KELOMPOK BUATAN

Kecerdasan Buatan F

ANGGOTA

5025211250

Syomeron Ansell Widjaya

5025211040

Fihriz Ilham Rabbany

5025211251

Muhammad Ahyun Irsyada

8-PUZZLE

8-PUZZLE

Teka-teki 8 adalah teka-teki geser yang memiliki 8 ubin persegi bernomor 1-8 dalam bingkai dengan tinggi 3 ubin dan lebar 3 ubin, menyisakan satu posisi ubin kosong. Ubin di baris atau kolom yang sama dengan posisi terbuka dapat dipindahkan dengan menggesernya masing-masing secara horizontal atau vertikal.Tujuan dari teka-teki ini adalah untuk menempatkan ubin dalam urutan yang diinginkan

8-PUZZLE BFS

BFS

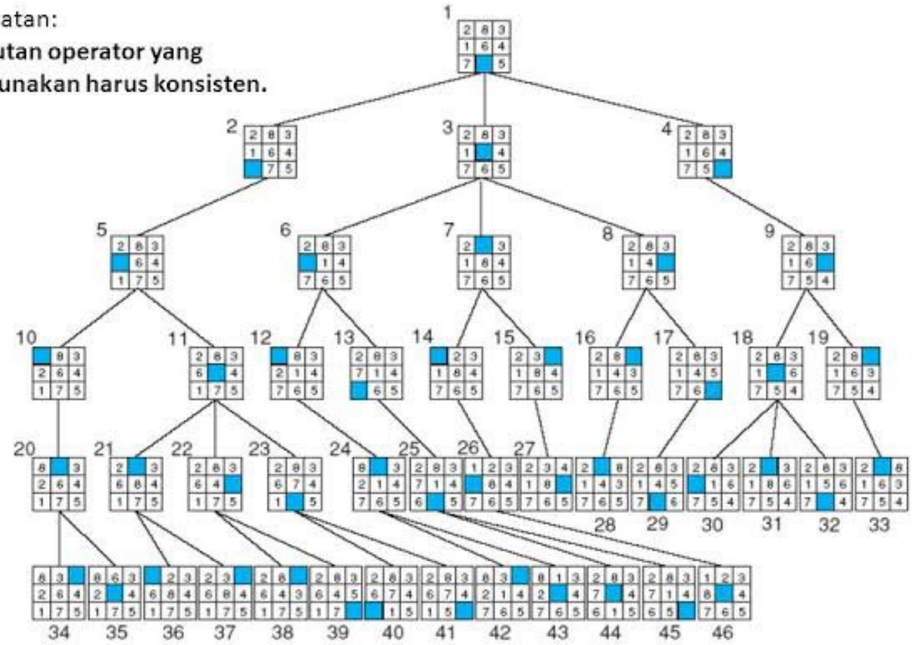
Breadth-first search (BFS) atau Breadth-fist traversal adalah algoritma traversing yang digunakan untuk melintasi atau mencari semua simpul atau node dari suatu struktur data tree atau graph.

Pada algoritma BFS, pencarian dimulai dari pemilihan node awal kemudian dilanjutkan dengan pencarian bertahap level demi level, memeriksa seluruh node pada kedalaman tertentu sebelum masuk ke level yang lebih dalam lagi hingga ditemukan tujuan atau goal state-nya.

ILUSTRATION

BFS untuk 8-Puzzle

Catatan:
Urutan operator yang digunakan harus konsisten.



SOURCE CODE

```cpp
#include <bits/stdc++.h>
using namespace std;

// A set to keep track of visited states
set<string> vis;

// A function to get the position of an adjacent tile given the current position of the blank tile
int getPos(int idx, int dest) {
    int row = idx / 3;
    int col = idx % 3;
    if(dest == 0)
        return (col == 0 ? -1 : 3 * row + col - 1); // move left
    else if(dest == 1)
        return (col == 2 ? -1 : 3 * row + col + 1); // move right
    else if(dest == 2)
        return (row == 0 ? -1 : 3 * (row - 1) + col); // move up
    else
        return (row == 2 ? -1 : 3 * (row + 1) + col); // move down
}
```

SOURCE CODE

```cpp
// The BFS function to solve the 8-puzzle problem
void bfs(string start, string finish) {
    // A queue to store the states that need to be expanded
    queue<pair<string, int>> q;
    // Initialize the queue with the starting state and a step count of 0
    q.push(make_pair(start, 0));
    while(!q.empty()) {
        // Dequeue the front element of the queue and store its state and step count
        string cur = q.front().first;
        int step = q.front().second;

        // Print the current state and step count for visualization
        cout << "Step " << step << endl;
        for(int i = 0; i < 9; ++i){
            cout << cur[i] << " ";
            if((i + 1) % 3 == 0)
                cout << endl;
        }
        cout << endl;

        // Mark the current state as visited
        if(!vis.count(cur))
            vis.insert(cur);
        q.pop();

        // Check if the goal state has been reached
        if(cur == finish){
            cout << "Total steps: " << step << endl;
            return;
        }

        // Find the position of the blank tile in the current state
        int blankPos = -1;
        for(int i = 0; i < 9; ++i) {
            if(cur[i] == '0') {
                blankPos = i;
                break;
            }
        }

        // Generate the child states by swapping the blank tile with its adjacent tiles
        for(int i = 0; i < 4; ++i) {
            int nextPos = getPos(blankPos, i);
            if(nextPos != -1) {
                string nextStr = cur;
                swap(nextStr[blankPos], nextStr[nextPos]);

                // Add the child state to the queue if it has not been visited before
                if(!vis.count(nextStr)) {
                    vis.insert(nextStr);
                    q.push(make_pair(nextStr, step + 1));
                }
            }
        }
    }
}
```

SOURCE CODE

```
78  // The main function to initialize the starting and goal states and call the BFS function
79  int main() {
80      // Initialize the starting and goal states as strings
81      string start = "123845760";
82      string finish = "123804765";
83      // Call the BFS function to solve the 8-puzzle problem
84      bfs(start, finish);
85      return 0;
86  }
```

PENJELASAN

Program mewakili keadaan teka-teki sebagai string dengan panjang 9, dengan '0' mewakili ruang kosong. Algoritma BFS digunakan untuk menemukan jalur terpendek dari keadaan awal ke keadaan tujuan, di mana setiap langkah sesuai dengan menggeser petak ke ruang kosong. Berikut ringkasan komponen utama program: set<string> vis digunakan untuk melacak status yang dikunjungi, untuk menghindari mengunjungi kembali status yang telah dieksplorasi. Fungsi getPos digunakan untuk menghitung posisi petak yang berdekatan berdasarkan posisi ruang kosong saat ini, berdasarkan arah pergerakan (kiri, kanan, atas, atau bawah).

PENJELASAN

Fungsi bfs adalah implementasi algoritma BFS utama. Ini menggunakan queue untuk menyimpan status yang perlu diperluas, dimulai dengan status awal. Algoritme secara iteratif mengeluarkan status dari queue, menghasilkan status turunannya, dan mengantrekannya jika belum dikunjungi. Algoritme berhenti saat status tujuan ditemukan atau saat semua kemungkinan status telah dieksplorasi. Fungsi utama menginisialisasi status awal dan tujuan sebagai string, dan memanggil fungsi bfs untuk memecahkan teka-teki. Dalam fungsi bfs, setiap status dicetak ke konsol beserta jumlah langkahnya, untuk memberikan visualisasi kemajuan algoritme. Setelah status tujuan ditemukan, jumlah total langkah yang diambil untuk mencapainya dicetak ke konsol.

OUTPUT

stdout

```
Step 0
1 2 3
8 4 5
7 6 0

Step 1
1 2 3
8 4 5
7 0 6

Step 1
1 2 3
8 4 0
7 6 5

Step 2
1 2 3
8 4 5
0 7 6

Step 2
1 2 3
8 0 5
7 4 6

Step 2
1 2 3
8 0 4
7 6 5

Total steps: 2
```

8-PUZZLE DFS

DFS

Algoritma Depth First Search (DFS) adalah suatu metode pencarian pada sebuah tree/pohon dengan menelusuri satu cabang sebuah tree sampai menemukan solusi. Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri dan dilanjutkan pada node sebelah kanan. Jika solusi ditemukan maka tidak diperlukan proses backtracking yaitu penelusuran balik untuk mendapatkan jalur yang diinginkan.

P&S

PROBLEM

Mengurutkan angka yang acak pada puzzle dengan menggeser kotak kosong yang tersedia sehingga angkanya urut dari 1-8

SOLUTION

1. Menentukan keadaan awal (start state) dan keadaan tujuan (goal state) dari permainan 8 puzzle.
2. Buat simpul root pada tree yang merepresentasikan keadaan awal.

P&S

SOLUTION

3. Lakukan ekspansi simpul root dengan memperoleh semua kemungkinan simpul turunan yang dapat dicapai dengan melakukan satu pergerakan puzzle pada keadaan root.
4. Buat simpul untuk setiap simpul turunan dan hubungkan simpul turunan tersebut dengan simpul parent-nya dengan sebuah edge yang merepresentasikan pergerakan puzzle yang dilakukan.

SOLUTION

5. Lakukan ekspansi simpul pada setiap simpul turunan yang belum dikunjungi sebelumnya, dan ulangi proses pembuatan simpul dan edge sampai kita mencapai simpul yang merepresentasikan keadaan tujuan.

P&S

SOLUTION

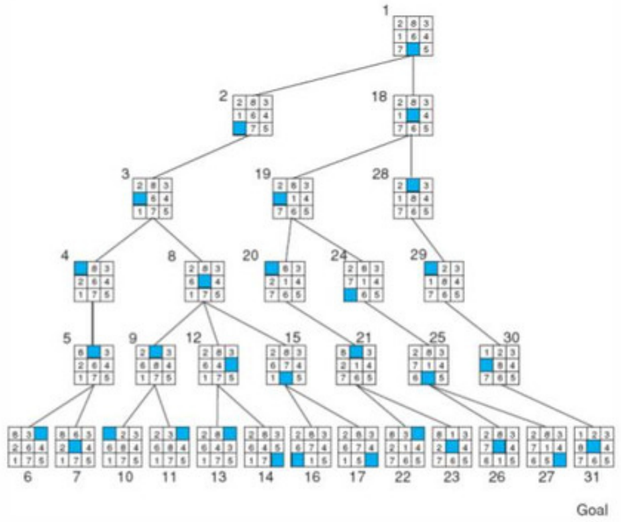
5. Jika keadaan tujuan sudah ditemukan, keluarkan jalur yang diambil dari simpul root menuju simpul tujuan sebagai solusi permainan 8 puzzle. Jalur tersebut merepresentasikan urutan pergerakan puzzle yang harus dilakukan untuk mencapai keadaan tujuan.

SOLUTION

6. Jika semua simpul sudah dikunjungi dan keadaan tujuan tidak ditemukan, keluarkan pesan bahwa permainan tidak dapat diselesaikan

ILUSTRATION

8-puzzle DFS



SOURCE CODE

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```cpp
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <deque>
#include <stack>
using namespace std;

//apothikeusi katastasis
class State{
public:
    int a[9];

    State(){
        for(int i=0; i<9; i++){
                a[i] = rand() % 9;
                while (found(i)) { a[i] = rand() % 9; }
        }
    }

    //ektypwsi katastasis
    void print(){
        cout << "-----------------" << endl;
        for(int i=0; i<9; i++){
            if (i>0 && i%3==0) cout << endl;
            cout << a[i] << "\t";
        }
```

8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```cpp
        cout << endl << "-----------------" << endl;
    }

    //gia tin anafwri epanalamvanomenwn stoixeiwn
    int found(int i){
        for(int j=0; j<i; j++){
            if (a[i]==a[j]) return 1;
        }
        return 0;
    }

    //epistrefei tin thesi tou kenou (0)
    int findzero(){
        for (int i=0; i<9; i++){
            if (a[i]==0) return i;
        }
    }

    //epistrefei neo state allazontas ta stoixeia i kai j metaksi tous
    State exch(int i, int j){
        State b;
        for (int k=0; k<9; k++)
            b.a[k]=a[k];
        int t=b.a[i];
        b.a[i]=b.a[j];
        b.a[j]=t;
        return b;
```

SOURCE CODE

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
52          b.a[j]=t;
53          return b;
54      }
55
56      //elegxei an to state einai idio me to s
57      int equal(State s){
58          for(int i=0; i<9; i++){
59              if (a[i]!=s.a[i]) return 0;
60          }
61          return 1;
62      }
63
64      //elegxei an to state einai to state stoxou
65      int goal(){
66          int g[9] = {1,2,3,8,0,4,7,6,5};
67          for (int i=0; i<9; i++){
68              if (a[i]!=g[i]) return 0;
69          }
70          return 1;
71      }
72
73  };
74
75  class Node{
76  public:
77      State s;
```

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
78      Node *father;
79      int action, cost, depth;
80
81      //dimiourgia rizas dentrou(arxikia katastasis)
82      Node(){
83          State s();
84          father=NULL;
85          action=-1;
86          cost=1;
87          depth=0;
88      }
89
90      //dimiourgia kombou mesa sto expand
91      Node(State _s, Node *_father, int _action, int _depth){
92          s=_s;
93          father=_father;
94          action=_action;
95          depth=_depth;
96      }
97
98      //antigrafi kombou
99      Node copy(){
100         Node b;
101         for (int i=0; i<9; i++)
102             b.s.a[i]=s.a[i];
103         b.father=father;
```

SOURCE CODE

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
103         b.father=father;
104         b.action=action;
105         b.depth=depth;
106
107         return b;
108     }
109
110     //ektypwsi cost+depth
111     void print(){
112         cout << "Cost: \t" << cost << endl;
113         cout << "Depth: \t" << depth << endl;
114     }
115
116     //expand kommou
117     void expand(deque<Node> *deque){
118         int p = s.findzero();
119         //moveup      (0)
120         if ((p!=0 && p!=1 && p!=2) && action!=1){
121             Node n(s.exch(p,p-3), this, 0, depth+1);
122             (*deque).push_back(n);
123         }
124         //m
125         if
126             Node n(s.exch(p,p+3), this, 1, depth+1);
127             (*deque).push_back(n);
128         }
```

Node::Node()
Node::Node(State _s, Node* _father, int _action, int _depth)
class Node {...}

bfs() : int

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
153 int bfs(){
154     deque<Node> toexpand;
155     deque<State> expanded;
156
157     toexpand.push_back(*this);
158     while ( !toexpand.empty() ){
159         if ( toexpand.front().s.goal()==1 ){
160             cout << "------|BFS|------" << endl;
161             cout << "Solution found!" << endl;
162             toexpand.front().print();
163             cost = toexpand.front().cost;
164             toexpand.clear();
165             return cost;
166         }
167         else{
168             if ( !(toexpand.front().expanded(&expanded)) ){
169                 toexpand.front().expand(&toexpand);
170                 expanded.push_front( toexpand.front().s );
171                 toexpand[1].cost=toexpand[0].cost+1;
172             }
173             toexpand.pop_front();
174         }
175     }
176     if ( toexpand.empty() ) cout << endl << "Solution NOT found!" << endl;
177     return 0;
178 }
```

SOURCE CODE

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
178     }
179
180     int dfs(int idsdepth){
181         deque<Node> toexpand;
182
183         if (idsdepth==-1) idsdepth = sizeof(int);
184
185         toexpand.push_back(*this);
186         while ( !toexpand.empty() ){
187                 if (toexpand.back().depth < idsdepth){
188                     if ( toexpand.back().s.goal()==1 ){
189                         if (idsdepth == sizeof(int))
190                             cout << "------|DFS|------" << endl;
191                         else
192                             cout << "------|IDS|------" << endl;
193                         cout << "Solution found!" << endl;
194                         toexpand.back().print();
195                         toexpand.clear();
196                         return cost;
197                     }
198                     else{
199                         Node t;
200                         t= toexpand.back().copy();
201                         toexpand.pop_back();
202                         t.expand(&toexpand);
203                     }
```

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
230     cout << "\nEnter number of problems to solve: ";
231     cin >> num;
232     cout << endl;
233
234     if (argc == 2){
235         cout << "-------DEMO------\n\n";
236     }
237
238     for(int i=0;i<num;i++){
239         int _bfs=0;
240         int _dfs=0;
241         int _ids=0;
242
243         cout << "\n    Problem " << i+1 << endl;
244         srand(argc==2?(2*i+1):time(NULL)+i);
245         Node n;
246         n.s.print();
247
248         _bfs=n.bfs();
249         // _dfs=n.dfs(-1);
250         // _ids=n.ids();
251
252         if (_bfs>0)
253             bfscost+=_bfs;
254         else
255             bfsfail+=1;
```

SOURCE CODE

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
256
257         if (_dfs>0)
258             dfscost+=_dfs;
259         else
260             dfsfail+=1;
261
262         if (_ids>0)
263             idscost+=_ids;
264         else
265             idsfail+=1;
266     }
267     if (num > 1){
268         cout << "\n\n------|BFS|-------";
269         cout << "\nAverage cost = " << bfscost/(num-bfsfail);
270         cout << "\nSolved " << num-bfsfail << "/" << num;
271         cout << "\n-------------------";
272         // cout << "\n\n------|DFS|-------";
273         // cout << "\nAverage cost = " << dfscost/(num-dfsfail);
274         // cout << "\nSolved " << num-dfsfail << "/" << num;
275         // cout << "\n-------------------";
276         // cout << "\n\n------|IDS|-------";
277         // cout << "\nAverage cost = " << idscost/(num-idsfail);
278         // cout << "\nSolved " << num-idsfail << "/" << num;
279         // cout << "\n-------------------";
280         cout << endl;
281     }
```

Start here | 8_puzzle DFS.cpp | hashtable.h | 8-puzzle DFS.cpp | 8-puzzle DFS c++.cpp

```
260             dfsfail+=1;
261
262         if (_ids>0)
263             idscost+=_ids;
264         else
265             idsfail+=1;
266     }
267     if (num > 1){
268         cout << "\n\n------|BFS|-------";
269         cout << "\nAverage cost = " << bfscost/(num-bfsfail);
270         cout << "\nSolved " << num-bfsfail << "/" << num;
271         cout << "\n-------------------";
272         // cout << "\n\n------|DFS|-------";
273         // cout << "\nAverage cost = " << dfscost/(num-dfsfail);
274         // cout << "\nSolved " << num-dfsfail << "/" << num;
275         // cout << "\n-------------------";
276         // cout << "\n\n------|IDS|-------";
277         // cout << "\nAverage cost = " << idscost/(num-idsfail);
278         // cout << "\nSolved " << num-idsfail << "/" << num;
279         // cout << "\n-------------------";
280         cout << endl;
281     }  ostream std::cout
282
283     return 0;
284 }
285
```

8 QUEEN PUZZLE BFS

SOURCE CODE

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

#define N 8

// Define a struct to represent a Queen, which has a row and column position
typedef struct {
    int row, col;
} Queen;

// Define a struct to represent a Node, which contains an array of Queens and a level indicating how many Queens are currently in the array
typedef struct {
    Queen queens[N];
    int level;
} Node;

// A function to check if a given Queen is safe to place on the board, given an array of other Queens already placed
bool isSafe(Queen queen, Queen queens[], int numQueens) {
    for (int i = 0; i < numQueens; i++) {
        if (queen.row == queens[i].row || queen.col == queens[i].col ||
            abs(queen.row - queens[i].row) == abs(queen.col - queens[i].col)) {
            // The new Queen is in the same row or column or diagonal as another Queen, so it's not safe
            return false;
        }
    }
    // The new Queen is safe to place
    return true;
}
```

SOURCE CODE

```c
// A function to perform a breadth-first search to find a solution to the N-Queens problem
void bfs() {
    // Initialize a queue to hold the Nodes that we need to explore
    Node queue[100000];
    int front = -1, rear = -1;
    // Initialize the first Node in the queue with an empty array of Queens and a level of 0
    Node node = { .level = 0 };
    queue[++rear] = node;

    // Keep exploring Nodes in the queue until we find a solution or the queue is empty
    while (front != rear) {
        // Remove the first Node from the queue and explore it
        node = queue[++front];
        // If the Node has N Queens, we have found a solution!
        if (node.level == N) {
            printf("Solution found:\n");
            for (int i = 0; i < N; i++) {
                printf("(%d,%d) ", node.queens[i].row, node.queens[i].col);
            }
            printf("\n");
            return;
        }
        // Try adding a Queen to each column in the next row of the board
        for (int col = 0; col < N; col++) {
            Queen queen = {node.level, col};
            // Check if the new Queen is safe to place
            if (isSafe(queen, node.queens, node.level)) {
                // If it is safe, create a new Node with the new Queen added to the array, and add it to the queue
                Node newNode = node;
                newNode.queens[newNode.level] = queen;
                newNode.level++;
                queue[++rear] = newNode;
            }
        }
    }
    // If we have explored all possible Nodes and haven't found a solution, there is no solution
    printf("Solution not found.\n");
}
```

SOURCE CODE

```
70  // The main function just calls the bfs function and returns 0
71  int main() {
72      bfs();
73      return 0;
74  }
```

PENJELASAN

Program dimulai dengan mendefinisikan dua struktur, Queen dan Node. Sebuah Queen mewakili satu ratu catur di papan dan memiliki dua bidang, baris dan kolom, yang mewakili posisinya di papan. Node mewakili kemungkinan konfigurasi ratu di papan dan memiliki susunan Ratu dan level yang menunjukkan berapa banyak ratu yang telah ditempatkan sejauh ini. Program mendefinisikan fungsi isSafe yang mengambil Ratu dan array Ratu yang mewakili ratu yang sudah ditempatkan di papan, dan mengembalikan nilai true jika ratu baru dapat ditempatkan dengan aman di papan tanpa mengancam ratu yang ada.

PENJELASAN

Fungsi memeriksa apakah ratu baru berada di baris atau kolom yang sama atau diagonal dengan ratu yang ada. Fungsi utama dari program ini adalah bfs, yang melakukan pencarian pertama untuk menemukan solusi untuk masalah N-Queens. Fungsi dimulai dengan menginisialisasi queue Node untuk menyimpan kemungkinan konfigurasi ratu untuk dijelajahi. Ini kemudian menginisialisasi Node pertama dalam queue dengan array ratu kosong dan level 0. Fungsi kemudian memasuki loop yang berlanjut hingga solusi ditemukan atau antrian kosong.

PENJELASAN

Di setiap iterasi loop, fungsi menghapus Node pertama dari antrian dan memeriksa apakah telah menempatkan N ratu di papan tulis. Jika sudah, solusi telah ditemukan dan fungsi mencetak posisi ratu di papan tulis. Jika Node belum menempatkan N ratu di papan, fungsi mencoba menambahkan ratu ke setiap kolom di baris papan berikutnya. Untuk setiap ratu baru yang ditambahkan, fungsi memeriksa apakah aman untuk ditempatkan menggunakan fungsi isSafe.

PENJELASAN

Jika aman, fungsi membuat Node baru dengan ratu baru ditambahkan ke array ratu dan level bertambah 1, dan menambahkannya ke akhir antrian. Jika antrian kosong dan tidak ada solusi yang ditemukan, fungsi akan mencetak pesan yang menunjukkan bahwa tidak ada solusi yang ditemukan. Terakhir, fungsi utama program cukup memanggil fungsi bfs dan return 0.

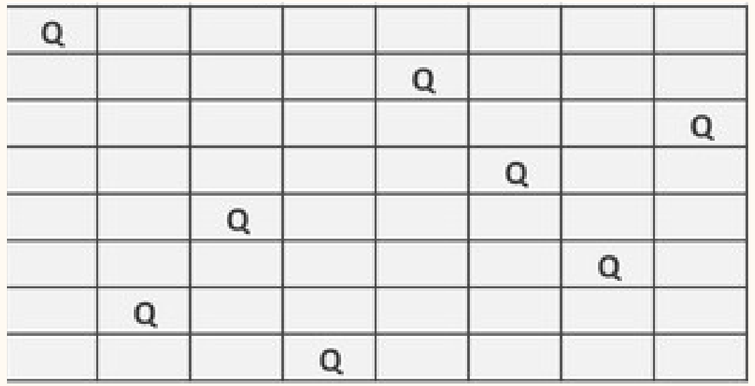
OUTPUT

stdout

```
Solution found:
(0,0) (1,4) (2,7) (3,5) (4,2) (5,6) (6,1) (7,3)
```

0 based index

REPRESENTASI OUTPUT



0 based index

8 QUEEN PUZZLE DFS

8 QUEEN PUZZLE

8-Queen Puzzle adalah teka-teki klasik di mana kita harus meletakkan 8 ratu pada papan catur standar berukuran 8x8 sedemikian rupa sehingga tidak ada dua ratu yang dapat menyerang satu sama lain, yaitu tidak ada dua ratu dalam baris, kolom, atau diagonal yang sama.

P&S

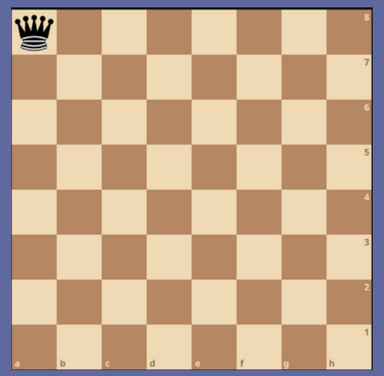
PROBLEM

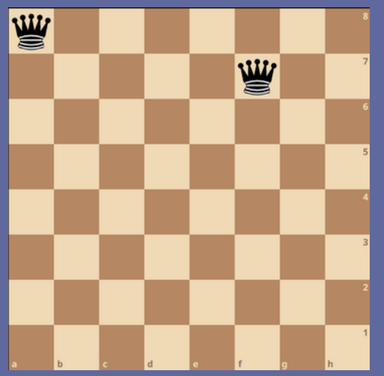
Meletakkan 8 ratu pada papan catur sehingga tidak ada dua ratu yang berada di baris, kolom, atau diagonal yang sama.

SOLUTION

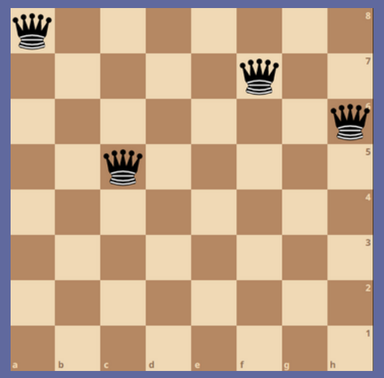
Meletakkan ratu satu per satu pada setiap kolom sambil memenuhi kriteria yang dibutuhkan. Jika ratu tidak dapat ditempatkan, melakukan backtrack sampai konfigurasi yang valid mungkin tercapai.

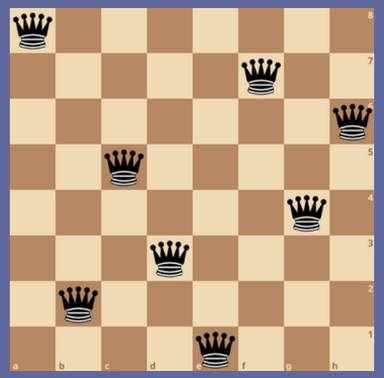
ILLUSTRATION





8 QUEEN DFS





SOURCE CODE

```python
import numpy as np

# membuat papan catur 8x8 tanpa ratu
# jika 0 maka tidak ada ratu
# Jika 1 maka ada ratu

langkah = 0

def Cek(catur, baris, kolom):
    #mengecek apakah baris terdapat ratu
    for i in range(0, 8):
        if catur[baris][i] == 1:
            return False

    temp = 0

    #mengecek apakah diagonal terdapat ratu
    while temp < 8:
        if  baris - temp > -1 and kolom - temp > -1 and catur[baris - temp][kolom - temp] == 1 or \
            baris - temp > -1 and kolom + temp <  8 and catur[baris - temp][kolom + temp] == 1 or \
            baris + temp <  8 and kolom - temp > -1 and catur[baris + temp][kolom - temp] == 1 or \
            baris + temp <  8 and kolom + temp <  8 and catur[baris + temp][kolom + temp] == 1:
            return False

        temp += 1

    return True

def dfs(catur, curr_kolom):
    global langkah

    if curr_kolom > 7:
        return

    for row in range(0,8):
        langkah += 1

        #menghapus baris sebelumnya
        dlt = row
        while dlt > -1:
            catur[dlt][curr_kolom] = 0
            dlt -= 1

        #mengeprint banyak langkah dan hasi akhir papan catur
        if Cek(catur, row, curr_kolom):
            catur[row][curr_kolom] = 1
            if curr_kolom == 7:
                print ('langkah:', langkah)
                print (catur)

            dfs (catur, curr_kolom + 1)


if __name__ == "__main__":
    #membuat papan catur kosong
    catur = np.full((8,8),0)

    #memasukan fungsi satu per satu
    dfs(catur, 0)
```

CEK

```
9   def Cek(catur, baris, kolom):
10      #mengecek apakah baris terdapat ratu
11      for i in range(0, 8):
12          if catur[baris][i] == 1:
13              return False
14
15      temp = 0
16
17      #mengecek apakah diagonal terdapat ratu
18      while temp < 8:
19          if  baris - temp > -1 and kolom - temp > -1 and catur[baris - temp][kolom - temp] == 1 or \
20              baris - temp > -1 and kolom + temp <  8 and catur[baris - temp][kolom + temp] == 1 or \
21              baris + temp <  8 and kolom - temp > -1 and catur[baris + temp][kolom - temp] == 1 or \
22              baris + temp <  8 and kolom + temp <  8 and catur[baris + temp][kolom + temp] == 1:
23              return False
24
25          temp += 1
26
27      return True
```

CEK

Fungsi "Cek" mengambil konfigurasi papan catur dan indeks baris dan kolom dan mengembalikan True jika aman untuk menempatkan seorang ratu di posisi tersebut, jika tidak mengembalikan False. Fungsi ini memeriksa apakah tidak ada ratu di baris, kolom, atau diagonal yang sama.

DFS

```
30  def dfs(catur, curr_kolom):
31      global langkah
32
33      if curr_kolom > 7:
34          return
35
36      for row in range(0,8):
37          langkah += 1
38
39          #menghapus baris sebelumnya
40          dlt = row
41          while dlt > -1:
42              catur[dlt][curr_kolom] = 0
43              dlt -= 1
44
45          #mengeprint banyak langkah dan hasi akhir papan catur
46          if Cek(catur, row, curr_kolom):
47              catur[row][curr_kolom] = 1
48              if curr_kolom == 7:
49                  print ('langkah:', langkah)
50                  print (catur)
51
52              dfs (catur, curr_kolom + 1)
```

DFS

Fungsi "dfs" melakukan pencarian kedalaman untuk menemukan solusi. Fungsi ini mengambil konfigurasi papan catur saat ini dan nomor kolom dari ratu berikutnya yang akan ditempatkan. Jika nomor kolom lebih besar dari 7, itu berarti semua 8 ratu telah ditempatkan dan solusi telah ditemukan. Jika tidak, ia melooping melalui setiap baris dari kolom saat ini dan memeriksa apakah aman untuk menempatkan seorang ratu di posisi tersebut. Jika aman, ia menempatkan seorang ratu di sana dan memanggil dirinya sendiri secara rekursif dengan papan catur yang diperbarui dan nomor kolom berikutnya. Jika tidak aman, ia melakukan backtracking dengan menghapus baris sebelumnya dan mencoba baris berikutnya.

OUTPUT

```
[[1 0 0 0 0 0 0 0]
 [0 0 0 0 0 1 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 0 1 0]
 [1 0 0 0 0 0 0 0]
 [0 0 1 0 0 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 1 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]]
```

```
[[0 1 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 0 0 1 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]]
```

1 menunjukan letak ratu

0 menunjukan papan kosong (tanpa ratu)

TERIMA KASIH