
O PASSEIO DO CAVALO

Eduardo Savian, Marcos Fehlauer

INTRODUÇÃO

- O passeio do cavalo tem como objetivo fazer o cavalo de xadrez percorrer um tabuleiro de xadrez de 8x8 de modo que cada casa do tabuleiro seja visitada exatamente uma vez;
- O problema pode ser estendido para tabuleiros de diferentes dimensões e formas;
- O cavalo se move em forma de "L" (duas casas em uma direção e uma casa perpendicularmente);
- A solução do passeio do cavalo pode ser encontrada por meio de força bruta ou de heurística como a de Wansdorff .

PONTOS POSITOS E NEGATIVOS

- A heurística de Warnsdorff implementado com um algoritmo guloso tem uma solução rápida, porém não tem solução garantida;
- O backtracking devido a ser força bruta terá uma tempo grande e necessitará de bastante memória para resolver, mas garante uma solução;
- O backtracking utilizando uma escolha aleatório tem os mesmos problemas do original, contudo ocasionalmente encontra a solução devido a sorte;
- O backtracking utilizando a escolha de ir para a próxima posição que tiver maior grau de possibilidades tem os mesmos problemas do original e pior desempenho devido a tendência de ir para posições sem saída.

ESTRUTUTAS UTILIZADAS

```
board := make([][]int, boardSize)
for i := range board {
    board[i] = make([]int, boardSize)
}

type Move struct {
    X, Y      int
    Priority int
}

var possibleMoves = [][]int{
    {-2, -1}, {-1, -2}, {1, -2}, {2, -1},
    {2, 1}, {1, 2}, {-1, 2}, {-2, 1},
}
```

VERIFICAR A VALIDADE DO MOVIMENTO

```
func isValidMove(x, y, boardSize int) bool {  
    return x >= 0 && x < boardSize && y >= 0 && y < boardSize  
}
```

ENCONTRAR O PRÓXIMO MOVIMENTO – ENCONTRAR MOVIMENTOS

```
func findNextMoves(x, y, boardSize int, board [][]int, searchType string) []Move {
    validMoves := []Move{}

    for _, move := range possibleMoves {
        nextX := x + move[0]
        nextY := y + move[1]

        if isMoveValid(nextX, nextY, boardSize) && board[nextX][nextY] == 0 {
            validMoves = append(validMoves, Move{nextX, nextY, 0})
        }
    }

    switch searchType {
```

ENCONTRAR O PRÓXIMO MOVIMENTO - WARNSDORFF

```
case "warnsdorff":  
    for i := range validMoves {  
        move := &validMoves[i]  
        move.Priority = len(findNextMoves(move.X, move.Y, boardSize, board, "default"))  
    }  
    sort.Slice(validMoves, func(i, j int) bool {  
        return validMoves[i].Priority < validMoves[j].Priority  
    })
```

ENCONTRAR O PRÓXIMO MOVIMENTO – MAIOR GRAU

```
case "highDegree":
    for i := range validMoves {
        move := &validMoves[i]
        move.Priority = len(findNextMoves(move.X, move.Y, boardSize, board, "default"))
    }
    sort.Slice(validMoves, func(i, j int) bool {
        return validMoves[i].Priority > validMoves[j].Priority
    })
```

ENCONTRAR O PRÓXIMO MOVIMENTO - ALEATÓRIO

```
case "shuffle":  
    rand.Shuffle(len(validMoves), func(i, j int) {  
        validMoves[i], validMoves[j] = validMoves[j], validMoves[i]  
    })
```

ENCONTRAR O PRÓXIMO MOVIMENTO - PADRÃO

```
}  
  
    return validMoves  
}
```

BUSCA GULOSA

```
func greedySearch(board [][]int, x, y, boardSize int, searchType string) bool {  
    board[x][y] = 1  
    for moveNum := 2; moveNum <= boardSize*boardSize; moveNum++ {  
        nextMoves := findNextMoves(x, y, boardSize, board, searchType)  
        if len(nextMoves) == 0 {  
            return false  
        }  
        move := nextMoves[0]  
        x, y = move.X, move.Y  
        board[x][y] = moveNum  
    }  
    return true  
}
```

BACKTRAKING

```
func backtrackSearch(board [][]int, moveNum, x, y, boardSize int, backtrackType string) bool {  
    board[x][y] = moveNum  
  
    if moveNum == boardSize*boardSize {  
        return true  
    }  
  
    nextMoves := findNextMoves(x, y, boardSize, board, backtrackType)
```

BACKTRAKING

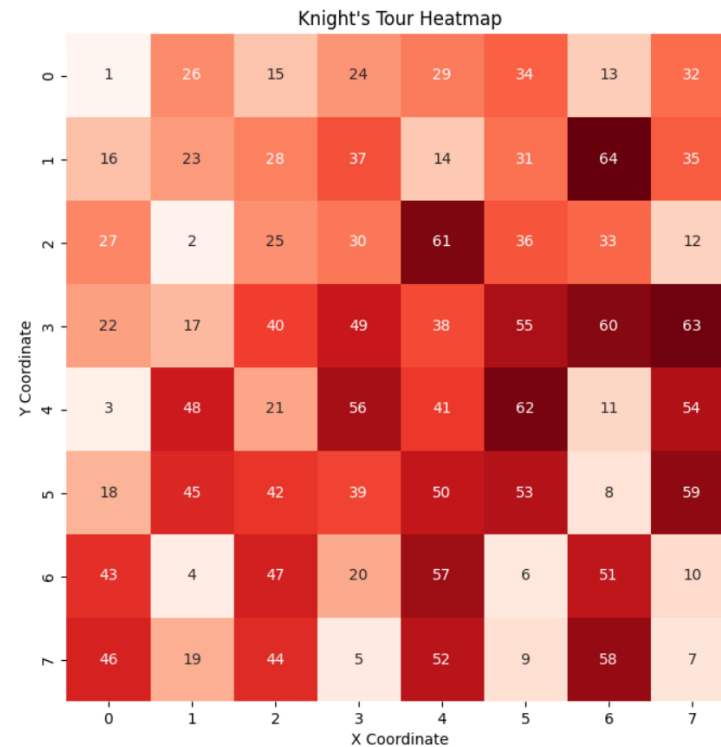
```
for _, move := range nextMoves {  
    if backtrackSearch(board, moveNum+1, move.X, move.Y, boardSize, backtrackType) {  
        return true  
    }  
}  
  
board[x][y] = 0  
return false  
}
```

CHAMADA DAS FUNÇÕES DE SOLUÇÃO

```
if (algorithm == "warnsdorff") {  
    solution = greedySearch(board, startX, startY, boardSize, algorithm)  
}  
  
if(algorithm == "backtrack" || algorithm == "highDegree" || algorithm == "shuffle") {  
    resultChan := make(chan bool)  
    ctx, cancel := context.WithTimeout(context.Background(), 15*time.Second)  
    defer cancel()  
  
    go func() {  
        resultChan <- backtrackSearch(board, 1, startX, startY, boardSize, algorithm)  
    }()  
  
    select {  
    case <-ctx.Done():  
        solution = false  
    case result := <-resultChan:  
        solution = result  
    }  
}
```

EXEMPLOS DE SOLUÇÃO – WARNSDORFF 8X8

Board Size: 8
Start X: 0
Start Y: 0
Algorithm: warnsdorff
☐ Animate
[Find Tour](#)




EXEMPLOS DE SOLUÇÃO – WARNSDORFF 8X8

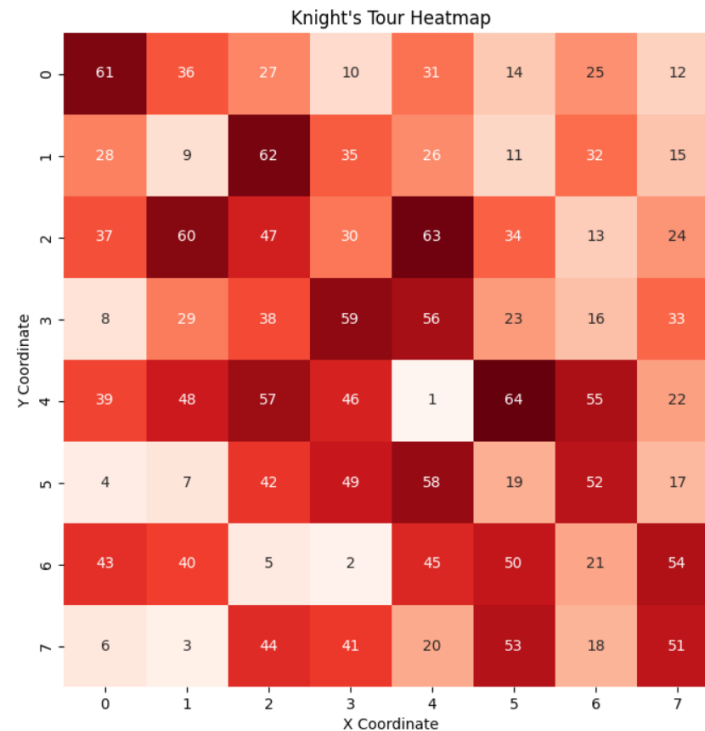
Board Size:

Start X:

Start Y:

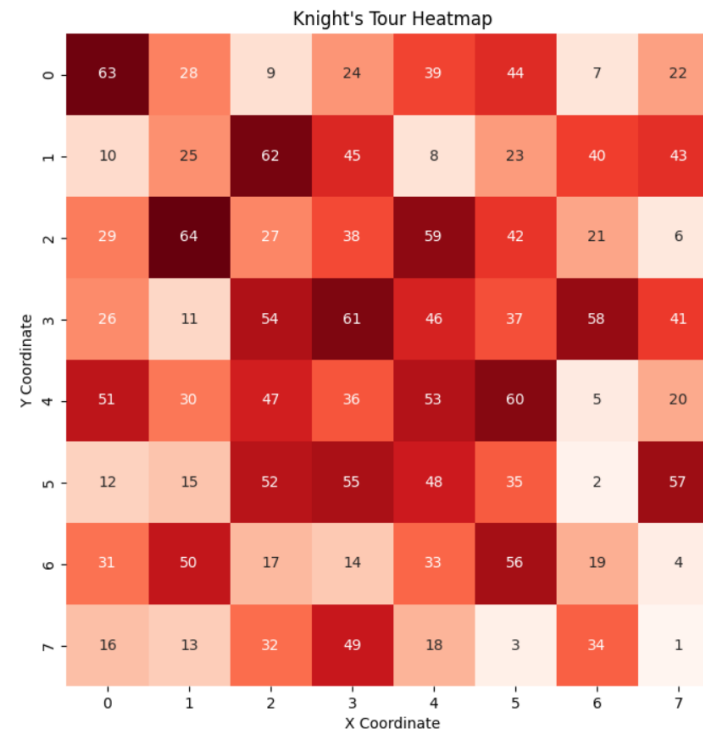
Algorithm: 

☐ Animate



EXEMPLOS DE SOLUÇÃO – WARNSDORFF 8X8

Board Size: 8
Start X: 7
Start Y: 7
Algorithm: warnsdorff
☐ Animate
Find Tour



EXEMPLOS DE SOLUÇÃO – OUTROS MÉTODOS 8X8

- Não foi possível encontrar a solução do passeio do cavalo usando outros métodos regularmente;
- O problema é o tempo de espera, devido a busca tentar todos os caminhos possíveis;
- Devido a sorte do backtracking com aleatoriedade, ocasionalmente era possível encontrar a solução, mas a maior parte das tentativas demoravam ou não encontravam solução.

EXEMPLOS DE SOLUÇÃO – WARNSDORFF 6X6

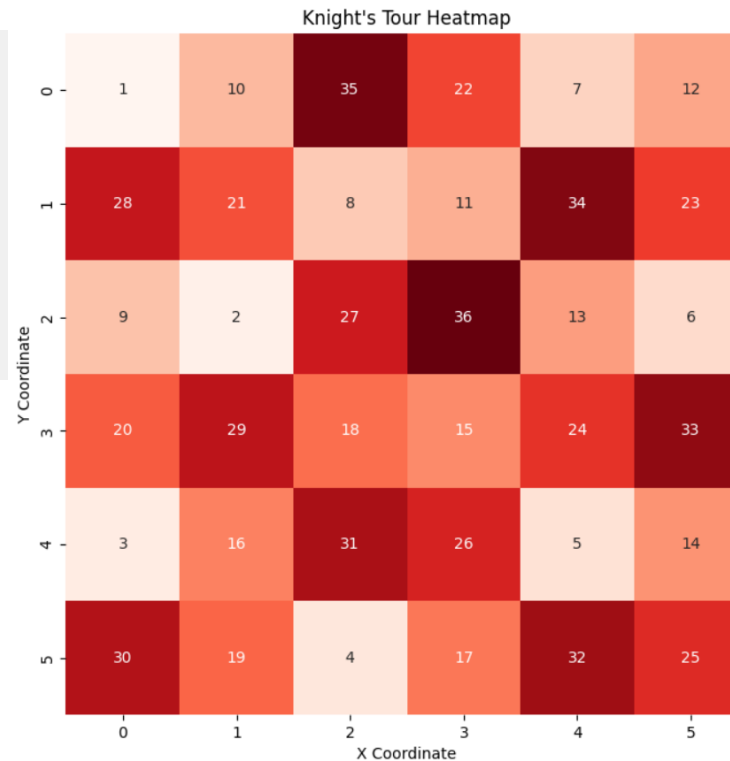
Board Size:

Start X:

Start Y:

Algorithm:

☐ Animate



EXEMPLOS DE SOLUÇÃO – BACKTRACK 6X6

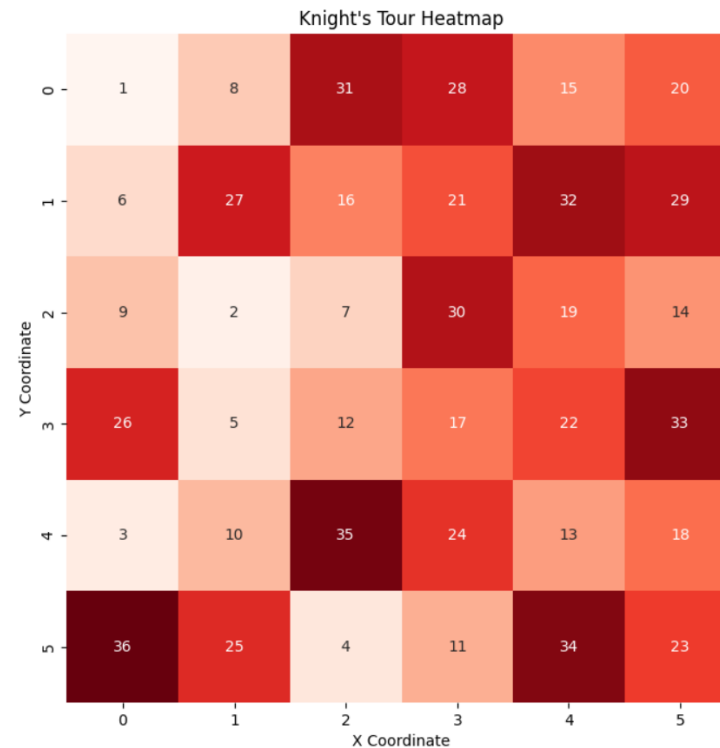
Board Size:

Start X:

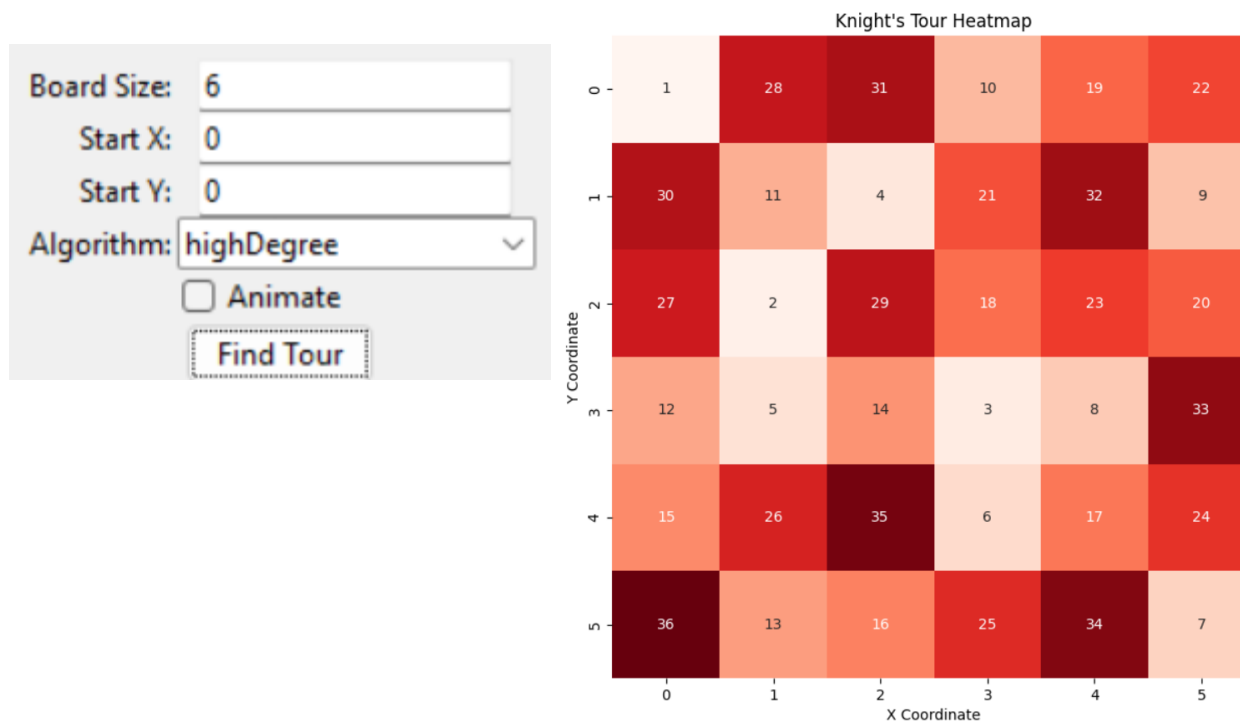
Start Y:

Algorithm:

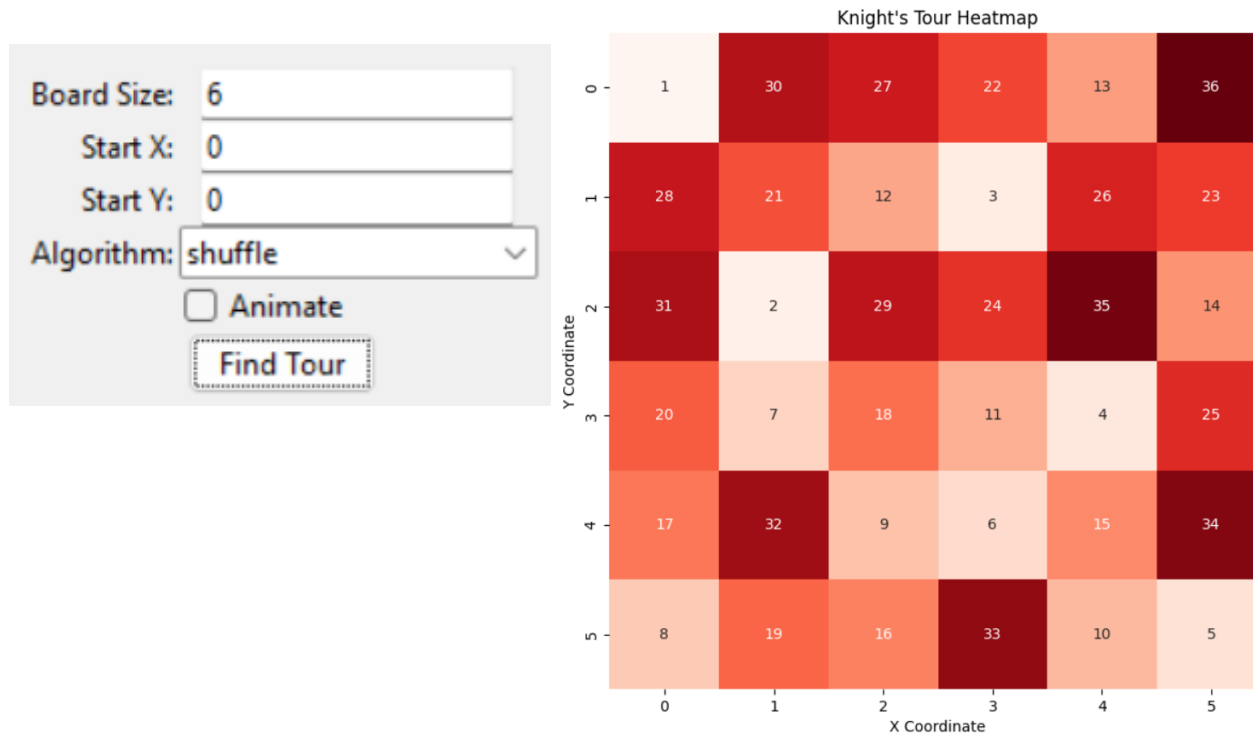
☐ Animate



EXEMPLOS DE SOLUÇÃO – MAIOR GRAU 6X6



EXEMPLOS DE SOLUÇÃO – ALEATÓRIO 6X6



REFERÊNCIAS BIBLIOGRÁFICAS

- PARBERRY, I. **An efficient algorithm for the Knight's tour problem**. Disponível em: <<https://core.ac.uk/download/pdf/81964499.pdf>>. Acesso em: 14 maio. 2024.
- WIKIPEDIA CONTRIBUTORS. **Knight's tour**. Disponível em: <https://en.wikipedia.org/w/index.php?title=Knight%27s_tour&oldid=1220995933>.

OBRIGADO