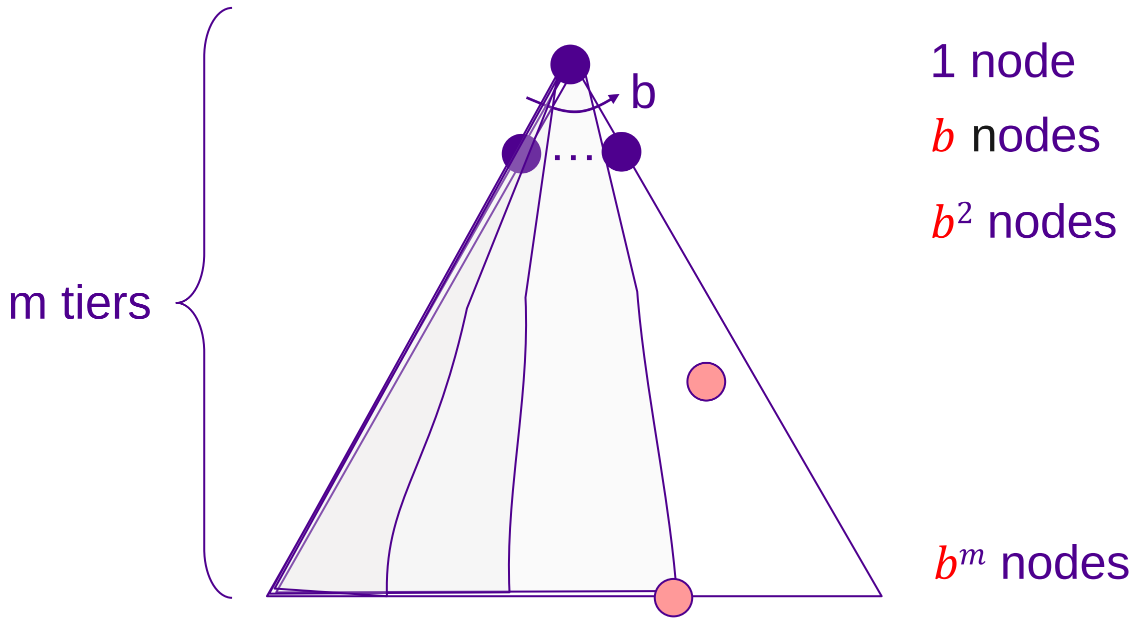
- The parameters:
  - b branching factor (max or avg.)
  - d depth of the (shallowest) goal
  - m max length of any path in the state space, could be ∞

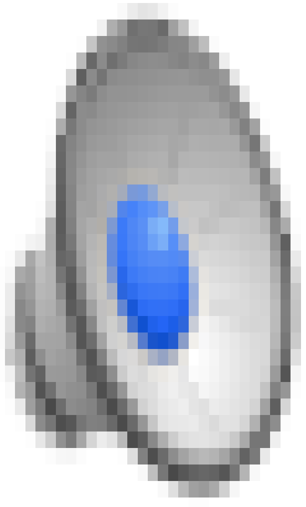


Brief analysis of the strategies: DFS

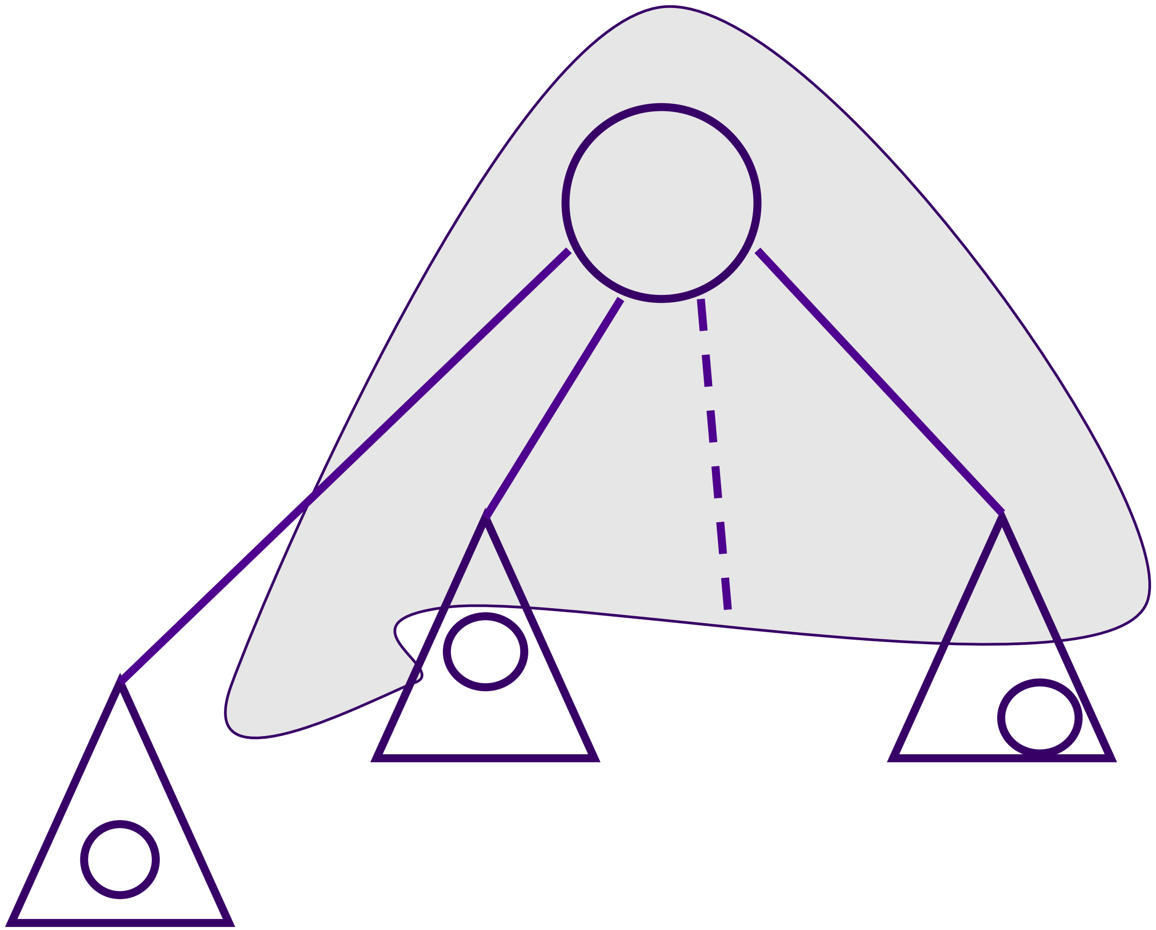
- Very modest memory requirements
  - Stores only one root-leaf path, along with the remaining unexpanded sibling nodes for each node on the path
  - DFS requires storage of only bm nodes
- The only goal node may be in the branch of the tree that is examined the last
  - In the worst case also DFS takes an exponential time: $O(b^m)$
- At its worst $m \gg d$,
  - the time taken by DFS may be much more than that of BFS




b
...
m tiers
1 node
b nodes
b^2 nodes
b^m nodes

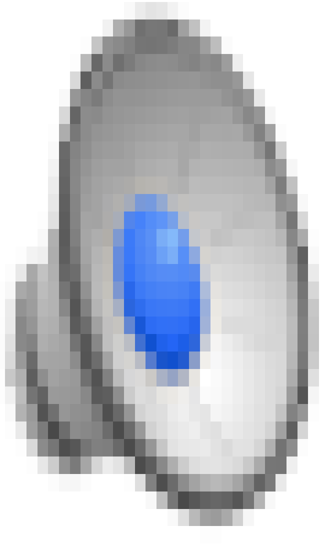


BFS analysis

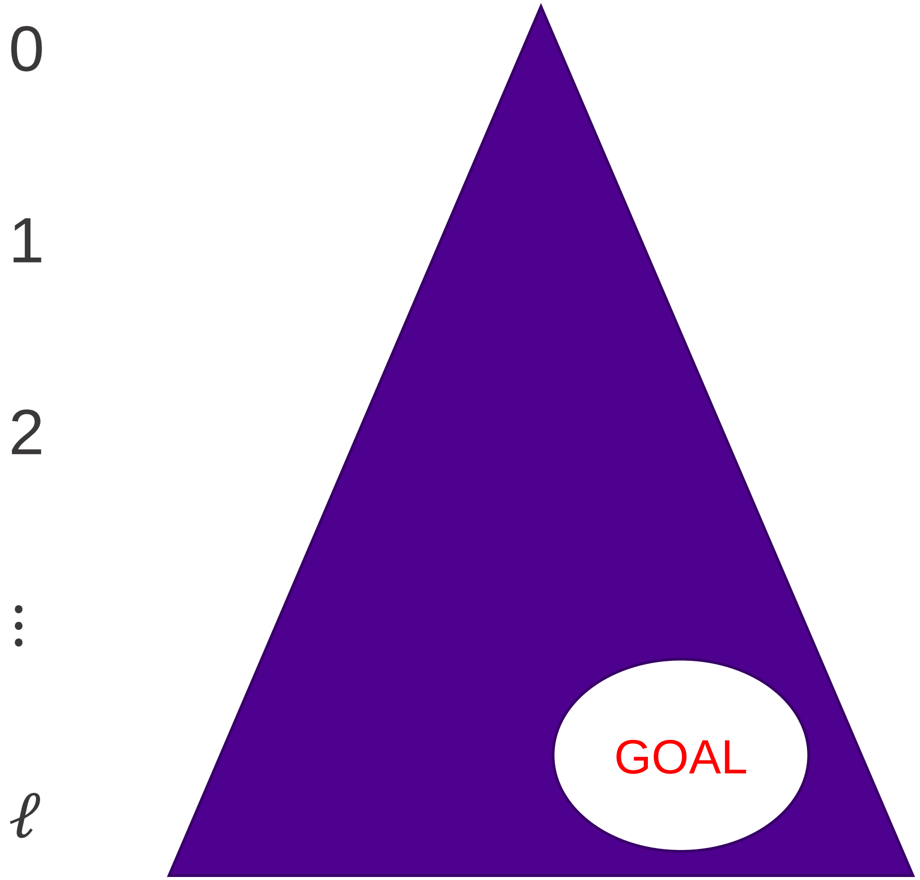


- In the worst case BFS expand all but the last node at level d
- Every node that is generated must remain in memory, because it may belong to the solution path
- Thus the worst-case time and space complexities of BFS are

$$b + b^2 + b^3 + \cdots + b^d = O(b^d)$$



Depth-limited search

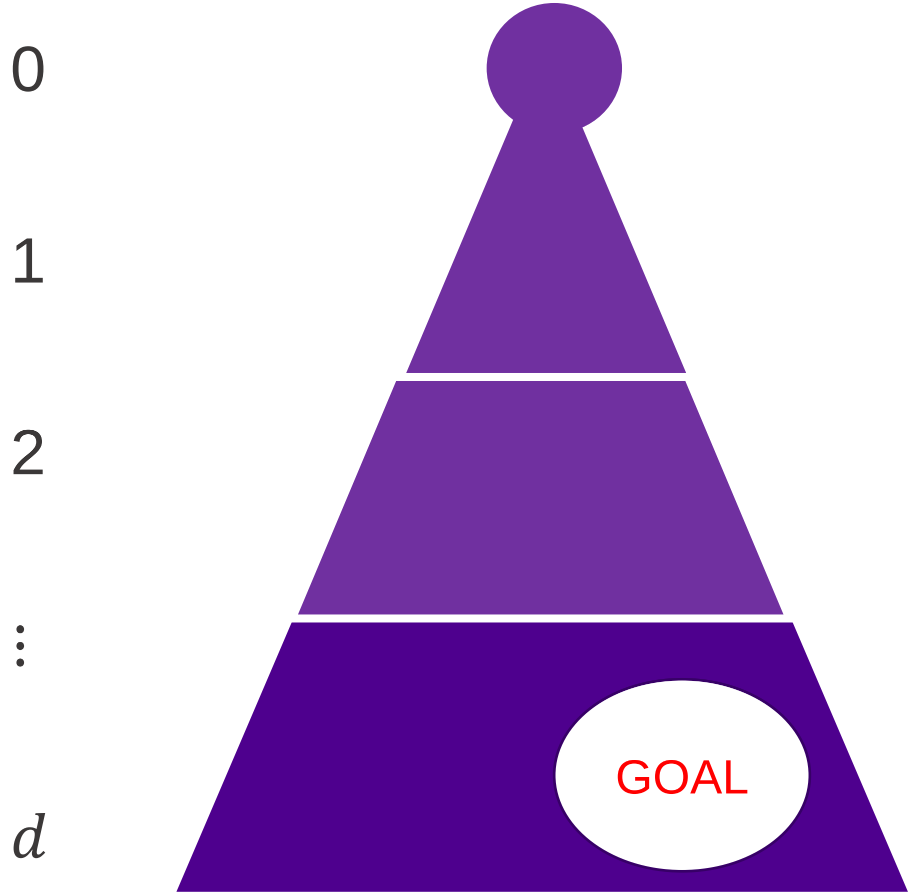


- To avoid infinite branches, limit the search to depth ℓ and use DFS
- Guaranteed to find the a goal, if the shallowest goal is at

$$\text{depth} \leq \ell$$

Iterative Deepening

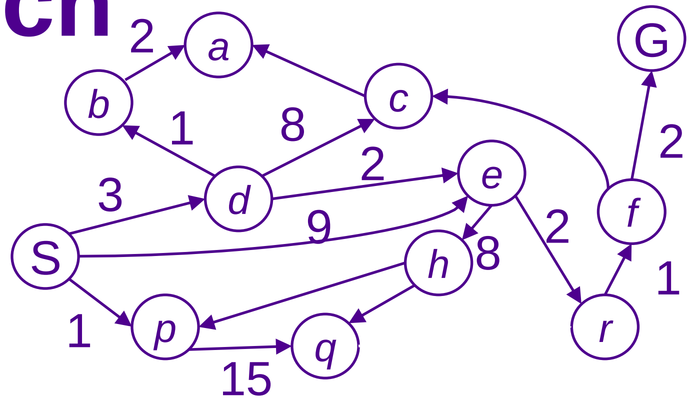
- To combine the good sides of BFS & DFS use Iterative Deepening:
- Let the parameter value grow gradually $\ell = 0,1,2,\ldots$ until the goal is found
- If a goal node exist, it will eventually be found
- Furthermore, we will find the optimal (shallowest) goal



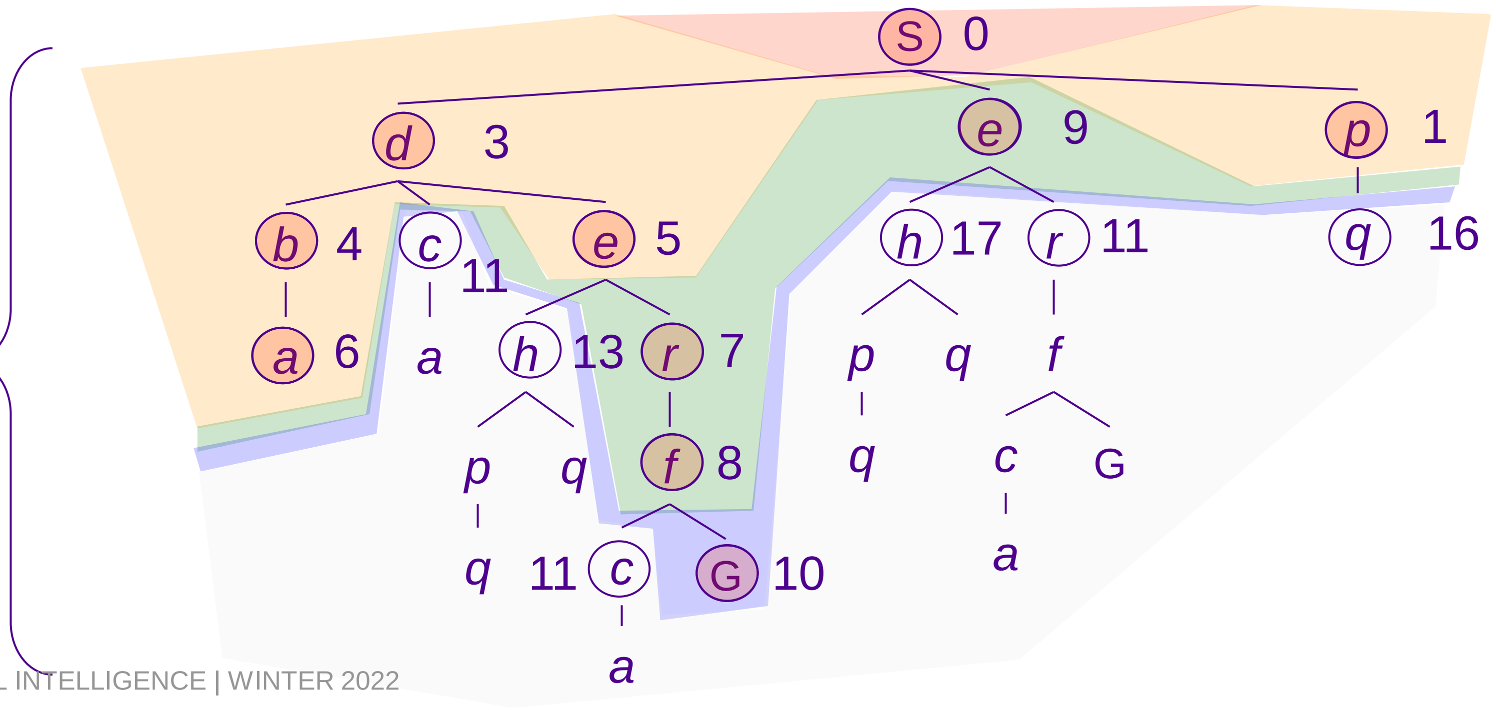
Uniform Cost Search

Strategy: expand a cheapest node first:

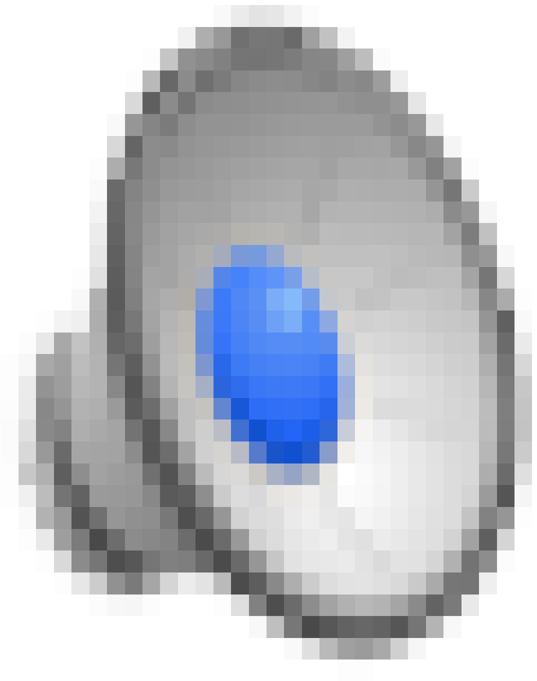
Fringe is a priority queue (priority: cumulative cost)



Cost contours



Demo Contours UCS Pacman Small Maze



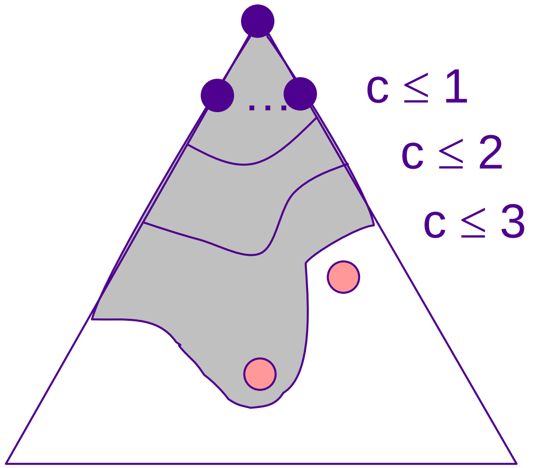
Uniform Cost Issues

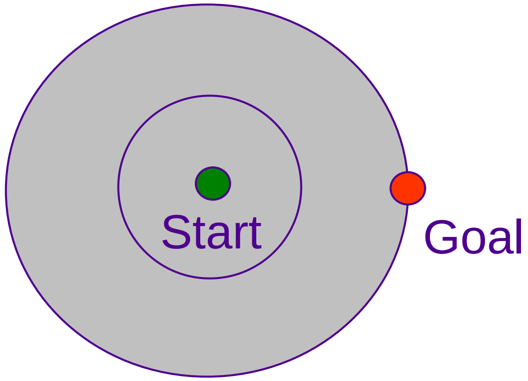
- Remember: UCS explores increasing cost contours

- The good: UCS is complete and optimal!

- The bad:
  - Explores options in every “direction”
  - No information about goal location

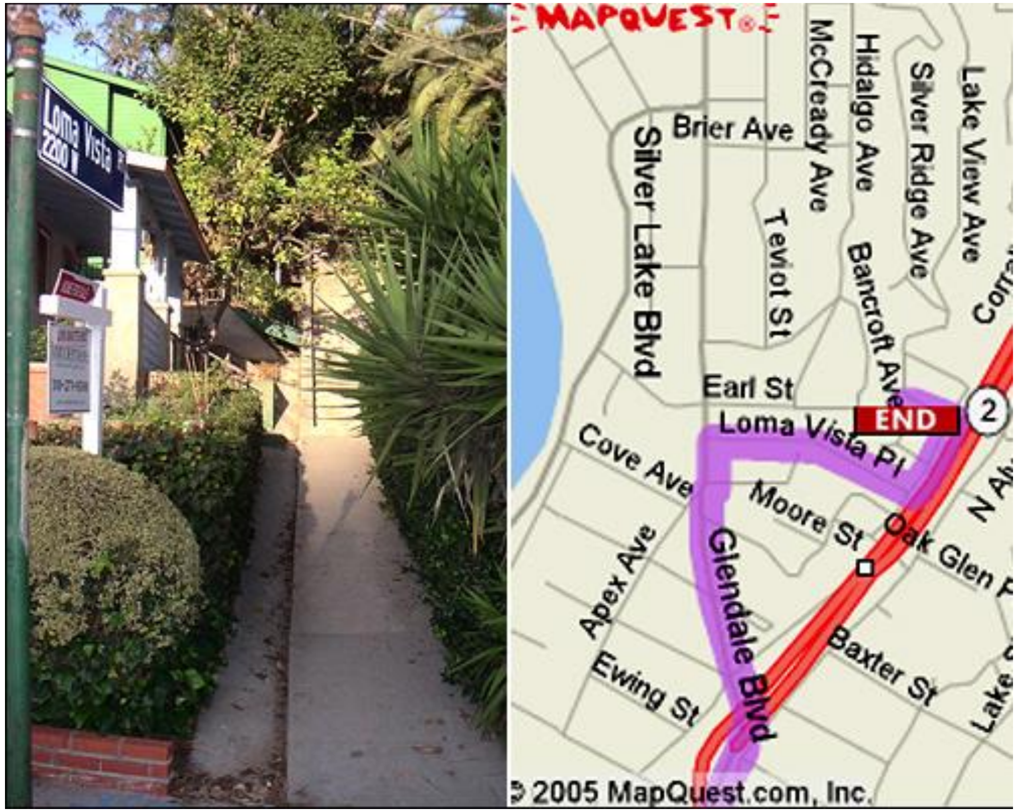
- We’ll fix that soon!





Tampere University

Search Gone Wrong



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