

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 konfigurasi, 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 $O((N^2)^{2N})$ 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 sampai solusi ditemukan atau semua konfigurasi ($O(N^{(2N)})$) telah dicoba

2. Source Code (Python)

1. Constrained Brute Force

```
1.from matrix import Matrix
2.
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:
25.             {iteration}\n")
26.         return None
27.     else:
28.         return M
29.
30.def bruteForce(board:Matrix, copyBoard:Matrix, k):
31.     global iteration
32.     if (k == 1):
33.         iteration += 1
34.         return validate(board, copyBoard)
35.
36.     M = bruteForce(board, copyBoard, k-1)
37.     if M is not None:
38.         return M
39.
40.     for i in range(0,k-1):
41.         if k % 2 == 0:
42.             swap(copyBoard, i, k-1)
```

```

43.         swap(copyBoard, 0, k-1)
44.         M = bruteForce(board, copyBoard, k-1)
45.         if M is not None:
46.             return M
47.     return None
48.
49. def swap(board:Matrix, colA, colB):
50.     for i in range(0, board.rows):
51.         if board.getElmnt(i,colA) == "#":
52.             rowA = i
53.     for i in range(0,board.rows):
54.         if board.getElmnt(i,colB) == "#":
55.             rowB = i
56.
57.     board.setElmnt(rowA, colA, "0")
58.     board.setElmnt(rowB, colB, "0")
59.     board.setElmnt(rowB, colA , "#")
60.     board.setElmnt(rowA, colB, "#")
61.
62. def validate(board:Matrix, newBoard:Matrix):
63.     global end
64.     arrPoints = []
65.     count = 0
66.     colorCheck = 1
67.     digonalCheck = 1
68.
69.     checkBoard = copyBoard(board)
70.     for i in range(board.rows):
71.         for j in range(board.cols):
72.             if newBoard.getElmnt(i,j) == "#":
73.                 p = Point(i,j,board.getElmnt(i,j))
74.                 arrPoints.append(p)
75.                 checkBoard.setElmnt(i,j,"#")
76.                 count += 1
77.
78.     #check color similarities
79.     check = []
80.     for i in range(0,count):
81.         if arrPoints[i].color in check:
82.             colorCheck = 0
83.         else:
84.             check.append(arrPoints[i].color)
85.
86.     #check diagonal
87.     for i in range(board.rows):
88.         for j in range(board.cols):
89.             if checkBoard.getElmnt(i,j) == "#":

```

```

90.         if (i+1) <= board.rows-1 and (j+1) <= board.cols-1 and
checkBoard.getElmnt(i+1,j+1) == "#":
91.             digonalCheck = 0
92.             break
93.         if (i+1) <= board.rows-1 and (j-1) >= 0 and
checkBoard.getElmnt(i+1,j-1) == "#":
94.             digonalCheck = 0
95.             break
96.     if not digonalCheck:
97.         break
98.
99.     if (colorCheck and digonalCheck):
100.         print("SUKSES pada ")
101.         end = time.perf_counter()
102.         displayBoard(checkBoard)
103.         return checkBoard
104.     else:
105.         if (iteration % interval == 0):
106.             end = time.perf_counter()
107.             displayBoard(checkBoard)
108.             return None
109.
110.     def copyBoard(board:Matrix):
111.         copyBoard = Matrix(board.rows, board.cols)
112.         for i in range(copyBoard.rows):
113.             for j in range(copyBoard.cols):
114.                 copyBoard.setElmnt(i,j, board.getElmnt(i,j))
115.         return copyBoard
116.
117.     def displayBoard(board:Matrix):
118.         print("iterasi ke - " + str(iteration) + "\n")
119.         for i in range(board.rows):
120.             for j in range(board.cols):
121.                 print(board.getElmnt(i,j), end=" ")
122.             print()
123.         print()
124.
125.     def countComplexity(n):
126.         return math.factorial(n)
127.
128.     start = 0
129.     end = 0
130.
131.
132.

```

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:
19.             {iteration}\n")
20.         return None
21.     else:
22.         return M
23.
24. def bruteForce(board: Matrix, newBoard: Matrix, N: int):
25.     global iteration
26.     cells = newBoard.rows * newBoard.cols
27.
28.     if N > cells:
29.         return None
30.
31.     pos = [i for i in range(N)]
32.
33.     used = [False] * cells
34.     for p in pos:
35.         used[p] = True
36.
37.     while True:
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.
```

```

45.         iteration += 1
46.         M = validate(board, newBoard)
47.         if M is not None:
48.             return M
49.
50.         if not increment_positions(pos, used, N, cells):
51.             break
52.
53.     return None
54.
55. def increment_positions(pos, used, N, cells):
56.     k = N - 1
57.     while k >= 0:
58.         used[pos[k]] = False
59.
60.         nxt = pos[k] + 1
61.         while nxt < cells and used[nxt]:
62.             nxt += 1
63.
64.         if nxt < cells:
65.             pos[k] = nxt
66.             used[nxt] = True
67.
68.             for t in range(k + 1, N):
69.                 x = 0
70.                 while x < cells and used[x]:
71.                     x += 1
72.                 if x >= cells:
73.                     return False
74.                 pos[t] = x
75.                 used[x] = True
76.
77.             return True
78.
79.         else:
80.             k -= 1
81.
82.     return False
83.
84. def clearnewBoard(newBoard: Matrix):
85.     for i in range(newBoard.rows):
86.         for j in range(newBoard.cols):
87.             newBoard.setElmnt(i, j, "0")
88.
89. def validate(board: Matrix, newBoard: Matrix):
90.     global end
91.     arrPoints = []

```

```

92.     count = 0
93.     rowColCheck = 1
94.     colorCheck = 1
95.     digonalCheck = 1
96.
97.     checkBoard = copyBoard(board)
98.     for i in range(board.rows):
99.         for j in range(board.cols):
100.             if newBoard.getElmnt(i,j) == "#":
101.                 p = Point(i,j,board.getElmnt(i,j))
102.                 arrPoints.append(p)
103.                 checkBoard.setElmnt(i,j,"#")
104.                 count += 1
105.
106.         #check row-col
107.         for i in range(board.rows):
108.             thisRowExist = 0
109.             for j in range(board.cols):
110.                 if checkBoard.getElmnt(i,j) == "#":
111.                     thisRowExist += 1
112.                 if thisRowExist > 1:
113.                     rowColCheck = 0
114.                     break
115.             if rowColCheck == 0:
116.                 break
117.         for i in range(board.cols):
118.             thisColExist = 0
119.             for j in range(board.rows):
120.                 if checkBoard.getElmnt(j,i) == "#":
121.                     thisColExist += 1
122.                 if thisColExist > 1:
123.                     rowColCheck = 0
124.                     break
125.             if rowColCheck == 0:
126.                 break
127.
128.         #check color similarities
129.         check = []
130.         for i in range(0,count):
131.             if arrPoints[i].color in check:
132.                 colorCheck = 0
133.             else:
134.                 check.append(arrPoints[i].color)
135.
136.         #check diagonal
137.         for i in range(board.rows):
138.             for j in range(board.cols):
139.                 if checkBoard.getElmnt(i,j) == "#":

```



```

140.             if (i+1) <= board.rows-1 and (j+1) <= board.cols-
141. 1 and checkBoard.getElmnt(i+1,j+1) == "#":
142.                 digonalCheck = 0
143.                 break
144.             if (i+1) <= board.rows-1 and (j-1) >= 0 and
145. checkBoard.getElmnt(i+1,j-1) == "#":
146.                 digonalCheck = 0
147.                 break
148.             if not digonalCheck:
149.                 break
150.             if (rowColCheck and colorCheck and digonalCheck):
151.                 print("SUKSES pada ")
152.                 end = time.perf_counter()
153.                 displayBoard(checkBoard)
154.                 return checkBoard
155.             else:
156.                 if (iteration % interval == 0):
157.                     end = time.perf_counter()
158.                     displayBoard(checkBoard)
159.                     return None
160.
161. def displayBoard(board:Matrix):
162.     print("iterasi ke - " + str(iteration) + "\n")
163.     for i in range(board.rows):
164.         for j in range(board.cols):
165.             print(board.getElmnt(i,j), end=" ")
166.         print()
167.     print()
168.
169. def copyBoard(board:Matrix):
170.     copyBoard = Matrix(board.rows, board.cols)
171.     for i in range(copyBoard.rows):
172.         for j in range(copyBoard.cols):
173.             copyBoard.setElmnt(i,j, board.getElmnt(i,j))
174.     return copyBoard
175.

```

3. Hasil Eksekusi

1. Eksekusi 1

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

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

2. Eksekusi 2

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

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

3. Eksekusi 3

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

TIDAK ADA solusi yang ditemukan, max iteration: 362880

Waktu pencarian solusi: 5394.0299999995837 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

```
AAABBBCCCD A A A B B C C # D
ABBBBCECD A B B B # C E C D
ABBBBDCECD A B B B D C # C D
AAABDCCCD A # A B D C C C D
BBBBDDDDDD B B B B D # D D D
FGGGDDHDD F G G # D D H D D
FGIGDDHDD # G I G D D H D D
FGIGDDHDD F G # G D D H D D
FGGGDDHHH 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		✓