N-QUEEN PROBLEM

USING SEARCHING ALGORITHMS

INTRODUCTION TO ARTIFICIAL INTELLIGENCE

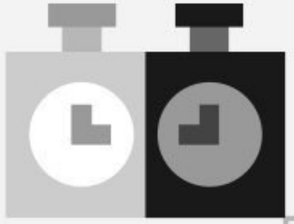
ASSIGNMENT 1

Presentation video: https://youtu.be/RrRy1sTh59M

MEMBERS LIST

#	Full name	ID
1	Tạ Đình Tiến	2110583
2	Trần Nguyễn Thái Binh	2110051
3	Nguyễn Tuấn Minh	2110359
4	Huỳnh Thái Học	2113443

TABLE OF CONTENTS

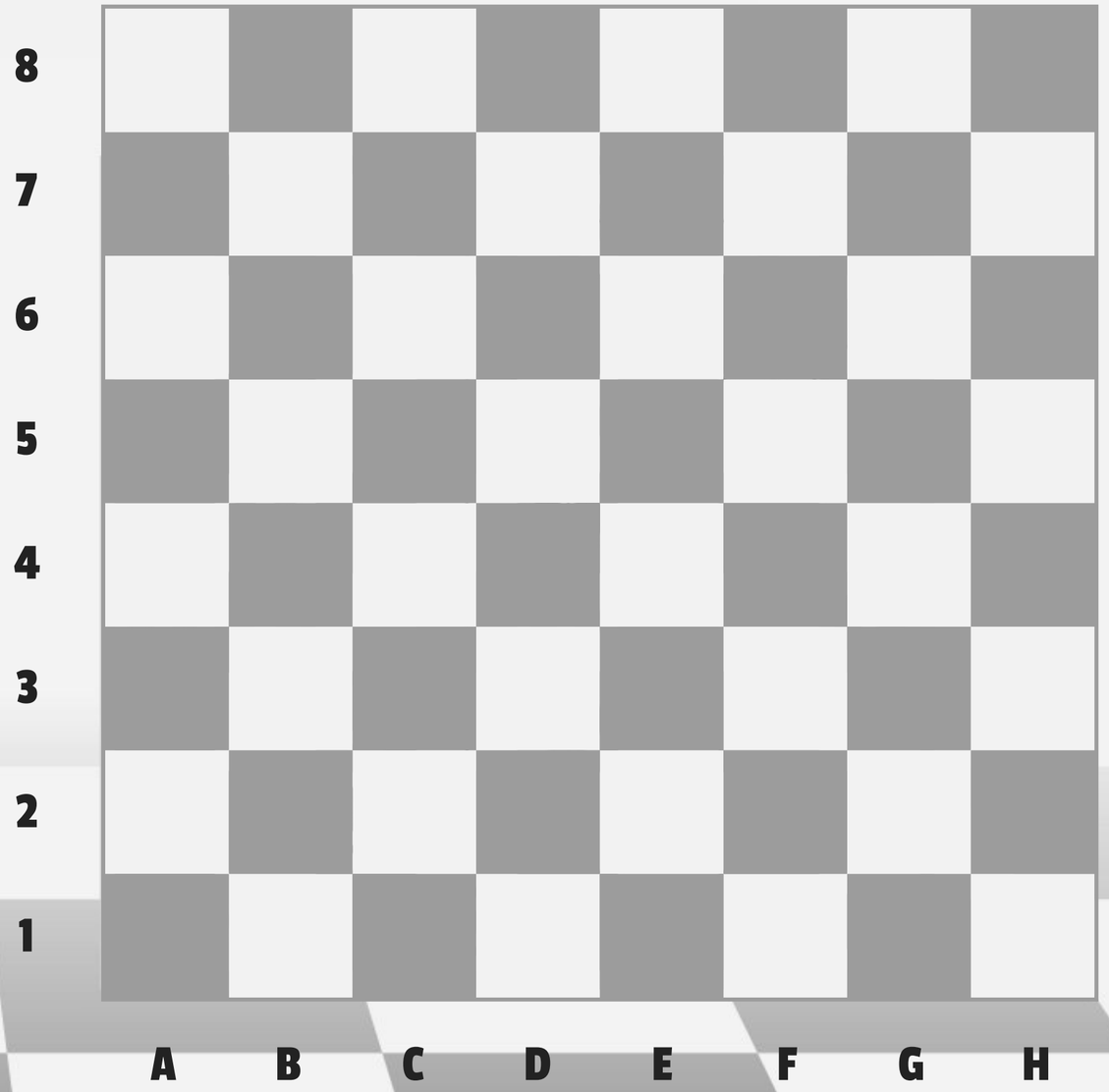


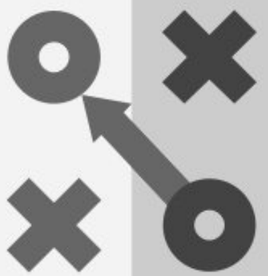
01

INTRODUCTION

CHESSBOARD (8x8)

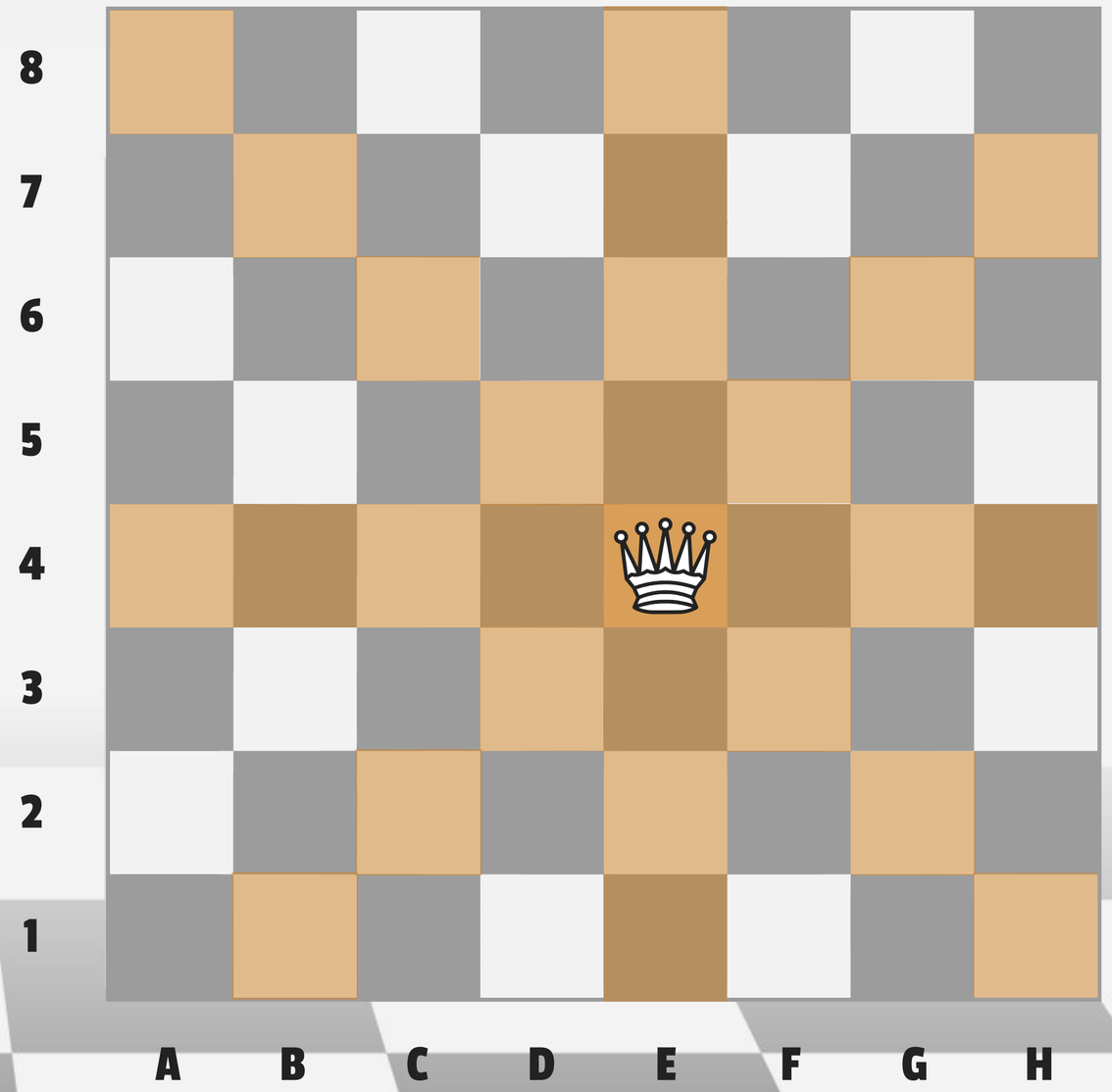
A basic chessboard is a gameboard used to play chess. It consists of 64 squares, 8 rows by 8 columns, on which the chess pieces are placed. It is square in shape and uses two colors of squares, one light and one dark, in a chequered pattern.





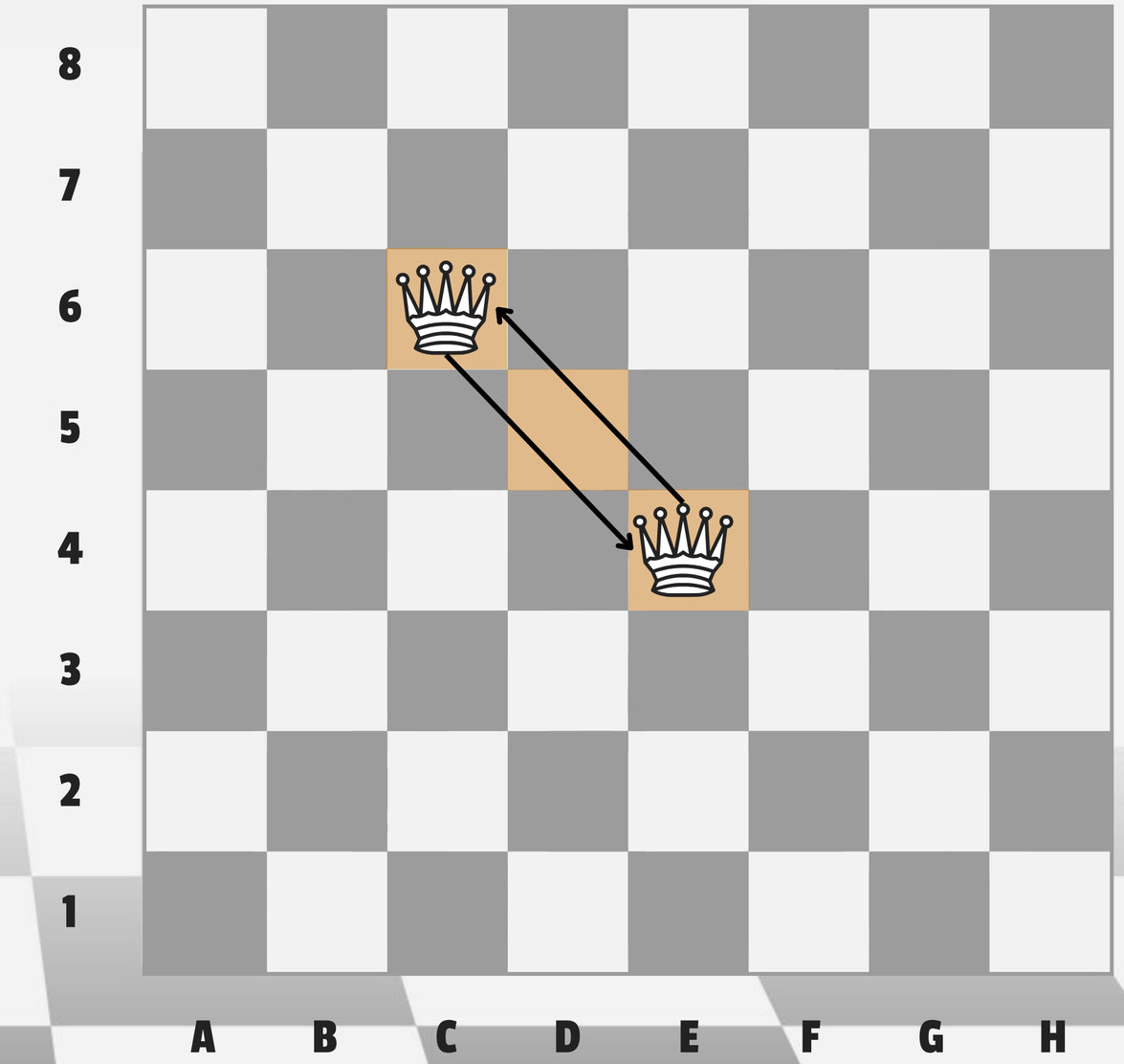
CHESS PIECE: QUEEN

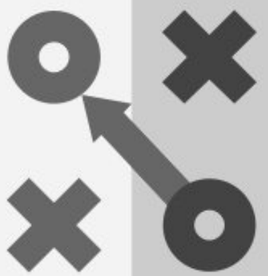
The queen ♕ is the most powerful piece in the game of chess. It can move and attack any number of squares vertically, horizontally or diagonally.



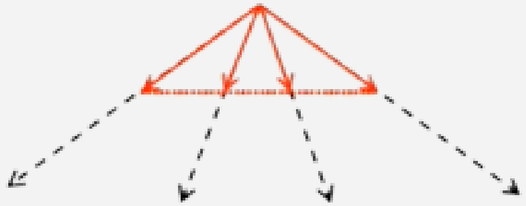
N-QUEEN PROBLEM

The N-Queen is the problem of placing N chess queens on an N×N chessboard so that no two queens attack each other.

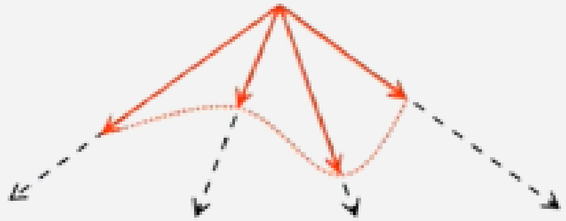




SEARCHING ALGORITHMS



Uninformed Search



Informed Search

SEARCHING ALGORITHMS: UNINFORMED SEARCH

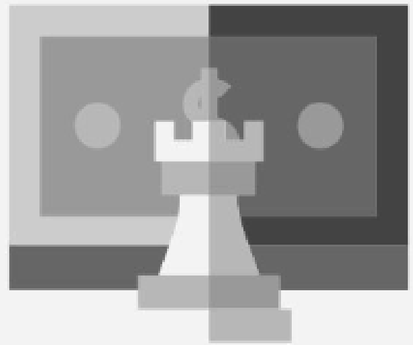
Uninformed search algorithms (also called **Blind Search**) have no additional information on the goal node other than the one provided in the problem definition.

The plans to reach the goal state from the start state differ only by the order and/or length of actions.

These algorithms can only generate the successors and differentiate between the goal state and non goal state.

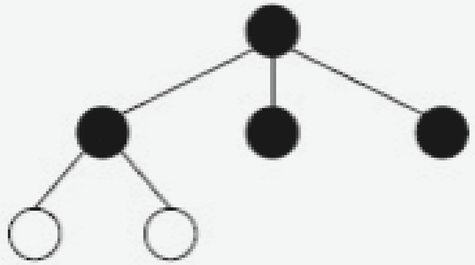
E.g. Breadth-First Search (BrFS), Depth-First Search (DFS), Uniform Cost Search,...

SEARCHING ALGORITHMS: UNINFORMED SEARCH



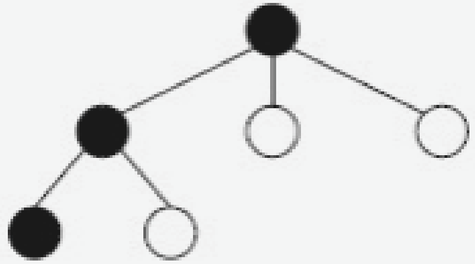
BREADTH-FIRST SEARCH

Expand all the nodes of one level first.



DEPTH-FIRST SEARCH

Expand one of the nodes at the deepest level.



SEARCHING ALGORITHMS: UNINFORMED SEARCH

Criterion	BrFS	DFS
Time	b^d	b^m
Space	b^d	bm
Optimal	Yes	No
Complete	Yes	No

Complexity of BrFS and DFS

b: branching factor m: maximum depth d: solution depth

SEARCHING ALGORITHMS: INFORMED SEARCH

Informed search algorithms (also called **Heuristic Search**) using additional information called “knowledge” in searching process.

Can evaluate the current state using function called heuristic function, thereby finding the way to the solution faster.

Traverse fewer nodes (states) than with **Blind Search.**

Sometimes optimization or completeness can be sacrificed compared to other search algorithms, but it still brings efficiency.

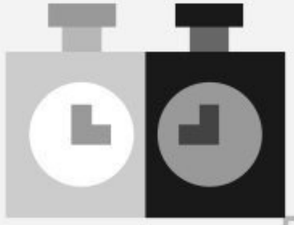
E.g. Best-First Search (BFS), A* Search, Greedy Search, Monte Carlo Search,...

SEARCHING ALGORITHMS: INFORMED SEARCH - QS2 ALGORITHM

The Queen Searching 2 (QS2) algorithm is based on the idea of setting up a heuristic function that counts **the number of collisions** between queens on the chessboard, then randomly selects a queen that is being attacked and swap rows with another queen, this could be performed only when the number of collisions between queens is reduced. The goal state is the state where number of collisions is equal to zero.

If the initial state spends too much time for random swapping attacked queens but the goal state can not be reached. A new randomly initial state will be generated. This can help avoiding stucking in some local state areas.

Link paper: https://sci-hub.se/https://doi.org/10.1109/21.135698?fbclid=IwAR2uuCqv9bPckQDUO0FajFNMCvvRyXgTmzmAur12jAdg3nkhb2IfoQ3e2w0



02

SPACE STATE

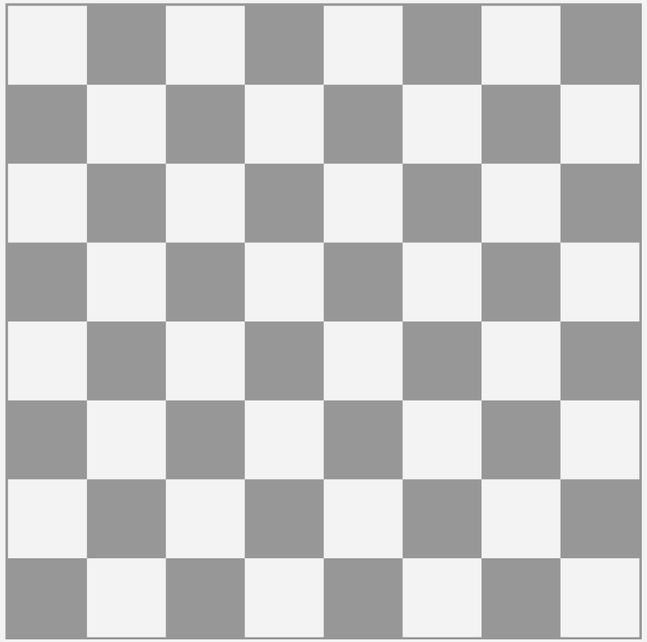
SPACE STATE: CHESSBOARD REPRESENTATION

In our algorithms, a square chessboard size N will be represented as **a list ‘queen’ size S** (S less than or equal to N).

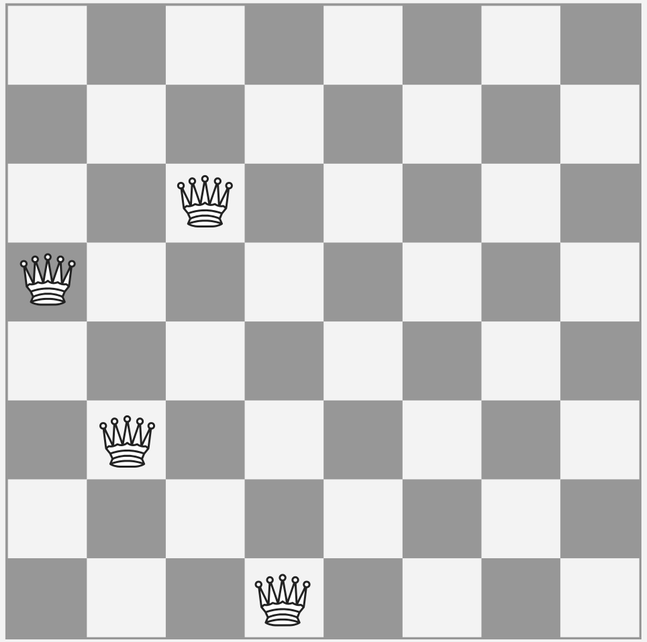
S represents numbers of queen placed in the chessboard, these queens are placed in column order from left to right.

The value of **i**th element of list ‘queen’ represents the order of row which the queen at column **i**th from top to bottom.

EXAMPLE

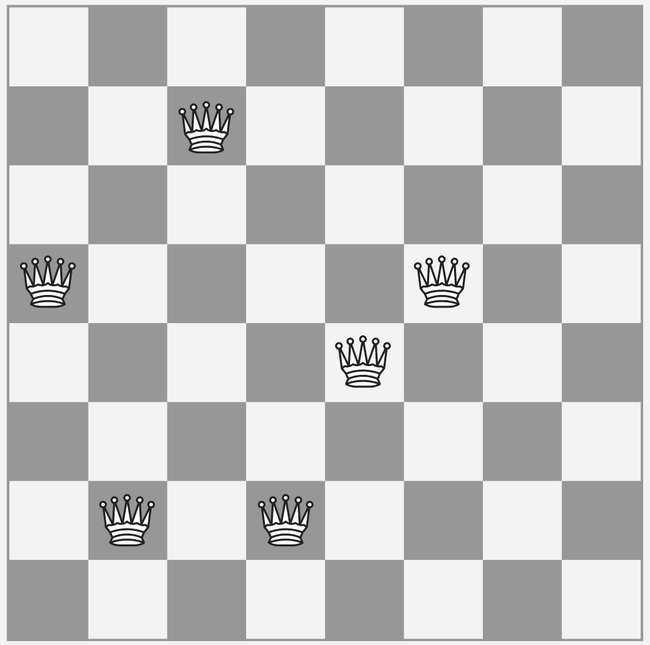


queen = []

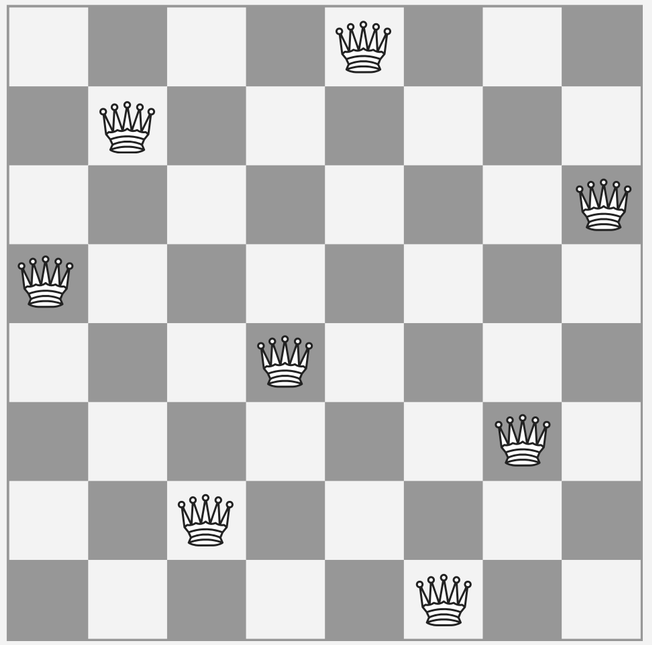


queen = [3, 5, 2, 7]

EXAMPLE

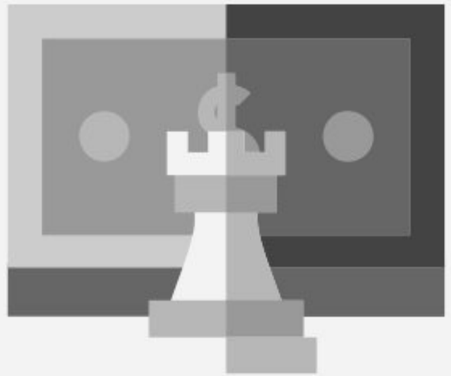


queen = [3, 6, 1, 6, 4, 3]



queen = [3, 1, 6, 4, 0, 7, 5, 2]

SPACE STATE: BrFS AND DFS



STATE

The chessboard (NxN) contains S queens (0 <= S <= N) on the first S columns, each column containing only one chess piece.

GOAL STATE

All N queens are placed on the chessboard and no two queens attack each other

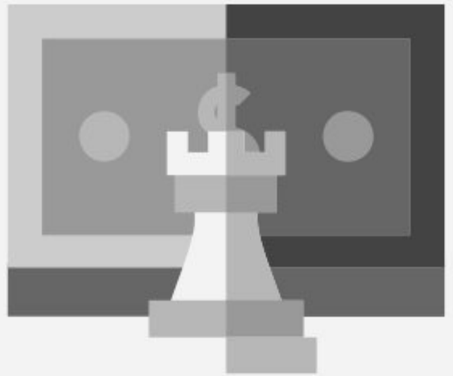
INITIAL STATE

The board is empty - no queen has been placed yet.

LEGAL MOVES

Place a queen to any position in the first empty column (from left to right) which is not attacked by existing queens

SPACE STATE: BrFS AND DFS



STATE

queen = [3, 1, 6, 4, 0]

queen = [3, 1, 6, 4, 0, 7, 5]

list ‘queen’ size S with queen[i]

value in range(0, N-1)

GOAL STATE

queen = [3, 1, 6, 4, 0, 7, 5, 2]

(solution for 8-Queen problem)

INITIAL STATE

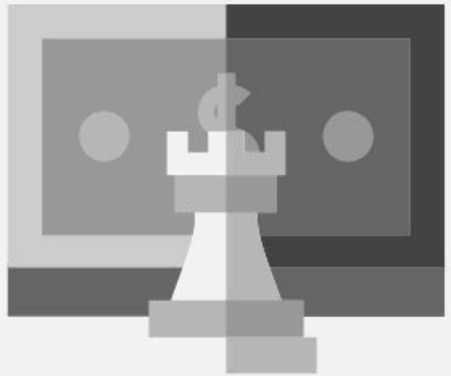
queen = []

LEGAL MOVES

queen = [3, 1, 6, 4, 0, 7, 5]

-> queen = [3, 1, 6, 4, 0, 7, 5, 2]

SPACE STATE: QS2



STATE

The chessboard (NxN) contains N queens, each placed on a different columns and rows

GOAL STATE

All N queens are placed on the chessboard and no two queens attack each other

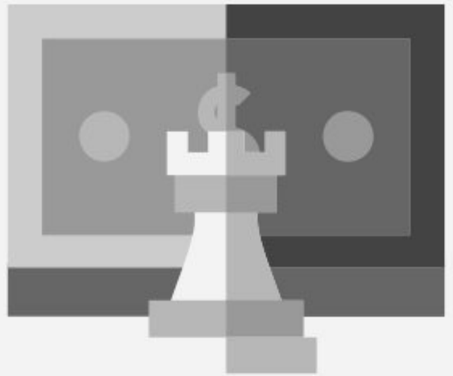
INITIAL STATE

N queens are placed on the chessboard in different columns and rows (random state)

LEGAL MOVES

Choose 2 columns which at least one of them has attacked queen, swap rows of 2 queen on those columns if collisions decreases

SPACE STATE: QS2



STATE

queen = [a, b, c, d, e, f, g, h]
where a, b, c, d, e, f, g, h are unique
in range(0, N-1)

GOAL STATE

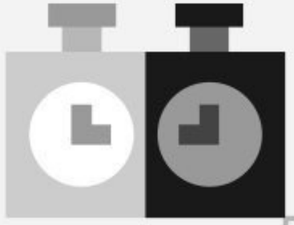
queen = [3, 1, 6, 4, 0, 7, 5, 2]
(solution for 8-Queen problem)

INITIAL STATE

queen = Permutation of [0, 1, 2,..,
N-1]

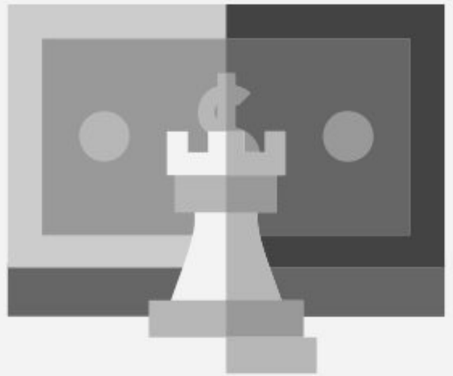
LEGAL MOVES

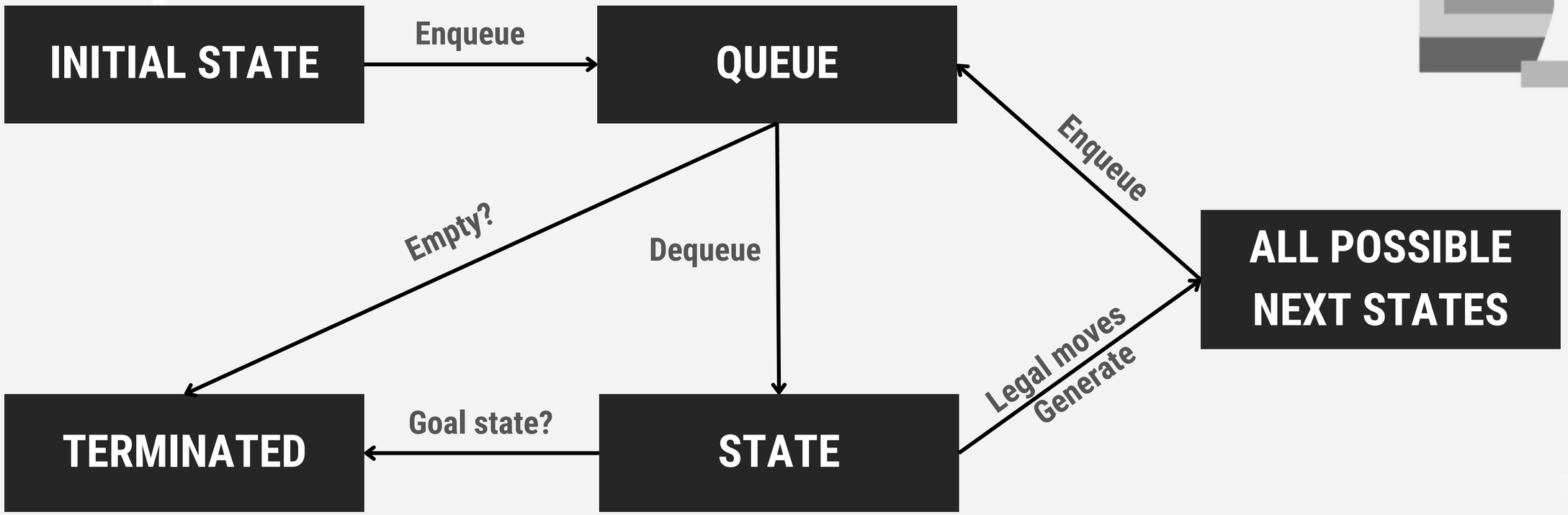
queen = [3, **7**, 6, 4, 0, **1**, 5, 2] (collisions = 2)
-> queen = [3, **1**, 6, 4, 0, **7**, 5, 2] (collisions = 0)



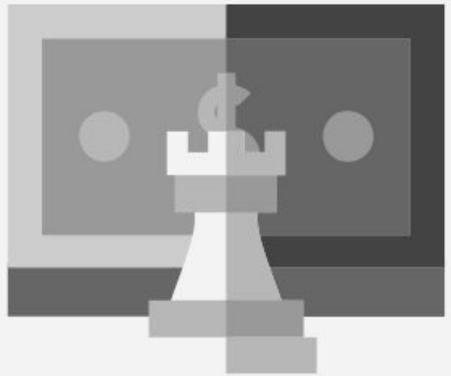
03 ALGORITHM IMPLEMENTATION

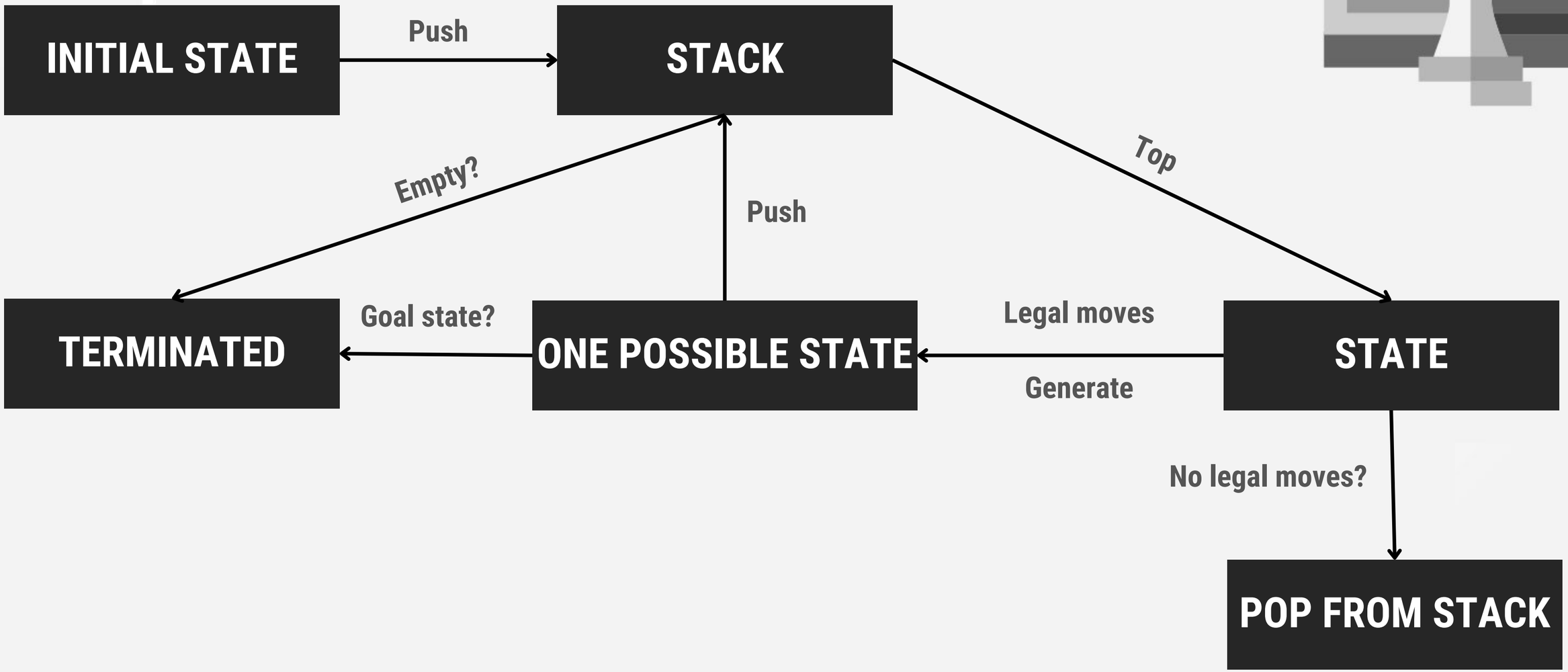
IMPLEMENTATION: Breadth-First Search (BrFS)



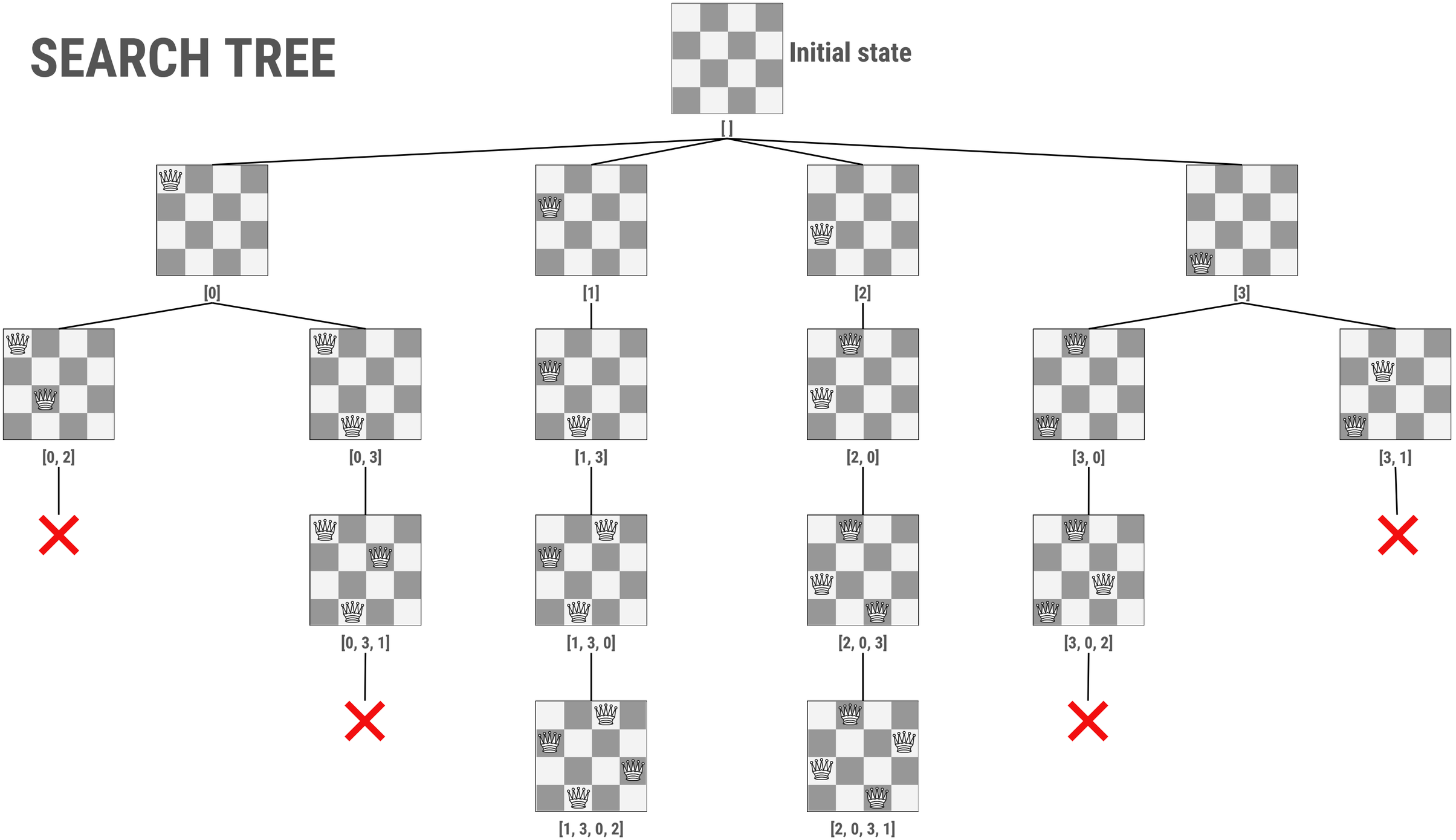


IMPLEMENTATION: Depth-First Search (DFS)

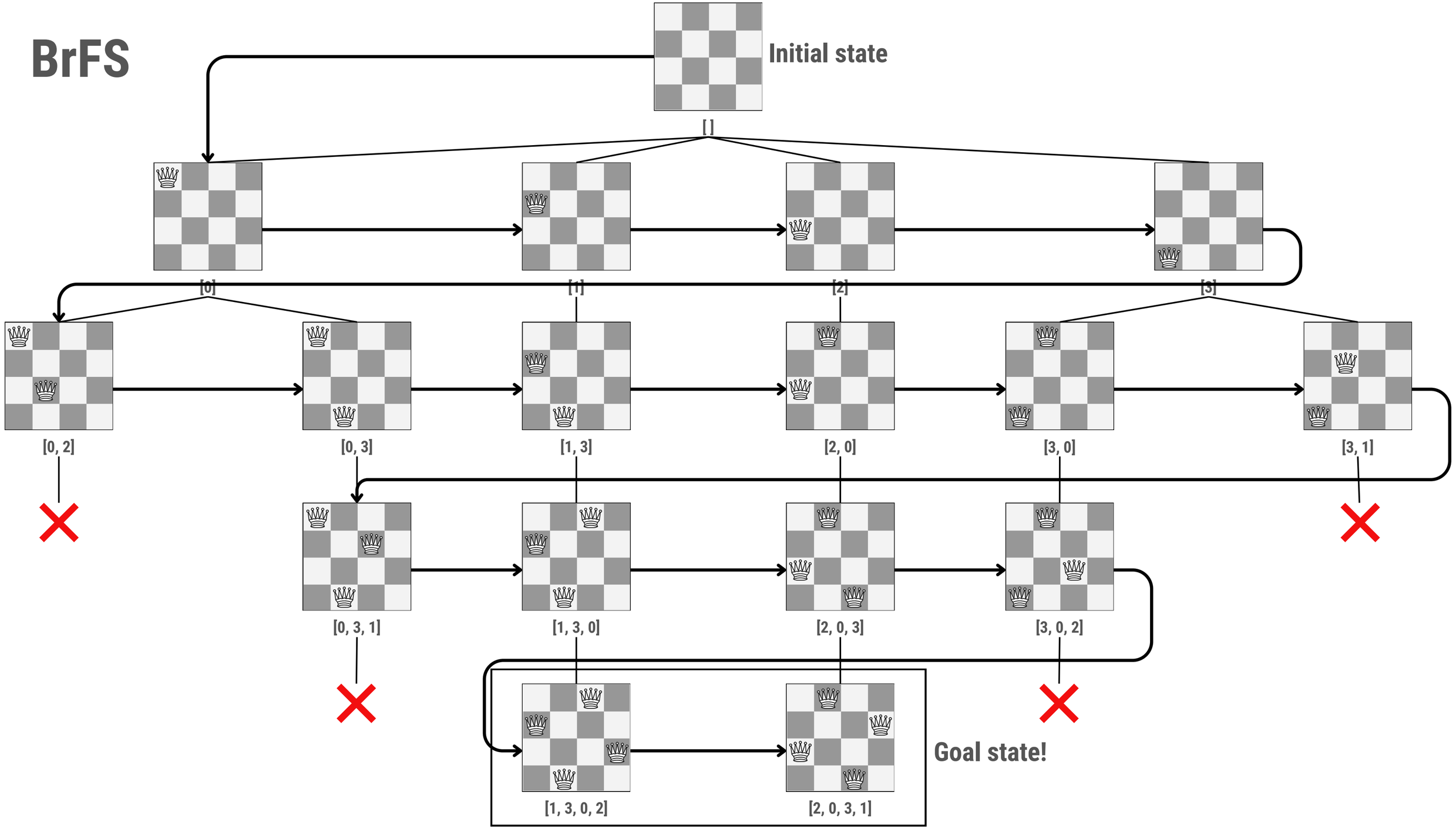




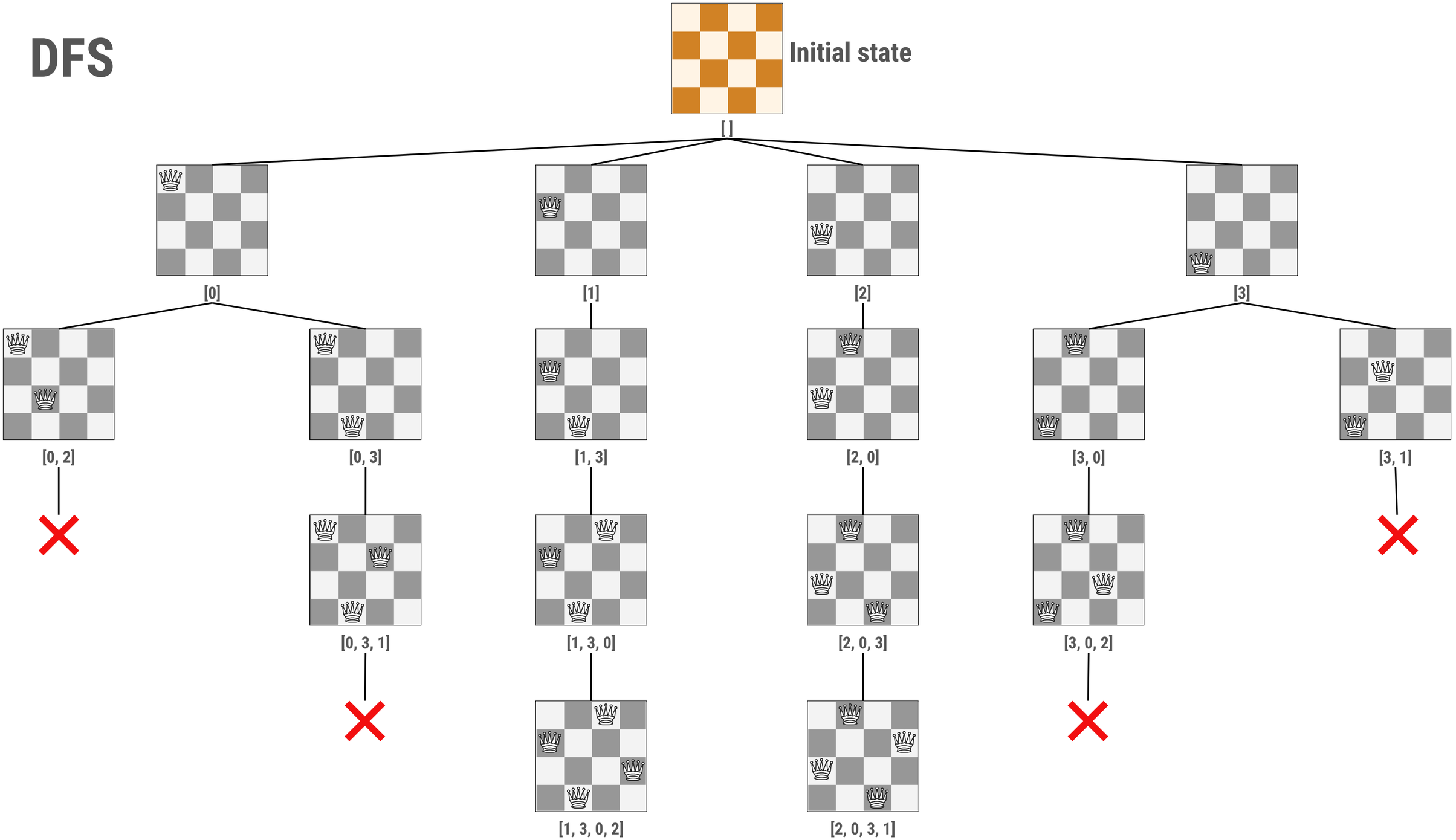
SEARCH TREE



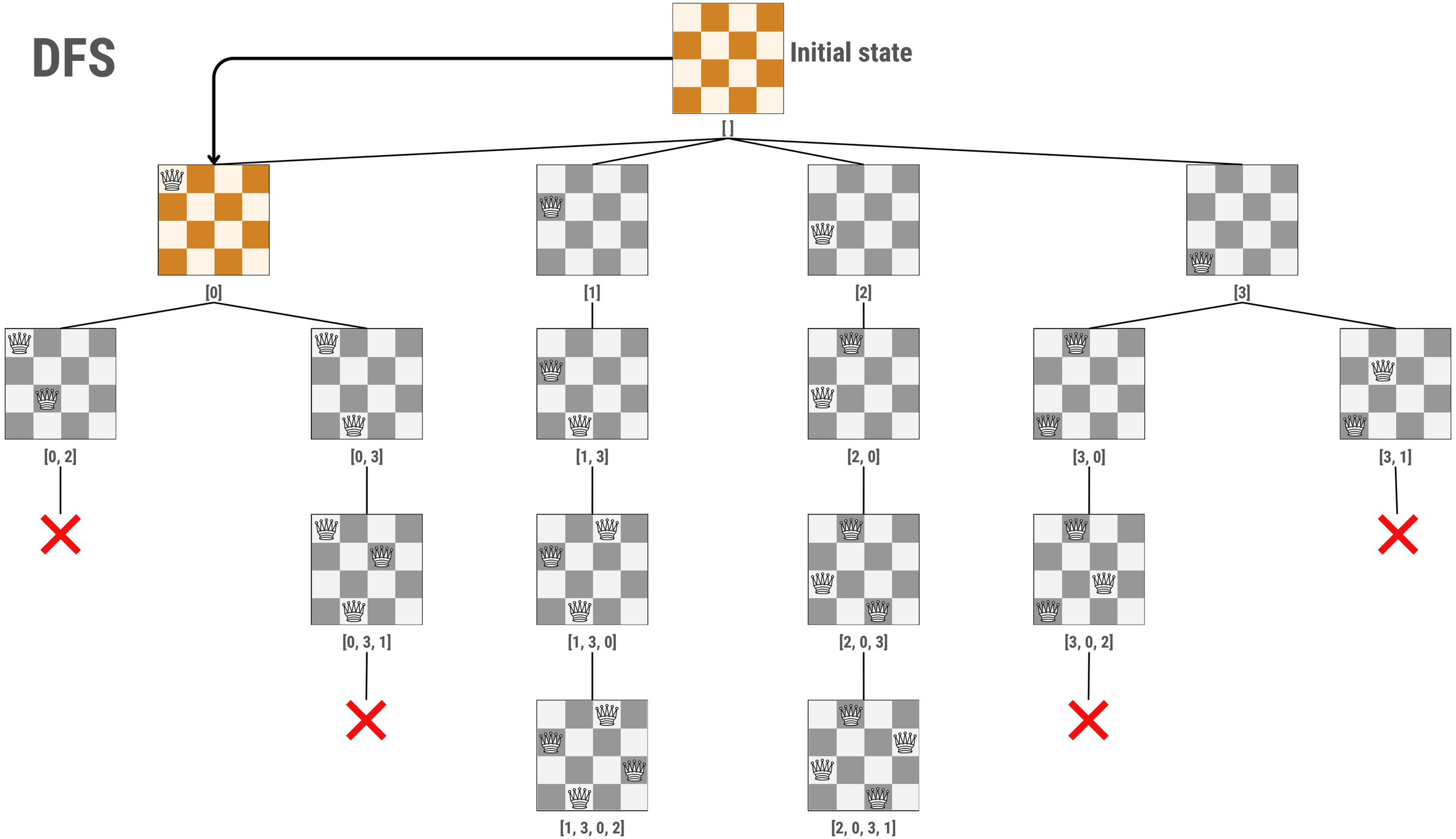
BrFS


Initial state
[ ]
[0]
[1]
[2]
[3]
[0, 2]
[0, 3]
[1, 3]
[2, 0]
[3, 0]
[3, 1]
[0, 3, 1]
[1, 3, 0]
[2, 0, 3]
[3, 0, 2]
[1, 3, 0, 2]
[2, 0, 3, 1]
Goal state!

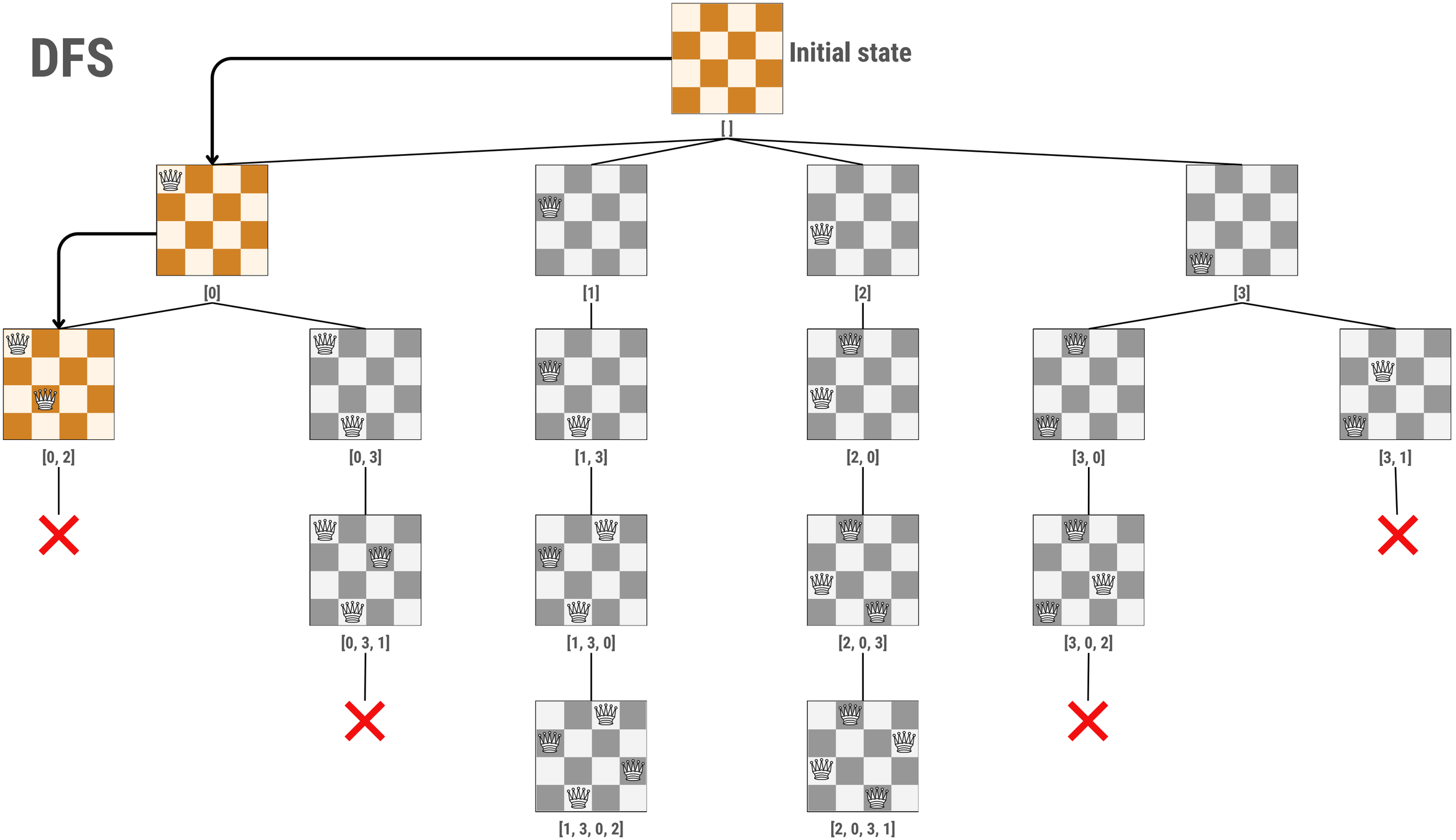
DFS



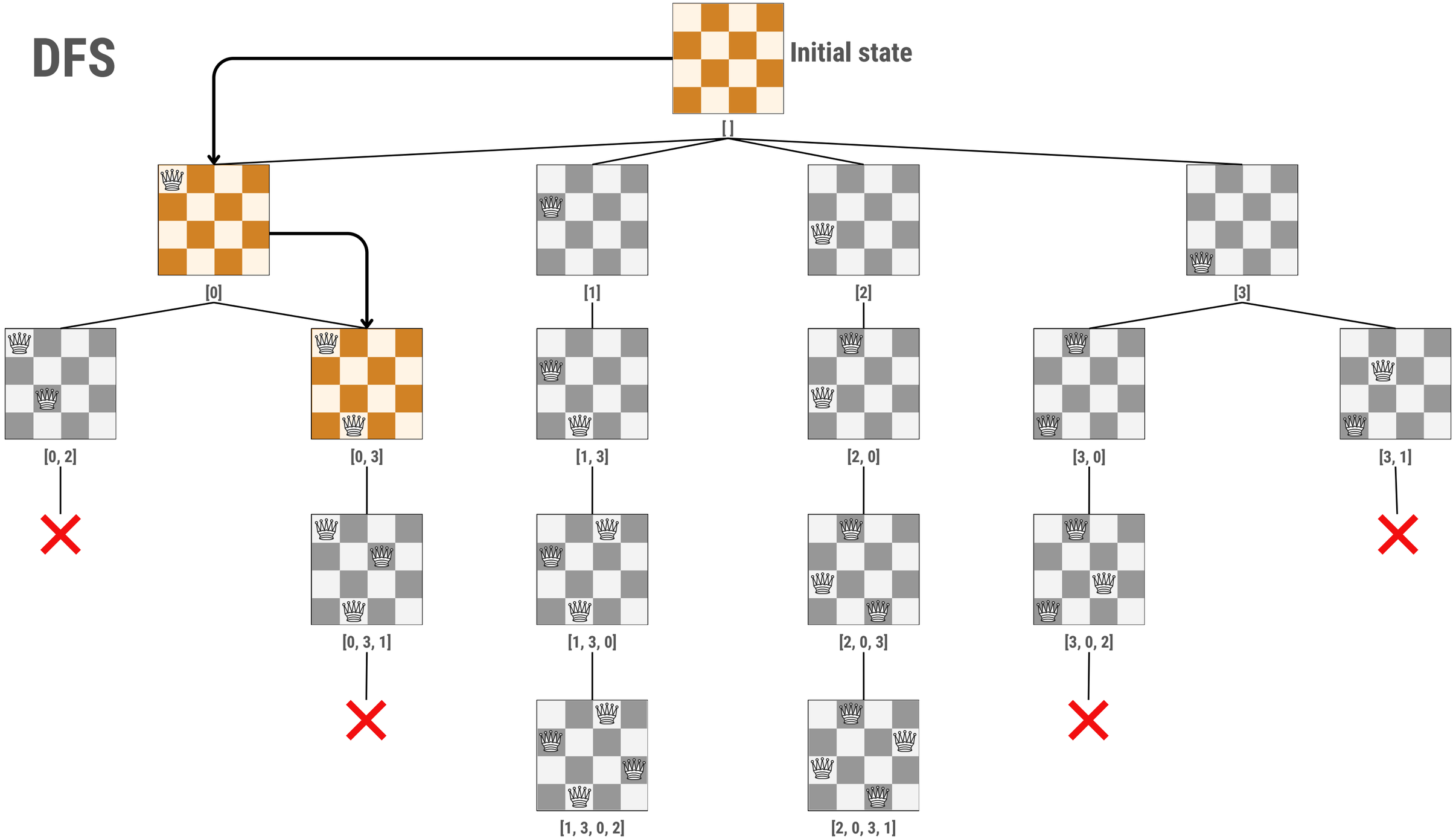
DFS



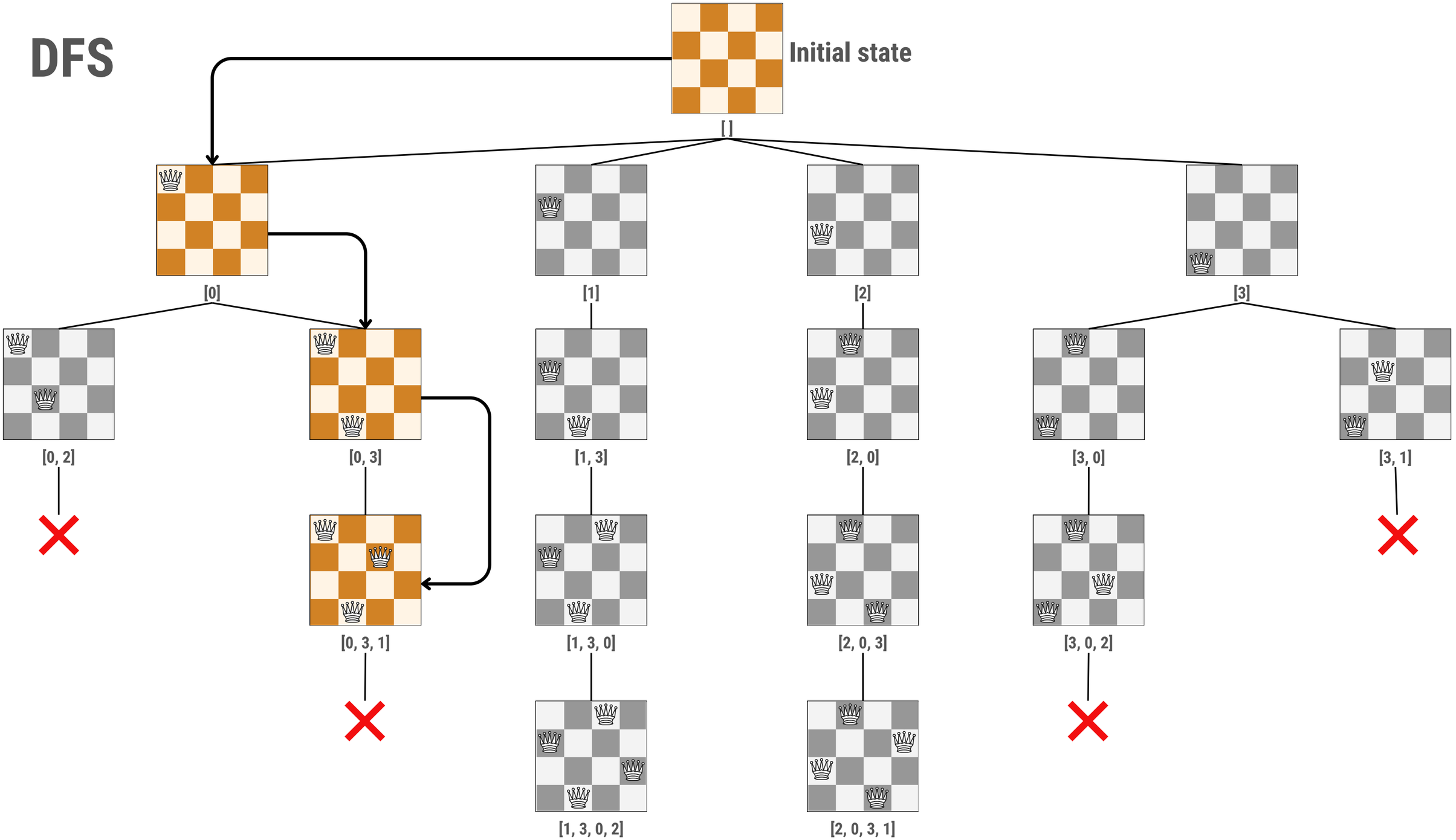
DFS



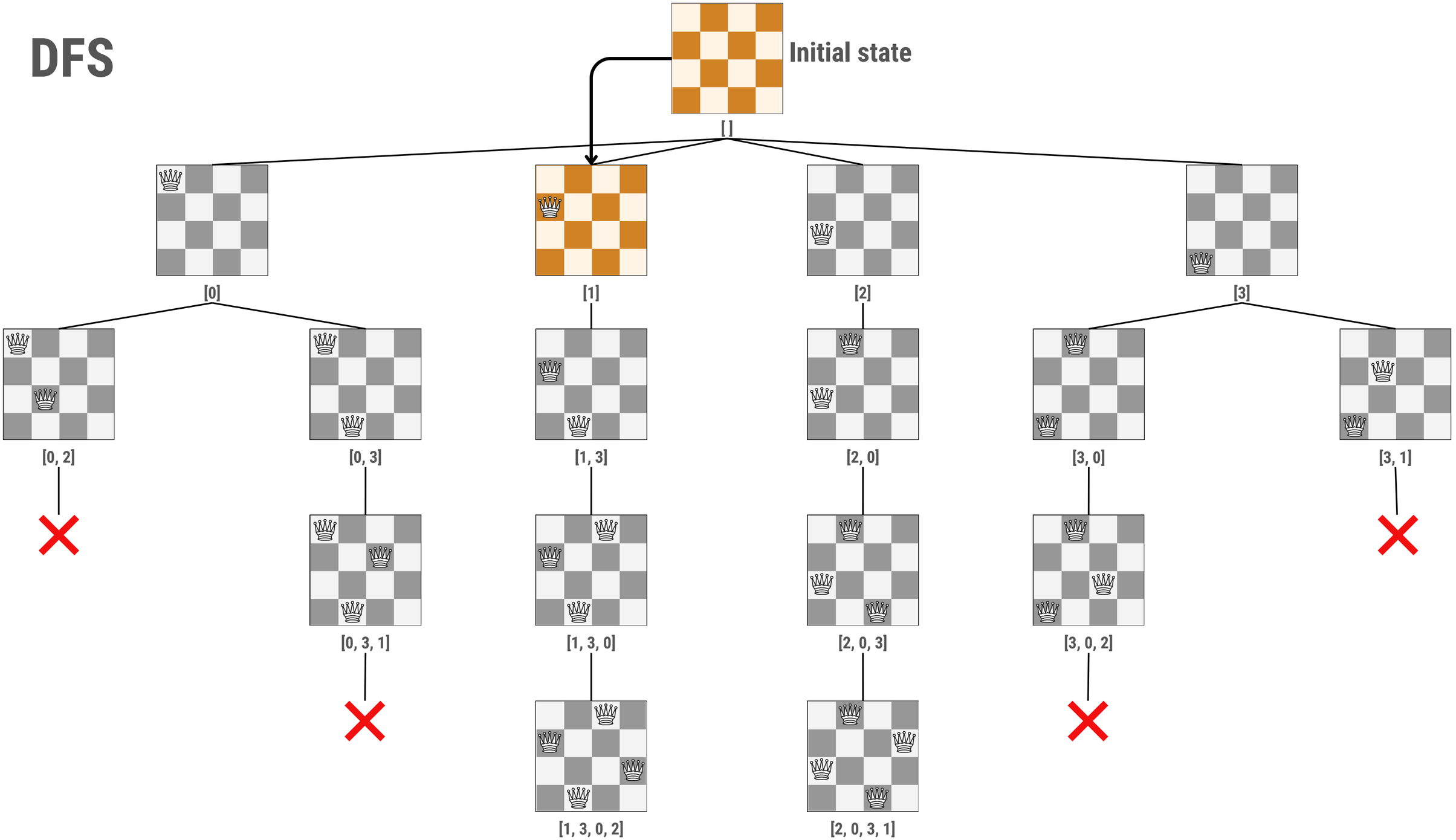
DFS



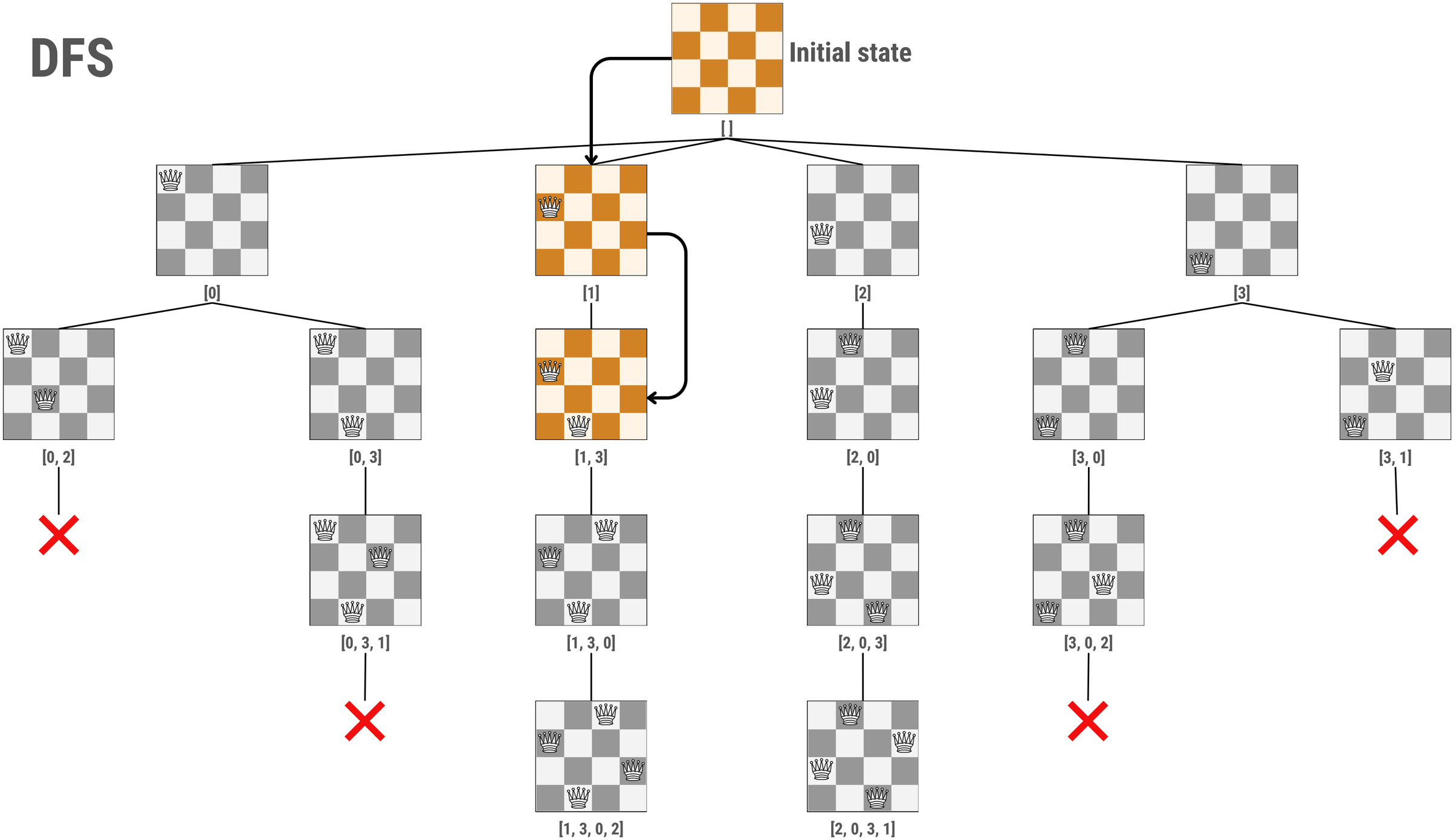
DFS



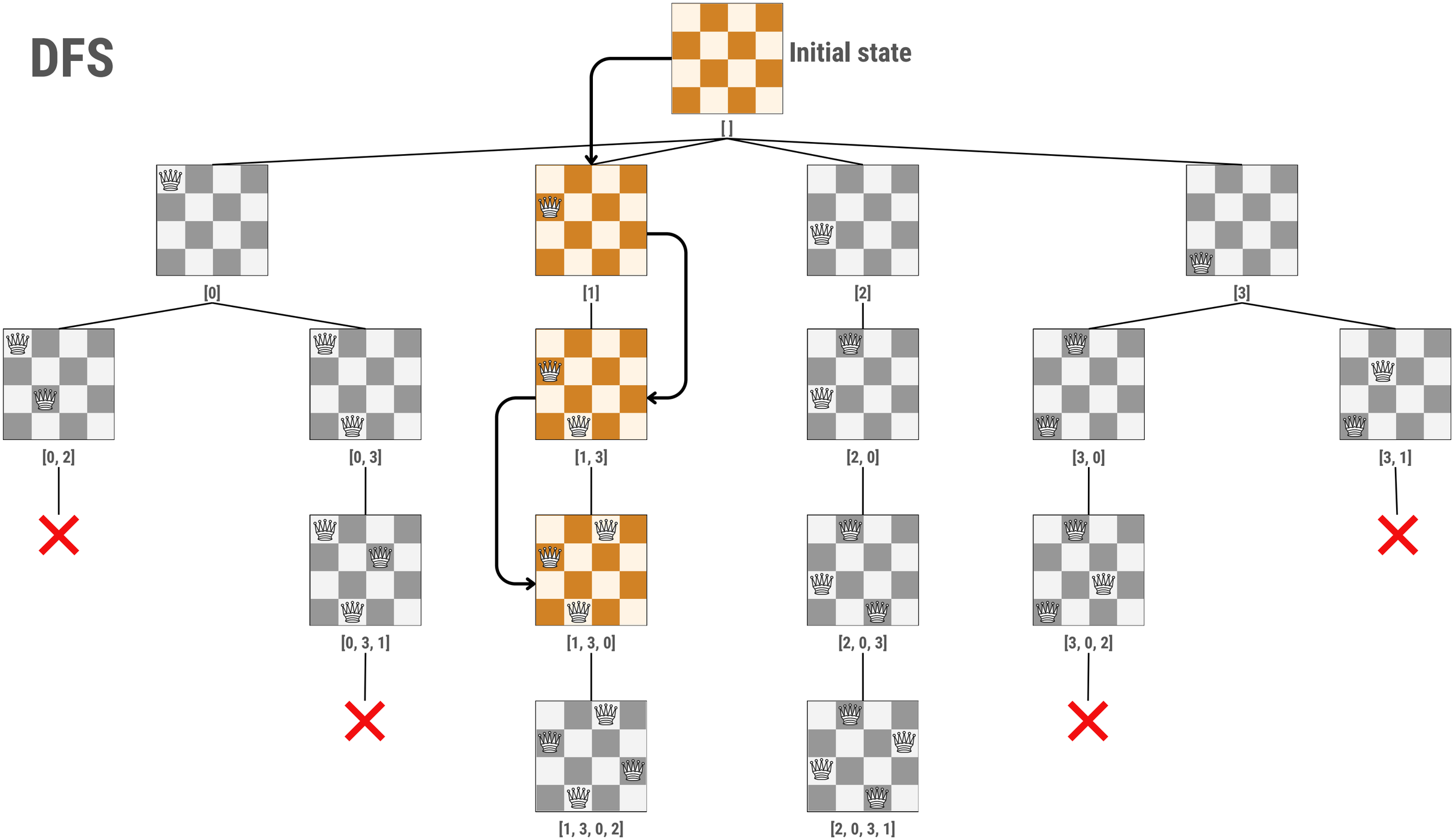
DFS



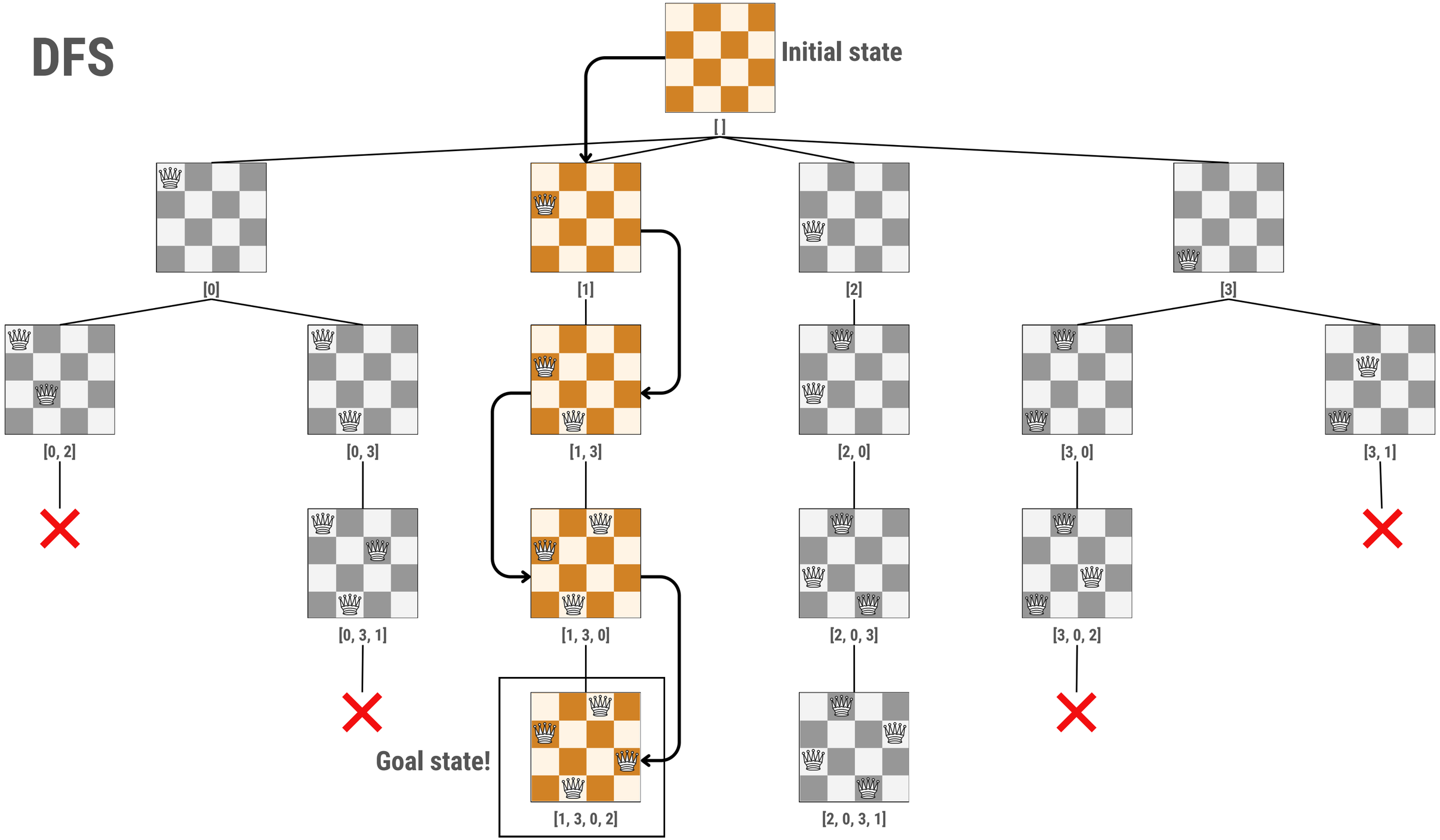
DFS



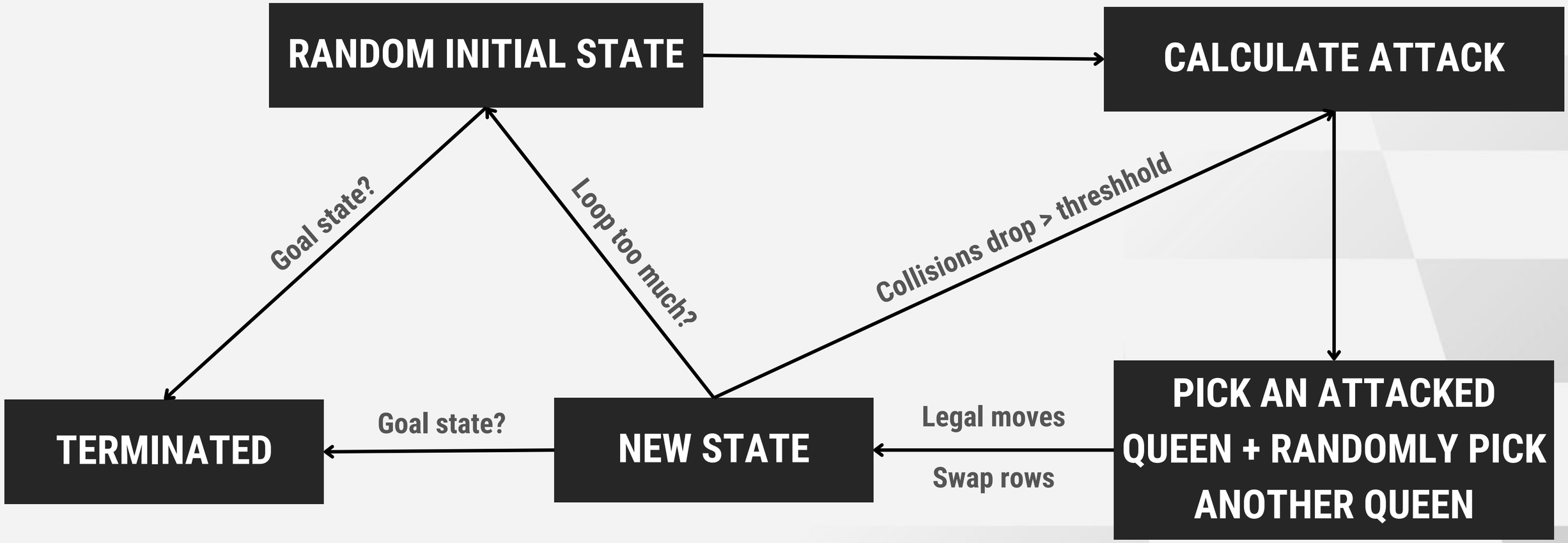
DFS



DFS


Initial state
[ ]
[0]
[1]
[2]
[3]
[0, 2]
[0, 3]
[1, 3]
[2, 0]
[3, 0]
[3, 1]
[0, 3, 1]
[1, 3, 0]
[2, 0, 3]
[3, 0, 2]
Goal state!
[1, 3, 0, 2]
[2, 0, 3, 1]

IMPLEMENTATION: QS2



IMPLEMENTATION: QS2

```
procedure queen_search2 (var queen : array [1 .. N] of integer);
var
    dn : array [2 .. 2 * N] of integer;
    dp : array [1 - N .. N - 1] of integer;
    attack : array [1 .. N] of integer;
    limit,collisions,number_of_attacks,loopcount : integer;
    i,j,k : integer;
```

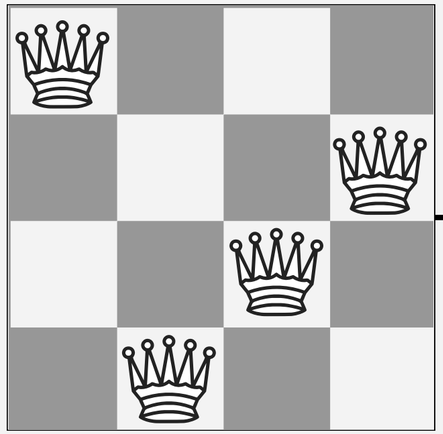
IMPLEMENTATION: QS2

```
begin
    repeat
                                { initialization }
        Generate a random permutation of queen[1] to queen[N];
        collisions := compute_collisions(queen,dn,dp);
        limit := C1 * collisions;
        number_of_attacks := compute_attacks(queen,dn,dp,attack);
        loopcount := 0;
                                { search }
        repeat
            for k := 1 to number_of_attacks do begin
                i := attack[k];
                Choose j in [1..N];
                if swap_ok(i,j,queen,dn,dp) then begin
                    perform_swap(i,j,queen,dn,dp,collisions);
                    if collisions = 0 then return;
                    if collisions < limit then begin
                        limit := C1 * collisions;
                        number_of_attacks := compute_attacks(queen,dn,dp,attack);
                    end;
                end;
            end;
            loopcount := loopcount + number_of_attacks;
        until loopcount > C2 * N;

    until collisions = 0;
end;
```

IMPLEMENTATION: QS2

Initial state



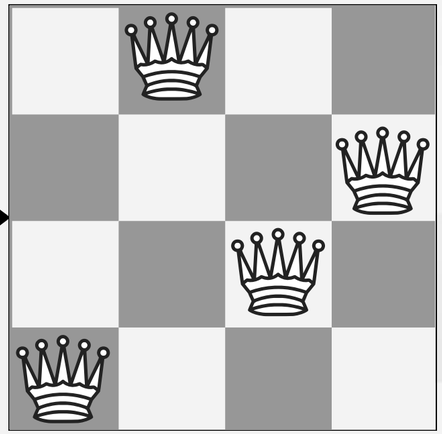
[0, 3, 2, 1]

Collisions = 3

Attacked queen = [0, 1, 2, 3]

Choose attacked queen in column 0

Another random queen is in column 1

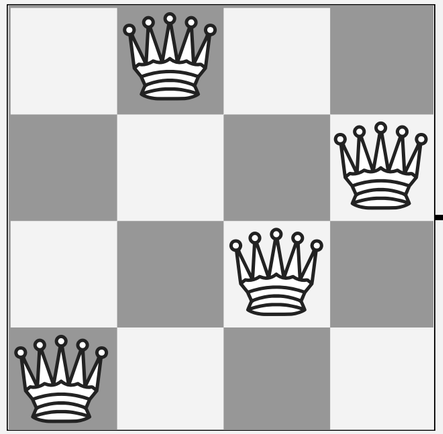


[3, 0, 2, 1]

Collisions = 1

Attacked queen = [2, 3]

IMPLEMENTATION: QS2



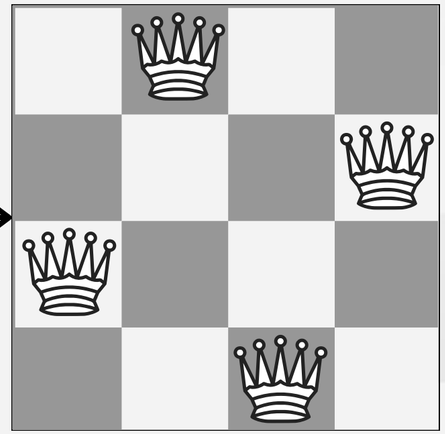
[3, 0, 2, 1]

Collisions = 1

Attacked queen = [2, 3]

Choose attacked queen in column 2

Another random queen is in column 0

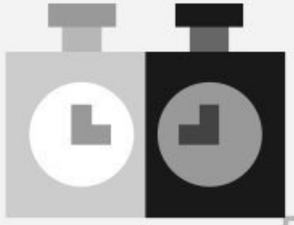


[2, 0, 3, 1]

Collisions = 0

Attacked queen = []

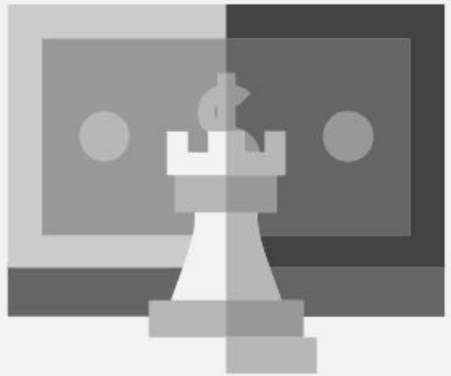
Goal state!

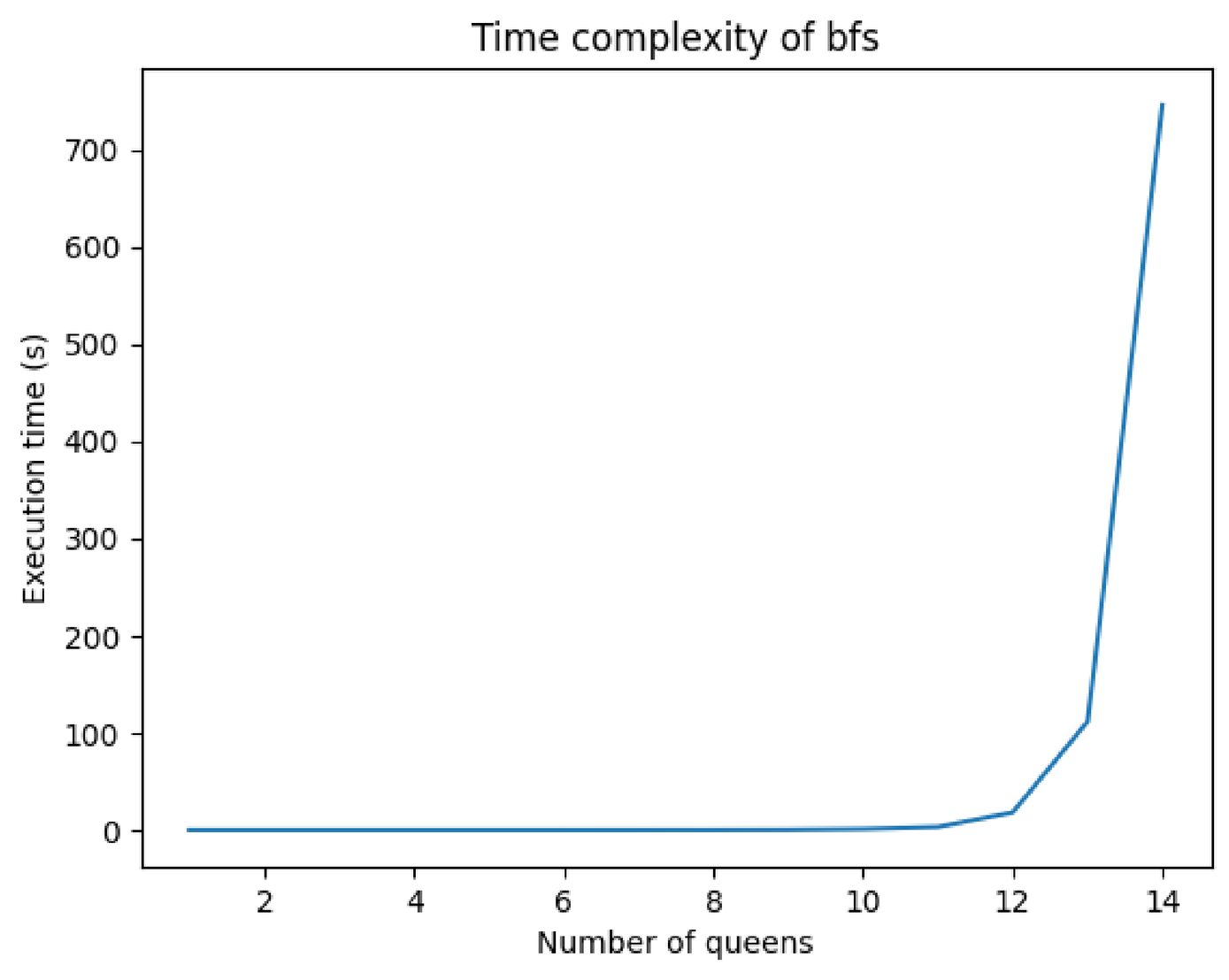


04

COMPARISON

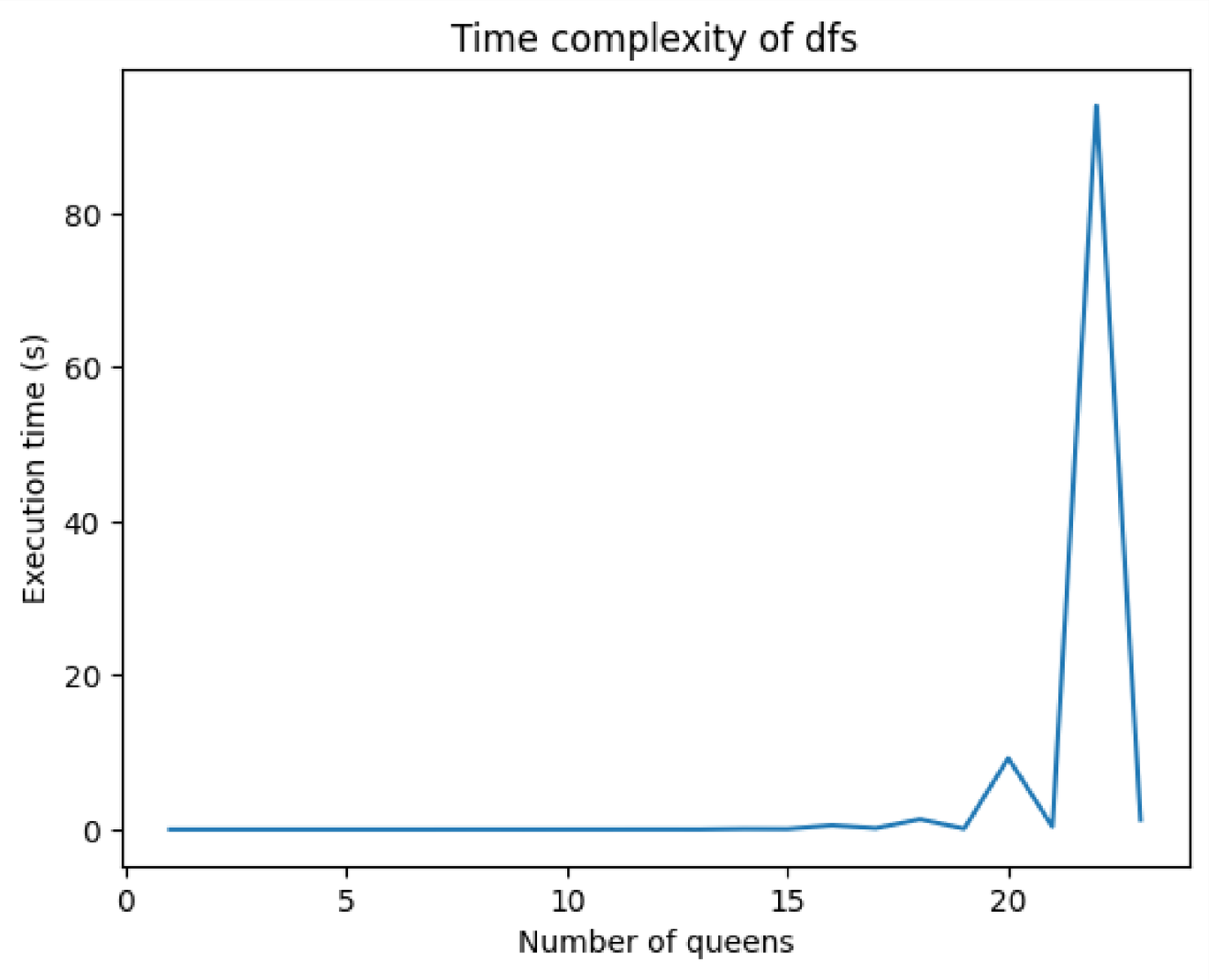
COMPARISON: TIME COMPLEXITY - BFS





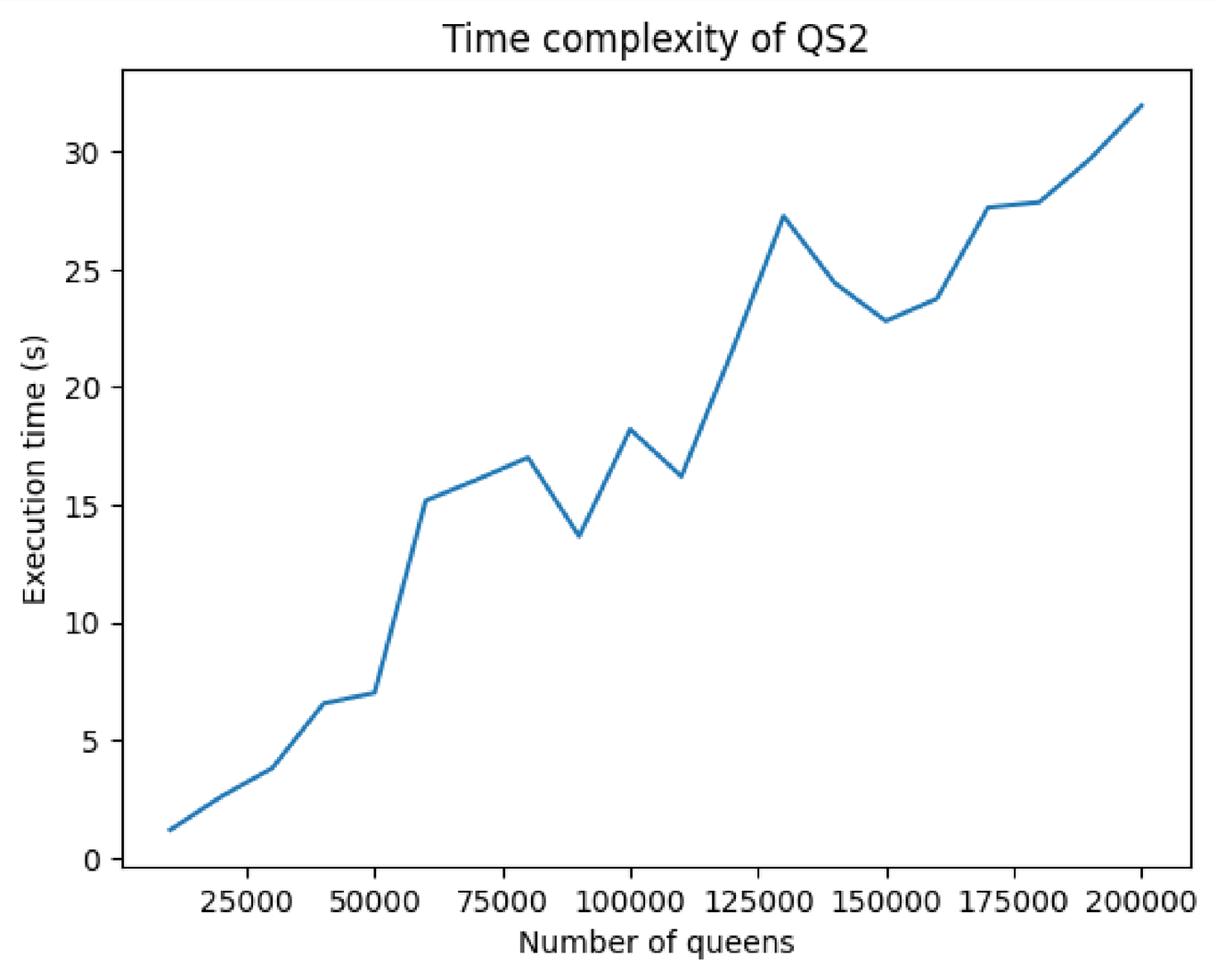
In the BFS algorithm, we need to traverse all the first n-1 levels before reaching the nth level to determine which state is the goal state. Therefore, the complexity of this algorithm is O(n^n).

COMPARISON: TIME COMPLEXITY - DFS



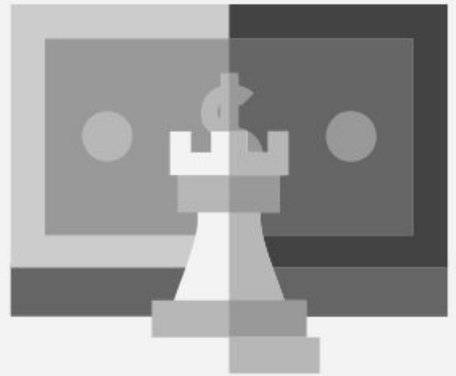
Similarly to BFS, in the worst-case scenario of the DFS algorithm, we still have to traverse all the first n-1 levels before considering the nth level. However, since there are multiple goal states, the probability of encountering a goal state in DFS is higher than in BFS, which often means that we don't need to traverse all the way to the n-1 levels. In general, the DFS algorithm tends to be a bit faster than BFS.

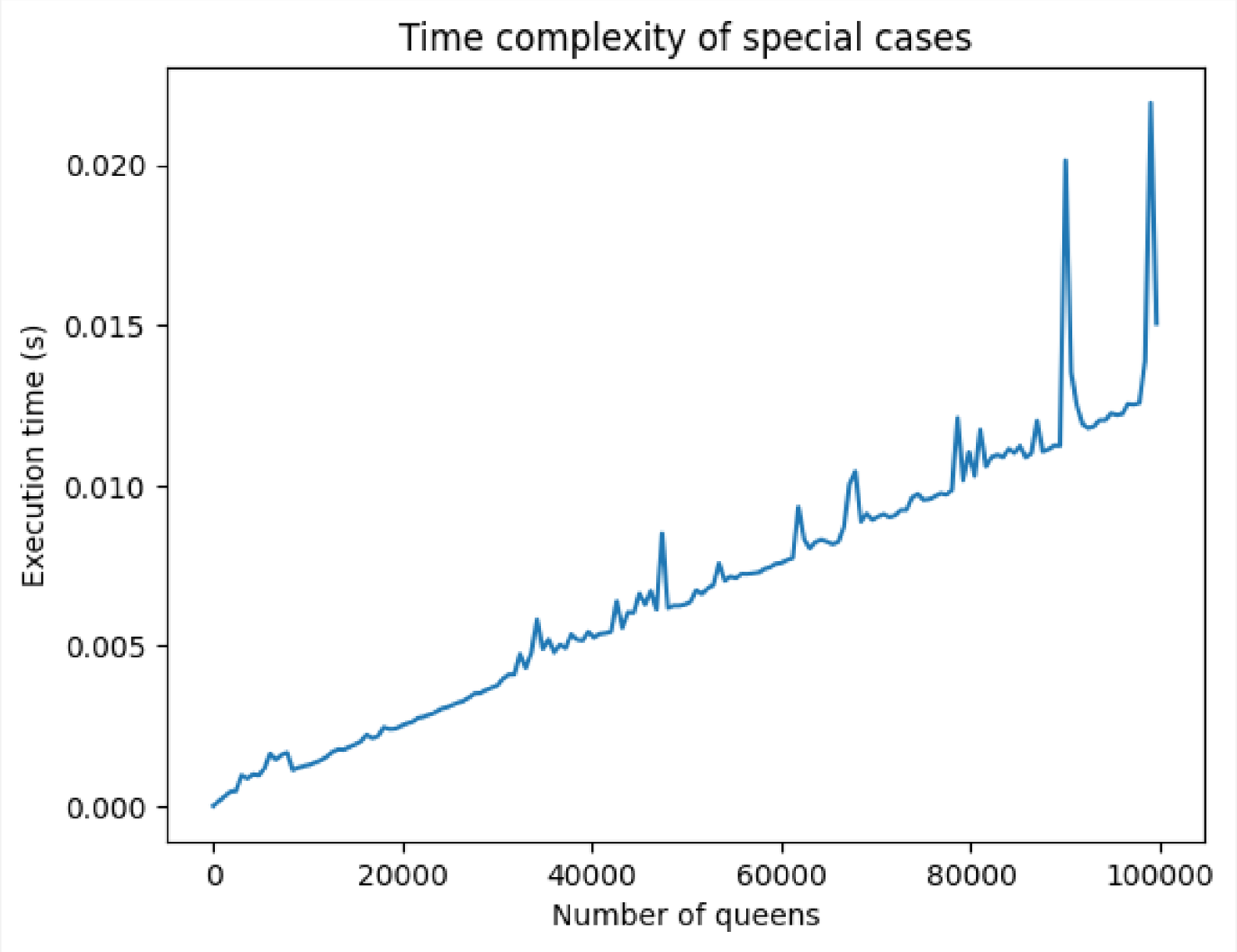
COMPARISON: TIME COMPLEXITY - QS2



In QS2, there can be multiple solutions for each board size, so the probability of randomly reaching a state that can lead to a goal state by swapping queens to reduce collisions is high. Here, the complexity is a polynomial function of the board size. In the step of moving from a random state towards a goal state, the complexity of swapping queens is also a polynomial function. Therefore, in general, the complexity of the QS2 algorithm is a polynomial function.

COMPARISON: TIME COMPLEXITY - SPECIAL CASES





For numbers that leave a remainder of 0, 1, 4, or 5 when divided by 6, we have a quick solution for this problem. The answer is in the form of [1, 3, ..., 2k+1, 0, 2, ..., 2k] with an odd length. Otherwise, the answer takes the form [1, 3, ..., 2k+1, 0, 2, ..., 2k+2] with an even length.

THANKS!

Do you have any questions?