Search algorithms

Introduction

In this report different search algorithms are being put to the test for solving a cube tower problem. The cubes are stacked on top of one another and can be rotated. Additionally a cube above the one being turned can be held, subsequently leaving the held cube and the ones above unrotated. The different search algorithms used in this solution are “depth first search”, “breadth first search”, “a-star search” and “greedy first search”. They will hereby be denoted as DFS, BFS, A* and GFS.

Due to the images of the solved towers for each algorithm being long and spacious they are appended at the end of the report, as well as in the task directory.

Technical background

Depth first search is an algorithm that is based on a progressive exploration of the current states unexplored next state. If regarding the problem as a binary tree the DFS would search toward the deepest nodes. The DFS can be viewed as a top to bottom linear search. The new nodes will be put first in the queue (Nordmo, 2024).

Breadth first search explores every child node of its explored ones until the solution is found. The child nodes of the current will be appended last in the queue, unlike in DFS. Breadth first will find the best depth of the solution of the problem as all states in the previous depths are explored (Nordmo, 2024).

A star search is an informed search in terms that it regards steps and heuristics for the current state (*Difference between Informed and Uninformed Search in AI*, 2020). The state with the combination of least steps (actions or moves) and best heuristic will be the next to be explored. This informed search will also find the solution with the fewest steps, but will disregard unnecessary branches that BFS will explore (Nordmo, 2024).

Greedy first search is semi informed in the regard that it evaluates a heuristic but is not concerned with the steps taken (*Difference between Informed and Uninformed Search in AI*, 2020). GFS, like A*, will explore the best option, but unlike A* that

evaluates every explored state's next state, GFS does only evaluate the child nodes of the current state, not the previous states. GFS looks at the best next step, and has no memory of previous states that may have been better (Nordmo, 2024).

A* and BFS is expected to find the optimal, shortest, solution. However, they are also expected to use more resources than their counterparts. This is the case for BFS because the branching factor is higher than the number of child nodes in DFS after only one exploration. The stack in A* is claiming resources in the algorithm, else it is not drastically more complex than GFS (Nordmo, 2024).

Design and implementation

Firstly, the method for rotating the cubes was implemented. A method that makes a new object from a new configuration-array. The new array is based on the order of the cubes (what colour comes next) and if there is a holding index. With the implementation the new object is a child, and the parent passed to it is the original object.

In this report's solution of the Cube tower problem DFS and BFS are similarly composed. A loop that explores the next node in line checks the collusion and if not solved the child nodes are added to the stack (queue). The difference between DFS and BFS is that DFS adds the child nodes first in the stack and BFS appends them last in the line.

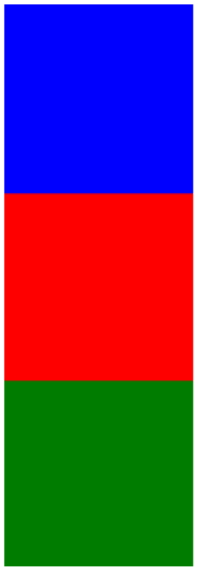
Similarly the A* and GFS are similar in the sense that they evaluate based on heuristics. A* star evaluates all child nodes of the current node and adds them sorted, based on steps taken and heuristic, in the stack. The stack in GFS is wiped clean every step as only the child nodes are evaluated, and since the steps taken for the child nodes are the same for all, only the heuristic is regarded.

The heuristic in this solution is the highest amount of cubes showing the same colour. The configuration [‘green’,’yellow’,’blue’,’blue’,’green’] will have a heuristic of two. This heuristic is easily measurable and is directly related to the solution of the problem. This specific heuristic can be viewed as distance from solution, and is intuitive to understand.

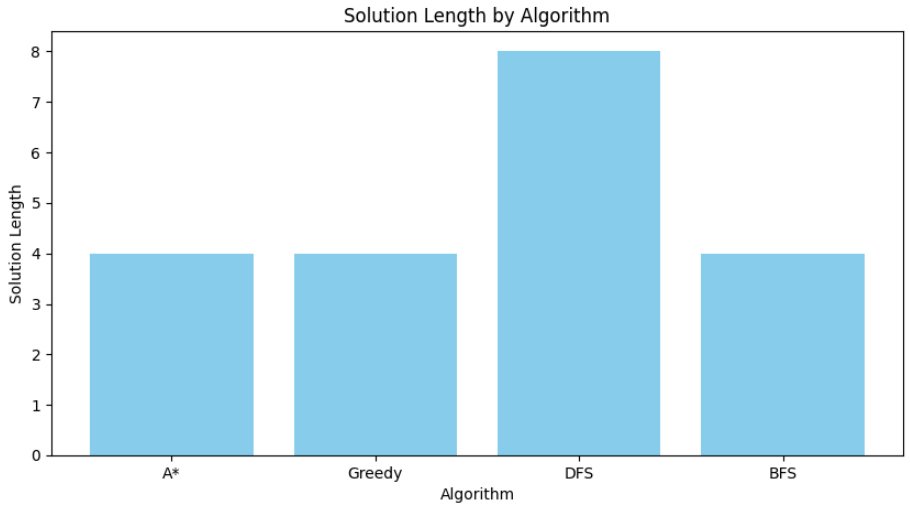
A problem in the implementation was the recursion limit in python, resulting in choosing a while loop in the algorithms. Aswell was the memory capture in the search algorithms difficult to perform without a little help. A thread at Stack Overflow resulted in using the ‘tracemalloc’-function (ARK1375, 2021).

Result

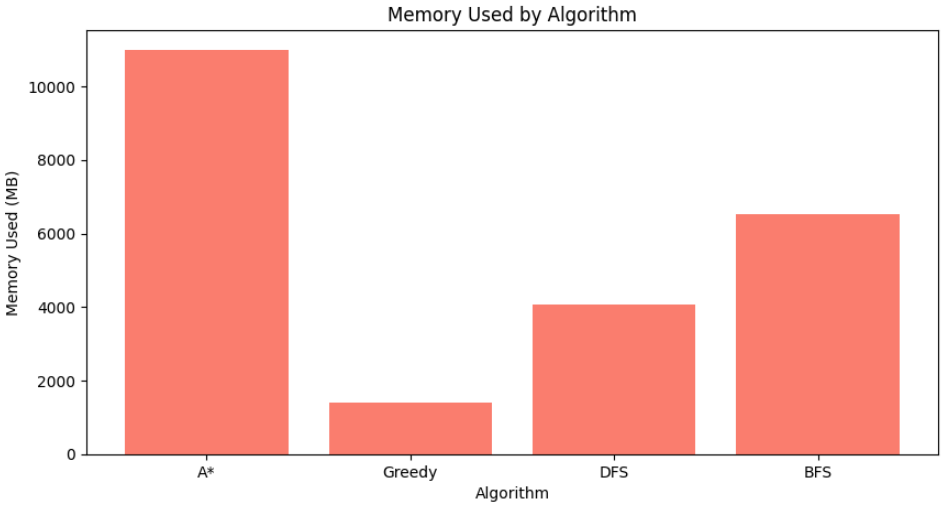
As mentioned in the technical background both BFS and A* found the optimal solutions for the problem in all cases. Though DFS was faster than BFS, and GFS was faster than A*. This time-complexity vs solution-complexity seems to have an interesting correlation. As the solution should become optimal, more nodes need to be explored, thus the time for the solution to be found is greater than the methods that only regard the states possible from the current. The retrospective insight in A* (and partially in BFS) will always find the optimal path, but will take longer than their respective counterparts.



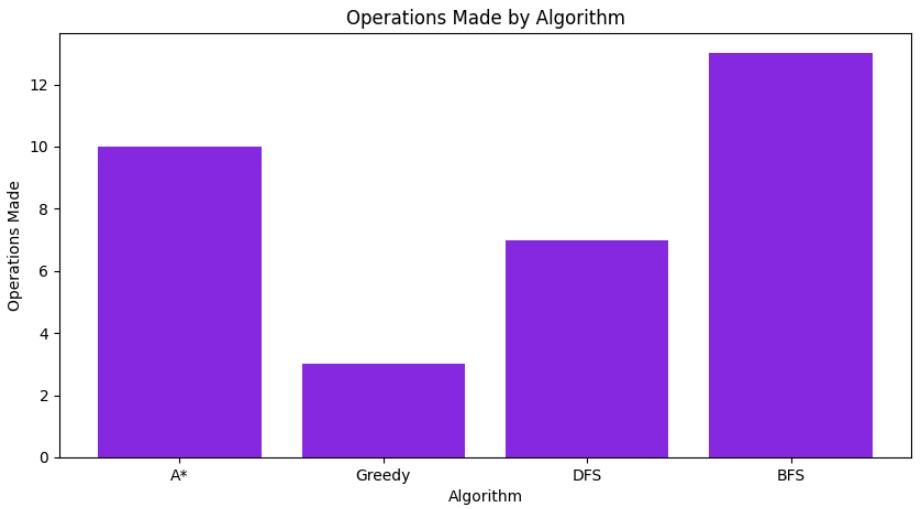
The original configuration for ‘Tower 1’.



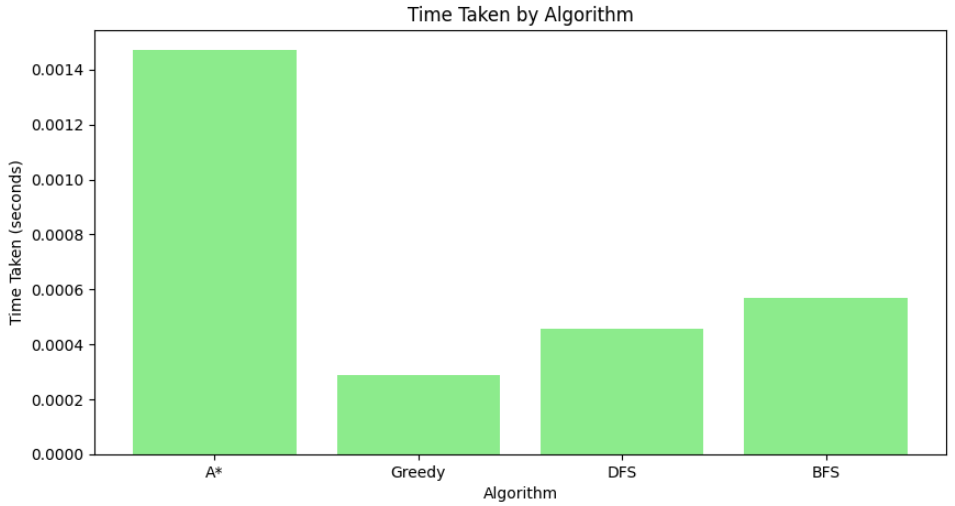
Solution length for 'Tower 1'



Memory usage for 'Tower 1'

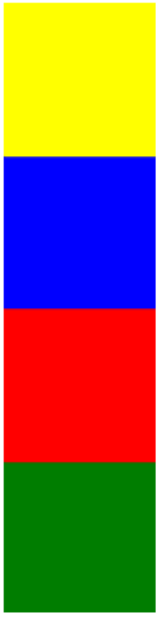


Operations for ‘Tower 1’

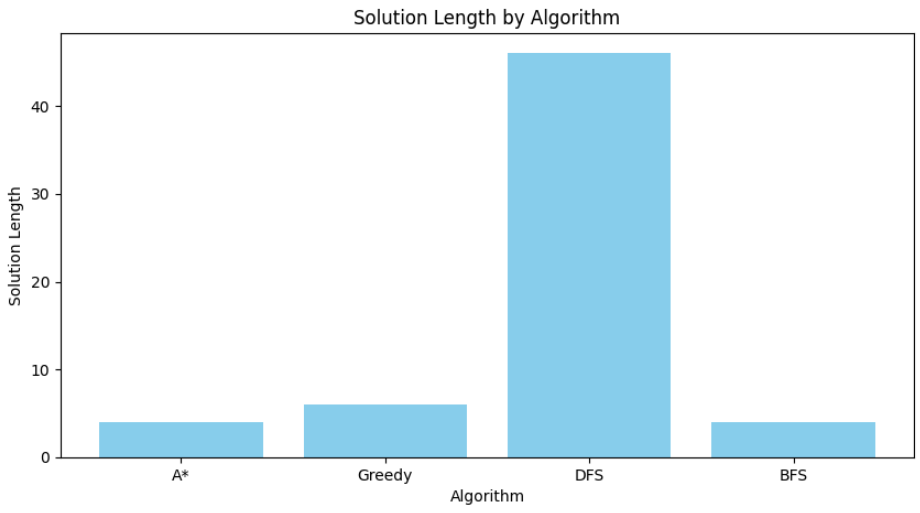


Time taken for ‘Tower 1’

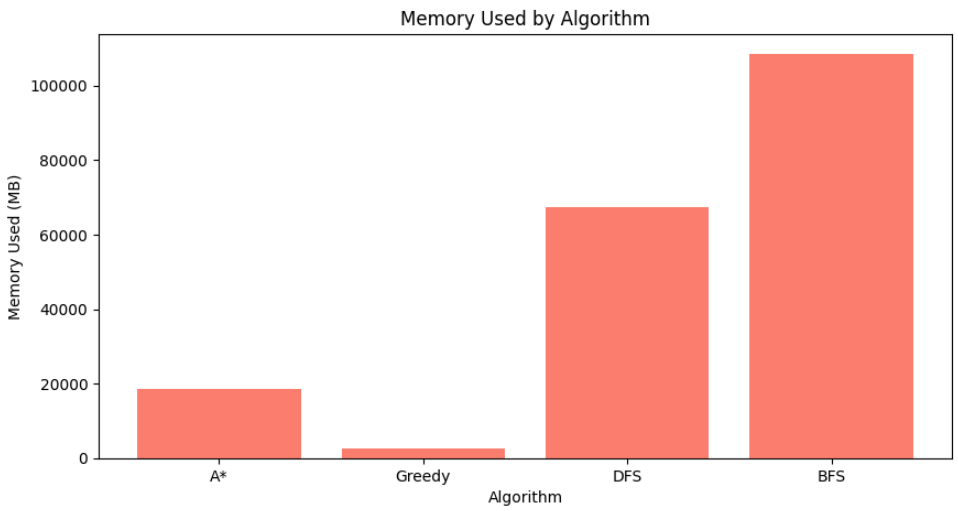
As visualised in the figures above, all algorithms except DFS found a length four solution. A* used the most time and memory, DFS used the least memory, performed the fewest operations and used the least time. ‘Tower 1’ is not a complex tower, given it is small.



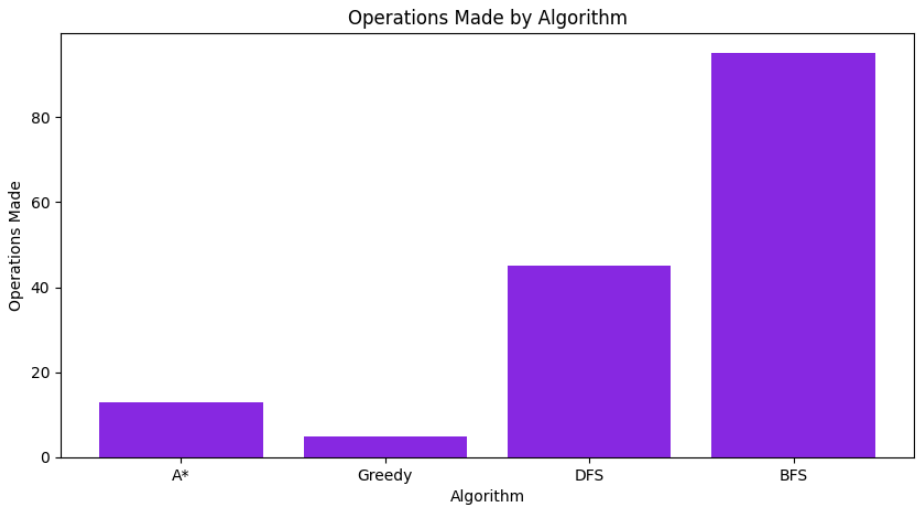
The original configuration for ‘Tower 2’.



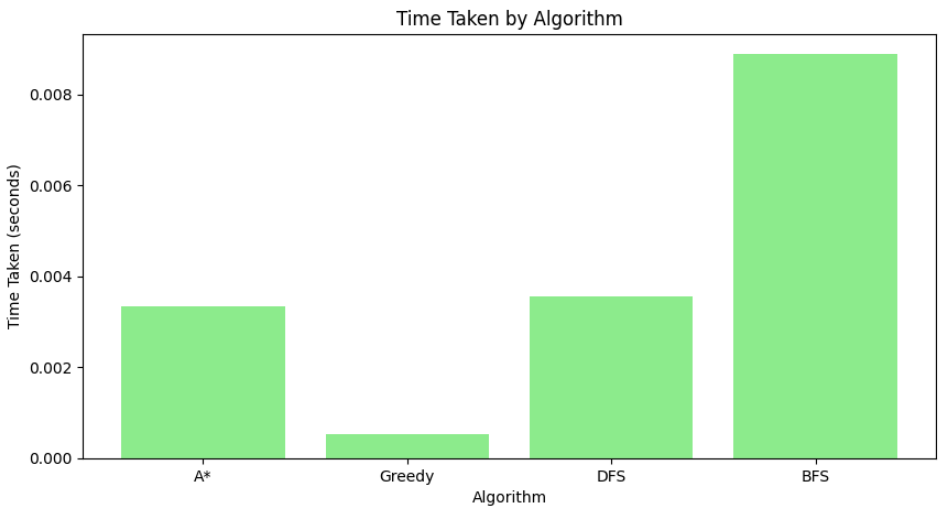
Solution length for ‘Tower 2’.



Memory used for 'Tower 2'.



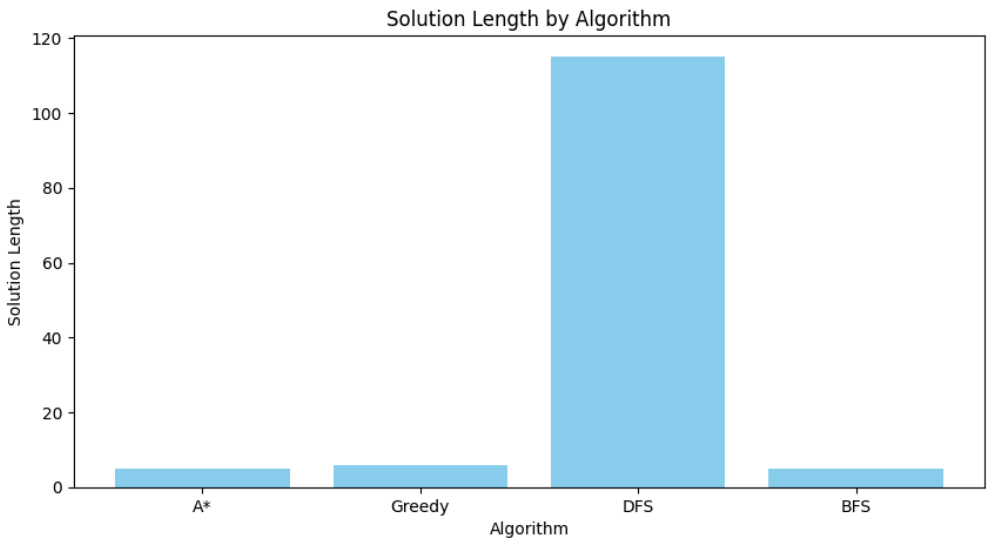
Operations for solving 'Tower 2'.



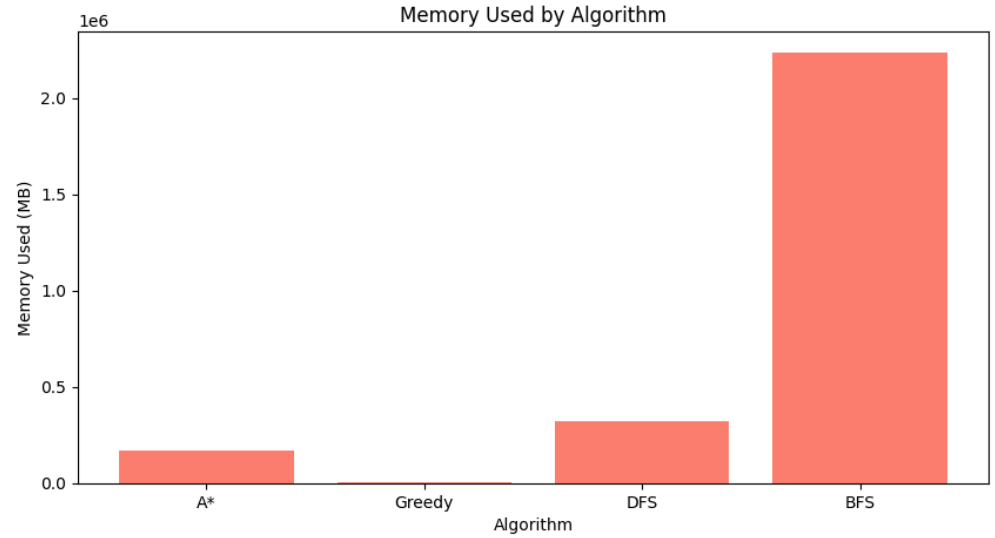
Time taken for ‘Tower 2’.

For ‘Tower 2’ BFS and A* finds the optimal, GFS is slightly worse, and DFS is far from optimal. GFS uses short time, little memory and few operations. A* is fairly good with operations and memory but uses some time. DFS and BFS are operation and memory consuming. BFS is by far the slowest.

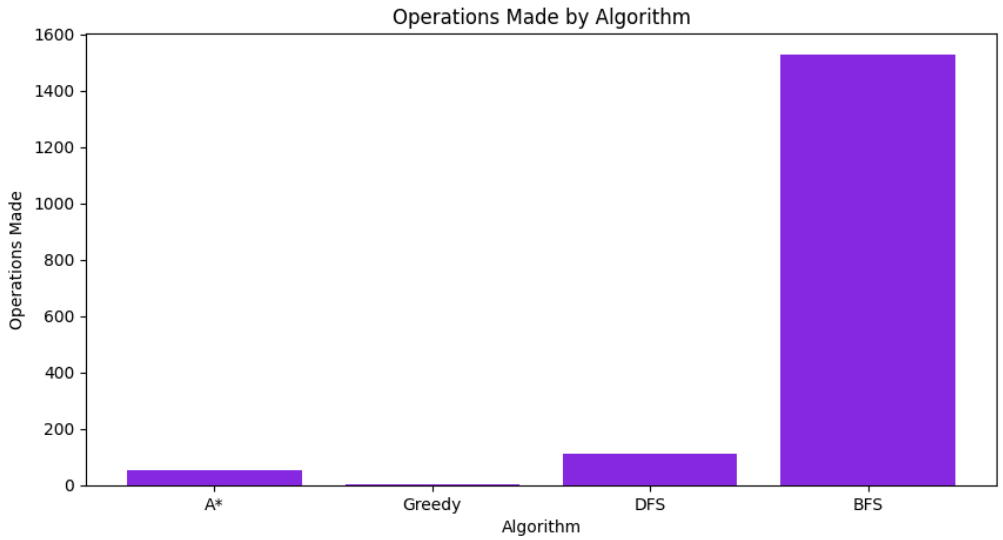
Original configuration for ‘Tower 3’.



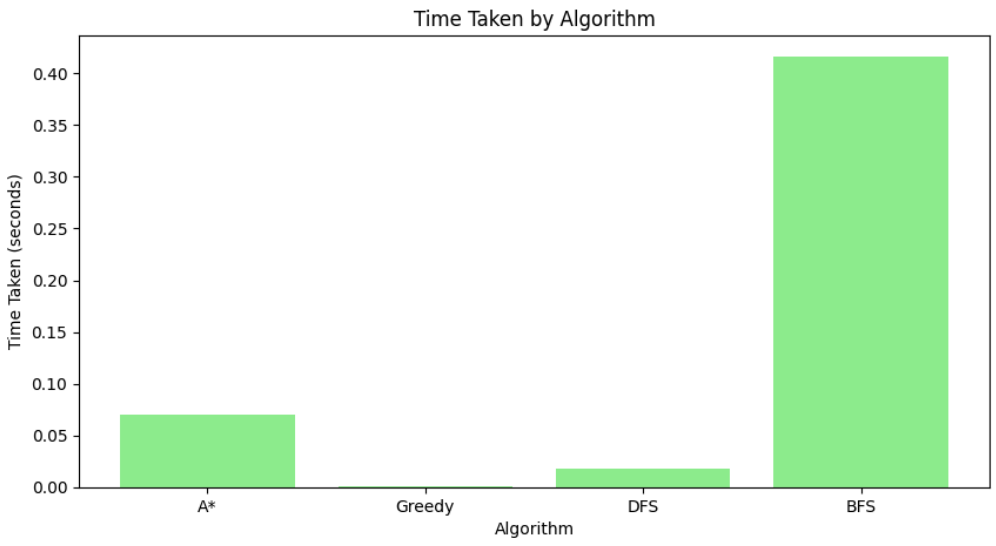
Solution length for ‘Tower 3’.



Memory used for 'Tower 3'.



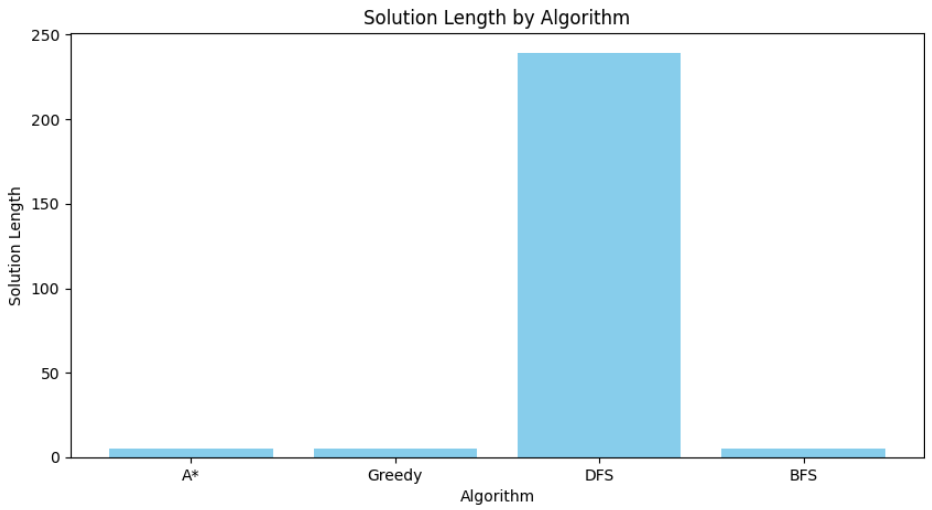
Operations for 'Tower 3'.



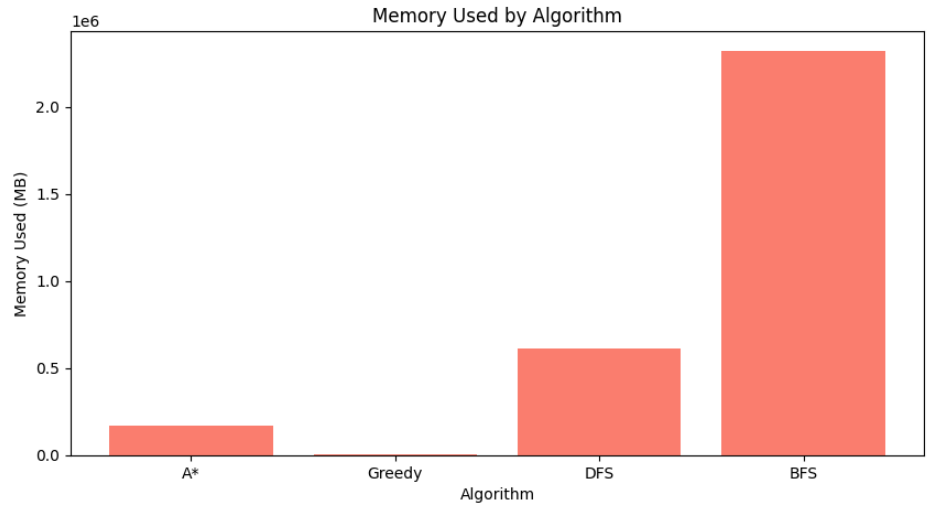
Time taken for 'Tower 3'.

The trends from Tower 2 are exaggerated when the complexity is increased. GFS is the top performer for all measures, except solution length. A* uses little memory and has the optimal solution length. BFS also finds the optimal solution, but takes a long time and consumes loads of memory in comparison to the other algorithms. DFS finds a solution second quickest, but performs many operations and uses some memory. The solution is also sub-optimal in that it is not short.

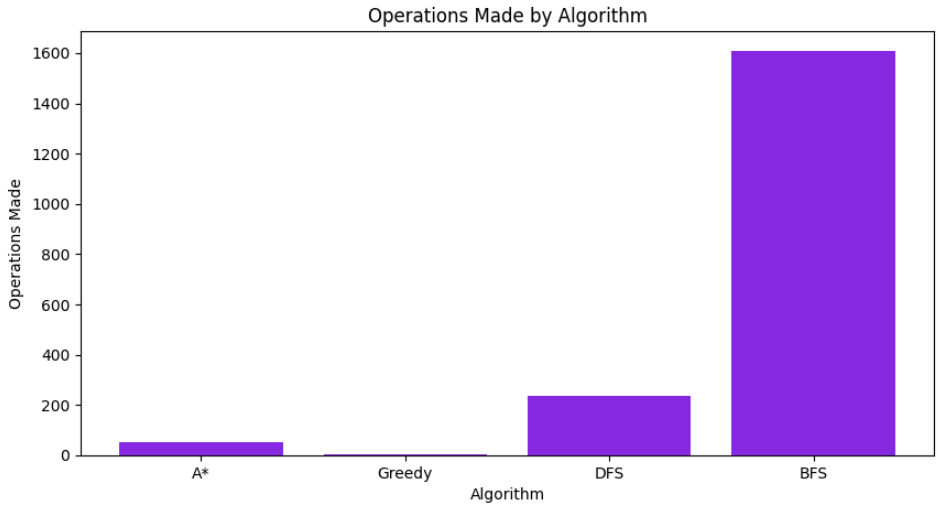
Original configuration for ‘Tower 4’.



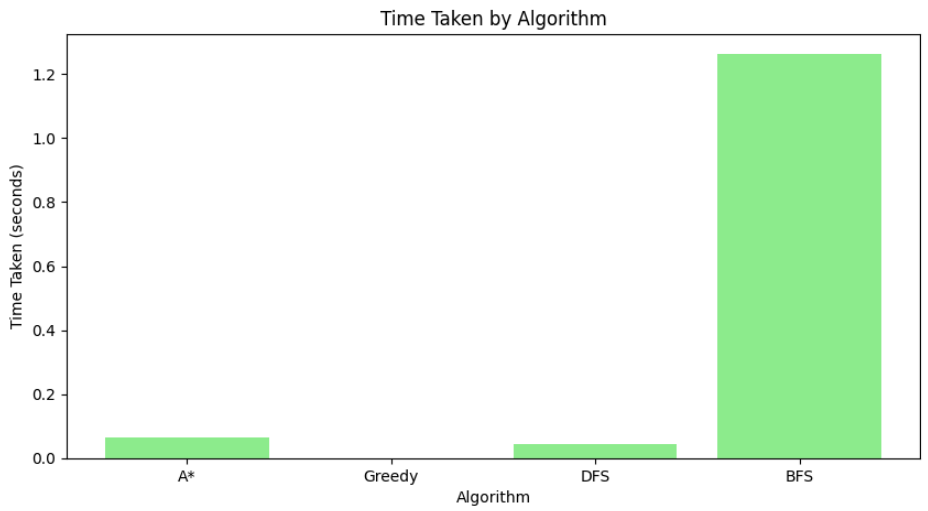
Solution length for ‘Tower 4’.



Memory used for 'Tower 4'.



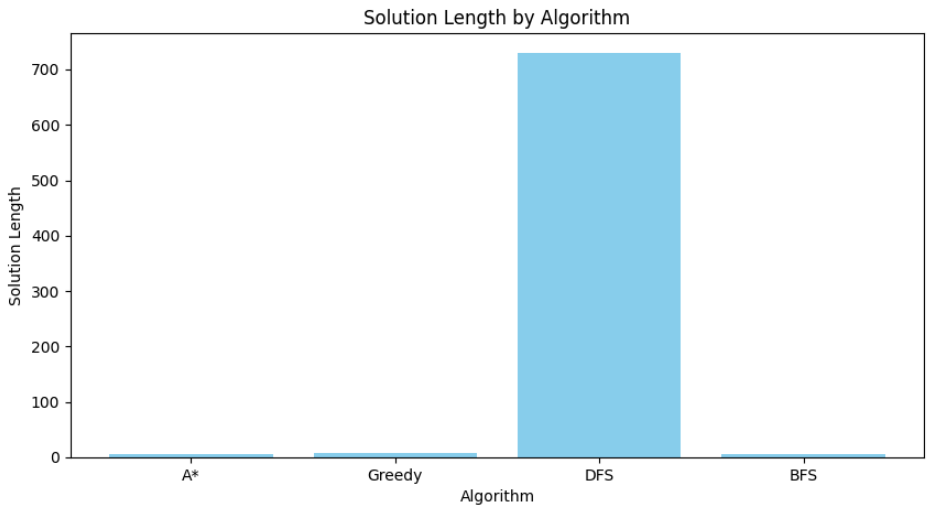
Operations for 'Tower 4'.



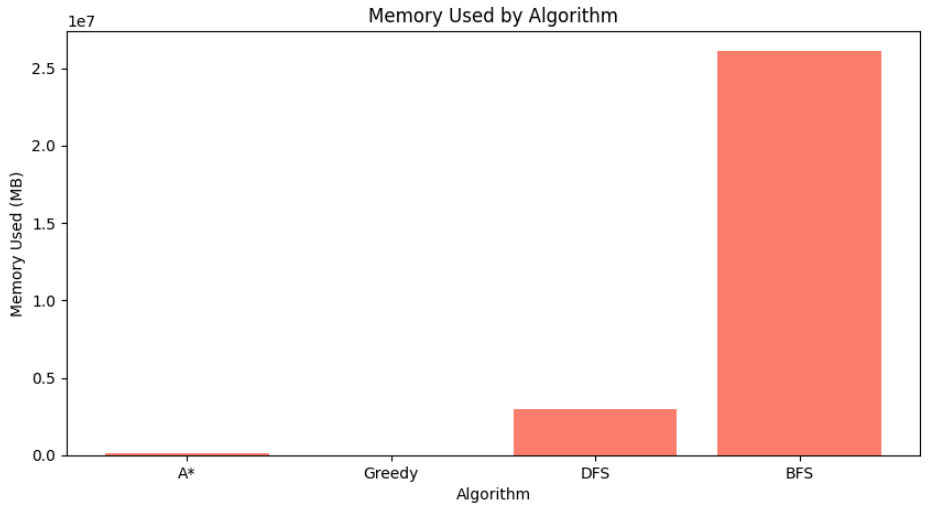
Time taken for ‘Tower 4’.

‘Tower 4’ is a more complex problem than ‘Tower 3’, given the initial configuration is farther from the solution although they are the same height. As seen in the difference between ‘Tower 2’ and ‘Tower 3’, with the increasing complexity the trends in the schematic portrayal are more scattered and apparent. The relative difference between GFS solution length and optimal solution length is less than for ‘Tower 2’, the memory, operations and time consumption is close to none. BFS uses tons of memory, time and operations compared to the other algorithms, but it finds the optimal solution.

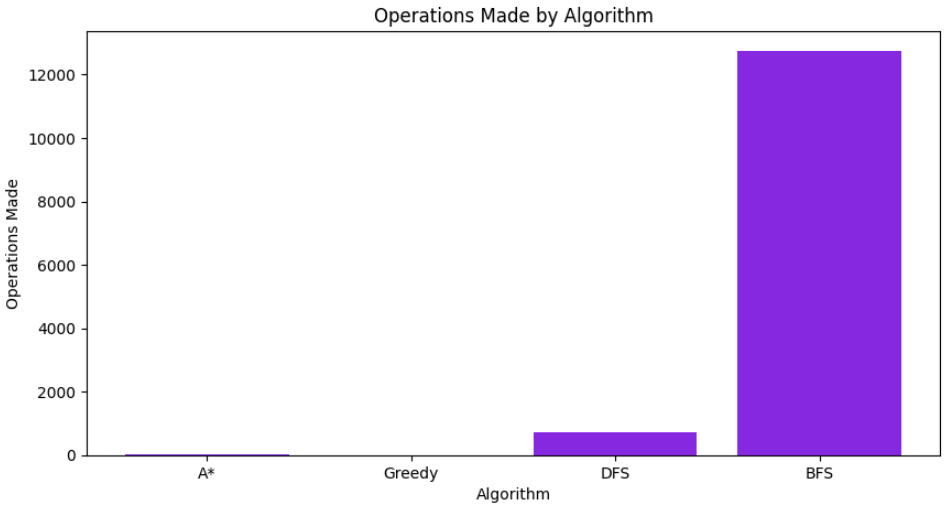
Original configuration for ‘Tower 5’.



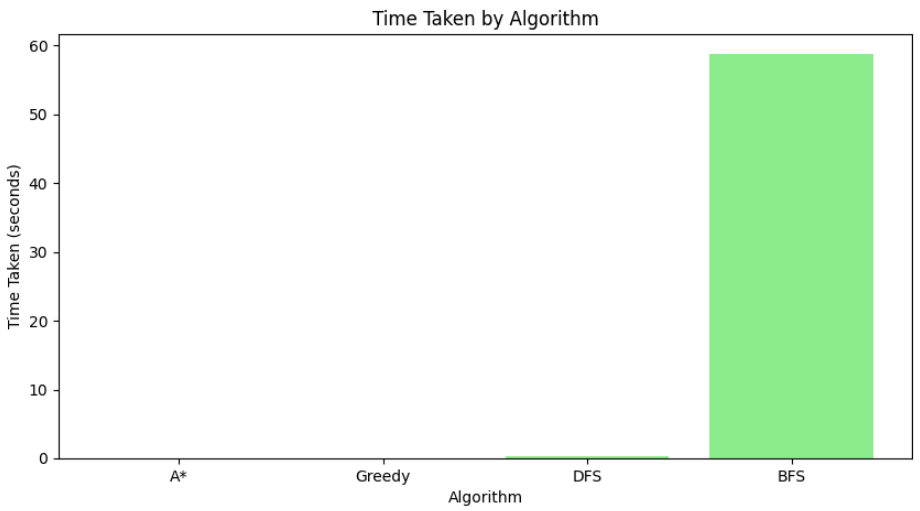
Solution length for ‘Tower 5’.



Memory used for 'Tower 5'.



Operations for 'Tower 5'.



Time taken for 'Tower 5'.

BFS' overwhelming time consumption, operations made and memory use is staggeringly high. The solution length is however optimal. GFS is top performer again, only being beat by BFS and A* for solution length.

Discussion

The heuristic seems to be a very positive integrated part of the algorithms. The heuristic works as a pruning of the state-branches not leading toward the end goal. The heuristic gives a measure of closeness to the solution, and only the best feasible states are being explored. The calculation of the heuristic is a computational setback, but the gain in not exploring useless branches is compensating this. Exploring random, non-feasible, states is memory and time consuming and the heuristic is eliminating this aspect.

The trade-off between space, time and solution optimality is interesting in this instance. The space and time are correlated in the sense that they indicate many possible states are being explored and kept in the queue. The branching factor in DFS is less than in BFS, thus resulting in less memory and time use. The high branching factor is however the saviour for BFS as it unveils the first instance of the solution, but unveiling all other possibilities before the final state. DFS is faster, less

memory consuming and performs fewer operations, in this instance, than BFS, but BFS finds the best solution. This is the same for GFS and A* respectively. GFS is semi-informed and evaluates the current states next best state, while A* has retrospective concern. A* therefore is more memory and time consuming, while also performing more operations than GFS. But it finds an equally short or shorter solution than GFS.

Conclusion

All in all, the complexity of the problem and the type of is determining the best algorithm. In this case the least memory, time and operations are yielded from GFS. A* and BFS yields the optimal solution always, but both are worse performing than GFS in other aspects. Thus leaving the best of both worlds being A*, since it is optimal and is less time, memory and operationally craving than BFS. If time and space is of the essence GFS is the best option, but if the solution should be optimal A* should be chosen.

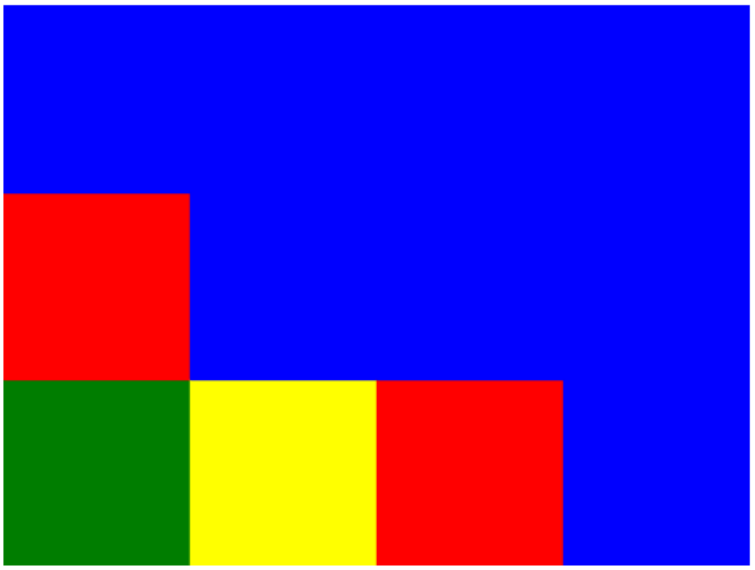
References

ARK1375. (2021, July 19). *Answer to 'Amount of memory used in a function?'* Stack Overflow. https://stackoverflow.com/a/68440973

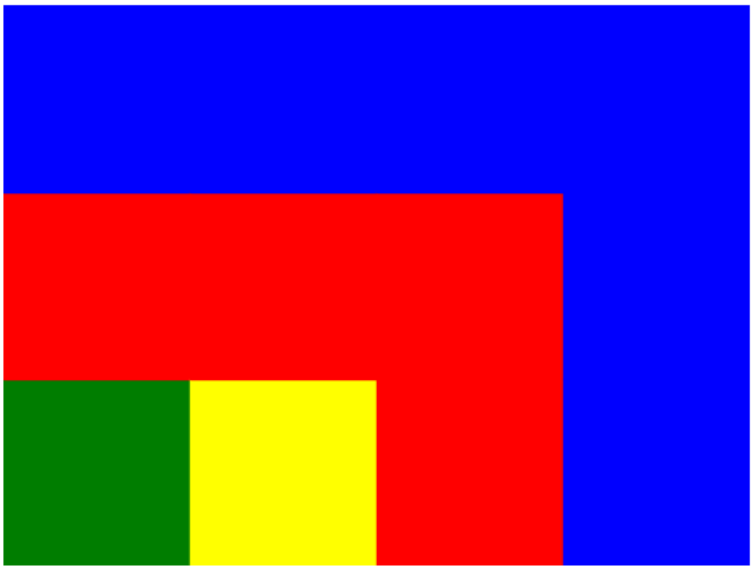
Difference between Informed and Uninformed Search in AI. (2020, April 3). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-informed-and-uninformed-search-in-ai/

Nordmo, T.-A. S. (2024, January 23). *Search, Uninformed—Informed INF-2600*.

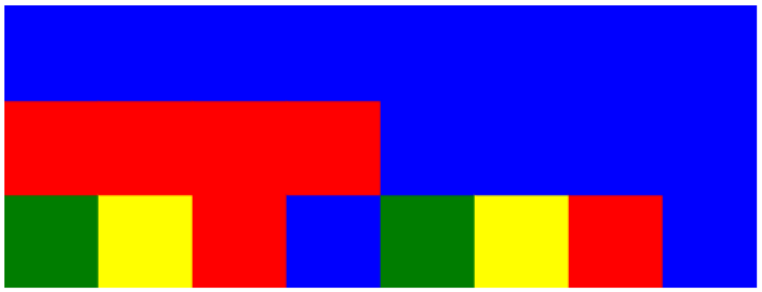
Appendix



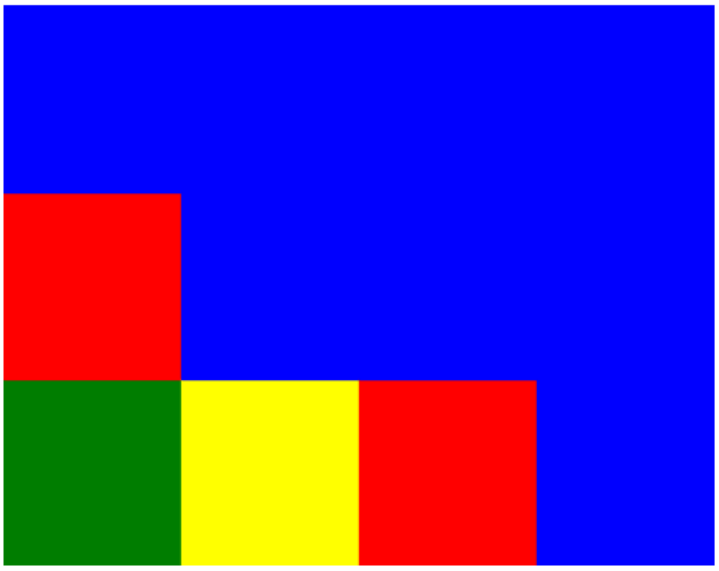
A* solution for ‘Tower 1’



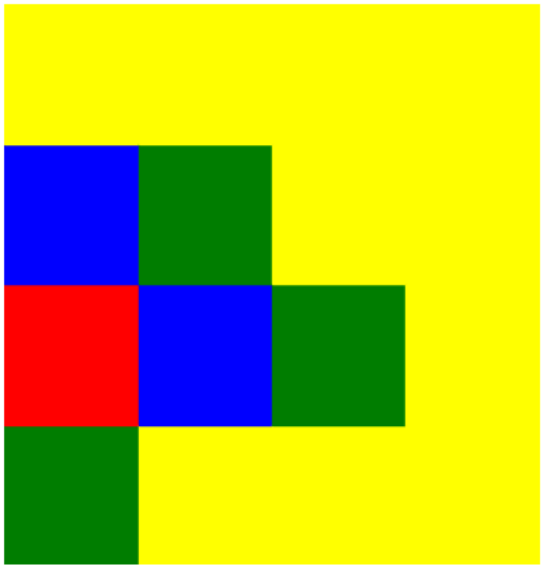
BFS solution for 'Tower 1’



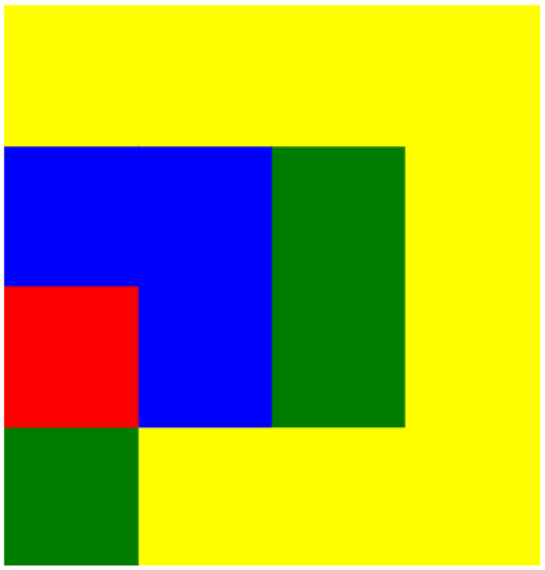
DFS solution for 'Tower 1’



GFS solution for 'Tower 1’

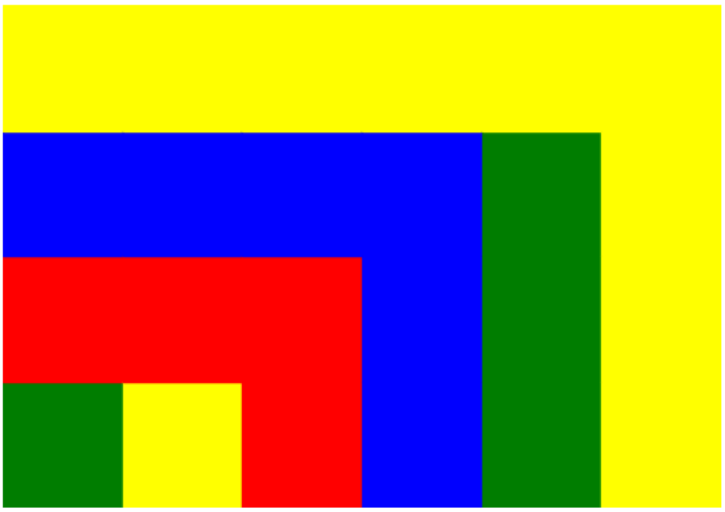


A* solution for ‘Tower 2’

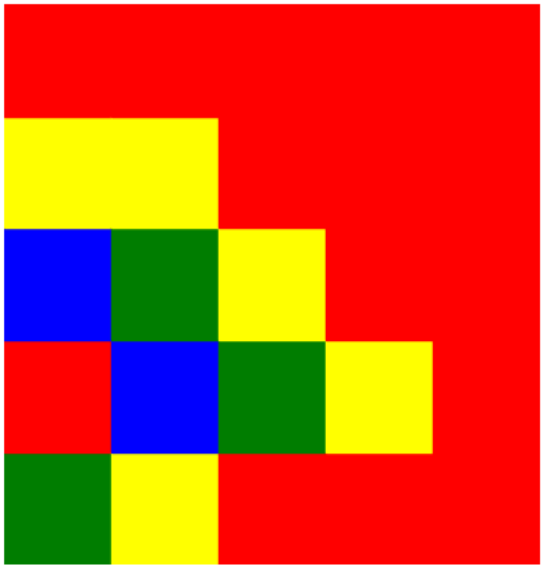


BFS solution for 'Tower 2’

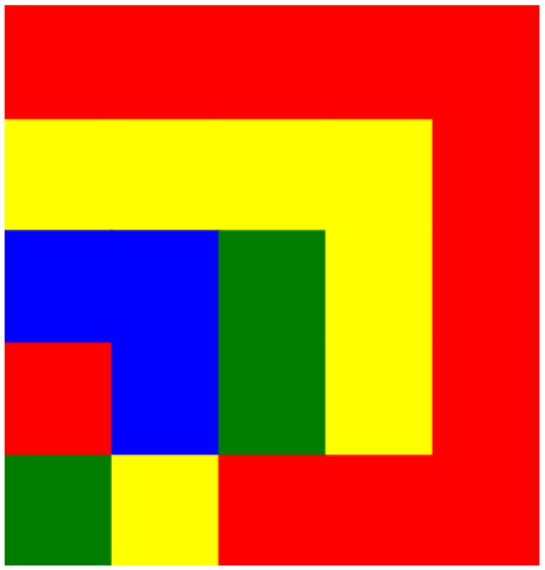
DFS solution for 'Tower 2’



GFS solution for 'Tower 2’

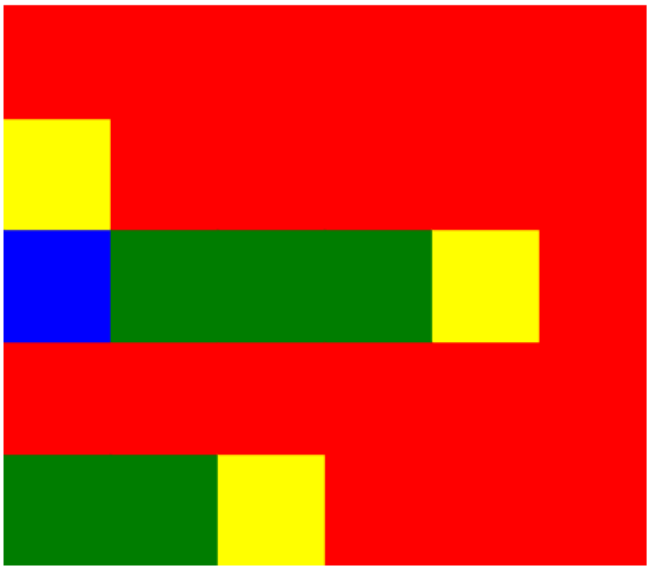


A* solution for ‘Tower 3’

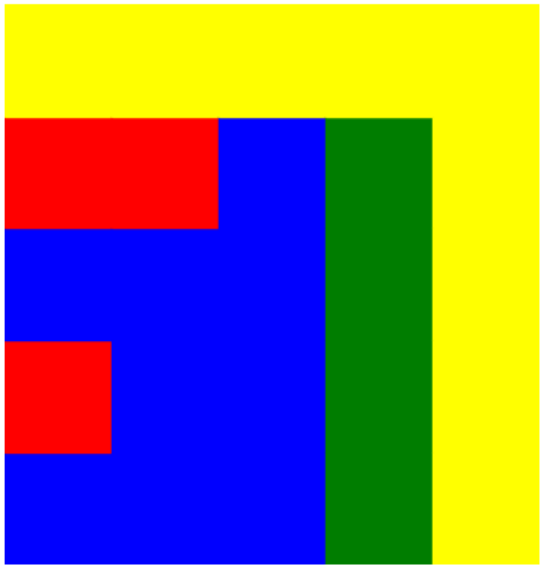


BFS solution for 'Tower 3’

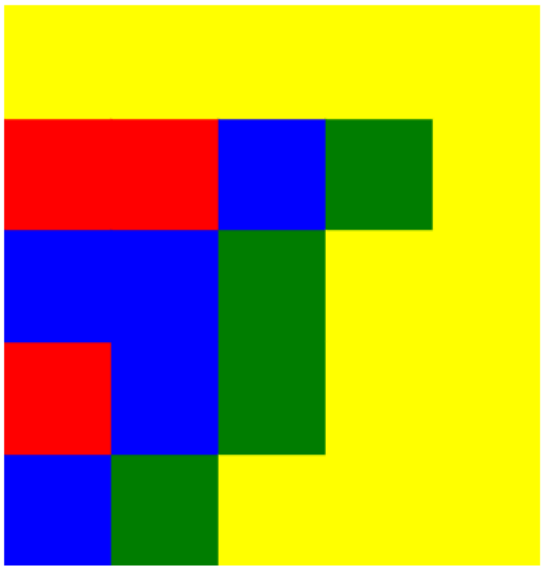
DFS solution for 'Tower 3’



GFS solution for 'Tower 3’



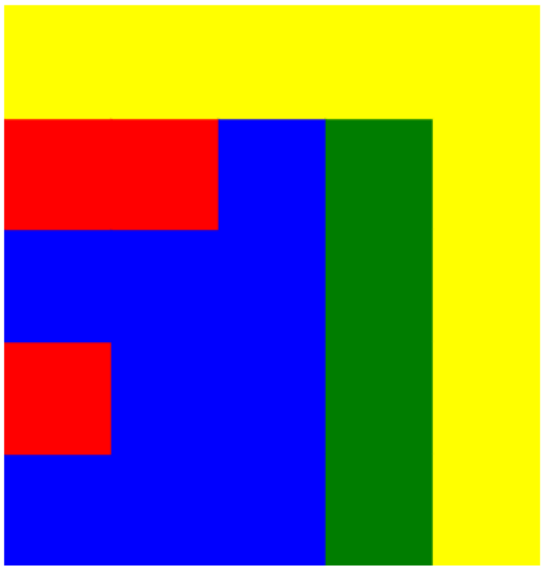
A* solution for ‘Tower 4’



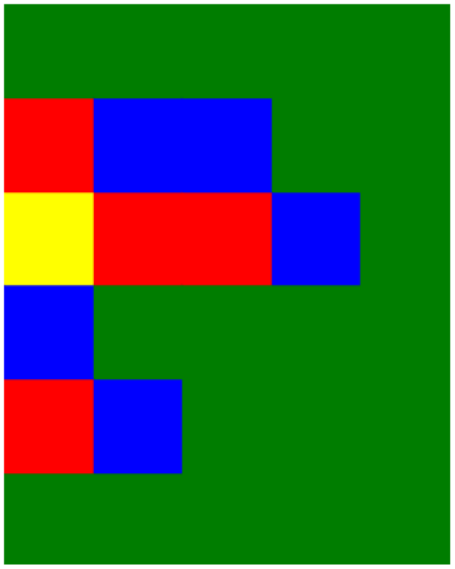
BFS solution for 'Tower 4’

DFS

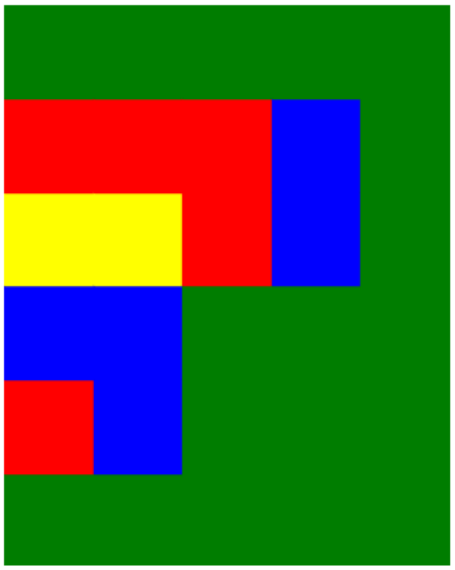
solution for 'Tower 4’



GFS solution for 'Tower 4’

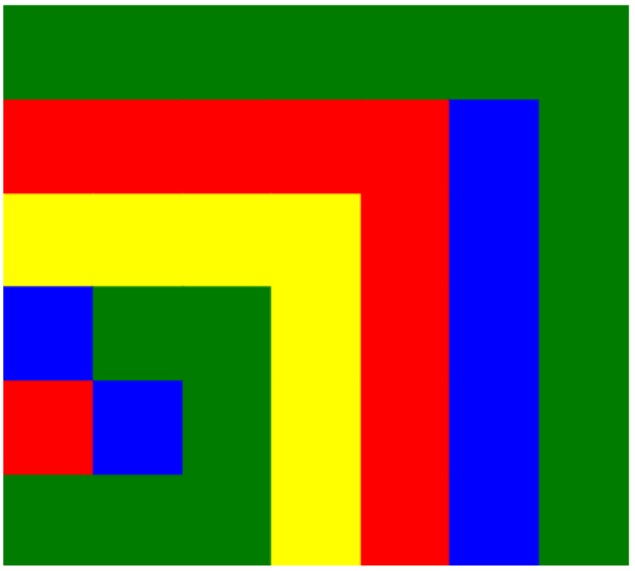


A* solution for ‘Tower 5’



BFS solution for 'Tower 5’

DFS solution for 'Tower 5’



GFS solution for 'Tower 5’