O Cavalo Perdido
Algoritmos e Estrutura de Dados II

Daniela Pereira Rigoli

[1]Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 1429 – 90619-900 – Porto Alegre – RS – Brazil

Resumo. *Este artigo descreve uma solução para o trabalho proposto na disciplina de Algoritmos e Estruturas de Dados II, que se trata de encontrar a saída usando um cavalo em um tabuleiro toroidal. Além disso, são apresentados mais detalhes sobre o desafio, assim como resultados para casos de teste propostos na disciplina, ideias, estruturas usadas, pseudo-código dos algoritmos, dificuldades encontradas e testes para validar os resultados.*

1. Introdução

Este trabalho visa resolver um desafio de xadrez espacial e aprimorar os conhecimentos de grafos vistos na disciplina de Algoritmos e Estruturas de Dados. Para o desafio foram preparados uma série de tabuleiros com tamanhos variados, onde foi colocado um único e solitário cavalo e o cavalo deve chegar até uma posição chamada de Saída no menor número possível de movimentos (se isto for possível). Em cada tabuleiro existe uma série de casas onde o cavalo não deve pisar de jeito nenhum.

No xadrez espacial os lados do tabuleiro se encostam, ou seja, se o cavalo sair do tabuleiro pelo lado esquerdo ele entra pelo lado direito (e vice-versa) e a mesma coisa acontece para os lados de cima e de baixo. Assim, pode-se afirmar que o tabuleiro é toroidal e infinito.

Os cavalos se movem de acordo com a Figura 1 e também estão inicialmente na posição C no tabuleiro. A saída está marcada com S e as posições com x não podem ser ocupadas.

Na Figura 2 é apresentado um caso de teste. Nesse exemplo o cavalo C pode chegar até a saída S em 10 pulos. Foram elaborados alguns casos para testar o algoritmo implementado neste trabalho.

Neste trabalho foi desenvolvido um algoritmo que recebe o tabuleiro em um arquivo e retorna o menor número de movimentos que são necessários para o cavalo chegar na saída quando isto for possível.

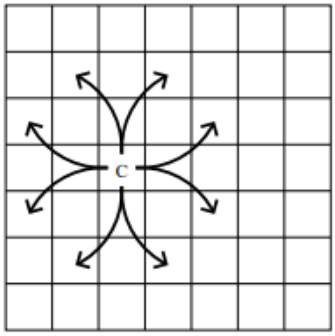

Figura 1. Destinos para onde o cavalo por ir.

```
..................x.....................x
.x......x..x..x..x..............x....x.
...........xx....x....x...........x...x
......x.x.........x..............x..x.x
..........C.....xx....x...xx...x........
.x....xx......xxx..xx....x.xx...........
.x............x.....x...x...........x.x.
........x...............x.......x...x..
x..x.x.....................x..x.........
............x.....x....x...............
.............................x..x.......
...x.....x.x.x.x.........x.xx.....x.....
.....x..x...x..........x................
.x............x.........xx..x.xx...x.x..
x..............x....x.x........x..S.....
..............x...x.x.........x....x....
.....x....x.......xxx......x........x.x.
............x.xx...xxx......x..........
........x..x.........x.....x....x........
.....xx...x.........x....x...x.....x.....
```

Figura 2. Entrada para tabuleiro do caso de teste 50.

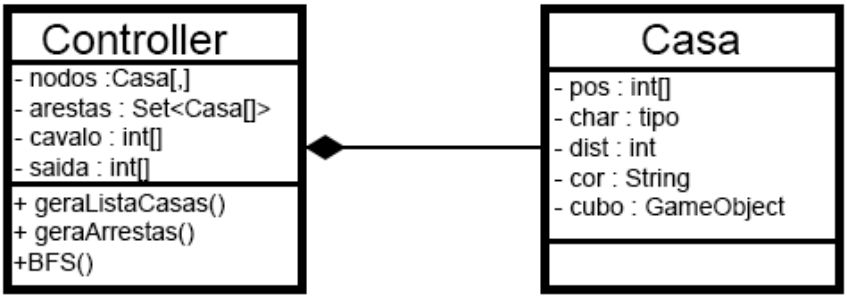

Figura 3. Diagrama de classes da solução.

2. Desenvolvimento

Para a solução desse desafio escolheu-se utilizar $C\#$ e $Unity$, para possibilitar a visualização e tornar mais didático o desenvolvimento do algoritmo. Na Figura 3 é possível ver o diagrama de classes utilizado na solução, contendo uma classe com todas as informações das casas do tabuleiro e uma principal onde é criado os nodos, arestas e o método utilizado para descobrir o mínimo de casas possíveis para chegar na saída.

Baseado nos algoritmos de grafos vistos em aula notou-se que o primeiro passo para fazer a implementação da solução do desafio era gerar os nodos, foi escolhido armazenar em uma matriz devido as duas dimensões, assim como o tabuleiro de xadrez. Para fazer isso apenas se percorreu os arquivos dos testes de caso e para cada caractere era gerado uma casa do tabuleiro. E é anotado a posição em que está o cavalo e a saída para serem utilizadas depois no método de busca.

Para efeitos visuais para cada casa é gerado um cubo como é possível ver na Figura 4. As cores escolhidas foram: preto para os nodos descobertos, onde o cavalo já passou; azul claro para os nodos de fronteira, nodos que estão na lista para serem descoberto; branco para os nodos que ainda não foram descobertos; vermelho para os nodos que não podem ser descobertos; inicialmente amarelo para o cavalo e verde para a saída.

O segundo passo foi a geração das arestas, para cada nodo diferente de "x" verificava-se com quais casas ele podia se conectar e essa informação era guardada em um set. Para isso, foi criado uma lista de movimentos para os quais o cavalo pode ir, considerando seus movimentos em eixos x e y. Para cada movimento do cavalo foi necessário garantir que não acabasse saindo do tabuleiro, verificando se havia passado de uma das

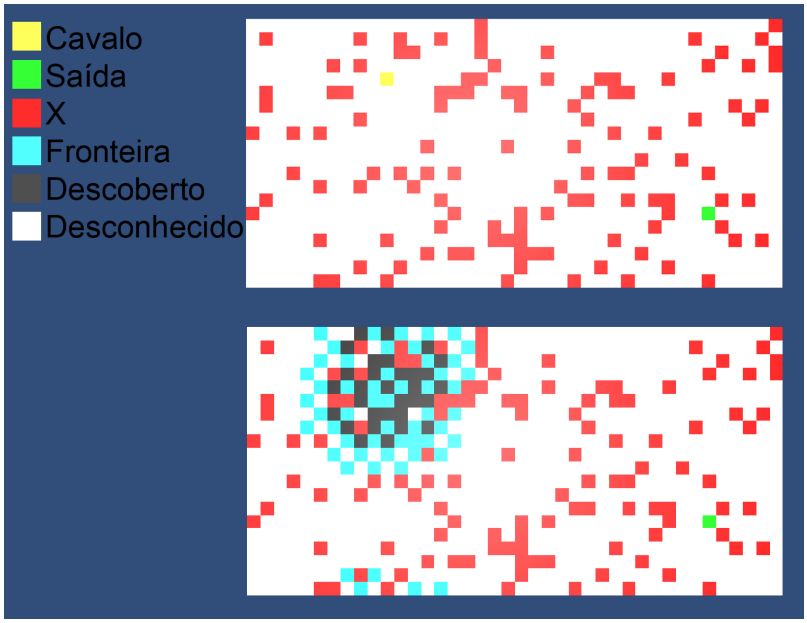

Figura 4. Tabuleiros gerados no Unity.

bordas para ser direcionado ao outro lado, assim como é representado no pseudocódigo abaixo.

```
geraArestas()
    moviments = ((2, 1), (1,2), (2,-1), (1,-2),
                 (-1, 2), (-2,1), (-2,-1), (-1,-2))

    for c in nodos:
        if c.getTipo() != 'x'
            for mov in moviments:
                x = c.getX() + mov[0]
                y = c.getY() + mov[1]

                if(x >= len(nodos))
                    if(x == len(nodos))
                        x = 0
                    else if(x == len(nodos) + 1)
                        x = 1
                else if(x < 0)
                    if(x == -1)
```

```
            x = len(nodos)-1
        else if(x == -2):
            x = len(nodos)-2
    if(y >= len(nodos[0]))
        if(y == len(nodos[0]))
            y = 0
        if(y == len(nodos[0]) + 1):
            y = 1
    else if(y < 0)
        if(y == -1)
            y = len(nodos[0])-1
        else if(y == -2)
            y = len(nodos[0])-2

    if(nodos[x][y].getTipo != 'x')
        arestas.add((c, nodos[x][y]))
```

Analisando as formas de caminhar em um grafo percebeu-se que utilizando busca por largura poderia se obter resposta de forma simples. A busca por largura foi adaptado para o tabuleiro toroidal.

Sendo assim, na classe Casa é armazenado o valor da distância até o cavalo, que inicialmente é infinito porque não se sabe a quantos nodos de distância está. Além de definir inicialmente a cor como branco, apenas para informar que é uma casa desconhecida ainda, lembrando que isso não influencia na cor visual do cubo.

O pseudocódigo abaixo representa o método de busca por largura do tabuleiro. Onde inicialmente é alterada a distância do cavalo para zero, inicializado a fila e adicionado o cavalo a fila. Então se tem um looping até encontrar a saída, onde o nodo atual é passado para a variável u é retirada da fila. Verifica se um dos nodos adjacentes de u é desconhecido, se for é adicionado como nodo de fronteira, alterado o valor da sua distância e colocado na fila. Após ver os nodos adjacentes de u é alterado sua cor para azul, significando que já foi descoberto.

```
buscaLargura(nodo c, nodo s)
    c.dist = 0
    Q = {}
    Q.enqueue(c)

    u
    enquanto u != s
```

```
u = Q.dequeue()
para cada vertice v adjacente u do grafo
        se v.cor == branco
                v.cor = cinza
                v.d = u.dist = u.dist + 1
                Q.enqueue(v)
u.cor = azul
```

Para demonstrara busca do cavalo funcionando foi gravado a animação do algoritmo do cavalo perdido (https://youtu.be/A6gQPXfO2gk), onde é possível ver que o cavalo usa a propriedade toroidal do tabuleiro para encontrar o menor caminho.

3. Resultados e Conclusão

Com esse trabalho foi possível aprimorar os conhecimentos em grafos, desde como gerar o grafo em si até formas de busca em grafos. Durante o trabalho foi testado as distâncias geradas utilizando busca por largura que permite responder o objetivo, qual o menor número possível de movimentos. Para os casos de teste desenvolvidos para esse desafio se obteve os resultados da tabela abaixo.

A utilização da ferramenta visual do $Unity$ permitiu identificar problemas no desenvolvimento mais facilmente e confirmar que a busca do cavalo estava funcionando. Também foi feito o algoritmo sem interface visual para avaliar questões de tempo, e caso que levou mais tempo foi de $1,19$ segundos, para o caso de teste 550.

Caso de Teste	**Máquinas em Equilíbrio**
100	68
150	64
200	108
250	156
300	197
350	170
400	185
450	186
500	225
550	223