IFS3101 - Kecerdasan Buatan

Week/session	**:**	**10/ 02**
Topic	**:**	**N-Queen Problem with Forward Checking Algorithm**
Due	**:**	**Rabu, 12 April 2016 Pukul 21.30 WIB (ecourse)**

Pada praktikum kali ini kita akan menyelesaikan permasalahan n-queen dengan menggunakan algoritma forward checking.

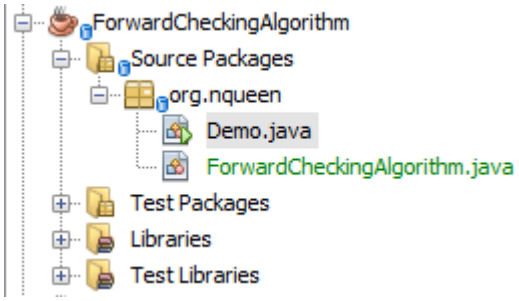
Algoritma ini hampir mirip dengan algoritma backtracking, yang membedakan antara kedua allgoritma ini hanya pada bagian pengecekannya saja.

Algoritma ini berkerja dengan menaruh tanda pada langkah yang tidak dapat dilalui oleh queen berikutnya. Ketika sebuah queen ditempatkan, maka algoritma ini akan menaruh tanda secara diagonal dan horizontal untuk langkah yang tidak dapat dilalui berikutnya. Sehingga pengecekan hanya dilakukan dengan melihat apakah tanda langkah tersebut ada pada baris dan kolom dimana queen akan ditempatkan berikutnya.

Misal n=4:

		X	X
	Q	X	X
	X	X	
Q	X	X	X

Buatlah project Java pada netbeans dan beri nama proyek **ForwardCheckingAlgorithm**. Kemudian tambahkan package seperti berikut pada proyek anda.



Setiap kelas memiliki fungsinya masing-masing yang dapat dijabarkan sebagai berikut:

- Demo

  Berfungsi sebagai driver utntuk menjalankan fungsi pencarian rute pada backtrack algorithm.

- Forward Checking Algorithm

Kelas ini berfungsi untuk melakukan pencarian sesuai dengan algoritma pencarian forward checking.

Perhatikan snippet code dari setiap kelas yang diuraikan dibawah ini

1. Kelas Forward Checking Algorithm

```
package org.nqueen;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class ForwardCheckingAlgorithm {
    int nQueen;
    int[] hasil;
    int[][] matrixForward;
    int iterator = 1;
    static int jumlahIterasi, jumlahSolusi, jumlahSimpulMati,
jumlahAnak;
    static List<String> listSimpulMati, listSimpulSolusi;
    static String judul;

    public ForwardCheckingAlgorithm(int nQueen) {
        this.nQueen = nQueen;
        this.hasil = new int[nQueen + 2];
        this.matrixForward = new int[nQueen + 2][nQueen + 2];
        jumlahIterasi = jumlahSolusi = jumlahSimpulMati =
jumlahAnak = 0;
        listSimpulMati = new ArrayList<>();
        listSimpulSolusi = new ArrayList<>();
        judul = "Forward Checking N = "+nQueen+" Information";
    }

    boolean check(int i) {
        int baris = hasil[i];
        int kolom = i;
        if (hasil[i] > 0 && hasil[i] <= nQueen) {
            jumlahIterasi++;
            if (matrixForward[baris][kolom] == 0) {
                return true;
            } else {
                return false;
            }
        } else {
            jumlahIterasi++;
            return false;
        }
    }

    void putTrace(int index) {
        int baris = hasil[index];
        int kolom = index;

        for (int i = index + 1; i <= nQueen; i++) {
            jumlahIterasi++;
```

```
                matrixForward[baris][i]++;
                if ((baris + Math.abs(index - i)) <= nQueen) {
                    matrixForward[baris    +    Math.abs(index    -
i)][i]++;
                }
                if ((baris - Math.abs(index - i)) >= 1) {
                    matrixForward[baris    -    Math.abs(index    -
i)][i]++;
                }
            }
        }

        void deleteTrace(int index) {
            int baris = hasil[index];
            int kolom = index;

            for (int i = index + 1; i <= nQueen; i++) {
                jumlahIterasi++;
                matrixForward[baris][i]--;
                if ((baris + Math.abs(index - i)) <= nQueen) {
                    matrixForward[baris + Math.abs(index - i)][i]--
;
                }
                if ((baris - Math.abs(index - i)) >= 1) {
                    matrixForward[baris - Math.abs(index - i)][i]--
;
                }
            }
        }

        void printTrace() {
            for (int i = nQueen; i >= 1; i--) {
                for (int j = 1; j <= nQueen; j++) {
                    System.out.printf("%d ", matrixForward[i][j]);
                }
                System.out.printf("\n");
            }
        }

        void takePlace(int i) {
            if (i == 1) {
                if (hasil[i] == 0) {
                    jumlahIterasi++;
                    hasil[i] = 1;
                    putTrace(i);
                } else {
                    jumlahIterasi++;
                    deleteTrace(i);
                    hasil[i] += 1;
                    putTrace(i);
                }
                return;
            } else if (i > 1 && i <= nQueen) {
                if (hasil[i] == 0) {
                    for (int j = 1; j <= nQueen; j++) {
```

```
                    jumlahIterasi++;
                    hasil[i] = j;
                    if (check(i)) {
                        putTrace(i);
                        return;
                    }
                }
                hasil[i] = nQueen + 1;
                return;
            } else {
                deleteTrace(i);
                for (int j = hasil[i] + 1; j <= nQueen; j++) {
                    jumlahIterasi++;
                    hasil[i] = j;
                    if (check(i)) {
                        putTrace(i);
                        return;
                    }
                }
                hasil[i] = nQueen + 1;
                return;
            }
        } else {
            jumlahIterasi++;
            hasil[i] = nQueen + 1;
            return;
        }
    }

    void cleanRoute(int i) {
        for (int j = i; j <= nQueen; j++) {
            jumlahIterasi++;
            hasil[j] = 0;
        }
    }

    public int[] routeUseForwardCheck() {
        jumlahAnak = 0;
        if (iterator > nQueen) {
            jumlahIterasi++;
            iterator = nQueen;
        }

        while (iterator <= nQueen) {
            if (iterator == 1) {
                takePlace(iterator);
                cleanRoute(iterator + 1);
                iterator++;
                jumlahIterasi++;
            } else if (iterator <= nQueen) {
                if (hasil[iterator] > nQueen) {
                    cleanRoute(iterator);
                    iterator--;
                    listSimpulMati.add(toStringFromListInt());
                    jumlahSimpulMati++;
```

```
                    jumlahIterasi++;
                    takePlace(iterator);
                } else if (hasil[iterator] <= nQueen) {
                    takePlace(iterator);
                }
                if   (check(iterator)   &&   hasil[iterator]   <=
nQueen) {
                    cleanRoute(iterator + 1);
                    iterator++;
                    jumlahIterasi++;
                } else {
                    iterator--;
                    listSimpulMati.add(toStringFromListInt());
                    jumlahSimpulMati++;
                    jumlahIterasi++;
                }
            }
        }

        if (hasil[1] > nQueen) {
            return null;
        } else {
            jumlahSolusi++;
            listSimpulSolusi.add(toStringFromListInt());
            jumlahAnak = jumlahSolusi + jumlahSimpulMati;

            return hasil;
        }
    }

    String toStringFromListInt() {
        String hasilString = new String();
        for (int i = 1; i <= nQueen; i++) {
            if (hasil[i] > nQueen || hasil[i] == 0) {
                hasilString += 'x';
            } else {
                hasilString += hasil[i];
            }
        }

        return hasilString;
    }

    public static String getInformation() {
        String info = new String();

        info = judul + "\n"
                + "========================================\n"
                + "Jumlah Iterasi       : " + jumlahIterasi +
"\n"
                + "Jumlah Solusi      : " + jumlahSolusi + "\n"
                + "Jumlah Simpul Mati : " + jumlahSimpulMati +
"\n"
                + "Jumlah Anak        : " + jumlahAnak + "\n"
                + "========================================\n";
```

```
            info += "Simpul Mati : \n";
            for (String simpulMati : listSimpulMati) {
                info += simpulMati + "\n";
            }

            info += "=====================================\n";
            info += "Solusi : \n";

            for (String solusiString : listSimpulSolusi) {
                info += solusiString + "\n";
            }

            return info;
        }
    }
```

Pada kelas tersebut memiliki beberapa atribut:

1. nQueen

   Atribut untuk menyimpan banyak n-queen dari papan catur, hal ini karena kelas ini bersifat dinamis.

2. Hasil

   Atribut ini berfungsi untuk menyimpan hasil data yang ada, setiap perubahan akan terjadi pada array ini.

3. Matrix Forward

   Atribut ini berfungsi untuk menyimpan lanhkah yang telah dilarang untuk ditempati oleh queen berikutnya. Matriks ini berkaitan dengan array hasil sebagai penaruhan langkah yang tidak dapat diambil sekaligus untuk mempermudah pengecekan nantinya.

4. Iterator

   Atribut ini berfungsi sebagai iterator untuk menunjukkan berada dimana kolom mana yang sedang dikerjakan pada suatu proses.

5. Jumlah iterasi, jumlah simpul mati, jumlah solusi dan jumlah anak.

   Berfungsi untuk menyimpan masing-masing informasi yang dibutuhkan.

6. List Simpul Mati dan List Simpul Solusi

   Berfungsi menyimpan seluruh solusi, dan simpul mati.

Pada Kelas ini juga terdapat beberapa method yang berguna dalam pencarian secara Forward Checking:

1. Check

   Berfungsi untuk mencek apakah langkah yang diambil saat ini sesuai atau tidak (melanggar aturan atau tidak). Fungsi ini hanya mengecek pada array matriks forward, apakah sudah ditandai atau belum.

2. Put Trace

   Berfungsi untuk menaruh tanda pada matriks forward, pada bagian ini matriks forward akan di tambah satu setiap kolom yang dilaluinya.

3. Delete Trace

   Berfungsi untuk menghapus tanda pada matriks forward, pada bagian ini matriks forward yang telah ditandai sebelumnya denga ditambah 1, maka akan dilakukan pengurangan 1 serhadap langkah yang telah diambil sebelumnya.

4. Print Trace

   Berfungsi untuk menggambarkan matriks forward untuk melihat bagaimana perubahan setiap pencarian solusi yang dilakukan.

5. Take Place

   Befungsi untuk mencari langkah yang sah secara perbaris. Fungsi ini berkerja secara iterasi sekaligus melakukan pengecekan dengan menggunakan fungsi check, bila tidak fungsi check mengembalikan nilai false maka iterasi akan dilanjutkan, dan bila mengembalikan nilai true maka langkah tersebut akan disimpan.

6. Route Use BackTrack

   Berfungsi untuk melakukan perjalanan pencarian secara perkolom, disetiap kolom fungsi ini akan memanggil fungsi Take Place untuk mencari langkah yang dapat ditempati pada kolom tersebut dan bila tidak ditemukan langkah yang tepat pada kolom tersebut maka fungsi ini akan mencari kembali ke belakang untuk mencari rute yang tepat.

7. Clean Route

   Berfungsi untuk membersihkan array hasil dari rute salah yang telah diambil sebelumnya agar tidak mengganggu pencarian selanjutnya.

8. To String From List Int

   Berfungsi untuk mengubah list integer menjadi suatu string. Hal ini untuk mempermudah penyimpanan list simpul mati dan list simpul solusi.

9. Get Information

   Berfungsi untuk mengembalikan suatu string berisikan informasi yang telah didapatkan secara terformat seperti yang diinginkan.

2. Kelas Demo

```
package org.nqueen;

public class Demo {
    public static void main(String[] args) {
        ForwardCheckingAlgorithm   forwardCheckingAlgorithm   =
```

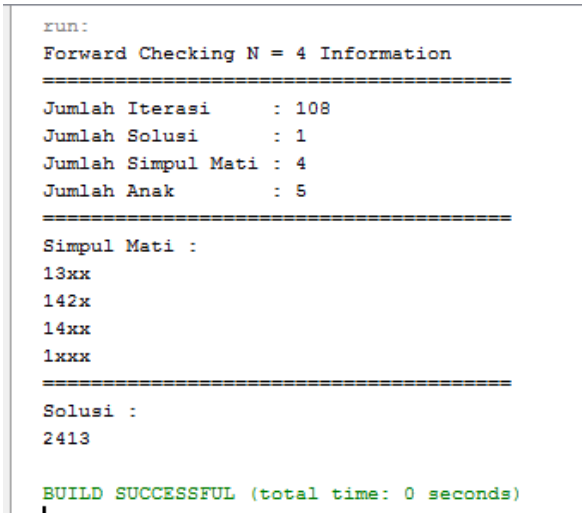
```
    new ForwardCheckingAlgorithm(4);

            int[]          solusiForwardCheckingAlgorithm          =
    forwardCheckingAlgorithm.routeUseForwardCheck();


    System.out.println(ForwardCheckingAlgorithm.getInformation());
        }
    }
```

Pada kelas demo, dilakukan pemanggilan terhadap kelas Forward Checking, sehingga dapat dilakukan pencarian rute sesuai dengan metode forward checking.

Pastikan tidak ada error pada setiap line kode, kemudian jalankan aplikasi maka akan muncul hasil sebagai berikut :



```
run:
Forward Checking N = 4 Information
=====================================
Jumlah Iterasi      : 108
Jumlah Solusi       : 1
Jumlah Simpul Mati : 4
Jumlah Anak         : 5
=====================================
Simpul Mati :
13xx
142x
14xx
1xxx
=====================================
Solusi :
2413

BUILD SUCCESSFUL (total time: 0 seconds)
```

Tugas:

1. Ubah jumlah queen pada class Demo, menjadi 5,6,7, dan 8!.
2. Apakah ada kemungkinan jumlah solusi yang dihasilkan lebih dari satu?
3. Apakah perbedaan utama antara back track algorithm dengan forward checking algorithm? Untuk N-Queen Problem, mana yang lebih efektif antara back track algorithm dengan forward checking algorithm?
4. Eksplorasi Forward Checking Algoritma dan buat dalam laporan singkat.

Selamat mengerjakan & have fun ☺