Os Anéis de Saturno

André Luiz Marques Macrini Leite, Giovanni Frozza

22 de novembro de 2020

Resumo

O artigo presente irá conduzir à uma das possíveis soluções de como se locomover nos diversos portais ou caminhos encontrados nos Anéis de Saturno. Os Anéis de Saturno são um ambiente que possuem vida inteligente e que nas bordas externas e internas do anel existem uma rede de portais (Cada portal é nomeado de A a Z para a rede externa e também de A a Z para a rede interna) que permitem a locomoção dos habitantes. Um habitante do Anel pode se teleportar por portais que contém o mesmo nome ou então caminhando de um portal ao outro. Neste artigo será apresentado um algoritmo que visa encontrar o menor caminho para chegar do portal A ao portal Z

Palavras-chaves: Algoritmos. Grafos. Caminhamento em largura.

Introdução

O problema apresentado nos Anéis de Saturno requer entender as complexas ligações entre os diversos portais que o Anel possui. A NASA instruiu o desenvolvimento de um algoritmo que irá percorrer essas ligações entre a rede de portais e retornar à quantidade de tempo que leva para percorrer do portal A ao portal Z da borda interna do Anel. Cada movimento realizado no Anel é equivalente a uma (1) unidade de tempo e podem ser feitos através dos espaços livres ou por uma letra do alfabeto respectivamente. Para conseguir entender as ligações a leitura do problema apresenta uma lista com as seguintes instruções:

- A – Z representa os portais;
- representa um rochedo por onde não se pode passar;
- . representa espaço livre.

A NASA apresentou uma foto para entender a estrutura do Anel, a localização dos portais, dos espaços livres e dos obstáculos. Para exemplificar, é apresentada a seguinte estrutura, que se refere ao caso de teste apresentado no enunciado do trabalho:

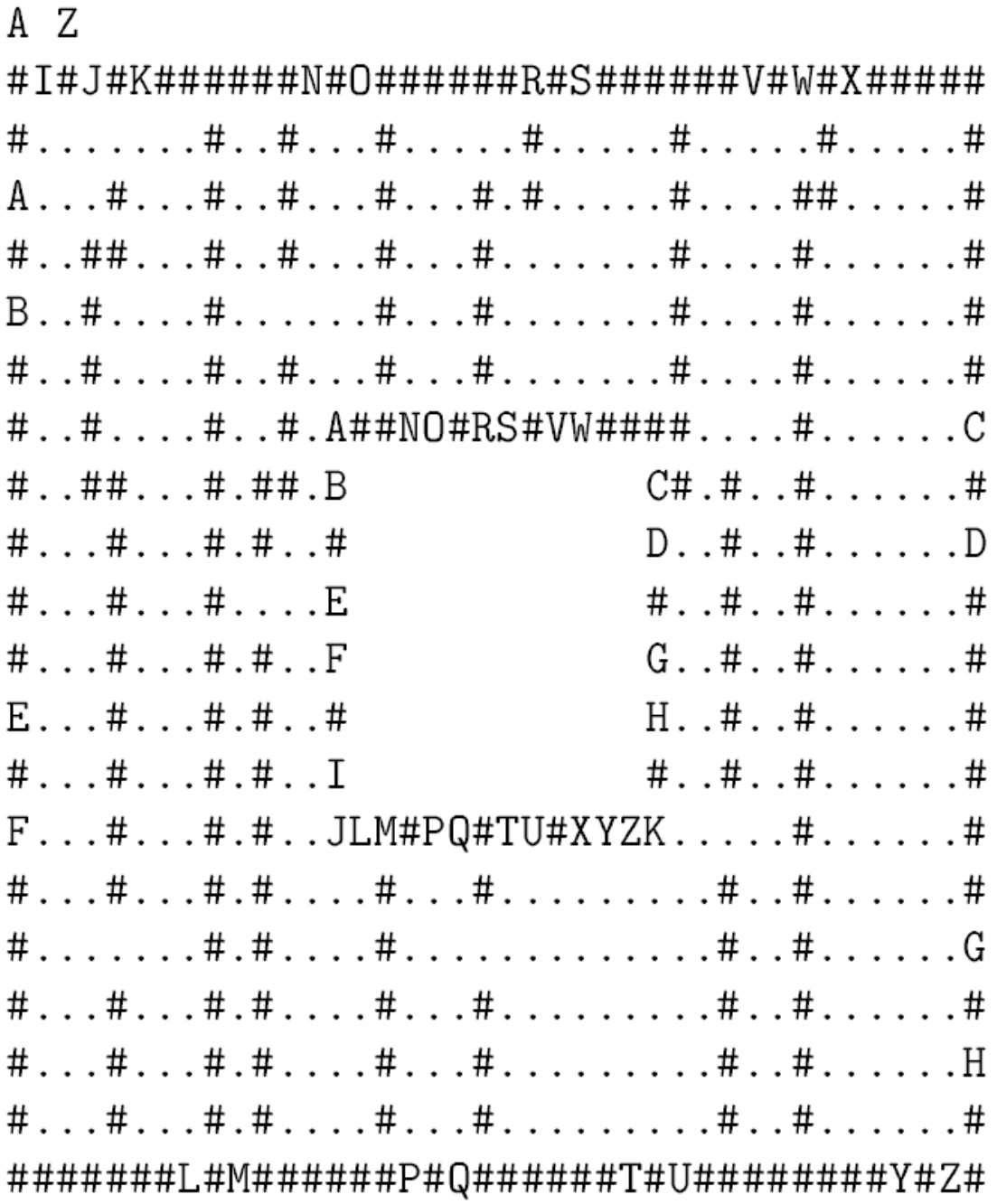

Figura 1 – Exemplo

Os habitantes não podem se mover nas diagonais e para ir de A (externo) a Z (interno) precisando somente de 9 unidades de tempo para se mover entre os portais de partida e destino.

RESULTADO DO CAMINHAMENTO A Z

A esta em 3 1

Z esta em 14 26

A 3 1 (0)

. 3 2 (1)

. 2 2 (2)

. 2 3 (3)

. 2 4 (4)

. 2 5 (5)

. 2 6 (6)

K 1 6 (7)

K 14 27 (8)

Z 14 26 (9)

Seguindo as regras da rede de portais do Anel foi possível desenvolver um algoritmo para a resolução do problema. Ao final do artigo será apresentado a identificação dos casos de teste contendo seus resultados.

Matrizes, Grafos e BFS

Matriz

Matrizes são estruturas de dados que apresentam um conjunto de valores do mesmo tipo. A linguagem de programação Java possui arrays e classes multidimensionais que permite a utilização de uma série de classes de coleções de objetos. Para a declaração de uma matriz em Java deverão ser fornecidas o tipo dos elementos que serão armazenados na matriz e o número de linhas e colunas que representam a dimensão. Logo, a linguagem Java permite ao programador criar uma estrutura de arrays "unidimensionais" no qual seus elementos recebem outro arrays "unidimensionais", alcançando assim a mesma estrutura de mesmo efeito que a "multidimensional".

Grafo

Um grafo é um tipo abstrato de dados que permite representar relações entre dois objetos. Os objetos são representados por vértices e as relações entre dois objetos por uma aresta. Os vértices do grafo são normalmente representados por círculos ou retângulos e suas relações formam um diagrama que podem ou não ser conectados por arestas que são tipicamente representadas por segmentos de retas ou arcos, com ou sem setas.

BFS - Busca em Largura

Para IME-USP (2019) a busca em largura, ou o termo em inglês breadth-first search(BFS), tem o objetivo de visitar a ponta inicial de um arco percorrendo pelos arcos de um vértice a outro, onde cada arco visitado é percorrido no máximo uma vez.

O algoritmo numera os vértices, em sequência, na ordem em que eles são descobertos (visitados pela primeira vez), ou seja, o algoritmo pega o vértice inicial, visita ele e depois todos os seus vizinhos, e depois os vizinhos dos vizinhos, e assim por diante. Através deste algoritmo é possível descobrir o menor caminho de um vértice a outro.

Solução

O algoritmo desenvolvido para este desafio utiliza a linguagem de programação JAVA e contém três classes App, Coordenadas e Grafo.

Classe App

Contém o método main e é onde faz a leitura dos dados de entrada e retorna os resultados com auxílio do método getTamanhoLinha(). Esta classe lê a segunda linha do arquivo de caso de teste e com base no tamanho encontrado dessa linha, é usado para instanciar a classe Grafo. Depois é realizado toda a leitura do arquivo onde vai ser montado uma matriz com auxilio do método insereLinha() da classe Grafo. Esta classe contém um método que recebe por parâmetro uma String, um arquivo .txt, e fará a conversão deste arquivo utilizando a interface das classes File, FileReader e BufferReader onde irá introduzir em um arranjo de String que irá ler os primeiros dois valores do arquivo .txt que irá determinar as coordenadas de origem e destino.

Classe Coordenadas

A classe Coordenadas apenas contém o método construtor que recebe dois parâmetros do tipo inteiro que representam o eixo x e y da matriz. Estes valores são importantes para a construção e mapeamento da matriz, que vão ser utilizados por quase todas as funções do algoritmo para diferentes finalidades.

Classe Grafo

A classe foi inspirada no problema do Passeio do Cavalo, onde decidimos utilizar como estrutura para o grafo uma matriz.

Esta classe é composta por 9 variáveis globais onde 4 são matrizes, 2 do tipo Coordenadas e 3 do tipo int. Além disso é composto também por um construtor que inicializa as matrizes e as variáveis do tipo inteiro.

O método encontraCaminho() que recebe a origem e destino que são lidas na primeira linha do arquivo de caso de teste. Neste método a primeira coisa a ser feita é definir as coordenadas do portal de origem e o portal de destino pelas variáveis globais Coordenadas portalInicio e portalFim respectivamente. Após definido as coordenadas são chamado o método bfs.

O método bfs() é possível saber a distância de um vértice para qualquer outro vértice. O método recebe duas coordenadas para descobrir a menor distância entre estas duas coordenadas. O algoritmo bfs utilizado neste trabalho é o que foi apresentado em aula, mas adaptado para usar matriz comum. Neste algoritmo cria-se uma Lista de Coordenadas e instancia uma fila. Usa-se uma matriz para sinalizar os marcados, uma matriz para sinalizar de qual vértice está vindo e a distância até a última marcação. Com uma estrutura de repetição “while” para que enquanto a fila estiver preenchida ele verifica se os vértices vizinhos já estão marcados, se são parede ou se estão em branco. Se o conteúdo da matriz for diferente de parede, branco ou marcado, ele marca que o vértice foi visitado, registra a origem da marcação e adiciona na lista. Se encontrado a coordenada destino o método é encerrado.

```
private void bfs(int x, int y) {
  List<Coordenadas> q = new LinkedList<>();
```

```
  Coordenadas cood = new Coordenadas(x, y);
  q.add(cood);
  marcados[cood.x][cood.y] = true;
  edgeTo[x][y] = new Coordenadas(-1, -1);
  distTo[x][y] = 0;
  boolean acabou = false;
  while (!q.isEmpty()) {
    Coordenadas v = q.remove(0);
    for (Coordenadas w : this.getAdj(v.x, v.y)) {
      if ( !marcados[w.x][w.y]
          && !mat[w.x][w.y].equals("#")
          && !mat[w.x][w.y].isBlank() ) {
        Coordenadas to = new Coordenadas(w.x, w.y);
        marcados[to.x][to.y] = true;
        edgeTo[to.x][to.y] = new Coordenadas(v.x, v.y);
        distTo[to.x][to.y] = distTo[v.x][v.y] + 1;
        q.add(to);
        if (w.x == this.portalFim.x && w.x == this.portalFim.y) {
          acabou = true;
        }
      }
    }
    if (acabou) break;
  }
}
```

O método getAdj() faz uma série de verificações de limites e é responsável por olhar para os vértices vizinhos e para o vértice corrente.

```
private List<Coordenadas> getAdj(int x, int y) {
  List<Coordenadas> adj = new ArrayList<>();
  if ( !(x - 1 < 0) ) {
    Coordenadas top = this.getCoordenadas(x - 1, y);
    if (top != null) adj.add(top);
  }
  if ( !(x + 1 >= this.altura) ) {
    Coordenadas bottom = this.getCoordenadas(x + 1, y);
    if (bottom != null) adj.add(bottom);
  }
  if ( !(y - 1 < 0) ) {
    Coordenadas left = this.getCoordenadas(x, y - 1);
    if (left != null) adj.add(left);
  }
  if ( !(y + 1 >= this.largura) ) {
    Coordenadas right = this.getCoordenadas(x, y + 1);
    if ( right != null ) adj.add(right);
  }
  if ( !this.mat[x][y].equals(".")
      && !this.mat[x][y].equals("#")
      && !this.mat[x][y].isBlank() ) {
```

```
    Coordenadas portal = this.getPortal(x, y);
    if (portal != null) adj.add(portal);
  }
  return adj;
}
```

Se o vértice corrente for um portal será chamado o método getPortal(). O método getPortal() faz duas verificações, se o portal é Externo ou Interno. Dependendo da condição válida ele vai chamar o método correspondente oposto. Por exemplo, se o vértice corrente for um portal Externo, então a condição irá chamar o método getPortalInterno() e vice-versa. Para saber se devemos ir para o anel interno ou externo, verificamos se a posição do vértice atual é na borda exterior, se for, então devemos ir para o anel interno, se não, devemos ir para o anel externo.

```
private Coordenadas getPortal(int x, int y) {
  if (x == 0 || x == this.altura - 1 || y == 0 || y == this.largura - 1) {
    return this.getPortalInterno(this.mat[x][y]);
  } else {
    return this.getPortalExterno(this.mat[x][y]);
  }
}
```

Os métodos getPortalExterno() e getPortalInterno() são semelhantes em seu funcionamento, mas contém algumas diferenças de cálculo que determina o tamanho das bordas. Este método contém uma estrutura de repetição "for" e faz duas verificações. Na primeira verificação olha se o portal inicial está em uma das bordas superior ou inferior, se verdadeiro ele usa uma estrutura de repetição para mapear todas as linhas borda superior ou inferior. Caso a verificação seja falsa, é verificado somente a primeira e a ultima coluna das linhas como uma forma de otimização.

```
private Coordenadas getPortalExterno(String portal) {
  Coordenadas portalExterno = null;
  boolean achou = false;
  for (int i = 0; i < this.altura; i++) {
    if (i == 0 || i == this.altura - 1) {
      for (int j = 0; j < this.largura; j++) {
        if (this.mat[i][j].equals(portal)) {
          portalExterno = new Coordenadas(i, j);
          achou = true;
          break;
        }
      }
    } else {
      if (this.mat[i][0].equals(portal)) {
        portalExterno = new Coordenadas(i, 0);
        break;
      } else if (this.mat[i][this.largura - 1].equals(portal)) {
        portalExterno = new Coordenadas(i, this.largura - 1);
        break;
      }
    }
```

```
    if (achou) break;
  }
  return portalExterno;
}
```

O método getPortalInterno(), diferente do getPortalExterno(), obtém a posição central da matriz e a partir da posição central encontramos o inicio do anel interno e o tamanho desse anel. Com isso podemos percorrer o anel interno da mesma forma que percorremos o anel externo

```
private Coordenadas getPortalInterno(String portal) {
  int x = ( this.altura / 2 );
  int y = ( this.largura / 2 );
  int nrDist = 0;
  Coordenadas portalInterno = null;
  for (int i = x; i > 0; i--) {
    if ( !this.mat[i][y].isBlank() ) break;
    nrDist++;
  }
  int posX = ( x - nrDist );
  int posY = ( y - nrDist );
  int tam = posX + nrDist * 2;
  boolean achou = false;
  for (int i = posX; i < tam; i++) {
    if (i == posX || i == tam - 1) {
      for (int j = posY; j < tam; j++) {
        if (this.mat[i][j].equals(portal)) {
          portalInterno = new Coordenadas(i, j);
          achou = true;
          break;
        }
      }
    } else {
      if (this.mat[i][posX].equals(portal)) {
        portalInterno = new Coordenadas(i, posX);
        break;
      } else if (this.mat[i][tam - 1].equals(portal)) {
        portalInterno = new Coordenadas(i, tam - 1);
        break;
      }
      if (achou) break;
    }
  }
  return portalInterno;
}
```

O método encontrarCaminho() retorna o custo do caminhamento. O método, após encontrar o vértice destino que pertence ao anel interno executa o método bfs e assim que chega no vértice de destino, para de executar. O método considera que toda a matriz edgeTo e distTo foi preenchida e então o algoritmo consegue saber a distância levada para

chegar nos vértices olhando a matriz distTo.

```
public int encontraCaminho(String valorInicio, String valorFim) {
  Coordenadas coordenadas = this.getPortalExterno(valorInicio);
  this.portalInicio = coordenadas;
  this.portalFim = this.getPortalInterno(valorFim);
  this.bfs(coordenadas.x, coordenadas.y);
  List<Coordenadas> caminho = this.getCaminho();
  return caminho.size();
}
```

Análise do Algoritmo

Este algoritmo determina qual o menor custo possível para fazer o caminhamento pela rede de portais nos Anéis de Saturno. A notação deste algoritmo pode se dar em seu melhor caso O(1) e no pior caso $O(n^2)$, pois devemos considerar que o Anel pode induzir a visitar todas as posições permitidas da matriz para chegar do portal A ao portal Z.

Resultados

O exercício consiste em apresentar os resultados dos oito estudos de casos, além do caso de teste que foi exemplificado na introdução do artigo. Segue abaixo a tabela contendo a resposta esperada e obtida para cada caso:

	CUSTO		
CASOS	ESPERADO	OBTIDO	TEMPO (milissegundos)
caso1	20	20	9
caso2	23	23	27
caso3	2	2	55
caso4	110	110	357
caso5	96	96	1038
caso6	486	486	3673
caso7	537	537	3564
caso8	1436	1436	3057

Figura 2 – Resultados

Considerações finais

O desafio apresentou grau de dificuldade médio e a resolução poderia ser desenvolvida usando até mesmo leitura simples de arranjos que nos resultaria em uma complexidade maior, mas foi compreendido que utilizando apenas uma matriz simples seria o suficiente para solucionar o esquema complexo da rede de portais dos Anéis de Saturno. O algoritmo implementado utilizou algumas noções grafos, como por exemplo vértices, arestas e o funcionamento sobre busca em largura, no qual foi implementado, para a construção e caminhamento de estruturas de grafos. O algoritmo processou as informações e obteve um resultado equivalente ao esperado, logo, mostrando que o algoritmo atingiu completamente as expectativas e chegou ao que foi proposto pelo desafio.

Referências

DEVMEDIA, Matrizes - Aprenda a trabalhar com vetores bidimensionais - Revista easy Java Magazine 22. Disponível em: <https://www.devmedia.com.br/matrizes-aprenda-a-

trabalhar-com-vetores-bidimensionais-revista-easy-java-magazine-22/25766> Acessado em 21 de novembro de 2020 20:30.

DEVMEDIA, Programando com Grafos - Revista easy Java Magazine 29. Disponível em: <https://www.devmedia.com.br/programando-com-grafos-revista-easy-java-magazine-29/28119> Acessado em 21 de novembro de 2020 21:34.