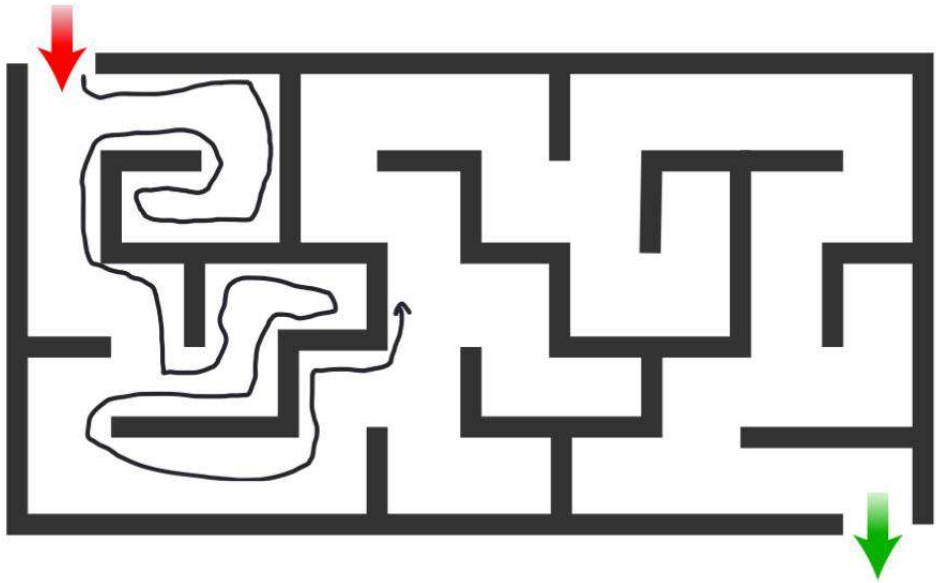
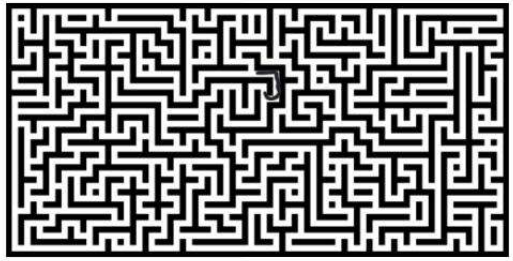
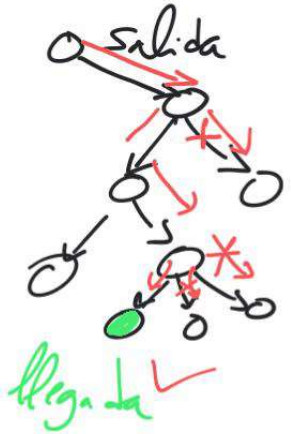
Backtracking

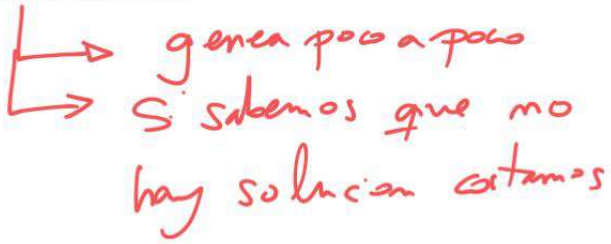




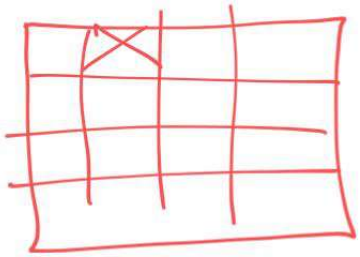

Salida
llegada

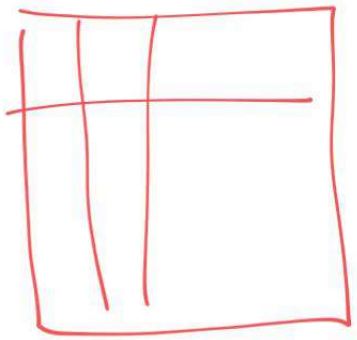
Algorítmicamente, ¿Cómo solucionarlo?

Backtracking:
Utilizar "fuerza bruta" para resolver un problema.
Se mira en todo el espacio de soluciones




5 3 1 7
6 12 4 1 9 5
7 9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9





todas las casillas 1 vez

caballo ajedrez

```cpp
// Código original: https://raw.githubusercontent.com/shoaibrayeen/Data-Structures-and-Algorithms/master/Sudoku/code_1.cpp
// Modificado (originalmente utilizaba arrays)
//
//  main.cpp
//  Algorithm
//
//  Created by Mohd Shoaib Rayeen on 31/07/18.
//  Copyright © 2018 Shoaib Rayeen. All rights reserved.
//
// A Backtracking program  in C++ to solve Sudoku problem
#include <iostream>
#include <vector>
#define UNASSIGNED 0
#define N 9
using namespace std;

bool FindUnassignedLocation(const vector <vector <int>> & grid , int& row, int& col);
bool isSafe(const vector <vector <int>> & grid, int row, int col, int num);

bool SolveSudoku(vector <vector  <int>>& grid) {
    int row, col;
    if (!FindUnassignedLocation(grid, row, col)) {
        return true; // success!
    }
    for (int num = 1; num <= 9; num++) {
        if (isSafe(grid, row, col, num)) {
            grid[row][col] = num;
            if (SolveSudoku(grid)) {
                return true;
            }
            grid[row][col] = UNASSIGNED;
        }
    }
    return false;
}

bool FindUnassignedLocation(const vector <vector <int>>& grid, int& row, int& col) {
    for (row = 0; row < N; row++) {
        for (col = 0; col < N; col++) {
            if (grid[row][col] == UNASSIGNED) {
                return true;
            }
        }
    }
    return false;
}

bool UsedInRow(const vector <vector <int>>& grid, int row, int num) {
    for (int col = 0; col < N; col++) {
```

```
50          if (grid[row][col] == num) {
51              return true;
52          }
53      }
54      return false;
55  }
56
57  bool UsedInCol(const vector <vector <int>>& grid, int col, int num) {
58      for (int row = 0; row < N; row++) {
59          if (grid[row][col] == num) {
60              return true;
61          }
62      }
63      return false;
64  }
65
66  bool UsedInBox(const vector <vector <int>>& grid, int boxStartRow, int
      boxStartCol, int num) {
67      for (int row = 0; row < 3; row++) {
68          for (int col = 0; col < 3; col++) {
69              if (grid[row + boxStartRow][col + boxStartCol] == num) {
70                  return true;
71              }
72          }
73      }
74      return false;
75  }
76
77  bool isSafe(const vector <vector <int>>& grid, int row, int col, int
      num) {
78      return !UsedInRow(grid, row, num) &&
79          !UsedInCol(grid, col, num) &&
80          !UsedInBox(grid, row - row % 3, col - col % 3, num);
81  }
82
83  void printGrid(const vector <vector <int>>& grid) {
84      for (int row = 0; row < N; row++) {
85          for (int col = 0; col < N; col++) {
86              printf("%2d", grid[row][col]);
87          }
88          printf("\n");
89      }
90  }
91
92  int main() {
93      vector <vector <int>> grid = { {3, 0, 6, 5, 0, 8, 4, 0, 0},
94                          {5, 2, 0, 0, 0, 0, 0, 0, 0},
95                          {0, 8, 7, 0, 0, 0, 0, 3, 1},
96                          {0, 0, 3, 0, 1, 0, 0, 8, 0},
97                          {9, 0, 0, 8, 6, 3, 0, 0, 5},
98                          {0, 5, 0, 0, 9, 0, 6, 0, 0},
99                          {1, 3, 0, 0, 0, 0, 2, 5, 0},
100                         {0, 0, 0, 0, 0, 0, 0, 7, 4},
```

```
101                         {0, 0, 5, 2, 0, 6, 3, 0, 0} };
102     if (SolveSudoku(grid)) {
103         printGrid(grid);
104     }
105     else {
106         printf("No solution exists");
107     }
108     return 0;
109 }
110
```

```cpp
// Código original:https://raw.githubusercontent.com/shoaibrayeen/Data-Structures-and-Algorithms/master/The%20Knight%E2%80%99s%20Tour%20Problem/code_1.cpp

//
//  code_1.cpp
//  Algorithm
//
//  Created by Mohd Shoaib Rayeen on 23/11/18.
//  Copyright © 2018 Shoaib Rayeen. All rights reserved.
//


#include <iostream>
using namespace std;
#define N 8

int solveKTUtil(int x, int y, int movei, int sol[N][N], int xMove[], int yMove[]);


bool isSafe(int x, int y, int sol[N][N]) {
    return (x >= 0 && x < N&& y >= 0 && y < N&& sol[x][y] == -1);
}


void printSolution(int sol[N][N]) {
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            cout << sol[x][y] << " ";
        }
        cout << endl;
    }
}


void solveKT() {
    int sol[N][N];
    for (int x = 0; x < N; x++) {
        for (int y = 0; y < N; y++) {
            sol[x][y] = -1;
        }
    }
    int xMove[8] = { 2, 1, -1, -2, -2, -1,  1,  2 };
    int yMove[8] = { 1, 2,  2,  1, -1, -2, -2, -1 };
    sol[0][0] = 0;
    if (solveKTUtil(0, 0, 1, sol, xMove, yMove) == false) {
        cout << "\nSolution does not exist\n";
        return;
    }
    else {
        cout << "\nSolution Exists for 8*8 Square\n";
        printSolution(sol);
```

```
51      }
52  }
53
54  int solveKTUtil(int x, int y, int movei, int sol[N][N], int xMove[N],
      int yMove[N]) {
55      int next_x, next_y;
56      if (movei == N * N) {
57          return true;
58      }
59      for (int k = 0; k < 8; k++) {
60          next_x = x + xMove[k];
61          next_y = y + yMove[k];
62          if (isSafe(next_x, next_y, sol)) {
63              sol[next_x][next_y] = movei;
64              if (solveKTUtil(next_x, next_y, movei + 1, sol, xMove,
                  yMove) == true) {
65                  return true;
66              }
67              else {
68                  sol[next_x][next_y] = -1;
69              }
70          }
71      }
72      return false;
73  }
74
75  int main() {
76      solveKT();
77      return 0;
78  }
79
```

```cpp
// Código original: https://raw.githubusercontent.com/shoaibrayeen/Data-Structures-and-Algorithms/master/N%20Queen%20Problem/code_1.cpp
// n-reinas
//  main.cpp
//  Algorithm
//
//  Created by Mohd Shoaib Rayeen on 02/08/18.
//  Copyright © 2018 Shoaib Rayeen. All rights reserved.
//

#include <vector>
#include <iostream>
using namespace std;
const int N=4;

void printSolution(vector <vector <int>> board ) {
    static int k = 1;
    cout << k++ << endl;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            cout << board[i][j] << "\t";
        }
        cout << "\n ";
    }
    cout << "\n";
}

bool isSafe(const vector <vector <int>> & board, int row, int col) {
    int i, j;
    for (i = 0; i < col; i++) {
        if (board[row][i]) {
            return false;
        }
    }
    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return false;
        }
    }

    for (i = row, j = col; j >= 0 && i < N; i++, j--) {
        if (board[i][j]) {
            return false;
        }
    }
    return true;
}

bool solveNQUtil(vector <vector <int>> & board, int col) {
    if (col == N) {
        printSolution(board);
        return true;
    }
```

```
53      bool res = false;
54      for (int i = 0; i < N; ++i) {
55          if (isSafe(board, i, col)) {
56              board[i][col] = 1;
57              res = solveNQUtil(board, col + 1) || res;
58              board[i][col] = 0;
59          }
60      }
61      return res;
62  }
63
64  void solveNQ() {
65      auto board = vector <vector <int>>(N);
66      for (int i = 0; i < N; ++i) {
67          board[i].resize(N);
68      }
69
70      if (solveNQUtil(board, 0) == false) {
71          cout << "\nSolution does not exist\n";
72          return;
73      }
74      return;
75  }
76
77  int main10() {
78      solveNQ();
79      return 0;
80  }
81
```