



# INTELIGENCIA ARTIFICIAL

*Profesor: DR. ISMAEL EVERARDO BARCENAS PATIÑO*

## PRÁCTICA 01

Busqueda exhaustiva

**Grupo: 08**

**Facultad de Ingeniería**

**Ciudad Universitaria, 18 de septiembre del 2025**

# Práctica 1 : búsqueda exhaustiva

Inteligencia Artificial

Grupo 4

Fecha 18/09/2025

## Descripción

El problema del caballo raro consiste en colocar en un tablero de n×n la mayor cantidad posible de estas piezas, donde cada una se mueve de forma similar a un caballo de ajedrez (en "L"), pero al final de dicho movimiento da un paso extra al frente o atrás y desde esa casilla es donde "ataca"; la restricción es que ningún caballo raro puede colocarse en una posición que quede bajo ataque de otro, de modo que el objetivo es encontrar mediante un algoritmo de búsqueda en profundidad (backtracking) la distribución de caballos raros que maximice su número sin que se ataquen entre sí.

## Equipo 1

- Programa: **caballo raro**

## Calificación

- Número de cuenta 1: 319236541
  - Preguntas:
  - Final:
- Número de cuenta 2: 320315064
  - Preguntas:
  - Final:
- Número de cuenta 3: 320213537
  - Preguntas:
  - Final:
- Número de cuenta 4: 319219917
  - Preguntas:
  - Final:
- Número de cuenta 5: 320252367
  - Preguntas:
  - Final:

# Código original

```
// Este codigo solo encuentra la primer solucion (no necesariamente la mejor)
#include<bits/stdc++.h>
using namespace std;

vector<pair<int, int>> moves = {
  {-1, -1}, {1, -1}, {1, 1}, {-1, 1},{-1, 3}, {1, 3},
  {-1, -3}, {1, -3}, {-3,-1},{-3, 1}, {3,-1},{3, 1}
};
int n;

bool check(vector<vector<bool>> board, int i, int j) {
  for (auto [x, y] : moves) {
    int a = i + x;
    int b = j + y;
    if (a >= 0 && a < n && b >= 0 && b < n && board[a][b]) {
      return false;
    }
  }
  return true;
}

void print(vector<vector<bool>> board) {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      cout << (int)board[i][j] << ' ';
    }
    cout << '\n';
  }
  cout << '\n';
}

void dfs(vector<vector<bool>> board, int i, int j, int knights) {
  if (i == n) {
    cout << "\nNumero de caballos: " << knights << "\n\n";
    print(board);
    exit(0);
  }

  print(board);

  int x = (j + 1 < n) ? i : i + 1;
  int y = (j + 1 < n) ? j + 1 : 0;

  if (check(board, i, j)) {
    board[i][j] = 1;
    dfs(board, x, y, knights + 1);
  }

  board[i][j] = 0;
  dfs(board, x, y, knights);
}

int main() {
  cout << "Ingrese el valor de n\n$ ";
```

```
  cin >> n;

  dfs(vector<vector<bool>>(n, vector<bool>(n, 0)), 0, 0, 0);
}
```

# Explicación del código

## 1. Movimientos

Se definen los movimientos posibles del "caballo raro":

```
vector<pair<int, int>> moves = {
  {-1, -1}, {1, -1}, {1, 1}, {-1, 1},{-1, 3}, {1, 3},
  {-1, -3}, {1, -3}, {-3,-1},{-3, 1}, {3,-1},{3, 1}
};
```

Estos movimientos no corresponden al caballo tradicional de ajedrez, sino a un conjunto de saltos diferentes.

## 2. Función `check`

- Revisa si es posible colocar un caballo en la celda $(i, j)$.
- Si alguna de las posiciones alcanzables desde $(i, j)$ ya tiene un caballo, se retorna `false`.

## 3. Función `print`

Muestra el tablero en consola:

- `1` significa que hay un caballo.
- `0` significa celda vacía.

## 4. Búsqueda recursiva `dfs`

- Recorre el tablero celda por celda.
- Caso base: cuando $i == n$, se terminó de recorrer el tablero y se imprime una solución.
- Calcula la siguiente posición $(x, y)$ en el tablero.

### 5. Núcleo del backtracking

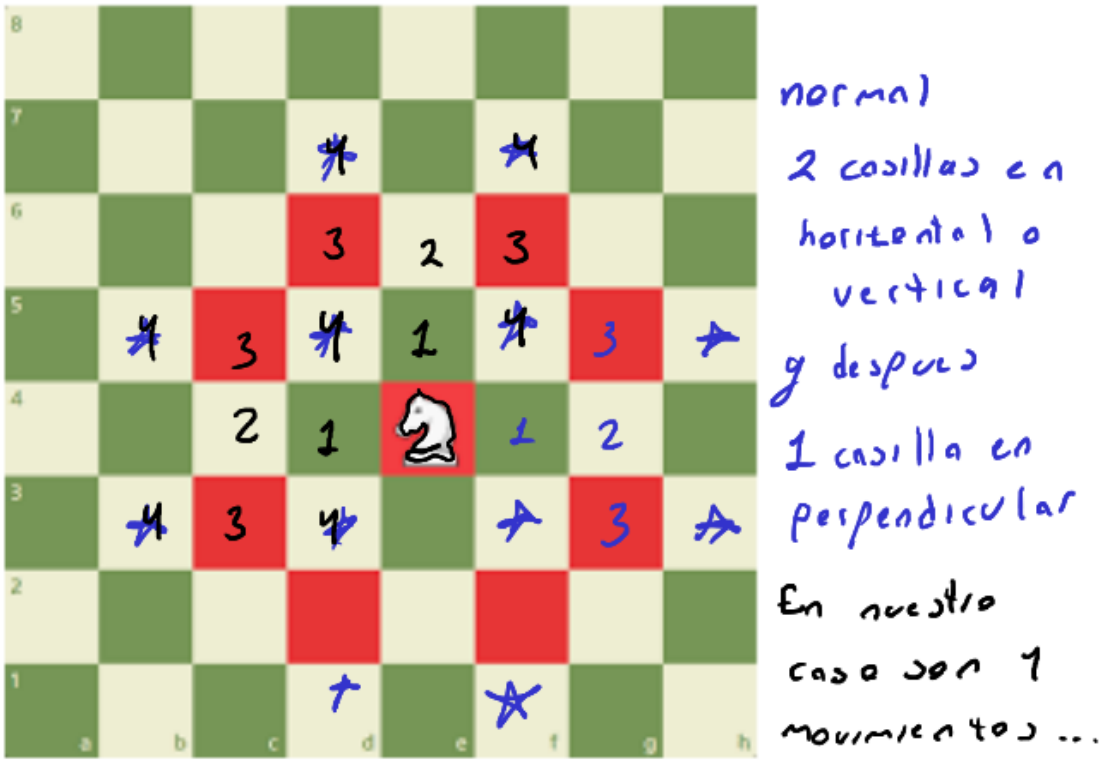
```
if (check(board, i, j)) {   //  puedo  poner caballo aqu ?
  board[i][j] = 1;          // decisi n: colocar caballo
  dfs(board, x, y, knights + 1); // recursi n
}
board[i][j] = 0;            // backtrack: deshacer decisi n
dfs(board, x, y, knights);  // recursi n sin colocar caballo
```

Aquí ocurre el **backtracking**:

- Se intenta colocar un caballo.
- Después se deshace la decisión (`board[i][j] = 0`) y se explora la rama alternativa: no colocar caballo.

## Logica de tablero con movimientos

### Movimientos



## Llenado de tabla comprobación