HEURISTIC SEARCH TECHNIQUES

TIM KECERDASAN BUATAN

AGENDA

- DEFINITION
- UNINFORMED VS INFORMED SEARCH
- CONSTRAINT SATISFACTION PROBLEMS
- LOCAL SEARCH TECHNIQUES
- GREEDY SEARCH
- SOLVING PROBLEM WITH CONSTRAINTS

DEFINITION

- There are many problems that require searching for an answer within the solution domain. There are many possible solutions to a given problem and we do not know which ones are correct. Also, going through every solution is not possible because it is prohibitively expensive
- In such cases, we rely on a rule of thumb that helps us narrow down the search by eliminating the options that are obviously wrong.
- This “rule of thumb” is called a heuristic. The method of using heuristics to guide the search is called heuristic search

WHY DO WE NEED HEURISTICS?

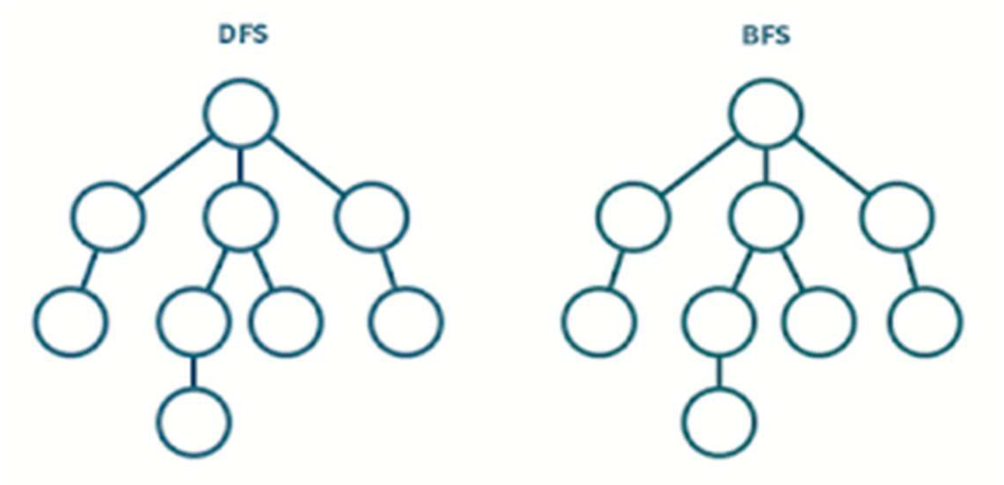
- To reduce a solution, in a reasonable amount of time. It doesn't have to be the best – an approxiamate solution will do since this is fast enough
- Reduce polynomial number for most problems that are exponential. And in situations where we can't find known algorithms.
- Heuristic techniques may be weak methods because they are vulnerable to combinatorial explosion.

UNINFORMED VS INFORMED SEARCH

- Other names for uninformed search are direct heuristic search, blind search, blind control strategy
- Uninformed search techniques do not take the goal into account. Uninformed search techniques search blindly and have no a priori knowledge about the final solution
- Depth first search (DFS), Breadth First Search (BFS)

- A heuristic search, on the other hand, is called an informed search. It uses prior information or rules to eliminate unnecessary paths.
- We need to note that heuristic searches might not always find the most optimal solution. This is because we are not exploring every single possibility and we are relying on a heuristic.
- In real-world scenarios, we need solutions that are fast and effective. Heuristic searches provide an efficient solution by arriving at a reasonable solution quickly.

DEPTH FIRST SEARCH (DFS) VS BREADTH FIRST SEARCH (BFS)

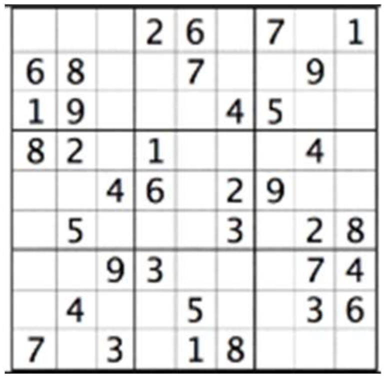


Breadth First Search (BFS) algorithm traverses a graph in a breadth-ward motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

Depth First Search (DFS) algorithm traverses a graph in a depth-ward motion and uses a stack to remember to get the next vertex to start a search when a deadend occurs in any iteration.

CONSTRAINT SATISFACTION PROBLEM

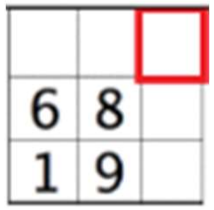
- There are many problems that must be solved under constraints. These constraints are basically conditions that cannot be violated during the process of solving the problem. These problems are referred to as Constraint Satisfaction Problems (CSPs).
- To gain some intuitive understanding, let's quickly look at an example section of a Sudoku puzzle. Sudoku is a game where we cannot have the same number twice acrosss a horizontal line, vertical line, or in the same square.



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

CSP

- Using constraint satisfaction and the rules of Sudoku we can quickly determine which numbers to try and which numbers not to try to solve the puzzle. For example, in this square:



- If we were not using CSP, one brute force approach would be to try all of the combinations of numbers in the slots and then check if the rules applied. Using CSP, we can prune the attempts before we try them

- We know that the number cannot be a 1, 6, 8 or a 9 because those numbers already exist in the square. We also know that it cannot be a 2 or 7 because those number exist in the horizontal line. We also know that it cannot be a 3 or a 4 because those number are already in the vertical line. That leaves us with the only possibility of what the number should be 5.

- CSPs are mathematical problems that are defined as a set of variables that must satisfy some constraints. When we arrive at the final solution, the states of the variables must obey all the constraints.

HILL CLIMBING

- Local search is a way of solving a CSP. It keeps optimizing the solution until all the constraints are satisfied. It iteratively keeps updating the variables until we arrive at the destination.
- A local search algorithm is a type of heuristic search algorithm. These algorithms use a function that calculates the quality of each update. For example, it can count the number of constraints that are being violated by the current update or it can see how the update affects the distance to the goal. This is referred to as the cost of the assignment. The overall goal of local search is to find the minimum cost update at each step.
- It uses a heuristic function that measures the difference between the current state and the goal. When we start, it checks if the state is the final goal. If it is, then it stops. If not, then it selects an update and generates a new state. If it's closer to the goal than the current state, then it makes that the current state. If not, it ignores it and continues the process until it checks all possible updates. It basically climbs the hill until it reaches the summit.

SIMULATED ANNEALING

- Based on physical process of annealing a metal to get the best (minimal energy) state.
- Hill climbing with a twist:
  - allow some moves downhill (to worse states)
  - start out allowing large downhill moves (to much worse states) and gradually allow only small downhill moves.
- The search initially jumps around a lot, exploring many regions of the state space.
- The jumping is gradually reduced and the search becomes a simple hill climb (search for local optimum).

SIMULATED ANNEALING (2)

- The reason it is called simulated annealing is because it is derived from the metallurgical process. In that process, we first heat metals up, allowing atoms to diffuse within the metal, and then let them cool until they reach their optimal desired state in terms of atomic structural arrangement. This is typically to change a metal’s physical properties to become softer and easier to work.

GREEDY SEARCH

- Greedy search is an algorithmic paradigm that makes the locally optimal choice at each stage in order to find the global optimum. But in many problems, greedy algorithms do not produce globally optimum solutions. An advantage of using greedy algorithms is that they produce an approximate solution in a reasonable time. The hope is that this approximate solution is reasonably close to the global optimal solution.
- Greedy algorithms do not refine their solutions based on new information during the search. For example, let's say you are planning on a road trip and you want to take the best route possible. If you use a greedy algorithm to plan the route, it might ask you to take routes that are have a shorter distance but might end up taking more time. It may also lead you to paths that may seem faster in the short term but might lead to traffic jams later. This happens because greedy algorithms only see the next step and not the globally-optimal final solution

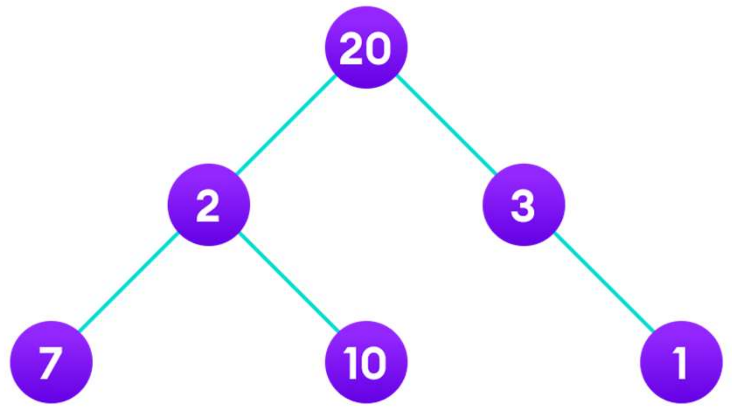
EXAMPLE GREEDY SEARCH

For example, suppose we want to find the longest path in the graph below from root to leaf. Let's use the greedy algorithm here.

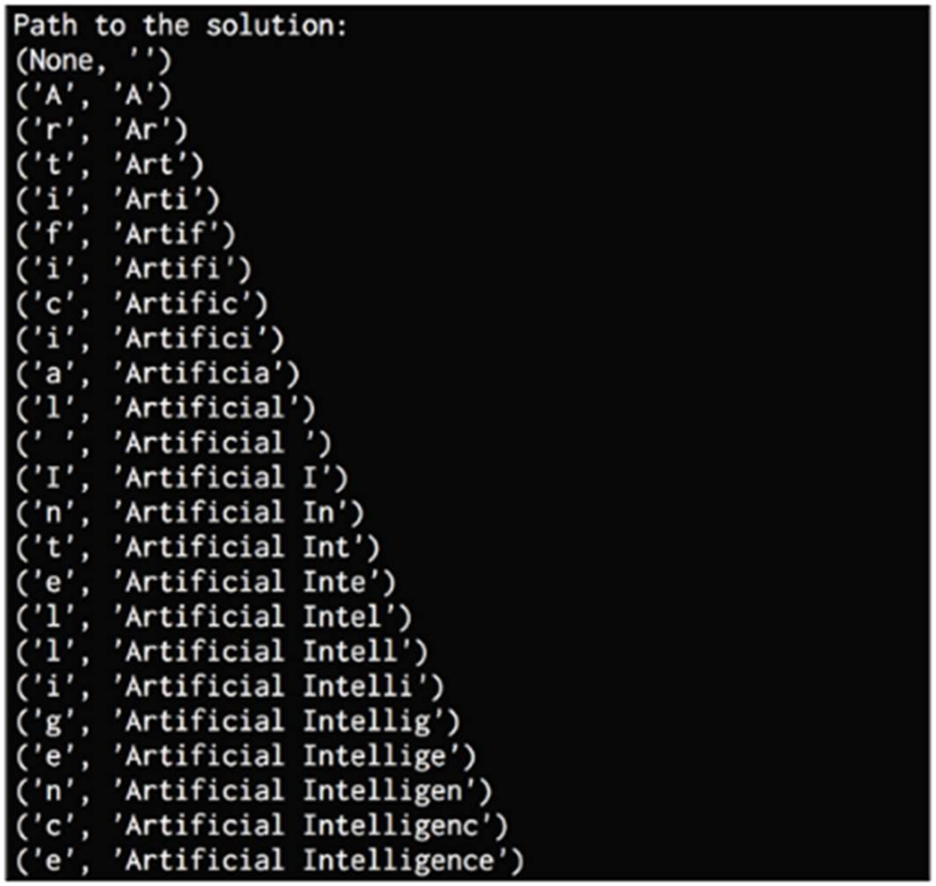
Greedy Approach:

1. Let's start with the root node 20. The weight of the right child is 3 and the weight of the left child is 2.
2. Our problem is to find the largest path. And, the optimal solution at the moment is 3. So, the greedy algorithm will choose 3.
3. Finally the weight of an only child of 3 is 1. This gives us our final result 20 + 3 + 1 = 24.

However, it is not the optimal solution. There is another path that carries more weight (20 + 2 + 10 = 32) as shown in the image below.

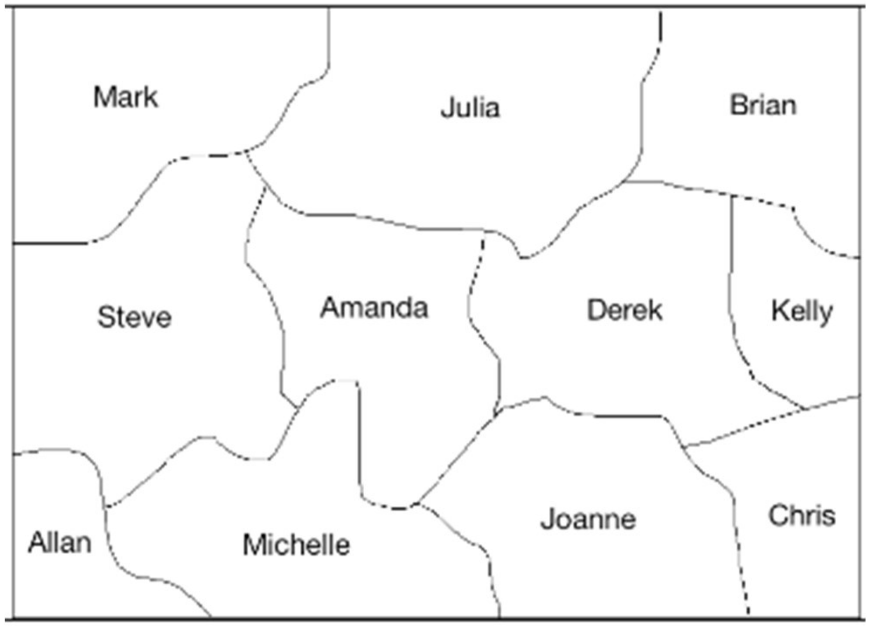


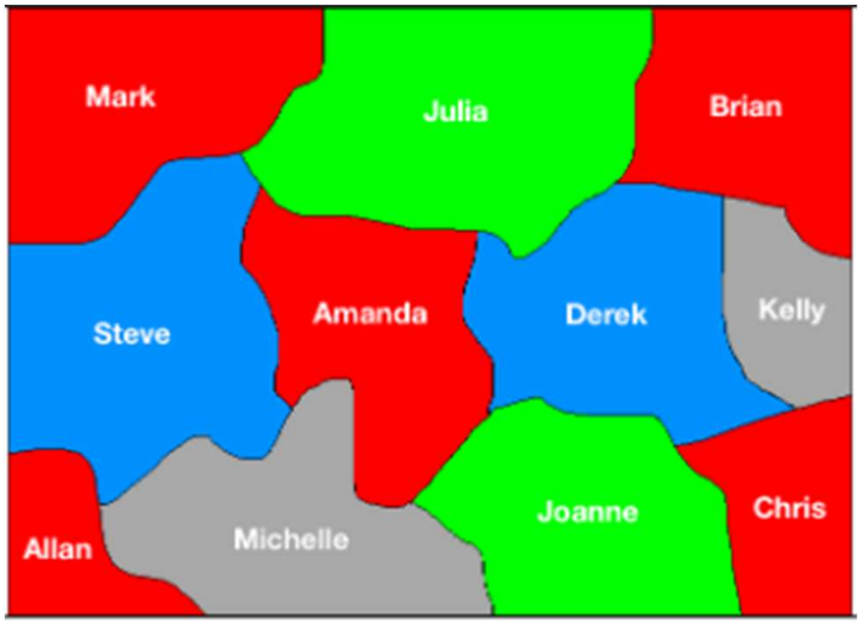
CONSTRUCTING STRING USING GREEDY SEARCH



```
Path to the solution:
(None, '')
('A', 'A')
('r', 'Ar')
('t', 'Art')
('i', 'Arti')
('f', 'Artif')
('i', 'Artifi')
('c', 'Artific')
('i', 'Artifici')
('a', 'Artificia')
('l', 'Artificial')
(' ', 'Artificial ')
('I', 'Artificial I')
('n', 'Artificial In')
('t', 'Artificial Int')
('e', 'Artificial Inte')
('l', 'Artificial Intel')
('l', 'Artificial Intell')
('i', 'Artificial Intelli')
('g', 'Artificial Intellig')
('e', 'Artificial Intellige')
('n', 'Artificial Intelligen')
('c', 'Artificial Intelligenc')
('e', 'Artificial Intelligence')
```

SOLVING REGION-COLORING PROBLEM

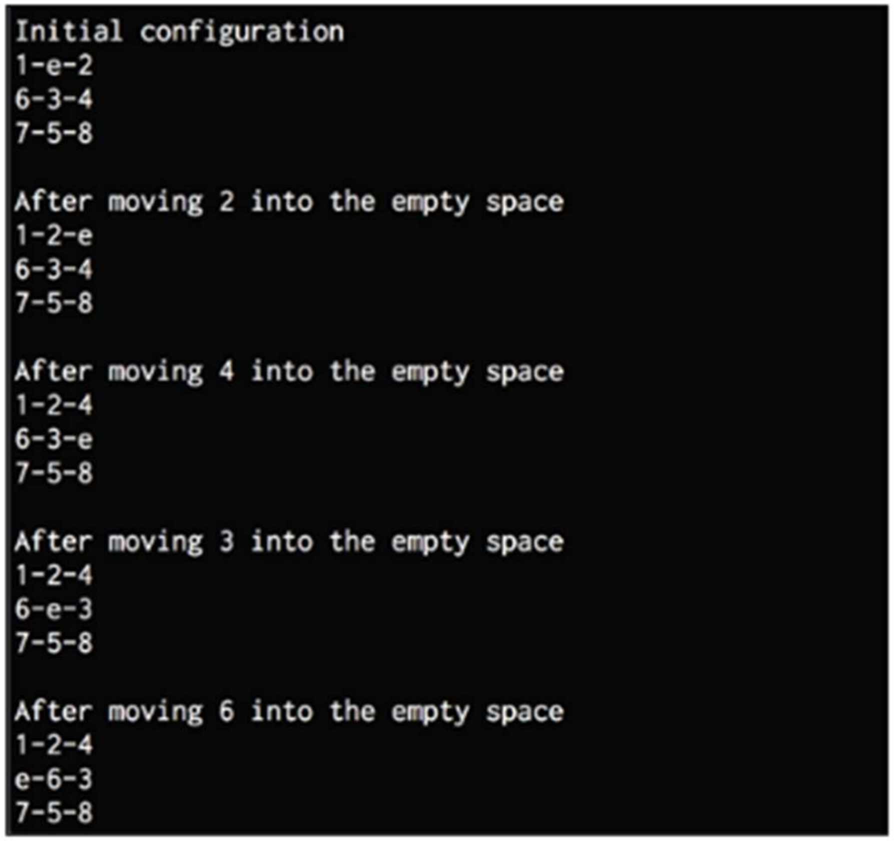




8-PUZZLE SOLVER

- We will use an A* algorithm to solve this problem. It is an algorithm that's used to find paths to the solution in a graph.
- The A* algorithm picks the one that looks the most promising. At each node, the list of all possibilities is generated and then the one with the minimal cost required to reach the goal is picked.

8-PUZZLE SOLVER



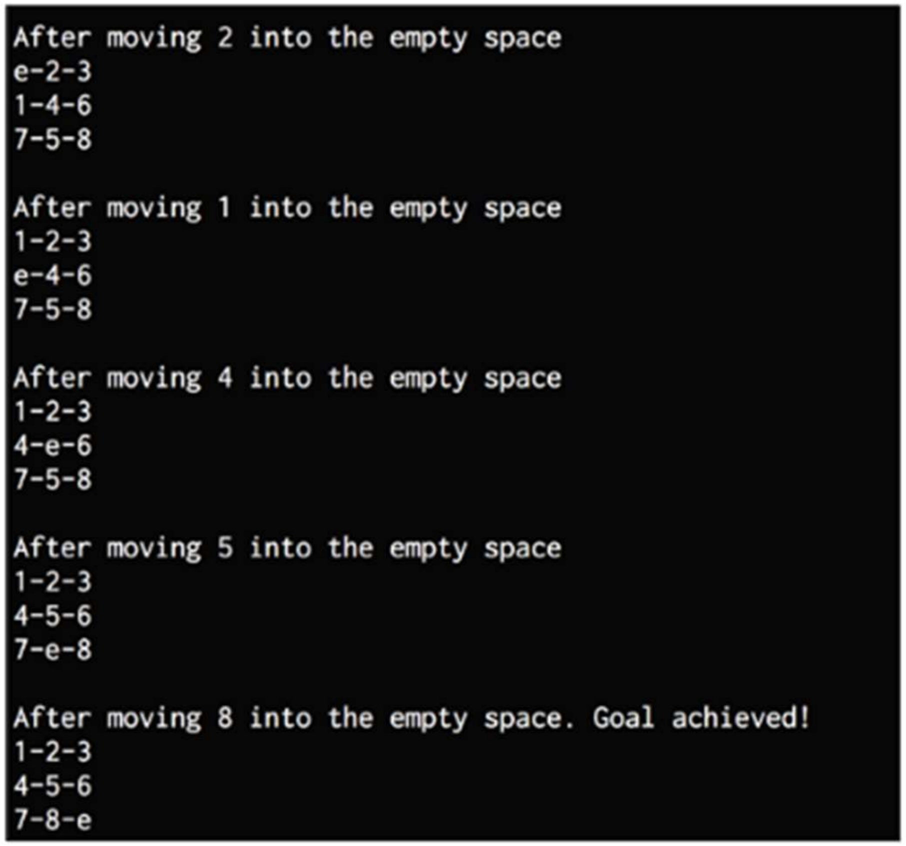
```
Initial configuration
1-e-2
6-3-4
7-5-8

After moving 2 into the empty space
1-2-e
6-3-4
7-5-8

After moving 4 into the empty space
1-2-4
6-3-e
7-5-8

After moving 3 into the empty space
1-2-4
6-e-3
7-5-8

After moving 6 into the empty space
1-2-4
e-6-3
7-5-8
```



```
After moving 2 into the empty space
e-2-3
1-4-6
7-5-8

After moving 1 into the empty space
1-2-3
e-4-6
7-5-8

After moving 4 into the empty space
1-2-3
4-e-6
7-5-8

After moving 5 into the empty space
1-2-3
4-5-6
7-e-8

After moving 8 into the empty space. Goal achieved!
1-2-3
4-5-6
7-8-e
```

THANK YOU