Design and Analysis of Algorithm

Lecture-21:
Backtracking

Contents

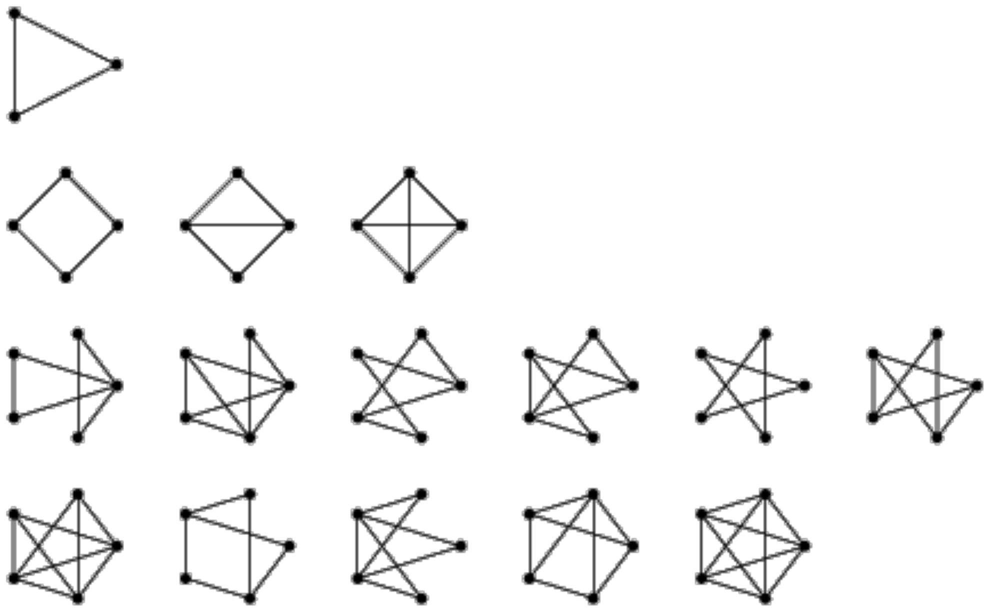
.

Connected Graph

Two vertices u and v are *connected* in an undirected graph iff there is a path from u to v (and v to u).

An undirected graph is *connected* iff for every pair of distinct vertices u and v in V(G) there is a path from u to v in G.

A *connected component* of an undirected is a maximal connected subgraph. A tree is a *connected acyclic* graph.



Connected Component

Algorithm 9.4 BFS

Input: A directed or undirected graph $G = (V, E)$.

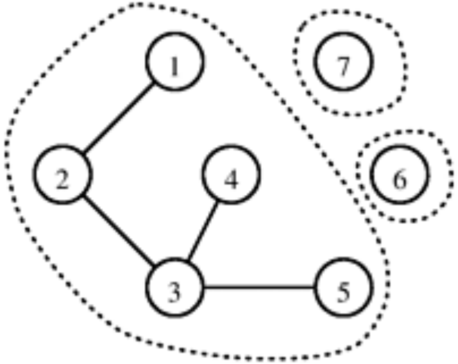
Output: Numbering of the vertices in breadth-first search order.

```
bfn ← 0
for each vertex v ∈ V
    mark v unvisited
end for
for each vertex v ∈ V
    if v is marked unvisited then bfs(v)
end for
```

Procedure *bfs*(v)

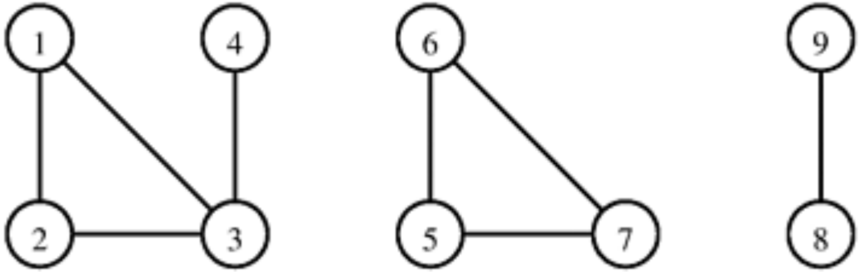
```
Q ← {v}
mark v visited
while Q ≠ {}
    v ← Pop(Q)
    bfn ← bfn + 1
    for each edge (v, w) ∈ E
        if w is marked unvisited then
            Push(w, Q)
            mark w visited
        end if
    end for
end while
```

If G is a connected undirected graph, then all vertices of G will get visited on the first call to BFS



Reachability Problem

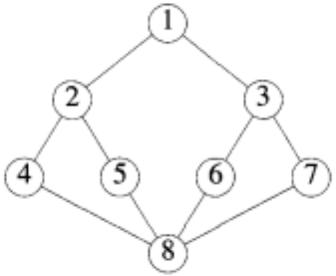
The connected components of an undirected graph are the equivalence classes of vertices under the "is reachable from" relation.

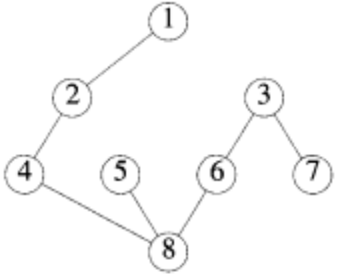


A graph with three connected components:

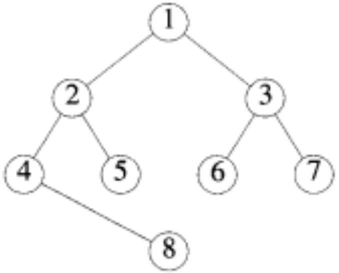
{1, 2, 3, 4}, {5, 6, 7}, and {8, 9}.

Spanning Tree





(a) DFS(1) spanning tree

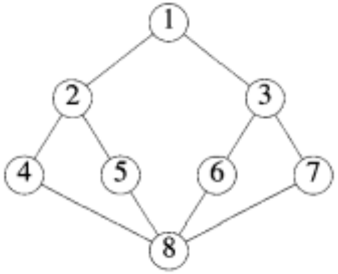


(b) BFS(1) spanning tree

Question

Let G be an undirected graph. Consider a depth-first traversal of G, and let T be the resulting depth-first search tree. Let u be a vertex in G and let v be the first new (unvisited) vertex visited after visiting u in the traversal. Which of the following statements is always true?

1. $\{u, v\}$ must be an edge in G, and u is a descendant of v in T
2. $\{u, v\}$ must be an edge in G, and v is a descendant of u in T
3. If $\{u, v\}$ is not an edge in G then u and v must have the same parent in T
4. If $\{u, v\}$ is not an edge in G then u is a leaf in T



Question

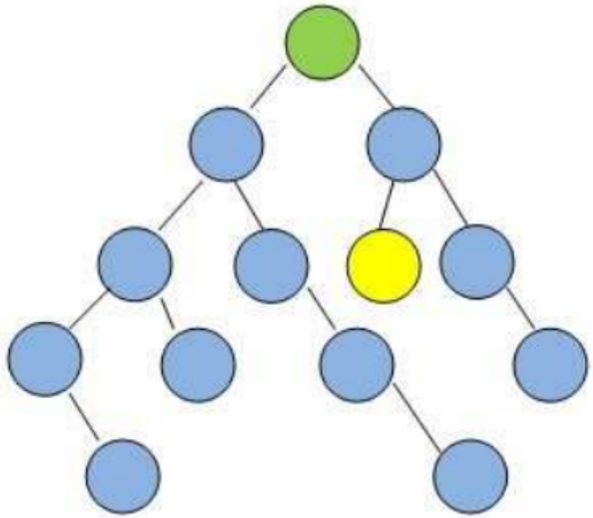
Given two vertices in a graph s and t, which of the two traversals (BFS and DFS) can be used to find if there is path from s to t?

1. Only BFS
2. Only DFS
3. Both BFS and DFS
4. Neither BFS nor DFS

Question

In the following graphs, assume that if there is ever a choice amongst multiple nodes, both the BFS and DFS algorithms will choose the left-most node first.

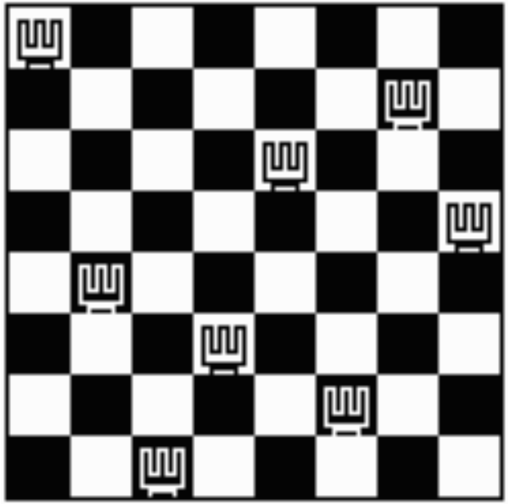
Starting from the green node at the top, which algorithm will visit the least number of nodes before visiting the yellow goal node?



1. BFS
2. DFS
3. Neither BFS nor DFS will ever encounter the goal node in this graph.
4. BFS and DFS encounter same number of nodes before encounter the goal node

Problem Definition

- Find an arrangement of **8** queens on a single chess board such that no two queens are attacking one another.
- In chess, queens can move all the way down any row, column or diagonal (so long as no pieces are in the way).



Generalization

Due to the the two restrictions, it's clear that each row and column of the board will have exactly one queen.

Since we are talking about the 8×8 chessboard to place 8 queens. The problem can be generalized as $n - queens\ problem$ of placing n queens on $n \times n$ chessboard

The solution exist for all natural numbers n with the exception of 2 and 3

The problem was originally proposed in 1848 by the German chess player *max bezel.*

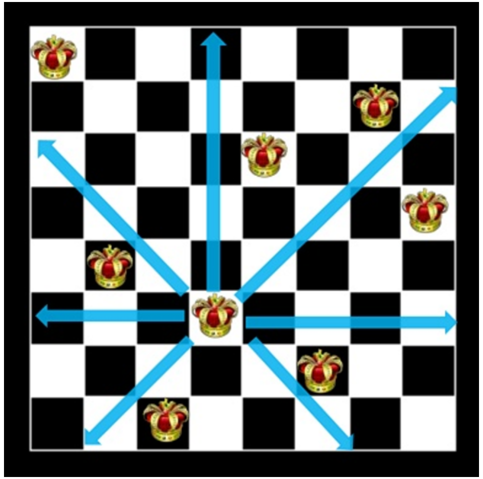
Terminology

States: Any arrangement of 0 to 8 queens on the board

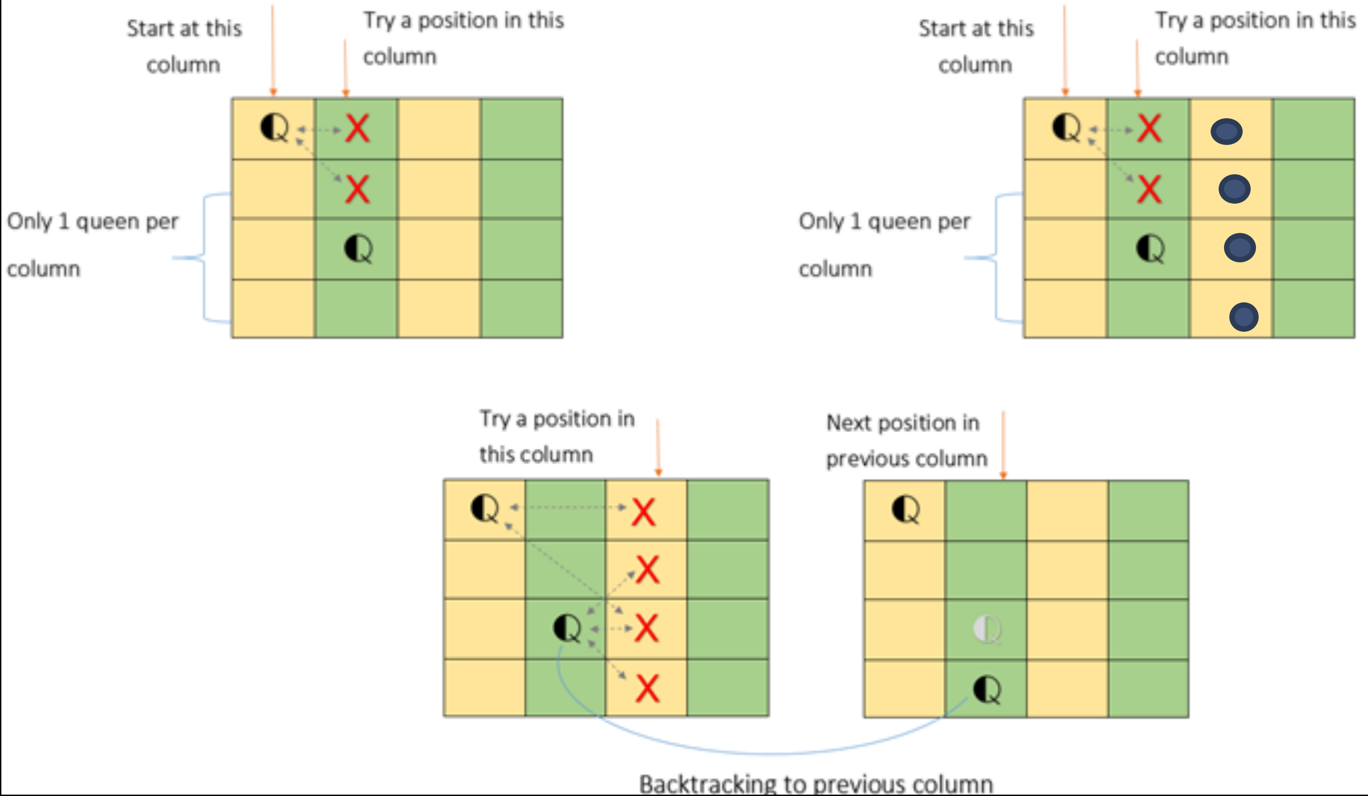
Initial State: 0 queens on the board

Successor function: Add a queen in any square

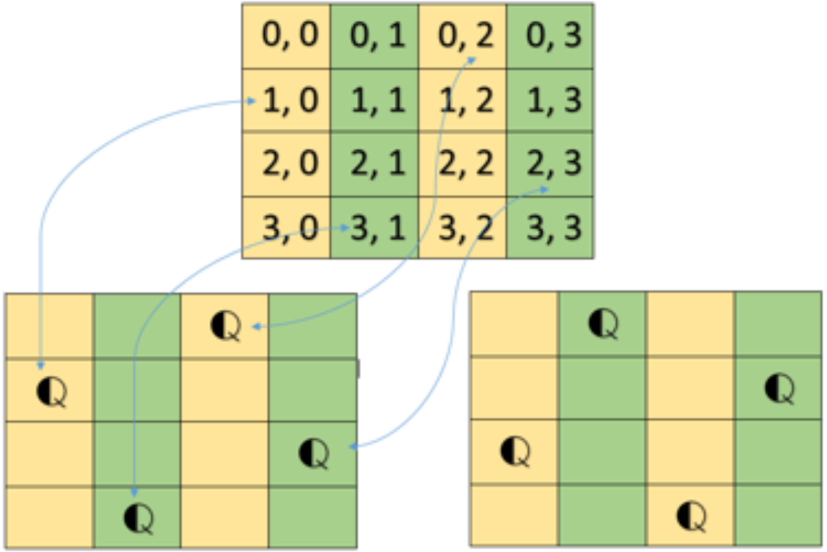
Goal test: 8 queens on the board, none attacked.



4 Queen's Problem

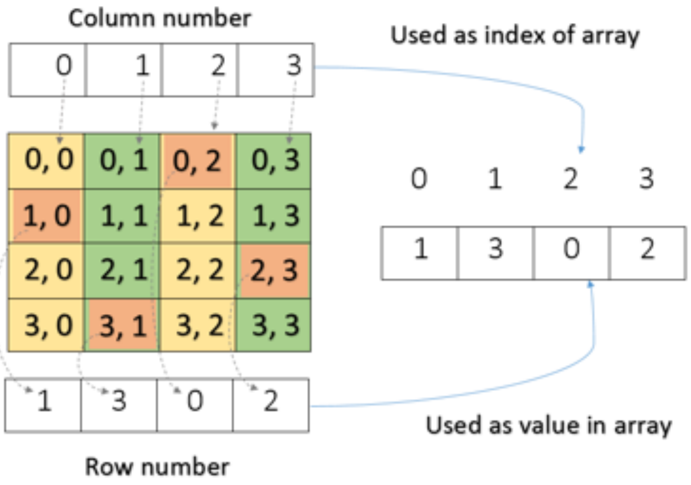


Solution of 4 Queen’s Problem



Solution one

Solution two



Diagonal Test

How do we test if two queens are on the same diagonal?

Consider the squares of the chessboard being numbered as the indices of the two dimensional array $A(1\colon n, 1\colon n)$

for every element on the same diagonal which runs from the upper left to the lower right, each element has the same $row - column$ value.

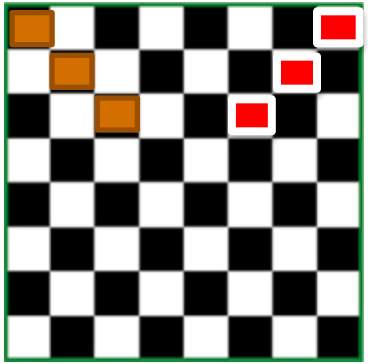
Also, every element on the same diagonal which goes from the upper right to the lower left has the same $row + column$ value.

Suppose two queens are placed at positions $(i,\ j)$ and (k, l).Then

$$i - j = k - l \text{ or } \quad i + j = k + l$$

$$i - k = j - l \text{ or } \quad i - k = l - j$$

$$|i - k| = |j - l|$$



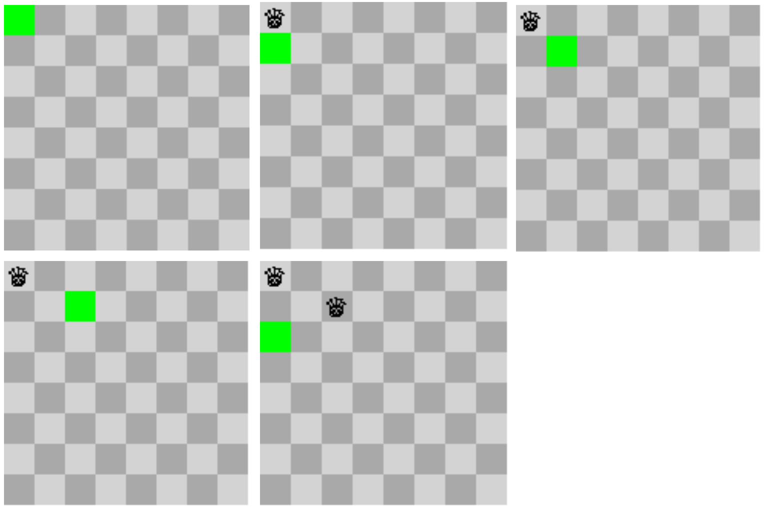
Algorithm

Worst case Time complexity: $O(n!)$

```
procedure NQUEENS(n)
  //using backtracking this procedure prints all possible placements of//
  //n queens on an n × n chessboard so that they are nonattacking//
  integer k, n, X(1:n)
  X(1) ← 0; k ← 1   //k is the current row; X(k) the current column//
  while k > 0 do   //for all rows do//
    X(k) ← X(k) + 1   //move to the next column//
    while X(k) ≤ n and not PLACE(k) do   //can this queen be placed?//
      X(k) ← X(k) + 1
    repeat
    if X(k) ≤ n   //a position is found//
      then if k = n   //is a solution complete?//
          then print(X)   //yes, print the array//
          else k ← k + 1; X(k) ← 0   //go to the next row//
          endif
      else k ← k − 1   //backtrack//
    endif
  repeat
end NQUEENS
```

```
procedure PLACE(k)
  //returns true if a queen can be placed in kth row and//
  //X(k)th column. Otherwise it returns false.//
  //X is a global array whose first k values have been set.//
  //ABS(r) returns the absolute value of r//
  global X(1: k); integer i, k
  for i ←1 to k  do
    if X(i) = X(k)   //two in the same column//
      or ABS(X(i) − X(k)) = ABS(i − k)   //in the same diagonal//
        then return(false)
    endif
  repeat
  return(true)
end PLACE
```

Demo



```
procedure PLACE(k)
  //returns true if a queen can be placed in kth row and//
  //X(k)th column. Otherwise it returns false.//
  //X is a global array whose first k values have been set.//
  //ABS(r) returns the absolute value of r//
  global X(1: k); integer i, k
  for i ← 1 to k do
    if X(i) = X(k)   //two in the same column//
      or ABS(X(i) − X(k)) = ABS(i − k)   //in the same diagonal//
        then return(false)
    endif
  repeat
  return(true)
end PLACE
```