

KELOMPOK X

llllgggg

# 8 QUEENS

LOCAL SEARCH

KECERDASAN BUATAN F



KELOMPOK X

llllllllll

NADIF MUSTAFA / 5025211127  
FREDERICK YONATAN / 5025211121  
NIZAM HAKIM / 5025211209

KECERDASAN BUATAN F



dr[ ]:  
array yang mendefinisikan nilai pergerakan pada row  
-1 jika bergerak ke atas,  
1 jika bergerak ke bawah,  
0 jika tidak berpindah row

dc[ ]:  
array yang mendefinisikan nilai pergerakan pada row  
-1 jika bergerak ke kiri,  
1 jika bergerak ke kanan,  
0 jika tidak berpindah column

makeRandom():  
buat state awal secara random

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define endl '\n'
4 #define vi vector<int>
5 #define vvi vector<vi>
6
7 int dr[] = {-1, -1, 0, 1, 1, 1, 0, -1};
8 int dc[] = {0, 1, 1, 1, 0, -1, -1, -1};
9
10 void makeRandom(vvi &board, vi &state)
11 {
12     for(int i = 0; i < 8; ++i){
13         state[i] = rand() % 8;
14         board[state[i]][i] = 1;
15     }
16 }
```

Fungsi makeRandom adalah sebuah fungsi yang menerima dua parameter yaitu sebuah vektor vektor integer bernama board dan sebuah vektor integer bernama state. Fungsi ini bertujuan untuk menempatkan N buah ratu pada papan catur yang berukuran  $N \times N$  secara acak. Pada setiap iterasi, fungsi ini mengisi elemen pada state dengan sebuah bilangan acak hasil modulo 8, yang merepresentasikan posisi kolom dari setiap ratu. Kemudian, pada papan catur yang terdapat pada variabel board, fungsi ini menandai posisi setiap ratu dengan mengisi elemen pada baris ke-state[i] dan kolom ke-i dengan nilai 1. Fungsi ini akan menghasilkan konfigurasi papan catur yang mungkin tidak benar, namun akan digunakan sebagai titik awal dari algoritma N-Queens.

```
18 void printBoard(vvi &board)
19 {
20     cout << endl;
21     for(int i = 0; i < 8; ++i){
22         for(int j = 0; j < 8; ++j)
23             cout << board[i][j] << " ";
24         cout << endl;
25     }
26     cout << endl;
27 }
28
29 vvi translate(vi state)
30 {
31     vvi res(8, vi(8));
32     for(int i = 0; i < 8; ++i)
33         res[state[i]][i] = 1;
34     return res;
35 }
36
37 bool validPos(int r, int c)
38 {
39     return (r >= 0 and r < 8 and c >= 0 and c < 8);
40 }
```

printBoard : print keadaan board saat ini

translate : men-translate suatu state ke dalam bentuk board

validPos : mengecek apakah suatu posisi (row, column) masih berada di dalam batasan board ( $0 \leq r, c \leq 8$ )

Fungsi printBoard adalah fungsi yang menerima sebuah vektor vektor integer bernama board dan bertujuan untuk mencetak ke layar seluruh elemen pada board. Fungsi ini melakukan iterasi pada setiap baris dan kolom pada board, dan mencetak setiap elemen dengan dilakukan sebuah space pada antara setiap elemen. Setelah mencetak seluruh elemen pada board, fungsi ini mencetak sebuah baris kosong untuk memisahkan tampilan output.

Fungsi translate adalah sebuah fungsi yang menerima sebuah vektor integer bernama state dan bertujuan untuk mengembalikan sebuah vektor vektor integer dengan ukuran 8x8 yang merepresentasikan papan catur dengan posisi ratu yang sesuai dengan nilai pada state. Fungsi ini membuat sebuah vektor vektor integer baru dengan ukuran 8x8, dan mengisi elemen pada posisi state[i] dan i dengan nilai 1, yang merepresentasikan posisi ratu pada baris ke-state[i] dan kolom ke-i. Fungsi ini kemudian mengembalikan vektor vektor integer yang telah diisi dengan posisi ratu.

Fungsi validPos adalah sebuah fungsi yang menerima dua buah parameter bilangan bulat r dan c, dan bertujuan untuk mengecek apakah posisi (r,c) merupakan posisi yang valid pada papan catur berukuran 8x8. Fungsi ini melakukan pengecekan apakah nilai r dan c berada pada rentang 0-7, yang merepresentasikan posisi baris dan kolom pada papan catur. Jika posisi tersebut valid, maka fungsi ini mengembalikan nilai true, dan false jika posisi tersebut tidak valid.

```
42 ~ int calcAttacked(vvi &board, vi &state)
43 {
44     int res = 0;
45
46     for(int i = 0; i < 8; ++i){
47         for(int j = 0; j < 8; ++j)
48         {
49             int r = state[i] + dr[j];
50             int c = i + dc[j];
51
52             while(validPos(r, c) and board[r][c] != 1){
53                 r += dr[j];
54                 c += dc[j];
55             }
56             if(validPos(r, c) and board[r][c] == 1)
57                 res++;
58         }
59     }
60     return res / 2;
61 }
```

calcAttacked : menghitung banyaknya queen yang saling serang pada suatu state

Fungsi calcAttacked digunakan untuk menghitung jumlah ratu pada papan catur yang diserang oleh ratu-ratu lain. Fungsi ini menerima dua parameter, yaitu board yang berisi letak ratu pada papan catur dan state yang berisi posisi baris dari setiap ratu pada papan catur. Fungsi ini melakukan iterasi pada setiap ratu, dan menghitung jumlah ratu yang diserang oleh ratu tersebut pada 8 arah yang mungkin. Fungsi akan mengembalikan jumlah ratu yang diserang pada akhir iterasi, dibagi 2 karena setiap ratu dihitung dua kali dalam proses iterasi.

getNext : mencari nextState yang  
nilai calcAttacked nya <= nilai calcAttacked state  
saat ini

```
63  void getNext(vvi &board, vi &state)
64  {
65      vi resState, nextState;
66      resState = nextState = state;
67      vvi resBoard, nextBoard;
68      resBoard = nextBoard = translate(state);
69
70      int minAttacked = calcAttacked(board, state);
71
72      for(int i = 0; i < 8; ++i){
73          for(int j = 0; j < 8; ++j)
74          {
75              if(j != state[i])
76              {
77                  nextState[i] = j;
78                  nextBoard[j][i] = 1;
79                  nextBoard[state[i]][i] = 0;
80
81                  int temp = calcAttacked(nextBoard, nextState);
82
83                  if(temp <= minAttacked){
84                      minAttacked = temp;
85                      resState = nextState;
86                      resBoard = translate(resState);
87                  }
88                  nextBoard[state[i]][i] = 1;
89                  nextBoard[j][i] = 0;
90                  nextState[i] = state[i];
91              }
92          }
93      }
94      state = resState;
95      board = translate(state);
96  }
```

Fungsi getNext berfungsi untuk mendapatkan solusi berikutnya dalam algoritma Hill Climbing untuk menyelesaikan masalah N-Queens. Fungsi ini mencari semua kemungkinan solusi yang berbeda dari solusi saat ini dengan memindahkan setiap ratu ke kolom yang berbeda di baris yang sama. Setiap kemungkinan solusi baru kemudian dihitung berapa banyak ratu yang saling menyerang dan kemudian solusi dengan jumlah serangan terkecil dipilih sebagai solusi berikutnya.

Fungsi ini mengambil dua argumen: papan catur saat ini dalam bentuk vektor 2 dimensi board dan posisi saat ini dari setiap ratu dalam bentuk vektor state. Fungsi mengembalikan argumen board dan state dengan solusi berikutnya yang ditemukan oleh algoritma Hill Climbing.

```
98 void hillClimbing(vvi &board, vi &state)
99 {
100     vi nextState = state;
101     vvi nextBoard = translate(state);
102
103     while(1)
104     {
105         state = nextState;
106         board = translate(state);
107         getNext(nextBoard, nextState);
108
109         if(state == nextState){
110             printBoard(board);
111             break;
112         }
113         else if(calcAttacked(board, state) == calcAttacked(nextBoard, nextState))
114         {
115             nextState[rand() % 8] = rand() % 8;
116             nextBoard = translate(nextState);
117         }
118     }
119 }
```

hillClimbing :  
menjalankan algoritma  
search Hill Climbing

Fungsi hillClimbing adalah implementasi dari algoritma Hill Climbing untuk menyelesaikan masalah N-Queens. Algoritma ini bekerja dengan mengevaluasi keadaan saat ini dan melakukan pergerakan satu langkah ke keadaan berikutnya dengan harapan dapat menemukan keadaan yang lebih baik.

Pada setiap iterasi, fungsi ini memeriksa apakah keadaan saat ini sama dengan keadaan berikutnya. Jika ya, maka pencarian dihentikan dan solusi ditemukan. Jika tidak, maka fungsi getNext dipanggil untuk menemukan keadaan berikutnya yang lebih baik. Jika fungsi getNext tidak menemukan keadaan yang lebih baik, maka salah satu ratu dipindahkan ke posisi acak untuk mencari keadaan yang lebih baik.

```
121 int main()
122 {
123     srand(time(NULL));
124     vi state(8);
125     vvi board(8, vi(8));
126
127     makeRandom(board, state);
128     hillClimbing(board, state);
129     return 0;
130 }
```

main

# **CONTOH HASIL OUTPUT**

0	0	0	0	1	0	0
0	0	0	0	0	0	1
0	0	0	1	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	1
0	0	0	0	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	1
1	0	0	0	0	0	0
0	0	0	0	1	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0

0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0

0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0



TERIMA KASIH