TUGAS 1

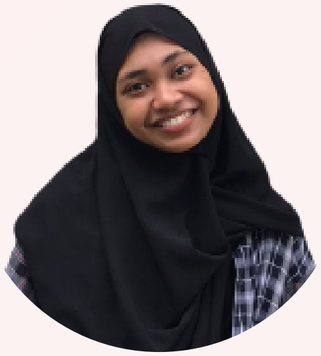
8-Puzzles and 8-Queens with BFS-DFS Solution

Dosen : Dr. Eng Darlis Herumurti, S.Kom, M.Kom

ANGGOTA KELOMPOK – Cucur Adabi

Rizky Alifiyah Rahma

5025211208



Salsabila Fatma Aripa

5025211057

Tsabita Putri Ramadhany

5025211130

Cucur Adabi

8-PUZZLES

1	8	2
0	4	3
7	6	5

Initial State

1	2	3
4	5	6
7	8	0

Goals

Terdapat 8 puzzle dengan posisi random yang tersusun didalam kotak dengan frame 3x3. 1 kotak terdapat sel yang kosong. Oleh karena itu memungkinkan untuk menggeser angka yang bersebelahan. Pada akhirnya nanti **diminta mengurutkan angka random tersebut menjadi berurutan 1 sampai 8**.

Cucur Adabi

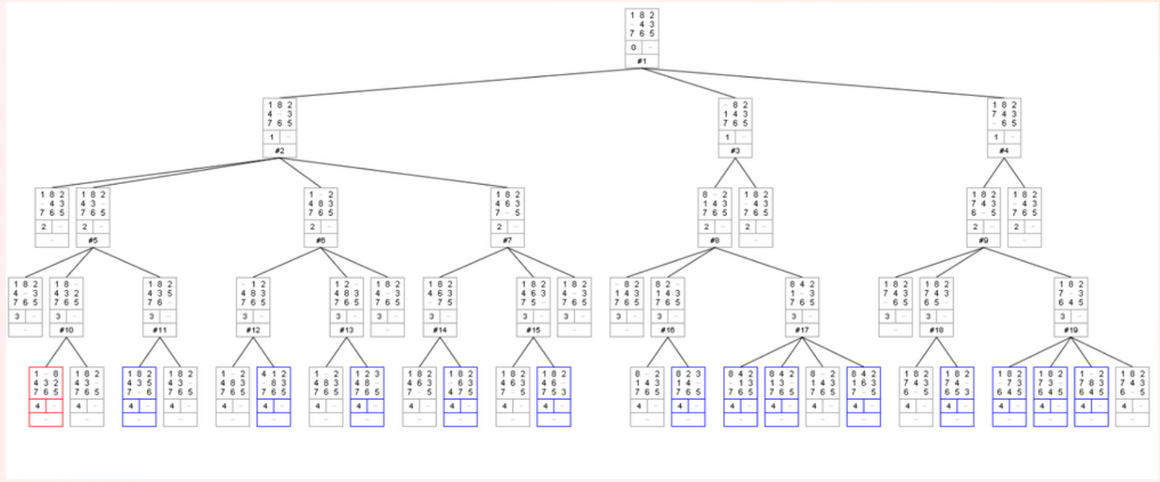
8-Puzzle

Pencarian solusi permasalahan

01

BFS

Breadth First Search
Mencari solusi pada setiap tingkat, dimulai dengan pengecekan dari kedalaman node saat ini baru berpindah ke node kedalaman berikutnya



Cucur Adabi

BFS Solution

```python
import numpy as np

class Node():
    def __init__(self,state,parent,action,depth,step_cost,path_cost,heuristic_cost):
        self.state = state
        self.parent = parent # node parent
        self.action = action # untuk bergerak ke atas,bawah,kiri,kanan
        self.depth = depth # merepresentasikan kedalaman (depth) dalam tree
        self.step_cost = step_cost # g(n), harga untuk melakukan langkah
        self.path_cost = path_cost # akumulasi dari g(n), harga untuk mencapai node saat ini
        self.heuristic_cost = heuristic_cost # h(n), harga untuk mencapai goal node dari node saat ini

        # children node
        self.move_up = None
        self.move_left = None
        self.move_down = None
        self.move_right = None

    # cek apabila bergerak ke bawah adalah valid
    def try_move_down(self):
        # index dari tile kosong
        zero_index=[i[0] for i in np.where(self.state==0)]
        if zero_index[0] == 0:
            return False
        else:
            up_value = self.state[zero_index[0]-1,zero_index[1]] # nilai dari tile atas
            new_state = self.state.copy()
            new_state[zero_index[0],zero_index[1]] = up_value
            new_state[zero_index[0]-1,zero_index[1]] = 0
            return new_state,up_value
```

```python
def breadth_first_search(self, goal_state):

    queue = [self] # # queue dari node yang ditemukan namun belum dikunjungi, FIFO
    queue_num_nodes_popped = 0 #jumlah dari node yang di pop pada queue, mengukur performa waktu
    queue_max_length = 1 # jumlah node maksimum dalam queue

    depth_queue = [0] # queue untuk node depth
    path_cost_queue = [0] # queue untuk harga path
    visited = set([]) # mencatat state-state yang pernah dilalui

    while queue:
        # update panjang maksimum dari queue
        if len(queue) > queue_max_length:
            queue_max_length = len(queue)

        current_node = queue.pop(0) # pilih dan pop node pertama dari queue
        queue_num_nodes_popped += 1

        current_depth = depth_queue.pop(0) # pilih dan pop depth dari node saat ini
        current_path_cost = path_cost_queue.pop(0) # pilih dan pop harga path untuk mencapai node saat ini
        visited.add(tuple(current_node.state.reshape(1,9)[0])) # hindari state yang berulang, yang direpresentasikan sebagai tuple

        # ketika goal state ditemukan, lakukan backtracking ke node root dan print pathnya
        if np.array_equal(current_node.state,goal_state):
            current_node.print_path()

            print ('Time performance:',str(queue_num_nodes_popped),'nodes popped off the queue.')
            return True

        else:
            # cek apakah menggerakan kotak atas ke bawah adalah langkah yang valid
            if current_node.try_move_down():
                new_state,up_value = current_node.try_move_down()
                # cek apakah node hasil sudah dikunjungi
                if tuple(new_state.reshape(1,9)[0]) not in visited:
                    # membuat child node baru
                    current_node.move_down = Node(state=new_state,parent=current_node,action='down',depth=current_depth+1,\
                                                  step_cost=up_value,path_cost=current_path_cost+up_value,heuristic_cost=0)
                    queue.append(current_node.move_down)
                    depth_queue.append(current_depth+1)
                    path_cost_queue.append(current_path_cost+up_value)
```

```python
# cek apabila bergerak ke kanan adalah valid
def try_move_right(self):
    zero_index=[i[0] for i in np.where(self.state==0)]
    if zero_index[1] == 0:
        return False
    else:
        left_value = self.state[zero_index[0],zero_index[1]-1] # nilai dari tile kiri
        new_state = self.state.copy()
        new_state[zero_index[0],zero_index[1]] = left_value
        new_state[zero_index[0],zero_index[1]-1] = 0
        return new_state,left_value

# cek apabila bergerak ke atas adalah valid
def try_move_up(self):
    zero_index=[i[0] for i in np.where(self.state==0)]
    if zero_index[0] == 2:
        return False
    else:
        lower_value = self.state[zero_index[0]+1,zero_index[1]] # nilai dari tile bawah
        new_state = self.state.copy()
        new_state[zero_index[0],zero_index[1]] = lower_value
        new_state[zero_index[0]+1,zero_index[1]] = 0
        return new_state,lower_value

# cek apabila bergerak ke kiri adalah valid
def try_move_left(self):
    zero_index=[i[0] for i in np.where(self.state==0)]
    if zero_index[1] == 2:
        return False
    else:
        right_value = self.state[zero_index[0],zero_index[1]+1] # value of the right tile
        new_state = self.state.copy()
        new_state[zero_index[0],zero_index[1]] = right_value
        new_state[zero_index[0],zero_index[1]+1] = 0
        return new_state,right_value
```

Cucur Adabi

BFS Solution

```python
# saat goal node ditemukan, lacak kembali sampai root node dan print pathnya
def print_path(self):
    # membuat stack FILO untuk menempatkan trace
    state_trace = [self.state]
    action_trace = [self.action]
    depth_trace = [self.depth]
    step_cost_trace = [self.step_cost]
    path_cost_trace = [self.path_cost]
    heuristic_cost_trace = [self.heuristic_cost]

    # menambah informasi node pada saat backtracking tree
    while self.parent:
        self = self.parent

        state_trace.append(self.state)
        action_trace.append(self.action)
        depth_trace.append(self.depth)
        step_cost_trace.append(self.step_cost)
        path_cost_trace.append(self.path_cost)
        heuristic_cost_trace.append(self.heuristic_cost)

    # untuk print path
    step_counter = 0
    while state_trace:
        print ('step',step_counter)
        print (state_trace.pop())


        step_counter += 1
```

```python
        # cek apakah menggerakan kotak kiri ke kanan adalah langkah yang valid
        if current_node.try_move_right():
            new_state,left_value = current_node.try_move_right()
            # cek apakah node hasil sudah dikunjungi
            if tuple(new_state.reshape(1,9)[0]) not in visited:
                # membuat child node baru
                current_node.move_right = Node(state=new_state,parent=current_node,action='right',depth=current_depth+1,\
                                step_cost=left_value,path_cost=current_path_cost+left_value,heuristic_cost=0)
                queue.append(current_node.move_right)
                depth_queue.append(current_depth+1)
                path_cost_queue.append(current_path_cost+left_value)

        # cek apakah menggerakan kotak bawah ke atas adalah langkah yang valid
        if current_node.try_move_up():
            new_state,lower_value = current_node.try_move_up()
            # cek apakah node hasil sudah dikunjungi
            if tuple(new_state.reshape(1,9)[0]) not in visited:
                # membuat child node baru
                current_node.move_up = Node(state=new_state,parent=current_node,action='up',depth=current_depth+1,\
                                step_cost=lower_value,path_cost=current_path_cost+lower_value,heuristic_cost=0)
                queue.append(current_node.move_up)
                depth_queue.append(current_depth+1)
                path_cost_queue.append(current_path_cost+lower_value)

        # cek apakah menggerakan kotak kanan ke kiri adalah langkah yang valid
        if current_node.try_move_left():
            new_state,right_value = current_node.try_move_left()
            # cek apakah node hasil sudah dikunjungi
            if tuple(new_state.reshape(1,9)[0]) not in visited:
                # membuat child node baru
                current_node.move_left = Node(state=new_state,parent=current_node,action='left',depth=current_depth+1,\
                                step_cost=right_value,path_cost=current_path_cost+right_value,heuristic_cost=0)
                queue.append(current_node.move_left)
                depth_queue.append(current_depth+1)
                path_cost_queue.append(current_path_cost+right_value)
```

```python
test = np.array([1,8,2,0,4,3,7,6,5]).reshape(3,3)

initial_state = test
goal_state = np.array([1,2,3,4,5,6,7,8,0]).reshape(3,3)
print (initial_state, '\n')
print (goal_state)

root_node = Node(state=initial_state, parent=None,action=None,depth=0,step_cost=0,path_cost=0,heuristic_cost=0)

# search level by level with queue
root_node.breadth_first_search(goal_state)
#root_node.depth_first_search(goal_state)
```

Cucur Adabi

Hasil BFS Solution

```
[[1 8 2]
 [0 4 3]
 [7 6 5]]

[[1 2 3]
 [4 5 6]
 [7 8 0]]
step 0
[[1 8 2]
 [0 4 3]
 [7 6 5]]
step 1
[[1 8 2]
 [4 0 3]
 [7 6 5]]
step 2
[[1 0 2]
 [4 8 3]
 [7 6 5]]
step 3
[[1 2 0]
 [4 8 3]
 [7 6 5]]
```

```
step 5
[[1 2 3]
 [4 8 5]
 [7 6 0]]
step 6
[[1 2 3]
 [4 8 5]
 [7 0 6]]
step 7
[[1 2 3]
 [4 0 5]
 [7 8 6]]
step 8
[[1 2 3]
 [4 5 0]
 [7 8 6]]
step 9
[[1 2 3]
 [4 5 6]
 [7 8 0]]
Time performance: 454 nodes popped off the queue.
```

Cucur Adabi

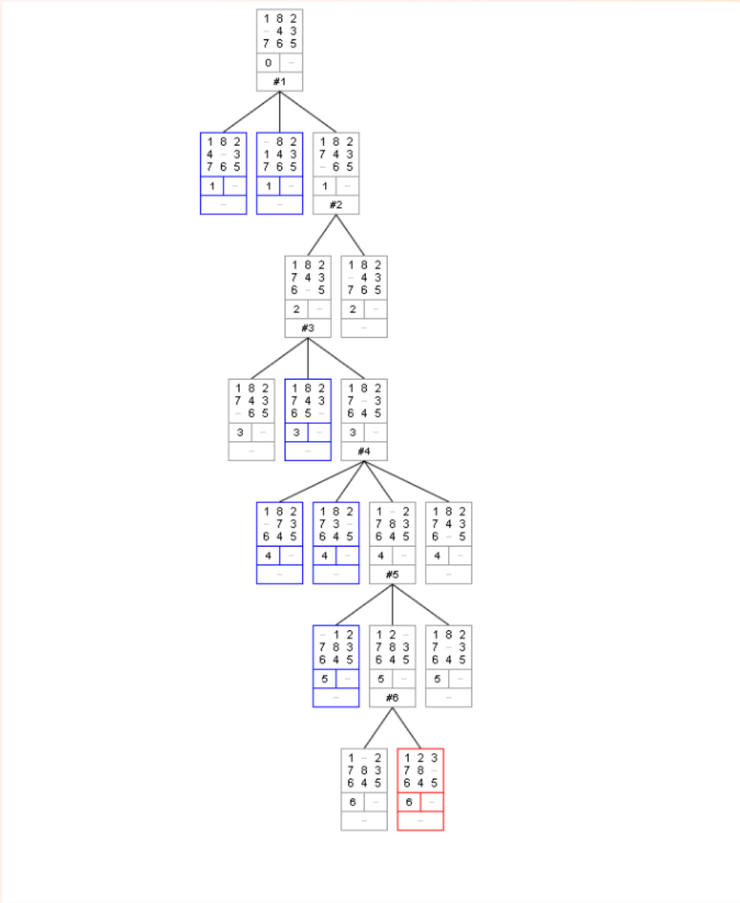
8-Puzzle

Pencarian solusi permasalahan

02

DFS

Depth First Search
Mencari solusi permasalahan dengan mengecek nodes dari yang paling dangkal menuju ke dalam sebelum akhirnya melakukan backtracking ke nodes sebelumnya hingga seluruh nodes selesai dicek



Cucur Adabi

DFS Solution

```python
import numpy as np

class Node():
    def __init__(self,state,parent,action,depth,step_cost,path_cost,heuristic_cost):
        self.state = state
        self.parent = parent # node parent
        self.action = action # untuk bergerak ke atas,bawah,kiri,kanan
        self.depth = depth # merepresentasikan kedalaman (depth) dalam tree
        self.step_cost = step_cost # g(n), harga untuk melakukan langkah
        self.path_cost = path_cost # akumulasi dari g(n), harga untuk mencapai node saat ini
        self.heuristic_cost = heuristic_cost # h(n), harga untuk mencapai goal node dari node saat ini

        # children node
        self.move_up = None
        self.move_left = None
        self.move_down = None
        self.move_right = None
```

```python
    def try_move_down(self):
        # index dari tile kosong
        zero_index=[i[0] for i in np.where(self.state==0)]
        if zero_index[0] == 0:
            return False
        else:
            up_value = self.state[zero_index[0]-1,zero_index[1]] # nilai dari tile atas
            new_state = self.state.copy()
            new_state[zero_index[0],zero_index[1]] = up_value
            new_state[zero_index[0]-1,zero_index[1]] = 0
            return new_state,up_value

    # cek apabila bergerak ke kanan adalah valid
    def try_move_right(self):
        zero_index=[i[0] for i in np.where(self.state==0)]
        if zero_index[1] == 0:
            return False
        else:
            left_value = self.state[zero_index[0],zero_index[1]-1] # nilai dari tile kiri
            new_state = self.state.copy()
            new_state[zero_index[0],zero_index[1]] = left_value
            new_state[zero_index[0],zero_index[1]-1] = 0
            return new_state,left_value
```

```python
    # cek apabila bergerak ke atas adalah valid
    def try_move_up(self):
        zero_index=[i[0] for i in np.where(self.state==0)]
        if zero_index[0] == 2:
            return False
        else:
            lower_value = self.state[zero_index[0]+1,zero_index[1]] # nilai dari tile bawah
            new_state = self.state.copy()
            new_state[zero_index[0],zero_index[1]] = lower_value
            new_state[zero_index[0]+1,zero_index[1]] = 0
            return new_state,lower_value

    # cek apabila bergerak ke kiri adalah valid
    def try_move_left(self):
        zero_index=[i[0] for i in np.where(self.state==0)]
        if zero_index[1] == 2:
            return False
        else:
            right_value = self.state[zero_index[0],zero_index[1]+1] # value of the right tile
            new_state = self.state.copy()
            new_state[zero_index[0],zero_index[1]] = right_value
            new_state[zero_index[0],zero_index[1]+1] = 0
            return new_state,right_value
```

```python
    def print_path(self):
        # membuat stack FILO untuk menempatkan trace
        state_trace = [self.state]
        action_trace = [self.action]
        depth_trace = [self.depth]
        step_cost_trace = [self.step_cost]
        path_cost_trace = [self.path_cost]
        heuristic_cost_trace = [self.heuristic_cost]

        # menambah informasi node pada saat backtracking tree
        while self.parent:
            self = self.parent

            state_trace.append(self.state)
            action_trace.append(self.action)
            depth_trace.append(self.depth)
            step_cost_trace.append(self.step_cost)
            path_cost_trace.append(self.path_cost)
            heuristic_cost_trace.append(self.heuristic_cost)

        # untuk print path
        step_counter = 0
        while state_trace:
            print ('step',step_counter)
            print (state_trace.pop())
```

Liceria & Co.

Cucur Adabi

DFS Solution

```
def depth_first_search(self, goal_state):

        queue = [self] # queue dari node yang ditemukan namun belum dikunjungi, FILO
        queue_num_nodes_popped = 0 #jumlah dari node yang di pop pada queue, mengukur performa waktu
        queue_max_length = 1 # jumlah node maksimum dalam queue

        depth_queue = [0] # queue untuk node depth
        path_cost_queue = [0] # queue untuk harga path
        visited = set([]) # mencatat state-state yang pernah dilalui

        while queue:
            # update panjang maksimum dari queue
            if len(queue) > queue_max_length:
                queue_max_length = len(queue)

            current_node = queue.pop(0) # pilih dan pop node pertama dari queue
            queue_num_nodes_popped += 1

            current_depth = depth_queue.pop(0) # pilih dan pop kedalaman dari node saat ini
            current_path_cost = path_cost_queue.pop(0) # # pilih dan pop harga path untuk mencapai node saat ini
            visited.add(tuple(current_node.state.reshape(1,9)[0])) # tambahkan state, yang direpresentasikan sebagai tuple
```

```
            # cek apakah menggerakan kotak kiri ke kanan adalah langkah yang valid
            if current_node.try_move_right():
                new_state,left_value = current_node.try_move_right()
                # cek apakah node hasil sudah dikunjungi
                if tuple(new_state.reshape(1,9)[0]) not in visited:
                    # membuat child node baru
                    current_node.move_right = Node(state=new_state,parent=current_node,action='right',depth=current_depth+1,\
                                                   step_cost=left_value,path_cost=current_path_cost+left_value,heuristic_cost=0)
                    queue.insert(0,current_node.move_right)
                    depth_queue.insert(0,current_depth+1)
                    path_cost_queue.insert(0,current_path_cost+left_value)

            # cek apakah menggerakan kotak bawah ke atas adalah langkah yang valid
            if current_node.try_move_up():
                new_state,lower_value = current_node.try_move_up()
                # cek apakah node hasil sudah dikunjungi
                if tuple(new_state.reshape(1,9)[0]) not in visited:
                    # membuat child node baru
                    current_node.move_up = Node(state=new_state,parent=current_node,action='up',depth=current_depth+1,\
                                                step_cost=lower_value,path_cost=current_path_cost+lower_value,heuristic_cost=0)
                    queue.insert(0,current_node.move_up)
                    depth_queue.insert(0,current_depth+1)
                    path_cost_queue.insert(0,current_path_cost+lower_value)
```

```
# ketika goal state ditemukan, lakukan backtracking ke node root dan print pathnya
if np.array_equal(current_node.state,goal_state):
    current_node.print_path()

    print ('Time performance:',str(queue_num_nodes_popped),'nodes popped off the queue.')
    return True

else:
    # cek apakah menggerakan kotak atas ke bawah adalah langkah yang valid
    if current_node.try_move_down():
        new_state,up_value = current_node.try_move_down()
        # cek apakah node hasil sudah dikunjungi
        if tuple(new_state.reshape(1,9)[0]) not in visited:
            # membuat child node baru
            current_node.move_down = Node(state=new_state,parent=current_node,action='down',depth=current_depth+1,\
                                          step_cost=up_value,path_cost=current_path_cost+up_value,heuristic_cost=0)
            queue.insert(0,current_node.move_down)
            depth_queue.insert(0,current_depth+1)
            path_cost_queue.insert(0,current_path_cost+up_value)
```

```
                # cek apakah menggerakan kotak kanan ke kiri adalah langkah yang valid
                if current_node.try_move_left():
                    new_state,right_value = current_node.try_move_left()
                    # cek apakah node hasil sudah dikunjungi
                    if tuple(new_state.reshape(1,9)[0]) not in visited:
                        # membuat child node baru
                        current_node.move_left = Node(state=new_state,parent=current_node,action='left',depth=current_depth+1,\
                                                      step_cost=right_value,path_cost=current_path_cost+right_value,heuristic_cost=0)
                        queue.insert(0,current_node.move_left)
                        depth_queue.insert(0,current_depth+1)
                        path_cost_queue.insert(0,current_path_cost+right_value)



test = np.array([1,3,4,8,6,2,7,0,5]).reshape(3,3)

initial_state = test
goal_state = np.array([1,2,3,8,0,4,7,6,5]).reshape(3,3)
print (initial_state, '\n')
print (goal_state)
root_node = Node(state=initial_state, parent=None,action=None,depth=0,step_cost=0,path_cost=0,heuristic_cost=0)
```

Liceria & Co.

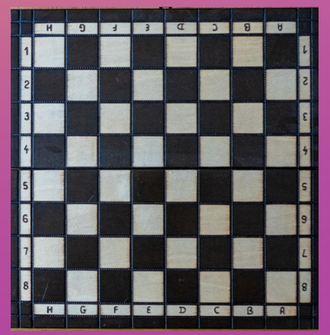
Cucur Adabi

Hasil DFS Solution

```
step 1129
[[1 2 3]
 [5 0 6]
 [4 7 8]]
step 1130
[[1 2 3]
 [0 5 6]
 [4 7 8]]
step 1131
[[1 2 3]
 [4 5 6]
 [0 7 8]]
step 1132
[[1 2 3]
 [4 5 6]
 [7 0 8]]
step 1133
[[1 2 3]
 [4 5 6]
 [7 8 0]]
Time performance: 1141 nodes popped off the queue.
```

Cucur Adabi

8-QUEENS



Initial State

Goals State

Mencari cara untuk meletakkan 8 Queens pada papan 8x8.
Dengan syarat pasangan queen tidak ada yang terancam (sebaris horizontal, vertikal, dan diagonal)

Cucur Adabi

BFS Solutions

```
void solveBFS(){
    queue<node*> q;
    int nodesCreated = 0;
    node *rootNode = new node;
    rootNode->prev = NULL;
    copy(queenPositions, queenPositions + 8, rootNode->qp);
    rootNode->queens = 0;
    q.push(rootNode);
    while(!q.empty()){
        node *currNode = q.front();
        q.pop();
        if(currNode->queens == 8){
            cout<<"HASIL BFS:\n";
            printStepByStep(*currNode);
            cout<<"nodes created: "<<nodesCreated<<"\n\n";
            return;
        }
        else{
            for(int i=0; i<8; i++){
                node *newNode = new node;
                *newNode = *currNode;
                newNode->qp[newNode->queens] = i;
                if(isValid(newNode->qp)){
                    newNode->prev = currNode;
                    newNode->queens = newNode->queens + 1;
                    nodesCreated++;
                    q.push(newNode);
                }
            }
        }
    }
}
```

```
void printBoard(node n){
    bool board[8][8];
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            board[i][j]=false;
        }
    }
    for(int i=0;i<8;i++){
        board[n.qp[i]][i] = true;
    }
    for(int i=0;i<8;i++){
        for(int j=0;j<8;j++){
            cout<<board[i][j]<<" ";
        }
        cout<<endl;
    }
    cout<<endl;
}

void printqp(){
    for(int i=0;i<8;i++){
        cout<<queenPositions[i]<<" ";
    }
    cout<<endl;
}
```

```
        void printqp(node n){
            for(int i=0;i<8;i++){
                cout<<n.qp[i]<<" ";
            }
            cout<<endl;
        }
};

int main(){
    QueenPuzzle dfsSolution;
    QueenPuzzle bfsSolution;
    dfsSolution.solveDFS();
    bfsSolution.solveBFS();
}
```

Cucur Adabi

DFS Solutions

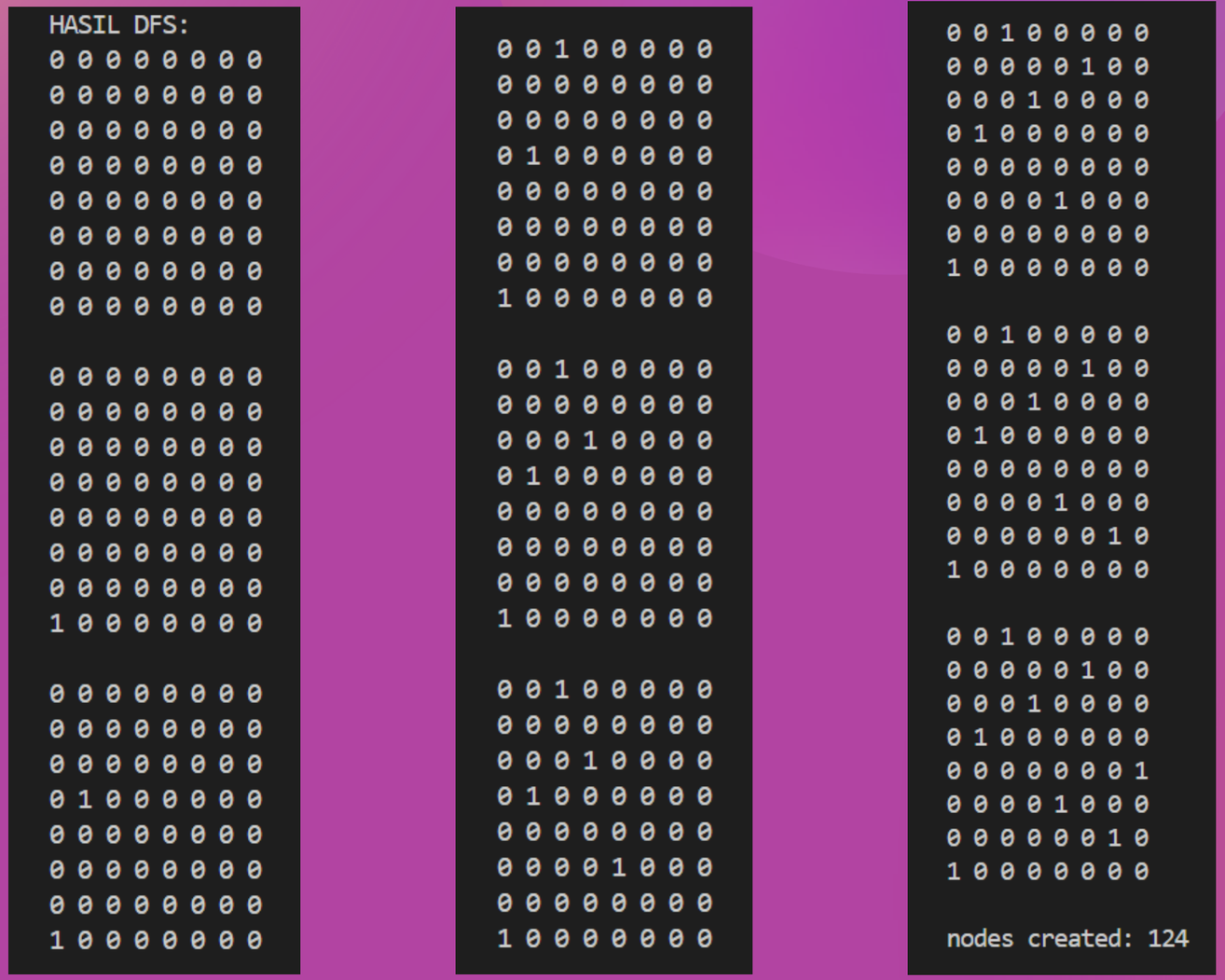
```
#include <bits/stdc++.h>
using namespace std;

class QueenPuzzle{
    public:
        class node{
            public:
                node *prev;
                int qp[8];
                int queens;
        };
        int queenPositions[8];
        QueenPuzzle(){
            for(int i=0; i<8; i++){
                queenPositions[i] = -1;
            }
        }

        bool isValid(int qp[8]){
            for(int i=0; i<7; i++){
                for(int j=i+1; j<8; j++){
                    if(qp[j] == -1 || qp[i] == -1){
                        continue;
                    }
                    if(qp[i] == qp[j]){ //cek horizontal
                        return false;
                    }
                    if(abs(qp[i]-qp[j]) == abs(i-j)){ //cek diagonal
                        return false;
                    }
                }
            }
            return true;
        }
```

```
void printStepByStep(node n){
    if(n.prev != NULL){
        printStepByStep(*n.prev);
    }
    printBoard(n);
}
```

```
void solveDFS(){
    stack<node*> q;
    int nodesCreated = 0;
    node *rootNode = new node;
    rootNode->prev = NULL;
    copy(queenPositions, queenPositions + 8, rootNode->qp);
    rootNode->queens = 0;
    q.push(rootNode);
    while(!q.empty()){
        node *currNode = q.top();
        q.pop();
        if(currNode->queens == 8){
            cout<<"HASIL DFS:\n";
            printStepByStep(*currNode);
            cout<<"nodes created: "<<nodesCreated<<"\n\n";
            return;
        }
        else{
            for(int i=0; i<8; i++){
                node *newNode = new node;
                *newNode = *currNode;
                newNode->qp[newNode->queens] = i;
                if(isValid(newNode->qp)){
                    newNode->prev = currNode;
                    newNode->queens = newNode->queens + 1;
                    nodesCreated++;
                    q.push(newNode);
                }
            }
        }
    }
}
```

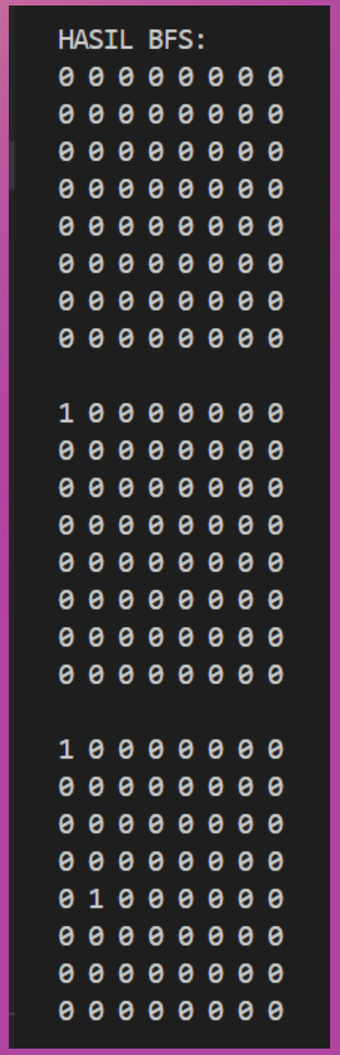
Cucur Adabi

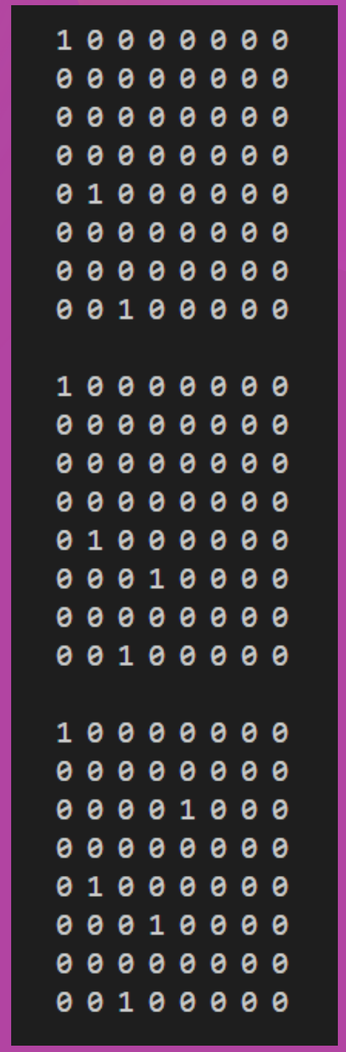
Hasil BFS Solution

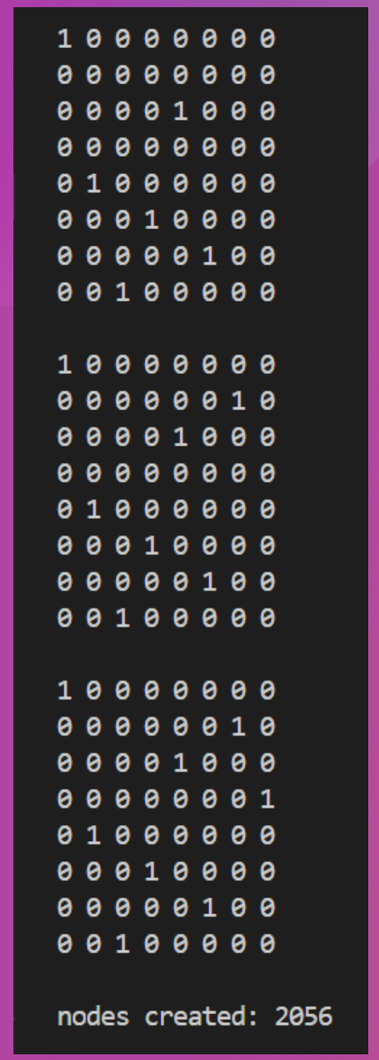


Cucur Adabi

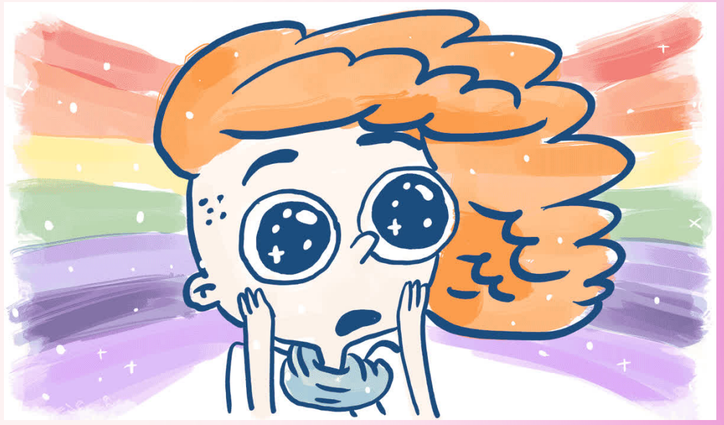
Hasil DFS Solution







Terima Kasih



by : **Cucur Adabi**