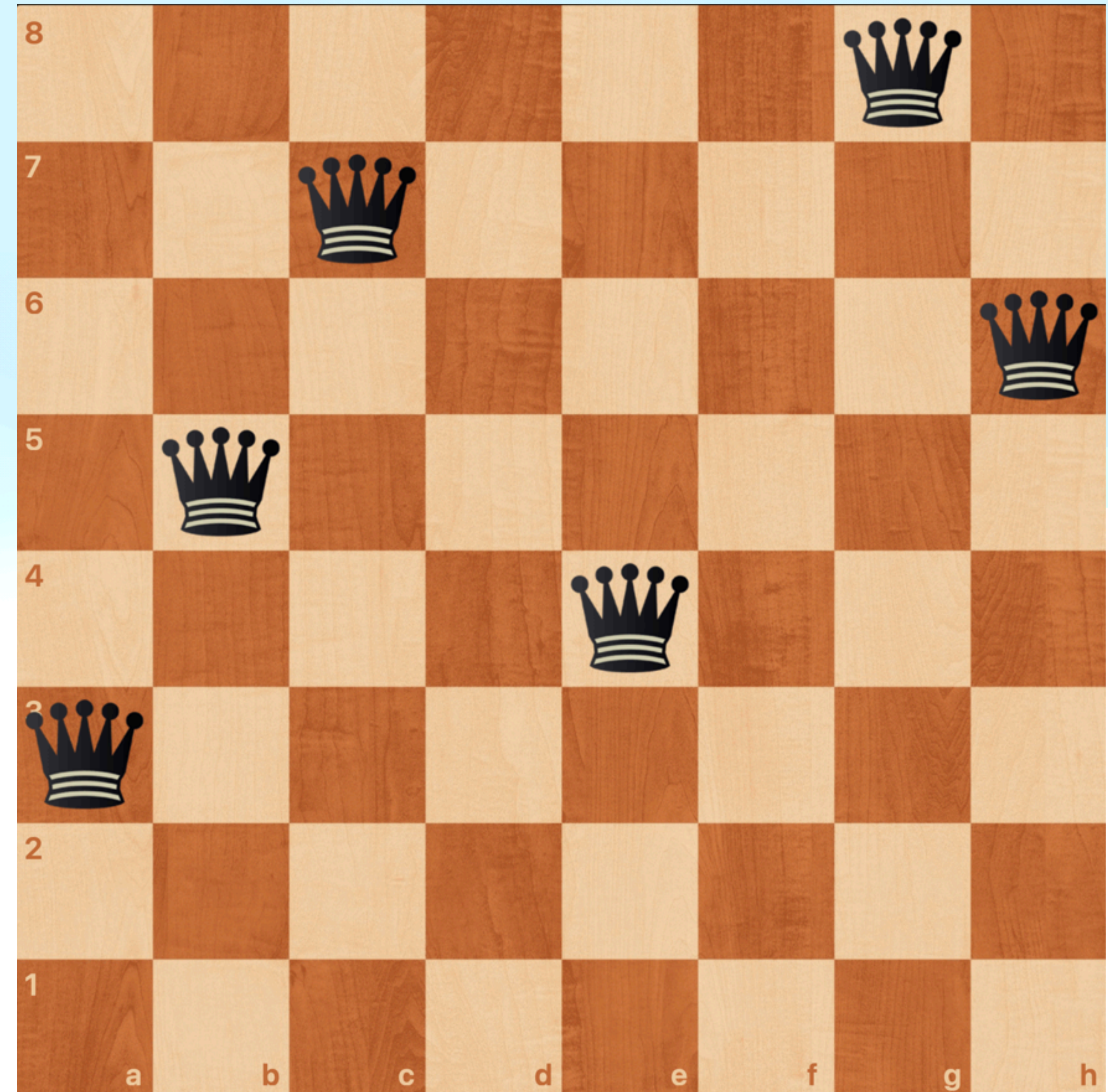


# Search algorithm

## Formulation of a search problem

- E.g. Eight Queen's Problem
- States
- Initial state
- Actions
- Successor function
- Goal test
- Path cost
- Solution





# Search algorithm

## Tree search algorithm

- States: physical config.
- Nodes: Relationship
  - Parent, Children, cost, etc.
- Expand function
- General tree search algorithm
  - Remove node -> Goal test -> Expand and insert

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE(node)) then return node
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

```
function EXPAND(node, problem) returns a set of nodes
  successors ← the empty set
  for each action, result in SUCCESSOR-FN(problem, STATE[node]) do
    s ← a new NODE
    PARENT-NODE[s] ← node; ACTION[s] ← action; STATE[s] ← result
    PATH-COST[s] ← PATH-COST[node] + STEP-COST(STATE[node], action, result)
    DEPTH[s] ← DEPTH[node] + 1
    add s to successors
  return successors
```