AQUARIUM

HEURISTIC SEARCH METHODS FOR PROBLEM SOLVING

Nuno Oliveira up201806525

Luís Miguel Pinto up201806206

Marcelo Reis up201809566

SPECIFICATION OF THE WORK

The rules of Aquarium are simple:

- The puzzle is played on a rectangular grid divided into blocks called "aquariums".
- The objective is to "fill" the aquariums up to a certain level or leave it empty.
- The water level in each aquarium is one and the same across its full width.
- The numbers outside the grid show the number of cells that must be filled horizontally and vertically.

FORMULATION OF THE PROBLEM

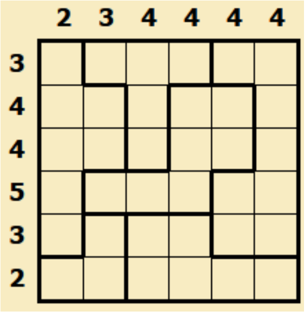
- Our problem is closer to a restriction-based problem than search problem (this has implications on the validity of heuristics and the cost value).

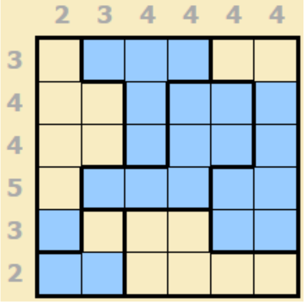
- The final solution is unique, so all that matters is achieving it the fastest way possible(even if with more depth)

- State : A state is composed of all aquariums which may or may not be filled up to a certain level. A state is valid if it respects all the restrictions put on by the horizontal and vertical numbers.

- Initial state : A state with all squares in every aquarium not filled.

- Objective Test : Verify first if the state is valid then if all restrictions are completed.





FORMULATION OF THE PROBLEM CNT.

- Operators:
  - Name - Fill up to level
  - Preconditions - The top level must be unfilled first.
  - Effect - The top level will be filled and so will the ones under it.
  - Cost – 1.
- Heuristics:
  - G(n) - Number of moves up to the current state (depth).
  - H1(n) - Minimum number of aquariums with levels yet unfilled needed to reach a solution (optimist estimative).
  - H2(n) - Number squares that still need be filled so every restriction is respected.

ALGORITHMS IMPLEMENTED

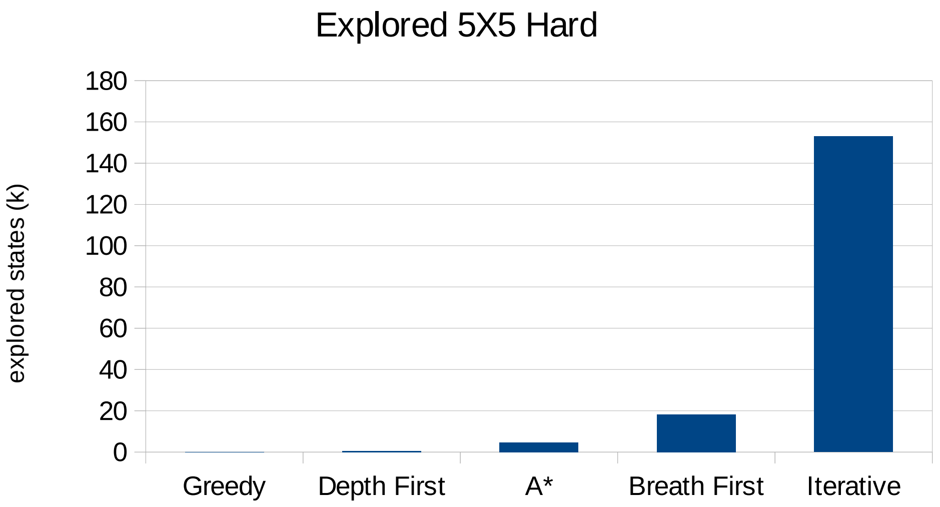
- Our application can solve the aquarium problem with multiple uninformed and heuristic search methods. The method the app uses can be altered using the main function's argument (further detail on README).
- Uninformed search methods used :
  - Breadth-first
  - Depth-first
  - Iterative Deepening
  - Uniform Cost(because the cost function used was the depth of search, this algorithm behaves like Breadth-first)
- Heuristic search methods used :
  - Greedy Search(using H2 as heuristic)
  - A* algorithm(using H1 as Heuristic,H2 could be used but it is not admissible)

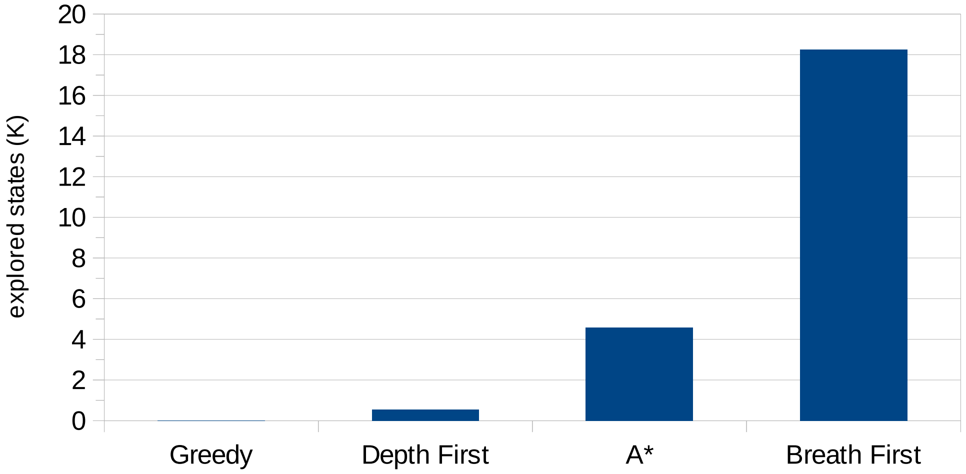
STATES EXPLORED BY EACH ALGORITHM

- Greedy and Depth-first search explore way less states than other search methods.

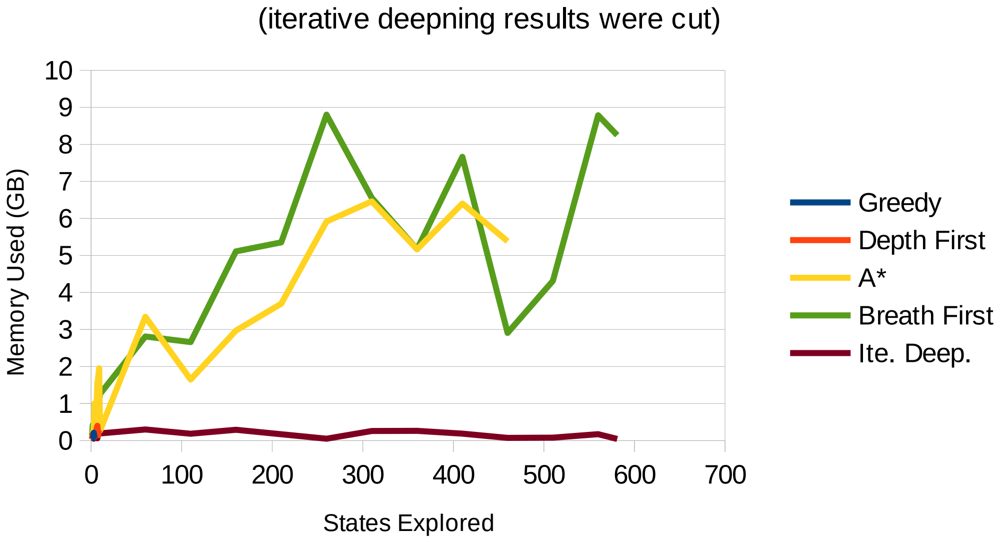
- A* search is not the best search method for this problem due to its nature being more like a restriction-based problem than a search one.

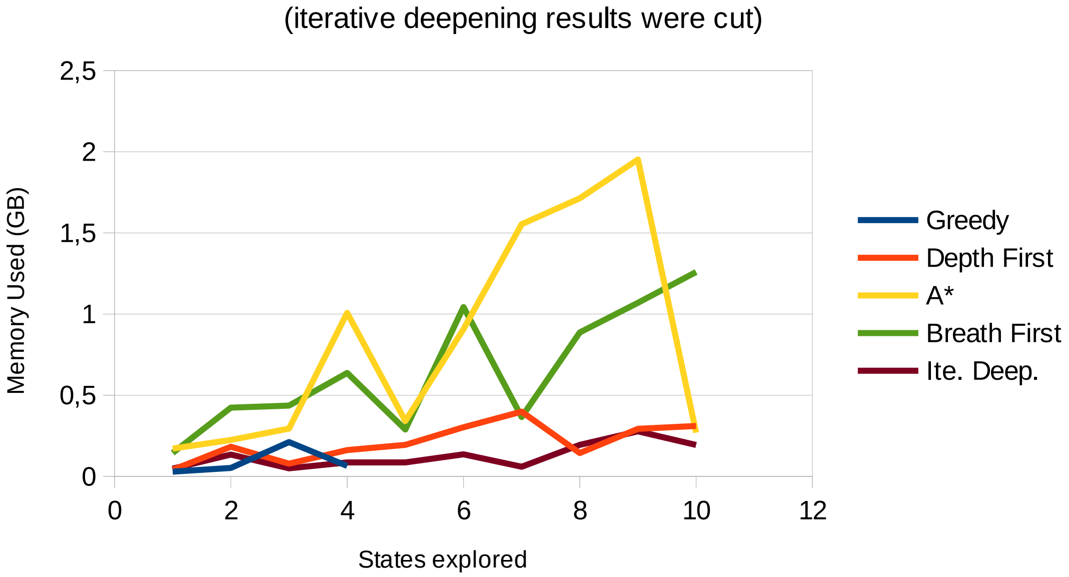
- As expected, Breadth-first and Iterative deepening need to explore much more states before reaching the solution





MEMORY USAGE BY DIFFERENT ALGORITHMS





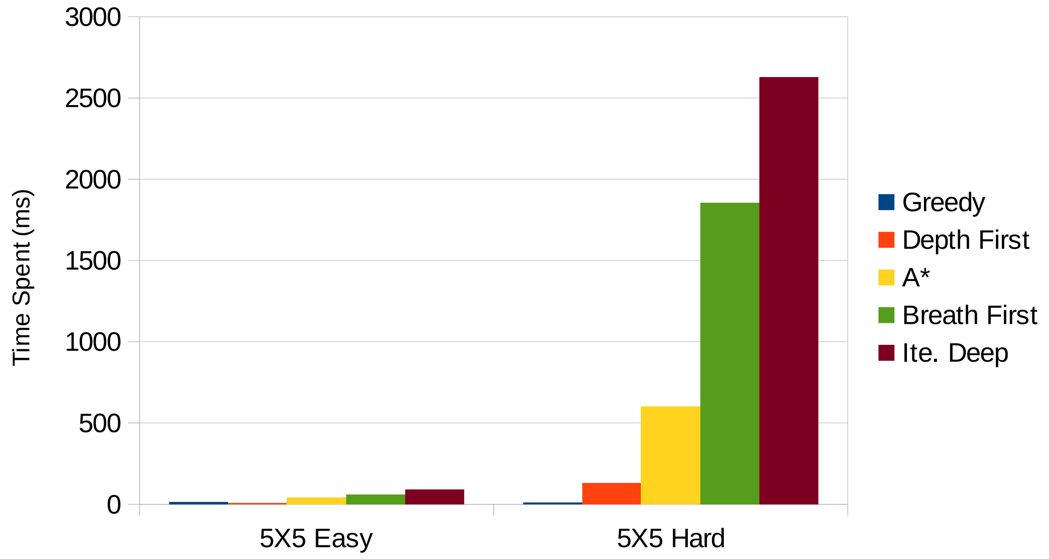
- Breadth first and A* consume more memory than Greedy and Depth First, due to the huge branching factor of the problem

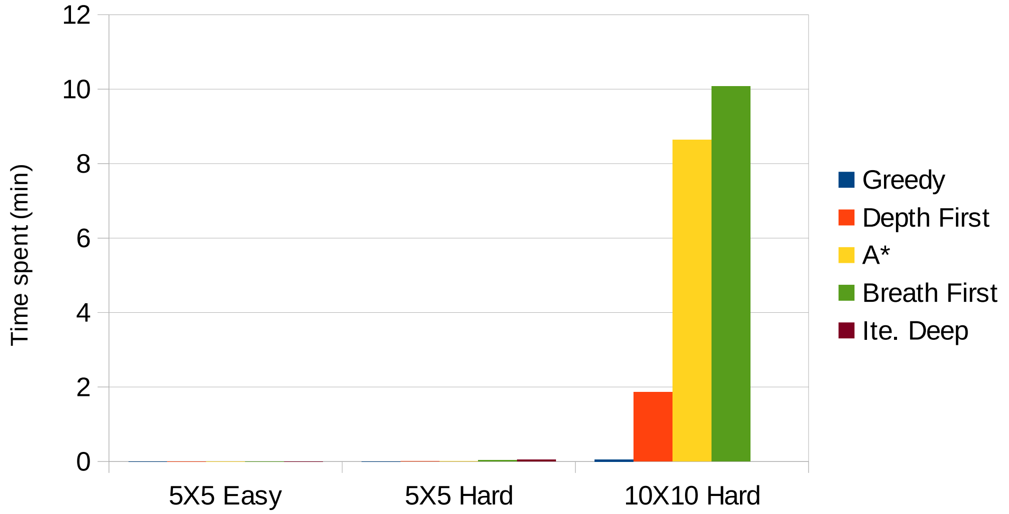
- Memory usage variates due to state limitations applied(no invalid state is put on the stack)

- A*, being a uniformed method, consumes in general less memory than breadth first

- As expected, iterative deepening has a lower memory usage for the same amount of states explored(same state multiple times)

TIME PERFORMANCE BY ALGORITHM





- Greedy is by far the best method.
- Restrictions of the problem made the heuristic used in A* less efficient.
- Problems like this are usually solved with backtracking, this explains why depth first is better than Breadth First.
- The time is in general proportional to the explored states
- The time greedy spends on the 10X10 Hard is negligible when compared to the other algorithms.
- Even though it is not in the graphics, Iterative Deepening spent 55 minutes solving 10X10 Hard

CONCLUSIONS

Given the problem(restriction-based), it was problematic to find an admissible heuristic to the A* algorithm which led to A* not being the best algorithm that was tested. Also, due to the tree pruning(invalid states were not explored) A* did not behave as good as expected.

Greedy and Depth first were the best ones because they were not focused on finding the "optimal"(least depth) path. Since there was no notion of optimal solution, those algorithms were able to achieve the final state much faster than the ones that guarantee the optimal path(A*, Breadth first and Uniform Cost).

Iterative deepening was by far the worst algorithm, it was slower than any of the other algorithms and because of its approach, the number of explored states was the biggest.

REFERENCES CONSULTED AND MATERIALS USED

- Software:
  - Java – IntelliJ, Gradle, Swing
  - LibreOffice - tables and plots
- Websites:
  - Aquarium Web Page
- Theorical Slides about search algorithms.