TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA

Penyelesaian Permainan Queens LinkedIn Menggunakan Algoritma Brute Force

D

I

S

U

S

U

N

O

L

E

H

Ishak Palentino Napitupulu

13524022

K02

Program Studi Teknik Informatika

STEI

ITB 2026

1. **Algoritma Brute Force yang digunakan**

   Program akan mencari semua kemungkinan konfigurasi penempatan queen pada board kemudian akan mengeluarkan solusi ketika konfigurasi tepat

   Sebelumnya, saya sebagai penulis bingung arti dari "brute force murni", apakah bisa menggunakan constraint atau tidak. Oleh karena itu saya menyediakan 2 algoritma yang menggunakan constraint dan yang tidak menggunakan constraint, yaitu

   a. Constrained Brute Force (mainSolution.py)

      Memiliki kompleksitas O(N!) atau O(N!(N^2) jika ikut algoritma validasi

      Langkah algoritma:

      1. Board diinisialisasi dengan menempatkan semua queen secara diagonal dari kiri atas sampai kanan bawah sehingga tidak ada 2 atau lebih queen yang memiliki baris atau kolom yang sama
      2. Enumerasi semua permutasi dengan operasi swap kolom antar 2 queen dengan heap Algorithm. Operasi swap dilakukan dengan procedure swa()
      3. Validasi tiap permutasi, untuk semua konfigruasi, akan dilakukan proses validasi untuk mengecek apakah konfigurasi tersebut sudah tepat

         Karena di awal sudah di constrained bahwa tidak ada 2 atau lebih queen yang 1 row/col, maka proses validasi hanya mengecek apakah tiap region memiliki tepat 1 queen atau tidak dan secara diagonal tidak berdekatan
      4. Proses enumerasi dan validasi akan terus dilakukan sampai solusi ditemukan atau semua konfigurasi (N!) sudah dicoba

   b. Blind Brute Force (solution.py)

      Memiliki kompleksitas O(N^(2N) atau O((N^2)N^(2N)) jika ikut algoritma validasi

      Langkah algoritma:

      1. Semua sel board diinisasi kosong, tidak ada pembatasan untuk tiap sel/kolom
      2. Untuk setiap enumerasi, program akan menempatkan setiap queen pada board mengikuti sequence odometer
      3. Untuk setiap enumerasi, akan dilakukan validasi, karena tidak ada dilakukan constraint, maka proses validasi akan mengecek tiap row/ tidak ada 2 atau lebih queen, tiap region/color hanya punya tepat 1 queen, dan tidak ada 2 queen yang secara diagonal berdekatan

4. Proses enumerasi dan validasi akan terus dilakukan sampa solusi ditemukan atau semua konfigurasi (O(N^(2N)) telah dicoba

2. Source Code (Python)

1. Constrained Brute Force

```
1.from matrix import Matrix
3. from point import Point
4. import math
5. import time
6.
7. interval = 10000
8. iteration = 0
9.
10.start = 0
11.end = 0
12.
13.def solve(board:Matrix):
14.    newBoard = copyBoard(board)
15.    for i in range(0,newBoard.rows):
16.        for j in range(0,newBoard.cols):
17.            if (i == j):
18.                newBoard.setElmnt(i,j,"#")
19.            else:
20.                newBoard.setElmnt(i,j,"0")
21.    M = bruteForce(board, newBoard, board.cols)
22.
23.    if M is None:
24.        print(f"TIDAK ADA solusi yang ditemukan, max iteration:
   {iteration}\n")
25.        return None
26.    else:
27.        return M
28.
29.def bruteForce(board:Matrix, copyBoard:Matrix, k):
30.    global iteration
31.    if (k == 1):
32.        iteration += 1
33.        return validate(board, copyBoard)
34.
35.    M = bruteForce(board, copyBoard, k-1)
36.    if  M is not None:
37.        return M
38.
39.    for i in range(0,k-1):
40.        if k % 2 == 0:
41.            swap(copyBoard, i, k-1)
42.        else:
```

```
            swap(copyBoard, 0, k-1)
        M = bruteForce(board, copyBoard, k-1)
        if M is not None:
            return M
    return None

def swap(board:Matrix, colA, colB):
    for i in range(0, board.rows):
        if board.getElmnt(i,colA) == "#":
            rowA = i
    for i in range(0,board.rows):
        if board.getElmnt(i,colB) == "#":
            rowB = i

    board.setElmnt(rowA, colA, "0")
    board.setElmnt(rowB, colB, "0")
    board.setElmnt(rowB, colA , "#")
    board.setElmnt(rowA, colB, "#")

def validate(board:Matrix, newBoard:Matrix):
    global end
    arrPoints = []
    count = 0
    colorCheck = 1
    digonalCheck = 1

    checkBoard = copyBoard(board)
    for i in range(board.rows):
        for j in range(board.cols):
            if newBoard.getElmnt(i,j) == "#":
                p = Point(i,j,board.getElmnt(i,j))
                arrPoints.append(p)
                checkBoard.setElmnt(i,j,"#")
                count += 1

    #check color similarities
    check  = []
    for i in range(0,count):
        if arrPoints[i].color in check:
            colorCheck = 0
        else:
            check.append(arrPoints[i].color)

    #check diagonal
    for i in range(board.rows):
        for j in range(board.cols):
            if checkBoard.getElmnt(i,j) == "#":
```

```
                if (i+1) <= board.rows-1 and (j+1) <= board.cols-1 and
   checkBoard.getElmnt(i+1,j+1) == "#":
                    digonalCheck = 0
                    break
                if (i+1) <= board.rows-1 and (j-1) >= 0 and
   checkBoard.getElmnt(i+1,j-1) == "#":
                    digonalCheck = 0
                    break
        if not digonalCheck:
            break

    if (colorCheck and digonalCheck):
            print("SUKSES pada ")
            end = time.perf_counter()
            displayBoard(checkBoard)
            return checkBoard
        else:
            if (iteration % interval == 0):
                end = time.perf_counter()
                displayBoard(checkBoard)
            return None

    def copyBoard(board:Matrix):
        copyBoard = Matrix(board.rows, board.cols)
        for i in range(copyBoard.rows):
            for j in range(copyBoard.cols):
                copyBoard.setElmnt(i,j, board.getElmnt(i,j))
        return copyBoard

    def displayBoard(board:Matrix):
        print("iterasi ke - " + str(iteration) + "\n")
        for i in range(board.rows):
            for j in range(board.cols):
                print(board.getElmnt(i,j), end=" ")
            print()
        print()

    def countComplexity(n):
        return math.factorial(n)

    start = 0
    end = 0


```

2. Blind Brute Force

```
1. from matrix import Matrix
2. from point import Point
3. import math
4. import time
5.
6. interval = 1000
7. iteration = 0
8.
9. def solve(board:Matrix):
10.     newBoard = Matrix(board.rows, board.cols)
11.     for i in range(newBoard.rows):
12.         for j in range(newBoard.cols):
13.             newBoard.setElmnt(i, j, "0")
14.
15.     M = bruteForce(board, newBoard, board.cols)
16.
17.     if M is None:
18.         print(f"TIDAK ADA solusi yang ditemukan, max iteration:
    {iteration}\n")
19.         return None
20.     else:
21.         return M
22.
23.def bruteForce(board: Matrix, newBoard: Matrix, N: int):
24.     global iteration
25.     cells = newBoard.rows * newBoard.cols
26.
27.     if N > cells:
28.         return None
29.
30.     pos = [i for i in range(N)]
31.
32.     used = [False] * cells
33.     for p in pos:
34.         used[p] = True
35.
36.     while True:
37.
38.         clearnewBoard(newBoard)
39.
40.         for p in pos:
41.             row = p // newBoard.cols
42.             col = p % newBoard.cols
43.             newBoard.setElmnt(row, col, "#")
44.
```

```
        iteration += 1
        M = validate(board, newBoard)
        if M is not None:
            return M

        if not increment_positions(pos, used, N, cells):
            break

    return None

def increment_positions(pos, used, N, cells):
    k = N - 1
    while k >= 0:
        used[pos[k]] = False

        nxt = pos[k] + 1
        while nxt < cells and used[nxt]:
            nxt += 1

        if nxt < cells:
            pos[k] = nxt
            used[nxt] = True

            for t in range(k + 1, N):
                x = 0
                while x < cells and used[x]:
                    x += 1
                if x >= cells:
                    return False
                pos[t] = x
                used[x] = True

            return True

        else:
            k -= 1

    return False

def clearnewBoard(newBoard: Matrix):
    for i in range(newBoard.rows):
        for j in range(newBoard.cols):
            newBoard.setElmnt(i, j, "0")


def validate(board:Matrix, newBoard:Matrix):
    global end
    arrPoints = []
```

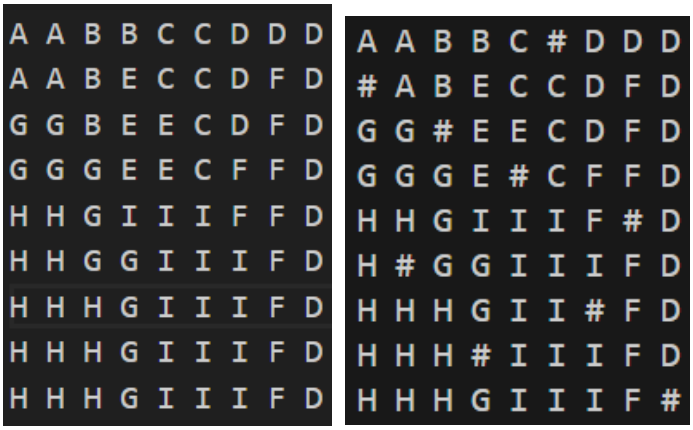
```
    count = 0
    rowColCheck = 1
    colorCheck = 1
    digonalCheck = 1

    checkBoard = copyBoard(board)
    for i in range(board.rows):
        for j in range(board.cols):
                if newBoard.getElmnt(i,j) == "#":
                    p = Point(i,j,board.getElmnt(i,j))
                    arrPoints.append(p)
                    checkBoard.setElmnt(i,j,"#")
                    count += 1

        #check row-col
        for i in range(board.rows):
            thisRowExist = 0
            for j in range(board.cols):
                if checkBoard.getElmnt(i,j) == "#":
                    thisRowExist += 1
                if thisRowExist > 1:
                    rowColCheck = 0
                    break
            if rowColCheck == 0:
                break
        for i in range(board.cols):
            thisColExist = 0
            for j in range(board.rows):
                if checkBoard.getElmnt(j,i) == "#":
                    thisColExist += 1
                if thisColExist > 1:
                    rowColCheck = 0
                    break
            if rowColCheck == 0:
                break

        #check color similarities
        check  = []
        for i in range(0,count):
            if arrPoints[i].color in check:
                colorCheck = 0
            else:
                check.append(arrPoints[i].color)

        #check diagonal
        for i in range(board.rows):
            for j in range(board.cols):
                if checkBoard.getElmnt(i,j) == "#":
```

```
                    if (i+1) <= board.rows-1 and (j+1) <= board.cols-1 and checkBoard.getElmnt(i+1,j+1) == "#":
                        digonalCheck = 0
                        break
                    if (i+1) <= board.rows-1 and (j-1) >= 0 and checkBoard.getElmnt(i+1,j-1) == "#":
                        digonalCheck = 0
                        break
            if not digonalCheck:
                break

        if (rowColCheck and colorCheck and digonalCheck):
            print("SUKSES pada ")
            end = time.perf_counter()
            displayBoard(checkBoard)
            return checkBoard
        else:
            if (iteration % interval == 0):
                end = time.perf_counter()
                displayBoard(checkBoard)
            return None

    def displayBoard(board:Matrix):
        print("iterasi ke - " + str(iteration) + "\n")
        for i in range(board.rows):
            for j in range(board.cols):
                print(board.getElmnt(i,j), end=" ")
            print()
        print()


    def copyBoard(board:Matrix):
        copyBoard = Matrix(board.rows, board.cols)
        for i in range(copyBoard.rows):
            for j in range(copyBoard.cols):
                copyBoard.setElmnt(i,j, board.getElmnt(i,j))
        return copyBoard

```

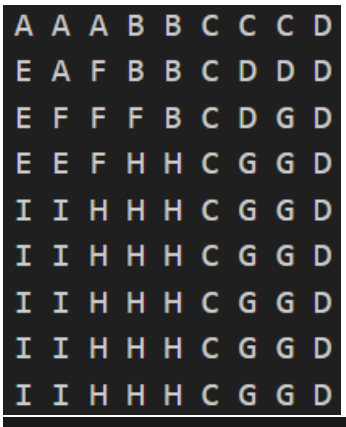
3. Hasil Eksekusi

1. Eksekusi 1

2. Eksekusi 2



3. Eksekusi 3




```
TIDAK ADA solusi yang ditemukan, max iteration: 362880

Waktu pencarian solusi: 5394.029999995837 ms
Banyak kasus yang ditinjau: 362880
Apakah anda ingin menyimpan solusi? (Ya/Tidak)
```

4. Eksekusi 4

```
A A B B B C C D D
A A B E B C D D D
F F B E E C D G D
F F F E E C G G D
H H F I I I G G G
H H F F I I I G G
H H H F I I I G G
H H H F I I I G G
H H H F I I I G G
```

```
TIDAK ADA solusi yang ditemukan, max iteration: 362880

Waktu pencarian solusi: 6853.824599995278 ms
Banyak kasus yang ditinjau: 362880
Apakah anda ingin menyimpan solusi? (Ya/Tidak)
```

5. Eksekusi 5

```
AAABBCCCD
ABBBBCECD
ABBBDCECD
AAABDCCCD
BBBBDDDDD
FGGGDDHDD
FGIGDDHDD
FGIGDDHDD
FGGGDDHHH
```

```
A A A B B C C # D
A B B B # C E C D
A B B B D C # C D
A # A B D C C C D
B B B B D # D D D
F G G # D D H D D
# G I G D D H D D
F G # G D D H D D
F G G G D D H H #

Waktu pencarian solusi: 534.0001000004122 ms
Banyak kasus yang ditinjau: 36201
```

4. Link Repository

https://github.com/Ishakpln/TUCIL1_13524022/tree/main

5. Pernyataan

Tugas ini disusun sepenuhnya tanpa bantuan kecerdasan buatan (Generative AI), melainkan hasil pemikiran dan analisis mandiri.

Ishak Palentino Napitupulu

No	Poin	Ya	Tidak
1	Program berhasil di kompilasi tanpa kesalahan	✓	
2	Program berhasil di jalankan	✓	
3	Solusi yang diberikan program benar dan mematuhi aturan permainan	✓	
4	Program dapat membaca masukan berkas .txt serta menyimpan solusi dalam berkas .txt	✓	
5	Program memiliki Graphical User Interface (GUI)		✓
6	Program dapat menyimpan solusi dalam bentuk file gambar		✓