

N-Rainhas Java

Pedro Loes

6/25/2021

Introdução

- O objetivo deste projeto foi desenvolver um programa para resolver o problema das N-Rainhas na linguagem Java. Este relatório apresenta a descrição do problema, explica o algoritmo, documenta o código e exemplifica o uso do programa.

Problema N-Rainhas

- O problema consiste em posicionar um número N de rainhas em um tabuleiro de xadrez $X_{n,n}$.
- A peça rainha pode ser movimentada em um número ilimitado de casas nas linhas, colunas ou diagonais.
- Uma posição é segura se não está na mesma linha, coluna ou diagonal de posições ocupadas.

Construção do Algoritmo

- O programa consiste em um algoritmo que recebe uma entrada do usuário e retorna todos os possíveis posicionamentos seguros das rainhas no tabuleiro $X_{n,n}$.
- O algoritmo foi desenvolvido com a implementação de 4 funções e 2 classes:
 1. **<verifique>**
 - Verifica se uma posição é segura.
 3. **<posicione>**
 - Executa laço recursivo para iterar sobre o espaço de busca.
 3. **<imprima>**
 - Imprime soluções encontradas pelo programa.
 4. **<nRainhas>**
 - Executa a função posicione na classe NRainhas.
 5. **<inteiroNaoPositivo>**
 - Estende a classe exceção.
 6. **<NRainhas>**
 - Classe principal de que recebe input do usuário e executa o programa.

- O funcionamento do algoritmo consiste em iterar sobre as linhas e colunas do tabuleiro $X_{n,n}$ procurando uma posição segura para cada rainha.
- O tabuleiro foi representado por um arranjo vetor $X_{\{n\}}$ onde as colunas são as posições do vetor e as linhas os valores.
- A verificação de segurança para linhas é realizada comparando os valores do vetor X_n . A verificação de segurança nas diagonais é realizada comparando índices e valores do vetor X_n .
- Caso uma posição selecionada inviabilize o posicionamento de qualquer rainha posterior, o algoritmo retorna e reposiciona as rainhas anteriores de forma recursiva até encontrar uma solução onde as n rainhas estarão seguras.

1. Função Verifique

- A primeira função denominada **<verifique>**, verifica se uma posição em uma determinada casa do tabuleiro é segura. As linhas, colunas e diagonais são inspecionadas e a função retorna verdadeiro se as condições de segurança da posição forem satisfeitas.
- A função recebe como parâmetros:
 1. Um arranjo de inteiros **<tabuleiro>** que representa o tabuleiro X_n do problema.
 2. Número inteiro coluna que representa as colunas ou rainhas.
- O laço itera sobre todas as colunas do tabuleiro verificando 2 condições:
 1. Se a rainha está posicionada na mesma linha que as rainhas anteriores.
 2. Se a rainha está posicionada na mesma diagonal que as rainhas anteriores.
- Se as condições retornarem o valor verdadeiro, ou seja, existem duas rainhas na mesma linha ou diagonal, o programa retorna falso, ou seja, a nova posição não é segura. Caso contrário, a posição verificada é segura e a função retorna o valor verdadeiro.

```
// Declara função de verificação privada com retorno booleano
private static boolean verifique(int[] tabuleiro, int coluna) {

    // Laço para verificar linha
    for (int i = 0; i < coluna; i++) {

        // Verifica condição de linha
        if (tabuleiro[i] == tabuleiro[coluna])

            // Se verdadeiro retorna falso ou posição não segura
            return false;

        // Verifica condição de diagonal
        if ((coluna - i) == Math.abs(tabuleiro[coluna] - tabuleiro[i])) {

            // Se verdadeiro retorna falso ou posição não segura
            return false;
        }
    }

    // Retorna verdadeiro se a posição é segura
    return true;
}
```

1. Função Posicione

- A segunda função denominada **<posicione>**, itera sobre o tabuleiro de forma recursiva.
- A função recebe como parâmetros:
 1. O arranjo vetor de inteiros **<tabuleiro>** X_n .
 2. O inteiro **<coluna>** que representa a coluna da vez.
 3. O inteiro **<n>** que representa o máximo de rainhas.
- O laço itera sobre as colunas executando 3 tarefas:
 1. Popula o arranjo **<tabuleiro>**.
 - A posição do arranjo **<tabuleiro>** recebe a linha segura para posicionamento.
 2. Verifica se a posição (coluna, linha) é segura.
 - Executa a função verifique com parâmetros **<tabuleiro>** e **<coluna>** da vez.
 3. Executa **<posicione>** de forma recursiva.
 - Caso a condição retornar verdadeiro, ou seja, posição segura, a função posicione é executada de forma recursiva alterando a coluna da vez para a próxima coluna.

```
// Declare função privada posicione com retorno vazio
private static void posicione(int[] tabuleiro, int coluna, int n) {

    // Condição de parada considerando o tamanho do problema
    if (coluna == n) {

        // Se verdadeiro imprime o tabuleiro
        imprima(tabuleiro);

        // Retorna vazio
        return;
    }

    // Laço para posicionar as rainhas
    for (int i = 0; i < n; i++) {

        // Popula rainha da vez para teste
        tabuleiro[coluna] = i;

        // Condição de verificação da posição
        if (verifique(tabuleiro, coluna)) {

            // Se verdadeiro popula tabuleiro
            posicione(tabuleiro, coluna + 1, n);
        }
    }
}
```

3. Função Imprima

- A terceira função denominada <imprima>, imprime os resultados do algoritmo no formato de um tabuleiro.
- Recebe como parâmetro de entrada o arranjo de inteiros <tabuleiro> da solução do problema.
- Itera sobre o tabuleiro imprimindo ‘R’ para posição com rainha e ‘.’ para posição vazia.
- Os tabuleiros foram separados por uma sequência de hífen ‘- - - - -’.
- A condição de tabuleiro é verificada em cada impressão para evitar impressão de tabuleiros com entradas inválidas.

```
// Declara função imprima estatica privada com retorno vazio
private static void imprima(int[] tabuleiro) {

    // Declara tamanho do tabuleiro
    int tamanho = tabuleiro.length;

    // Condição de Separação
    if(tamanho > 1)

        // Quebra de linha e divisão
        System.out.print("-----");
        System.out.print("\n");

    // Laços para impressão do tabuleiro
    for (int i = 0; i < tamanho; i++) {
        for (int valor : tabuleiro) {

            // Condição de Rainhas
            if (valor == i && tamanho > 1)

                // Imprime rainha
                System.out.print("R\t");

            else if(valor != i && tamanho > 1)

                // Imprime posição vazia
                System.out.print(".\t");
        }

        // Condição de quebra de linha
        if(tamanho > 1)

            // Imprime quebra de linha e separador
            System.out.print("\n");
    }
}
```

4. Função nRainhas

- A função `<n_rainhas>` foi construída para gerar o arranjo de inteiros tabuleiro X_n que será populado pela solução construída na execução da função posicione iniciada na coluna 0.

```
// Declara função estática privada NRainhas
private static void nRainhas(int n) {

    // Declara tabuleiro com dimensão de N_rainhas
    int [] tabuleiro = new int[n];

    // Posiciona rainhas no tabuleiro
    posicione(tabuleiro, 0, n);
}
```

5. Classe inteiroNaoPositivo

- A biblioteca `java.util` foi utilizada para estender a classe `<Exception>`.
- A classe `<inteiroNaoPositivo>` foi construída para gerar exceção se a entrada do usuário não é um número inteiro. O construtor `<this>` recebe o parâmetro `<numero>` com tipo inteiro que será verificado pela extensão da exceção.

```
// Importa biblioteca de utilidades para escanear entrada do usuário
import java.util.*;

// Declara nova classe de exceção
class inteiroNaoPositivo extends Exception {

    // Declara variável
    private int naoPositivo = 0;

    // Construtor com argumento de inteiro
    public inteiroNaoPositivo(int numero) {

        // Declara atributo número
        this.naoPositivo = numero;
    }
}
```

6. Classe NRainhas

- A classe principal NRainhas foi desenvolvida para controlar e executar o programa.
- Composta de uma função principal e uma declaração `<try>` bloco de teste.

1. Função Principal `<main>`:

- Recebe o argumento `<args>` que é uma estrutura de dados arranjo do tipo `String`:
- Declara inteiro `<numeroEntrada>` inicializado com valor 0.
- Declara inteiro `<n>` de entrada inicializado com valor 1.
- Declara objeto entrada do tipo `<Scanner>` para recuperar a escolha do usuário.

2. Bloco de Testes <try>:

- Imprime mensagem requisitando a escolha do usuário para determinar o número de rainhas.
- Aciona módulo próximo inteiro do objeto <entrada> tipo *Scanner*.
- Verifica se número de entrada é negativo:
 - * Se verdadeiro lança exceção *inteiroNaoPositivo*.
 - * Se falso pula.
- Atribui número de entrada a variável *n*.
- Declarações <catch>:
 - * 1º <catch> imprime mensagem de erros se a entrada é negativa e reinicia o programa.
 - * 2º <catch> imprime mensagem de erros se a entrada é diferente de inteiro e reinicia o programa.
- A declaração <finally> permite executar código depois de <try>
 - * Executa a função <n_rainhas> com parâmetro <n>.

```
// Declara classe publica principal
public class NRainhas {

    // Declara classe publica estática com retorno vazio
    public static void main(String[] args) {

        // Declara variáveis
        int numeroEntrada = 0; int n = 1;

        // Inicializa escaneador entrada
        Scanner entrada = new Scanner(System.in);

        // Teste de entrada
        try {

            // Solicita entrada do usuário
            System.out.println("Por favor insira a quantidade de Rainhas:");

            // Declara entrada
            numeroEntrada = entrada.nextInt();

            // Condição de negativo
            if (numeroEntrada < 0) {

                // Lança exceção de inteiro não negativo
                throw new inteiroNaoPositivo(numeroEntrada);
            }

            // Caso contrário declara variável n com entrada do usuário
            n = numeroEntrada;

            // Captura exceção de não positivo
        } catch (inteiroNaoPositivo e) {

            // Mensagem para usuário
            System.out.println("Entrada de inteiro negativo recusada!\n");
            main(args);
        }
    }
}
```

```

// Captura exceção de entrada diferente de inteiro
} catch (InputMismatchException e) {

    // Mensagem de entrada diferente de inteiro
    System.out.println("Entrada diferente de inteiro recusada!\n");
    main(args);

// Retorno dos testes
} finally {

    // Chama a função nRainhas com parâmetro n
    nRainhas(n);
}
}

```

Exemplo

- Um exemplo da execução do programa é a resolução do problema com o parâmetro <4> para indicar a resolução do problema com 4 rainhas em um tabuleiro 4 x 4.
- A JVM do **replit** permite executar o programa no sítio:
 - <https://replit.com>
- O Programa pode ser compilado e executado de 2 formas:
 1. Acionando o botão Run do menu superior central.
 2. Digitando os seguintes comando no Shell da janela principal da direita:
 - <javac Main.java>
 - <java Main>
- Saída do Programa para o exemplo com 4 rainhas no tabuleiro 4 x 4:

```

Por favor insira a quantidade de Rainhas:
4
-----
.   .   R   .
R   .   .   .
.   .   .   R
.   R   .   .
-----
.   R   .   .
.   .   .   R
R   .   .   .
.   .   R   .

```

Referências

- <https://homepages.dcc.ufmg.br/bigonha>
- www.tutorialspoint.com/java
- [You Tube](#)