Algoritmia e Estrutura de Dados

Trabalho Prático 1
Problema das *N Rainhas* com estrutura de dados *Stack*

Discente:
João Oliveira

Docente:
Doutor Tiago Candeias

24 de Maio de 2018

Conteúdo

1 Introdução

Este trabalho prático tem como objectivo resolver o problema das *N-Rainhas* utilizando a estrutura de dados *Stack*.

1.1 Descrição do problema

O problema das *N-Rainhas* pertence à classe de problemas de preenchimento de *puzzles* baseado em restrições.

Dado um tabuleiro de xadrez com dimensão *N x N*, é pretendido encontrar o conjunto de todas as soluções possíveis.

Uma solução consiste em encontrar a posição *N rainhas* sobre o tabuleiro *N x N*, em que nenhuma destas se ataque. As rainhas movem-se em *linhas*, *colunas* e *diagonais*.

A figura seguinte, mostra uma possível solução para a instância do problema das *N-Rainhas*, quando $\boldsymbol{N = 4}$.

Observamos que como temos de ter 4 rainhas em cada coluna, para representar uma solução podemos omitir a coluna e apenas representar a linha. Por exemplo, no diagrama anterior podemos representar a solução como:

[3, 1, 4, 2]

1.2 Abordagem ao problema

O problema foi abordado utilizando uma stack, e utilizando backtracking para encontrar todas as soluções possíveis num tabuleiro *NxN* com *N* rainhas.

Para tal, seguiu-se a ideia de começar pela coluna mais à esquerda, e pela primeira linha, seguindo para a próxima coluna, testando cada linha até ser possível posicionar e avançar para a coluna seguinte, ou então voltar uma

coluna atrás (caso em que chegou à linha N e não foi possível posicionar) e refazer os mesmos passos.

Sempre que a coluna mais à esquerda tenha sido totalmente testada, marcamos a mesma como ***colDone***, significando que não adianta fazer mais backtracking quando se voltar a essa coluna (isto apenas acontece na coluna mais à esquerda que esteja por completar os testes).

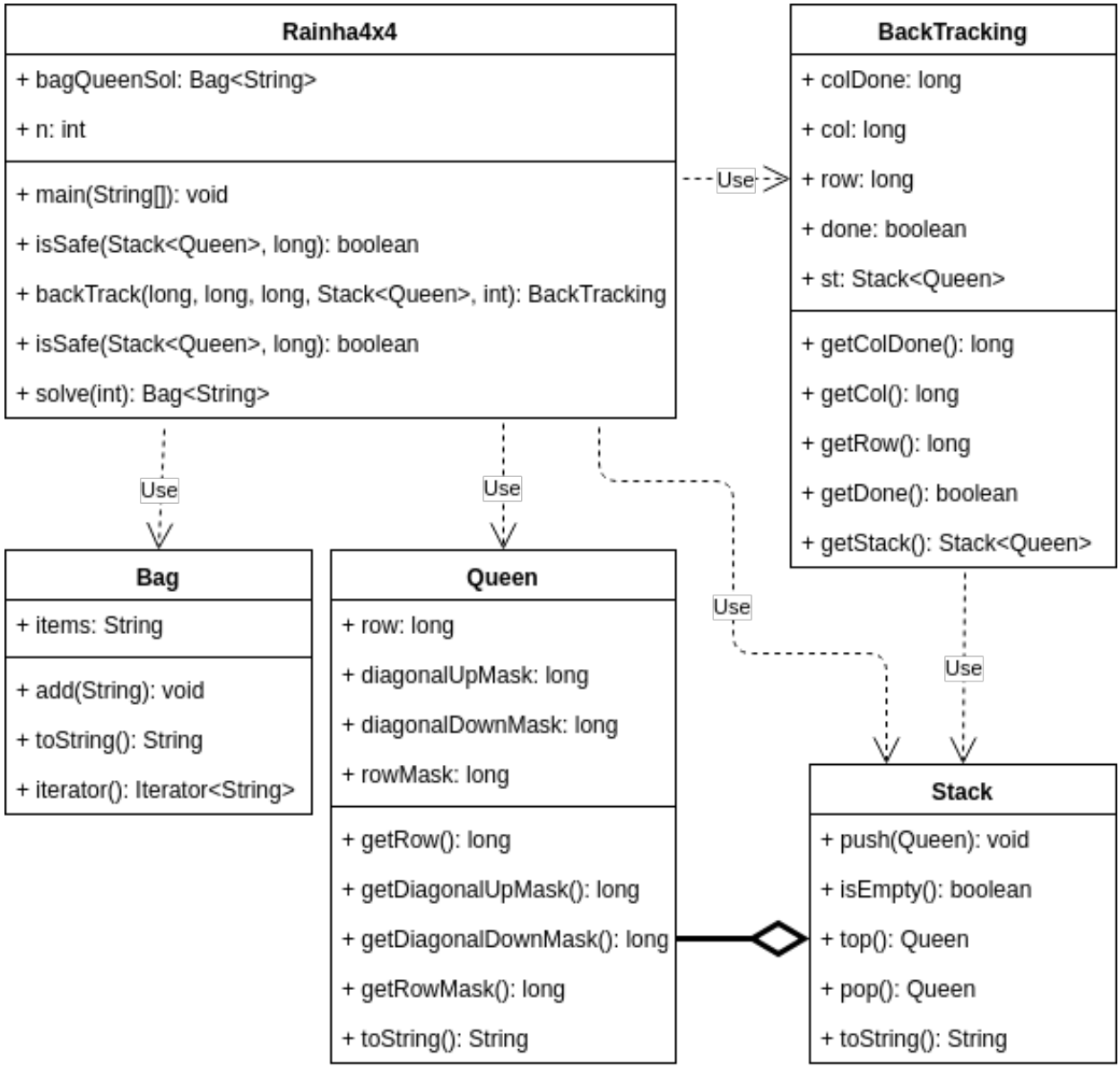
Desta forma, utilizando a variável *colDone*, evitamos testes desnecessários.

Na primeira versão do algoritmo, os testes de posicionamento obrigavam a ciclos extra para verificar contra todos os elementos da *Stack* se era seguro posicionar a Rainha.

Optou-se então por uma aproximação mais *matemática*, utilizando máscaras de bits, para marcar, e actualizar cumulativamente (isto é, por cada elemento de Stack) as posições atacáveis, em linhas ***Horizontais***, ***Diagonais para cima*** e ***Diagonais para baixo***, relativamente à posição de cada rainha, evitando assim os ciclos desnecessários para verificação das posições seguras.

1.3 Diagrama de classes

Diagrama de classes, com a análise realizada.



1.4 Pseudo-código do algoritmo desenvolvido

A listagem do pseudo-código de cada método implementado:

1.4.1 Método backTrack()

```
backTrack(colDone, col, row, st, n)
 while colDone < (col - 1)
  col--
  tmp = st.pop()
  row = tmp.getRow() + 1

  if row <= n
   break

 return new BackTracking(colDone, col, row, (row > n ? true
     : false), st)
```

1.4.2 Método calcMask()

```
calcMask(col, row, st)
 rowMask = 1 << (row - 1)
 diagUpMask = rowMask << 1
 diagDownMask = rowMask >>> 1

 if col > 1
  tmp = st.top()
  diagUpMask |= tmp.getDiagonalUpMask() << 1
  diagDownMask |= tmp.getDiagonalDownMask() >>> 1
  rowMask |= tmp.getRowMask()

 return new Queen(row, rowMask, diagUpMask, diagDownMask)
```

1.4.3 Método isSafe()

```
isSafe(st, row)
 if st.isEmpty()
  return true

 tmp = st.top()
 testRow = tmp.getRowMask() | tmp.getDiagonalUpMask() | tmp
    .getDiagonalDownMask()
 thisRow = 1 << (row - 1)

 if (~testRow & thisRow) == thisRow
  return true

 return false
```

1.4.4 Método solve()

```
solve(n)
 bagQueens = new Bag<String>()
 s = new Stack<Queen>()

 if n >= 4
  done = false
  colDone =  0
  col = row = 1
  tmp = calcMask(col, row, s)
  initTime = System.nanoTime()

  while !done
   if isSafe(s, row)
    tmp = calcMask(col, row, s)
    s.push(tmp)

    if col < n
     col = col + 1
     row = 1
    else
     bagQueens.add( s.toString() )

     tmp = s.pop()
     row = tmp.getRow() + 1

     if row > n
      if colDone == (col - 1)
       done = true
      else
       bkt = backTrack(colDone, col, row, s)
       s = bkt.getStack()
       col = bkt.getCol()
       row = bkt.getRow()
       done = bkt.getDone()

     if row == n E colDone == (col - 1)
      colDone = colDone + 1
   else
    if row < n
     row = row + 1
    else
     if colDone < (col - 1)
      bkt = backTrack(colDone, col, row, s)
      s = bkt.getStack()
      col = bkt.getCol()
      row = bkt.getRow()
      done = bkt.getDone()
```

```
    else
     done = true

 endTime = System.nanoTime()

 write "Solucoes=" + bagQueens.size() + " :: => Tempo=" +
    ((endTime - initTime) / 1000000000) + "s"
else
 write "Minimum is 4!!"

return bagQueens
```

2 Enquadramento

O trabalho descrito neste relatório foi realizado em linguagem Java, implementando testes unitários do mesmo.

2.1 Motivação

A principal motivação para a realização deste trabalho, resulta da importância em resolver de forma eficiente, o problema das N Rainhas e demonstrar os conhecimentos alcançados na disciplina de Algoritmia e Estrutura de Dados.

2.2 Objectivos

O objectivo deste trabalho prático é a criação de um algoritmo que consiga resolver de modo mais eficiente o problema das *N Rainhas*.

3 Conclusões

Foi possível melhorar a eficiência do algoritmo desde a primeira versão até esta última, ao utilizar matemática com manipulação de bits, resultando nos seguintes valores médios entre ambas as versões:

Versão inicial			Versão final		
N	Soluções	Tempo (s)	N	Soluções	Tempo (s)
4	2	0.001373873	4	2	7.37E-05
5	10	0.002581748	5	10	2.35E-04
6	4	0.00258968	6	4	5.95E-04
7	40	0.004265591	7	40	0.001474831
8	92	0.015334179	8	92	0.003521791
9	352	0.013674552	9	352	0.011897093
10	724	0.033703422	10	724	0.017972943
11	2,680	0.085065652	11	2,680	0.059050854
12	14,200	0.313915151	12	14,200	0.156339025
13	73,712	1.313999855	13	73,712	0.391390124
14	365,596	7.530787424	14	365,596	3.006554531
15	2,279,184	47.51982108	15	2,279,184	20.02295484
16	14,772,512	344.9248243	16	14,772,512	117.829401
17	95,815,104	2607.882779	17	95,815,104	872.035982
			18	666,090,624	6697.386393

4 Anexos

Ficheiros ***relatorio.pdf*** e os ficheiros ***BackTracking.java***, ***Bag.java***, ***Queen.java***, ***QueenSolver.java***, ***Stack.java*** e ***TestQueenSolver.java*** compactados num ficheiro ***Rainha4x4 (Joao Oliveira, a21501152).zip***, dentro da pasta ***src***.

Não existem quaisquer códigos ou listagens adicionais.