# PENYELESAIAN KASUS 8-PUZZLE, 8-QUEEN MENGGUNAKAN BFS DAN DFS


## Anggota Kelompok :

| | |
|---|---|
| Abdurrahman Farimza | 5025201125 |
| Kartika Diva Asmara Gita | 5025211039 |
| Hana Maheswari | 5025211182 |

# BSF

Algoritma pencarian secara melebar dengan menggunakan queue untuk menyimpan node.

Keuntungan :

- Tidak akan menemui jalan buntu.
- Jika ada satu solusi, maka BFS akan menemukannya. Dan jika ada lebih dari satu solusi, maka solusi minimum akan ditemukan.

Kekurangan :

- Membutuhkan memori yang cukup banyak, karena menyimpan semua node dalam satu pohon.
- Membutuhkan waktu yang cukup lama, karena akan menguji n level untuk mendapatkan solusi pada level ke-(n+1)

# DFS

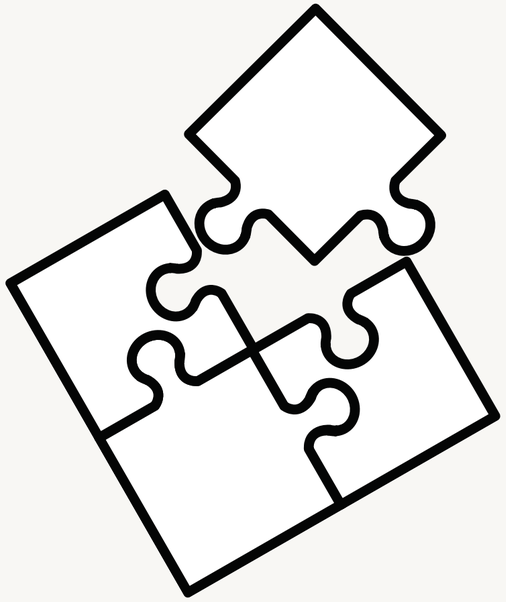
Algoritma pencarian secara mendalam dengan menggunakan stack untuk menyimpan node.

Keuntungan :

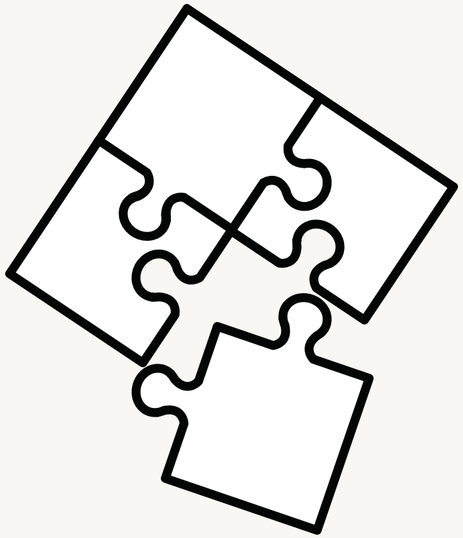
- Pemakain memori hanya sedikit karena tidak menyimpan semua node yang pernah dibangkitkan.
- Jika solusi yang dicari berada pada level yang dalam dan paling kiri, maka DFS akan menemukannya secara cepat.

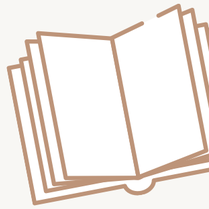
Kekurangan :

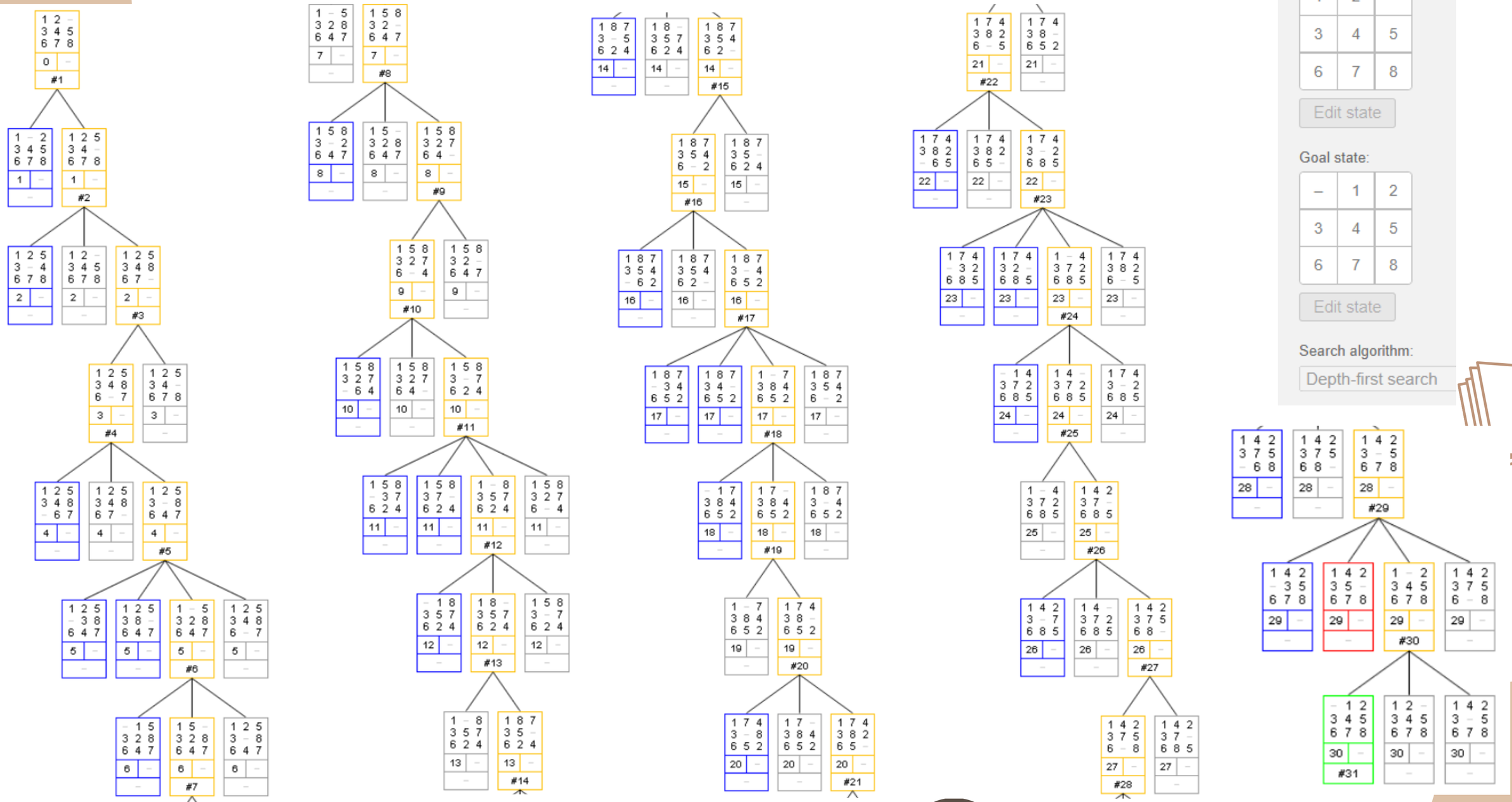
- Jika pohon yang dibangkitkan mempunyai level yang dalam (tak terhingga), maka tidak ada jaminan untuk menemukan solusi (Tidak Complete).



# 8 Puzzle
## (DFS)






Initial state:
1 2 –
3 4 5
6 7 8
Edit state
Goal state:
– 1 2
3 4 5
6 7 8
Edit state
Search algorithm:
Depth-first search

# Code DFS

```python
#DFS
def dfs(startState):

    global GoalNode

    boardVisited = set()
    stack = list([PuzzleState(startState, None, None, 0, 0, 0)])
    while stack:
        node = stack.pop()
        boardVisited.add(node.map)
        if node.state == GoalState:
            GoalNode = node
            return stack
        #inverse the order of next paths for execution purposes
        posiblePaths = reversed(subNodes(node))
        for path in posiblePaths:
            if path.map not in boardVisited:
                stack.append(path)
                boardVisited.add(path.map)
```

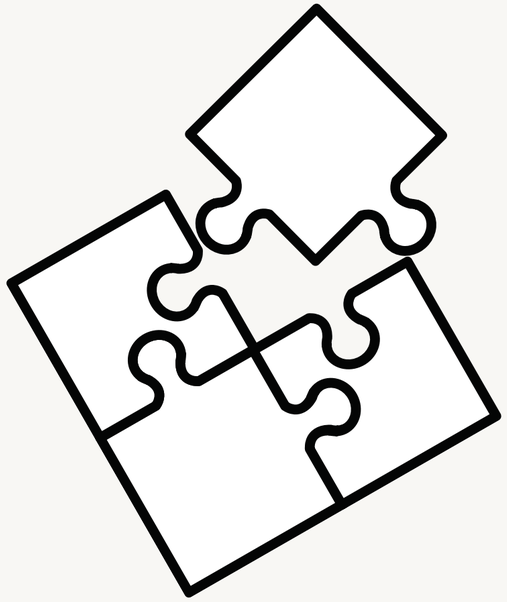
```python
class PuzzleState:
    def __init__(self, state, parent, move, depth, cost):
        self.state = state
        self.parent = parent
        self.move = move
        self.depth = depth
        self.cost = cost
        if self.state:
            self.map = ''.join(str(e) for e in self.state)
    def __eq__(self, other):
        return self.map == other.map
    def __lt__(self, other):
        return self.map < other.map
    def __str__(self):
        return str(self.map)

#Global variables
GoalState = [0, 1, 2, 3, 4, 5, 6, 7, 8]
GoalNode = None # at finding solution
NodesExpanded = 0 #total nodes visited
MaxSearchDeep = 0 #max deep
```

# Code DFS

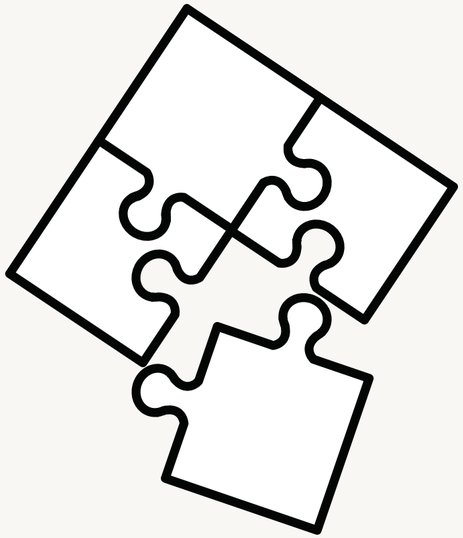
```python
def subNodes(node):

    global NodesExpanded
    NodesExpanded = NodesExpanded+1

    nextPaths = []
    nextPaths.append(PuzzleState(move(node.state, 1), node, 1, node.depth + 1, node.cost + 1, 0))
    nextPaths.append(PuzzleState(move(node.state, 2), node, 2, node.depth + 1, node.cost + 1, 0))
    nextPaths.append(PuzzleState(move(node.state, 3), node, 3, node.depth + 1, node.cost + 1, 0))
    nextPaths.append(PuzzleState(move(node.state, 4), node, 4, node.depth + 1, node.cost + 1, 0))
    nodes=[]
    for procPaths in nextPaths:
        if(procPaths.state!=None):
            nodes.append(procPaths)
    return nodes
```

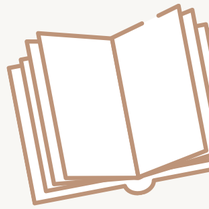
```python
    #Start operation
    start = timeit.default_timer()

    dfs(InitialState)

    stop = timeit.default_timer()
    time = stop-start

    #Save total path result
    deep=GoalNode.depth
    moves = []
    while InitialState != GoalNode.state:
        if GoalNode.move == 1:
            path = 'Up'
        if GoalNode.move == 2:
            path = 'Down'
        if GoalNode.move == 3:
            path = 'Left'
        if GoalNode.move == 4:
            path = 'Right'
        moves.insert(0, path)
        GoalNode = GoalNode.parent

    #Print results
    print("path: ",moves)
    print("cost: ",len(moves))
    print("nodes expanded: ",str(NodesExpanded))
    print("search_depth: ",str(deep))
    print("MaxSearchDeep: ",str(MaxSearchDeep))
    print("running_time: ".format(time, '.8f'))
```

```
PS C:\Users\abdur\Downloads\8puzzledfs\8puzzledfs> python 8puzzle_dfs.py 1,2,5,3,4,8,6,7,0
path:  ['Up', 'Up', 'Left', 'Left']
cost:  4
nodes expanded:  181437
search_depth:  4
running_time:  2.23132810
```



# 8 Puzzle
## (BFS)





# N-Puzzle

Initial state:

| | | |
|---|---|---|
| 3 | 1 | 2 |
| 4 | – | 5 |
| 6 | 7 | 8 |

Edit state

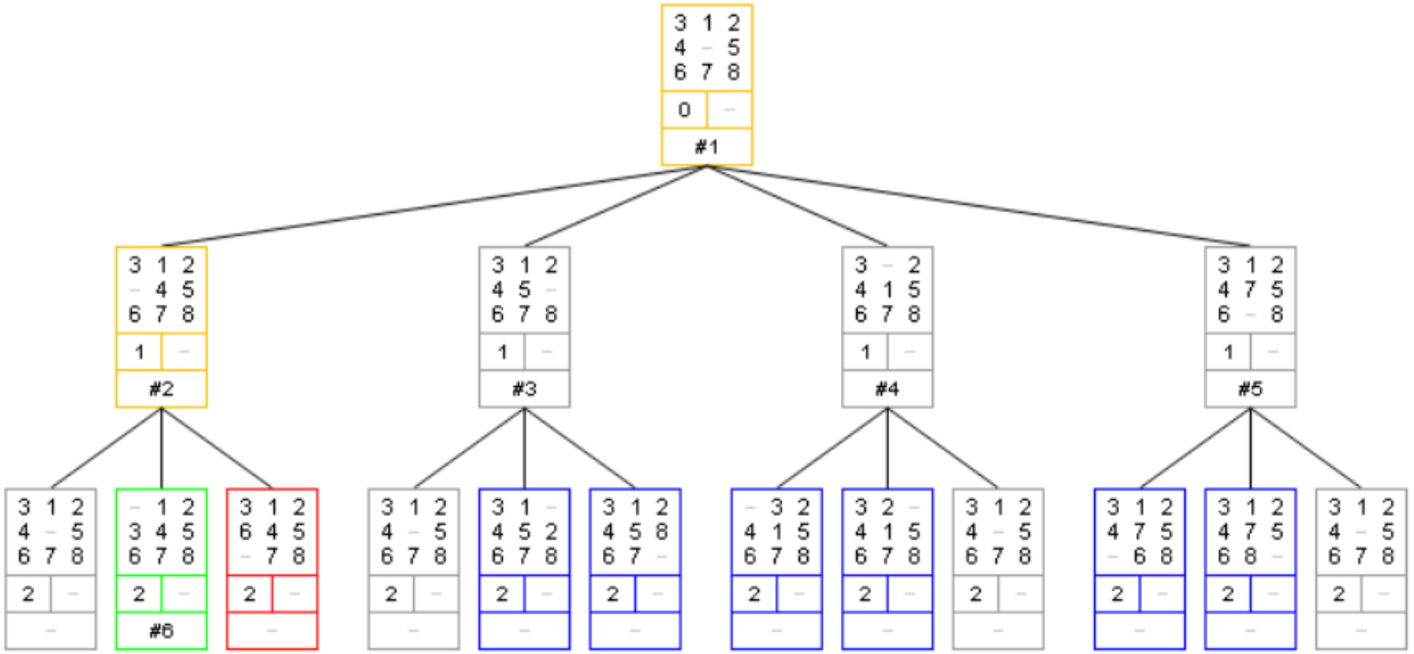
Goal state:

| | | |
|---|---|---|
| – | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Edit state

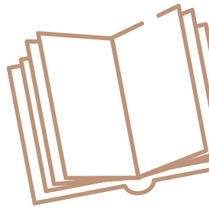
Search algorithm:

Breadth-first search



```c
int main(void)
{

    State initial;   // initial board state
    State goalState; // goal board configuration

    // solution path of each search method
    SolutionPath *bfs;

    // input initial board state
    printf("INITIAL STATE:\n");
    inputState(&initial);

    // input the goal state
    printf("\nGOAL STATE:\n");
    inputState(&goalState);

    printf("INITIAL BOARD STATE:\n");
    printBoard(initial.board);

    printf("GOAL BOARD STATE:\n");
    printBoard(goalState.board);

    // perform breadth-first search
    bfs = BFS_search(&initial, &goalState);
    printf("\nBFS ALGORITHM\n");
    printSolution(bfs);

    return 0;
}
```

```c
void inputState(State *const state)
{
    state->action = NOT_APPLICABLE;
    char row, col;
    int symbol;

    // flags for input validation
    char isNumUsed[9] = {0};

    for (row = 0; row < 3; ++row)
    {
        for (col = 0; col < 3; ++col)
        {
            printf("    board[%i][%i]: ", row, col);
```



```c
printf("SOLUTION: \n");

    // will use hash map to speed up the proccess a bit
    char *move[4] = {"UP", "DOWN", "LEFT", "RIGHT"};
    int counter = 1;

    // will be skipping the first node since it represents the initial state with no action
    for (path = path->next; path; path = path->next, ++counter)
    {
        printf("%i. Move %s\n", counter, move[path->action]);
    }

    printf(
        "DETAILS:\n"
        " - Solution length : %i\n"
        " - Nodes expanded  : %i\n"
        " - Nodes generated : %i\n"
        " - Runtime         : %g milliseconds\n"
        " - Memory used     : %i bytes\n", // only counting allocated `Node`s
        solutionLength, nodesExpanded, nodesGenerated, runtime, nodesGenerated * sizeof(Node));
}
```

```c
SolutionPath *BFS_search(State *initial, State *goal)
{
    NodeList *queue = NULL;
    NodeList *children = NULL;
    Node *node = NULL;

    // start timer
    clock_t start = clock();

    // initialize the queue with the root node of the search tree
    pushNode(createNode(0, manhattanDist(initial, goal), initial, NULL), &queue);
    Node *root = queue->head->currNode; // for deallocating the generated tree

    // while there is a node in the queue to expand
    while (queue->nodeCount > 0)
    {
        // pop the last node (tail) of the queue
        node = popNode(&queue);
        // if the state of the node is the goal state
        if (statesMatch(node->state, goal))
            break;
        // else, expand the node and update the expanded-nodes counter
        children = getChildren(node, goal);
        ++nodesExpanded;
        // add the node's children to the queue
        pushList(&children, queue);
    }
    // determine the time elapsed
    runtime = (double)(clock() - start) / CLOCKS_PER_SEC;
```

```c

    // get solution path in order from the root, if it exists
    SolutionPath *pathHead = NULL;
    SolutionPath *newPathNode = NULL;

    while (node)
    {
        newPathNode = malloc(sizeof(SolutionPath));
        newPathNode->action = node->state->action;
        newPathNode->next = pathHead;
        pathHead = newPathNode;

        // update the solution length and move on the next node
        ++solutionLength;
        node = node->parent;
    }
    --solutionLength; // uncount the root node
    return pathHead;
}
```

```c
Node *createNode(unsigned int d, unsigned int h, State *s, Node *p)
{
    Node *newNode = malloc(sizeof(Node));
    if (newNode)
    {
        newNode->depth = d;
        newNode->hCost = h;
        newNode->state = s;
        newNode->parent = p;
        newNode->children = NULL;
        ++nodesGenerated; // update counter
    }
    return newNode;
}
```

```c
int totalCost(Node *const node)
{
    return node->depth + node->hCost;
}
```

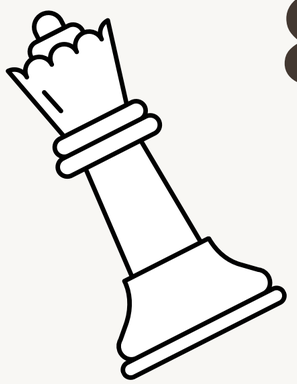
```c
NodeList *getChildren(Node *parent, State *goalState)
{
    NodeList *childrenPtr = NULL;
    State *testState = NULL;
    Node *child = NULL;
    // attempt to create states for each moves, and add to the list of children if true
    if (parent->state->action != DOWN && (testState = createState(parent->state, UP)))
    {
        child = createNode(parent->depth + 1, manhattanDist(testState, goalState), testState, parent);
        pushNode(child, &parent->children);
        pushNode(child, &childrenPtr);
    }
    if (parent->state->action != UP && (testState = createState(parent->state, DOWN)))
    {
        child = createNode(parent->depth + 1, manhattanDist(testState, goalState), testState, parent);
        pushNode(child, &parent->children);
        pushNode(child, &childrenPtr);
    }
    if (parent->state->action != RIGHT && (testState = createState(parent->state, LEFT)))
    {
        child = createNode(parent->depth + 1, manhattanDist(testState, goalState), testState, parent);
        pushNode(child, &parent->children);
        pushNode(child, &childrenPtr);
    }
    if (parent->state->action != LEFT && (testState = createState(parent->state, RIGHT)))
    {
        child = createNode(parent->depth + 1, manhattanDist(testState, goalState), testState, parent);
        pushNode(child, &parent->children);
        pushNode(child, &childrenPtr);
    }
    return childrenPtr;
}
```

```c
#define BLANK_CHARACTER '0'

typedef enum Move {
    UP, DOWN, LEFT, RIGHT,
    NOT_APPLICABLE
} Move;

typedef struct State {
    Move action;
    char board[3][3];
} State;
```

```c
State* createState(State *state, Move move) {
    State *newState = malloc(sizeof(State));

    char i, j;
    char row, col;

    for(i = 0; i < 3; ++i) {
        for(j = 0; j < 3; ++j) {
            if(state->board[i][j] == BLANK_CHARACTER) {
                row = i;
                col = j;
            }
            newState->board[i][j] = state->board[i][j];
        }
    }

    if(move == UP && row - 1 ≥ 0) {
        char temp = newState->board[row - 1][col];
        newState->board[row - 1][col] = BLANK_CHARACTER;
        newState->board[row][col] = temp;
        newState->action = UP;
        return newState;
    }
    else if(move == DOWN  && row + 1 < 3) {
        char temp = newState->board[row + 1][col];
        newState->board[row + 1][col] = BLANK_CHARACTER;
        newState->board[row][col] = temp;
        newState->action = DOWN;
        return newState;
    }
    else if(move == LEFT  && col - 1 ≥ 0) {
        char temp = newState->board[row][col - 1];
        newState->board[row][col - 1] = BLANK_CHARACTER;
        newState->board[row][col] = temp;
        newState->action = LEFT;
        return newState;
    }
    else if(move == RIGHT && col + 1 < 3) {
        char temp = newState->board[row][col + 1];
        newState->board[row][col + 1] = BLANK_CHARACTER;
        newState->board[row][col] = temp;
        newState->action = RIGHT;
        return newState;
    }
    free(newState);
    return NULL;
}
```

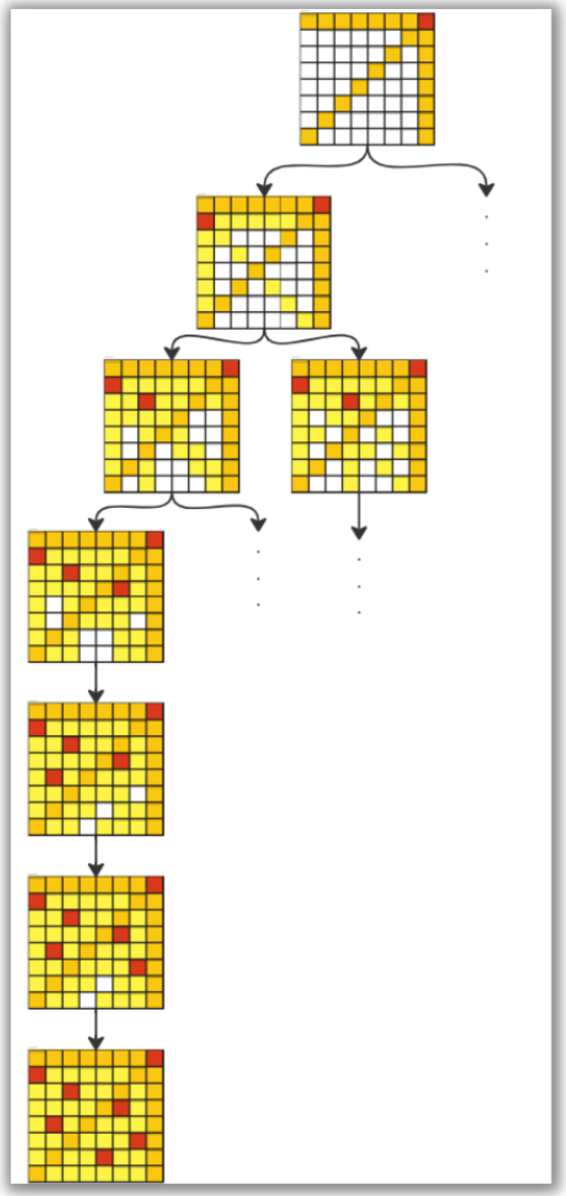
```
INITIAL STATE:
    board[0][0]: 1
    board[0][1]: 2
    board[0][2]: 3
    board[1][0]: 4
    board[1][1]: 0
    board[1][2]: 5
    board[2][0]: 6
    board[2][1]: 7
    board[2][2]: 8


GOAL STATE:
    board[0][0]: 0
    board[0][1]: 1
    board[0][2]: 2
    board[1][0]: 3
    board[1][1]: 4
    board[1][2]: 5
    board[2][0]: 6
    board[2][1]: 7
    board[2][2]: 8

INITIAL BOARD STATE:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 0 | 5 |
+---+---+---+
| 6 | 7 | 8 |
+---+---+---+
GOAL BOARD STATE:
+---+---+---+
| 0 | 1 | 2 |
+---+---+---+
| 3 | 4 | 5 |
+---+---+---+
| 6 | 7 | 8 |
+---+---+---+

 USING BFS ALGORITHM
SOLUTION: (Relative to the space character)
1. Move LEFT
```

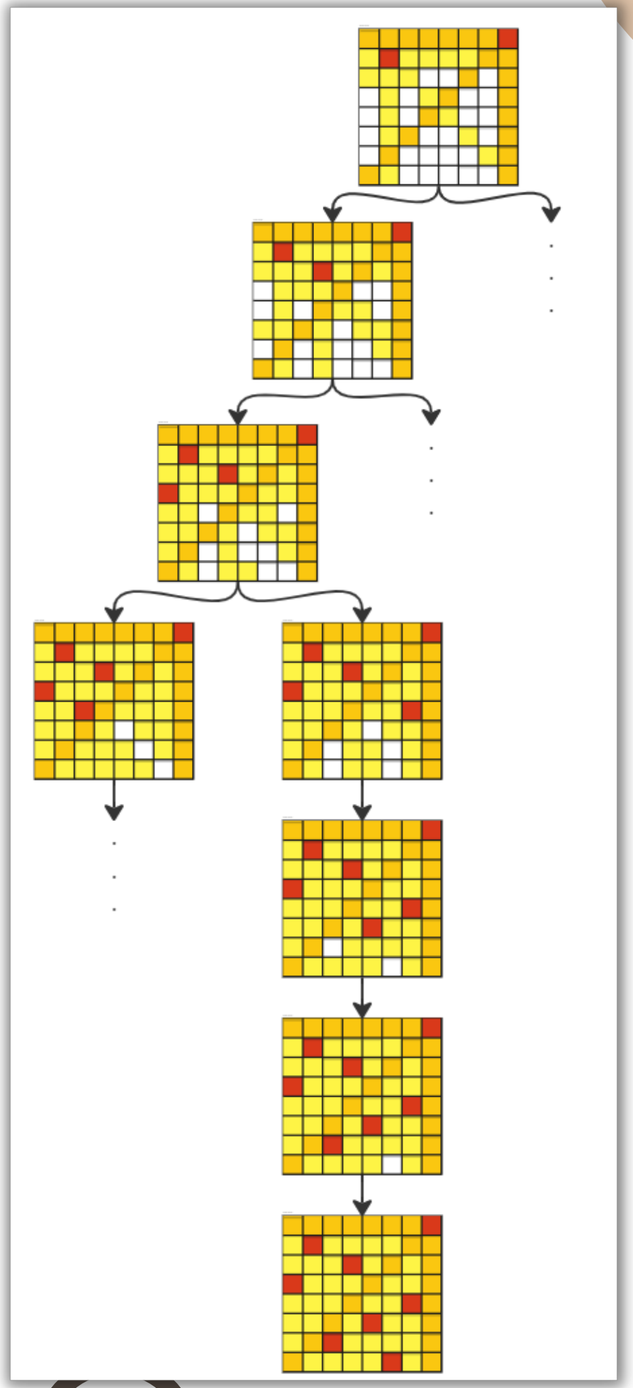
```
 USING BFS ALGORITHM
SOLUTION: (Relative to the space character)
1. Move LEFT
2. Move UP
3. Move RIGHT
4. Move RIGHT
5. Move DOWN
6. Move LEFT
7. Move UP
8. Move LEFT
9. Move DOWN
10. Move RIGHT
11. Move RIGHT
12. Move UP
13. Move LEFT
14. Move LEFT
DETAILS:
 - Solution length : 14
 - Nodes expanded  : 8706
 - Nodes generated : 14732
 - Runtime         : 0.006 milliseconds
 - Memory used     : 471424 bytes
PS C:\Users\Kartika Diva\Documents\8-Puzzle-Solver
```



# 8 Queen
## (DFS)

# Skema (column = 8)



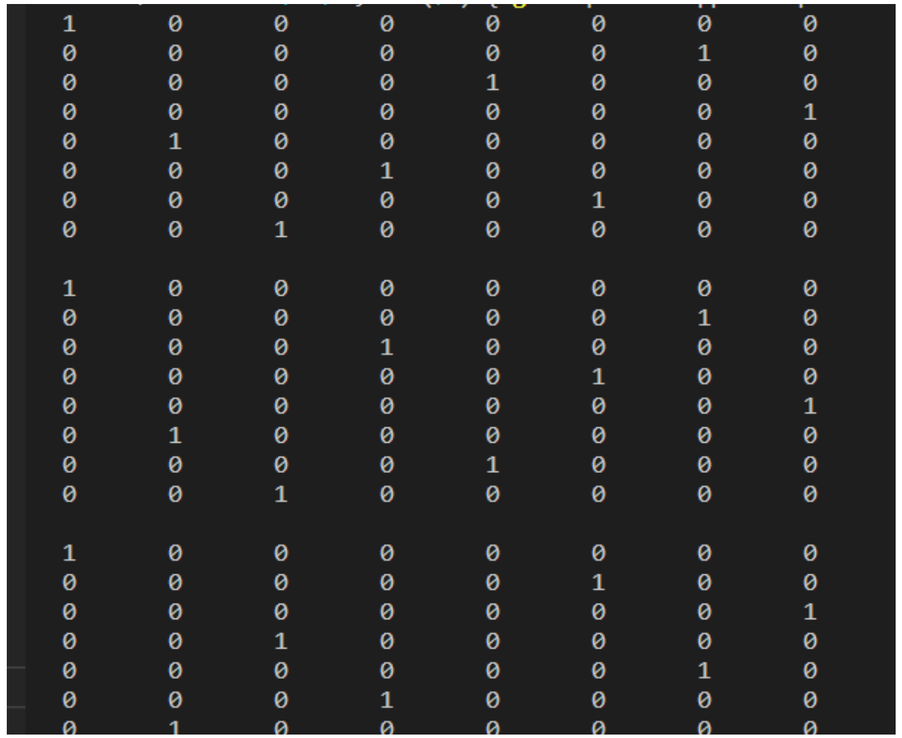


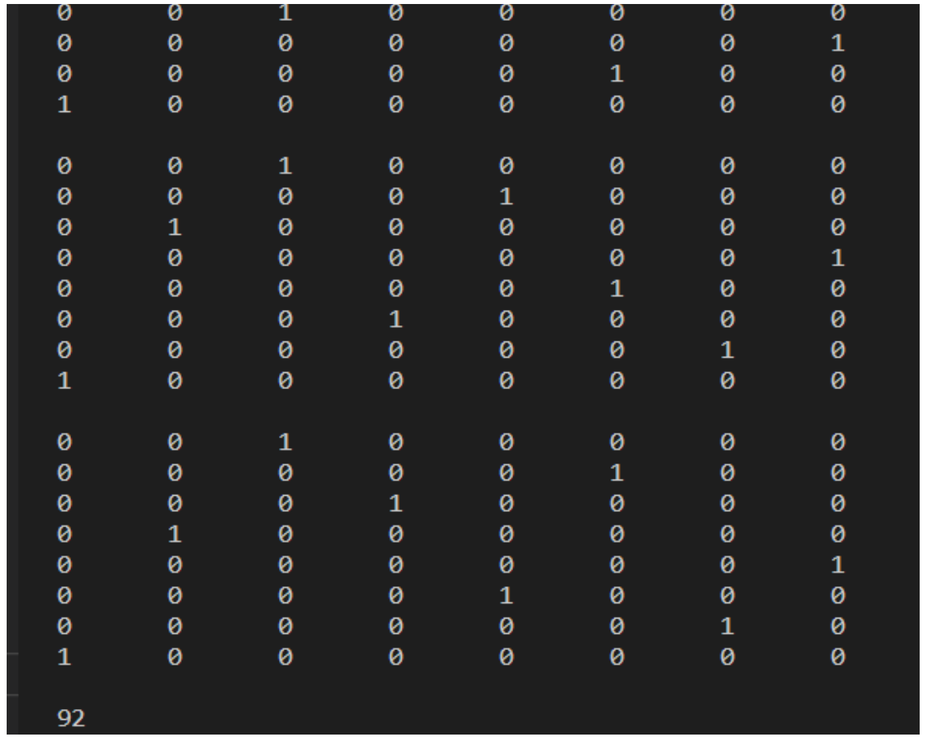
```cpp
#include<iostream>

using namespace std;

int N = 8;
int board[8][8];
int numQueens = 0;
int sols = 0;

void searchCol(int col) {
    // base case
    if(numQueens >= N) {
        // print solution and exit
        for(int i = 0; i < N; i++) {
            for(int j = 0; j < N; j++) {
                if(board[i][j] < 1)
                    cout << "0" << "\t";
                else
                    cout << board[i][j] << "\t";
            }
            cout << "\n";
        }
        cout << "\n";
        sols++;
        return;
    }
```

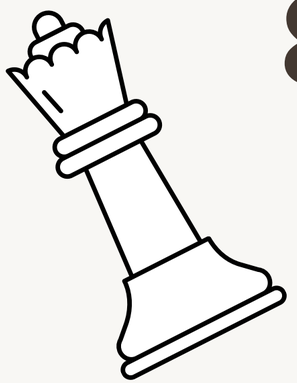
```cpp
int main() {
    searchCol(0);
    cout << sols << "\n";
    return 0;
}
```

```
    // for each row
    for(int i = 0; i < N; i++) {
        // if cell is not attacked
        if(board[i][col] > -1) {
            // set queen and mark all cells in cross and diagonal
            board[i][col] = 1;
            for(int j = 0; j < N; j++) {
                if(j != col) board[i][j]--;
                if(j != i) board[j][col]--;
            }
            for(int j = 1; j < N; j++) {
                if(i - j >= 0 && col - j >= 0) board[i - j][col - j]--;
                if(i + j < N && col + j < N) board[i + j][col + j]--;
                if(i - j >= 0 && col + j < N) board[i - j][col + j]--;
                if(i + j < N && col - j >= 0) board[i + j][col - j]--;
            }

            numQueens++;
            searchCol(col + 1);
            // remove queens and markers
            board[i][col] = 0;
            for(int j = 0; j < N; j++) {
                if(j != col) board[i][j]++;
                if(j != i) board[j][col]++;
            }
            for(int j = 1; j < N; j++) {
                if(i - j >= 0 && col - j >= 0) board[i - j][col - j]++;
                if(i + j < N && col + j < N) board[i + j][col + j]++;
                if(i - j >= 0 && col + j < N) board[i - j][col + j]++;
                if(i + j < N && col - j >= 0) board[i + j][col - j]++;
            }
            numQueens--;
        }
    }
}
```

# Output:

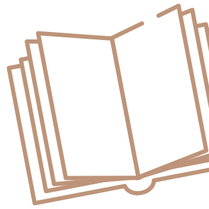

1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 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 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
1 0 0 0 0 0 0 0
0 0 1 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 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
1 0 0 0 0 0 0 0
0 0 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 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
92



# 8 Queen
## (BFS)

```python
class Queens:
    def __init__(self, size):
        self.size = size

    def solve_bfs(self):
        solutions = []
        queue = Queue()
        queue.put([])
        while not queue.empty():
            solution = queue.get()
            if self.conflict(solution):
                continue
            row = len(solution)
            if row == self.size:
                solutions.append(solution)
                continue
            for col in range(self.size):
                queen = (row, col)
                queens = solution.copy()
                queens.append(queen)
                queue.put(queens)
        return solutions
```



```python
    def conflict(self, queens):
        for i in range(1, len(queens)):
            for j in range(0, i):
                a, b = queens[i]
                c, d = queens[j]
                if a == c or b == d or abs(a - c) == abs(b - d):
                    return True
        return False

    def print(self, queens):
        for i in range(self.size):
            print(' ---' * self.size)
            for j in range(self.size):
                p = 'Q' if (i, j) in queens else ' '
                print('| %s ' % p, end='')
            print('|')
        print(' ---' * self.size)
```

```python
from queens import Queens


def main():
    size = 8
    queens = Queens(size)
    bfs_solutions = queens.solve_bfs()
    for i, solution in enumerate(bfs_solutions):
            print('BFS Solution %d:' % (i + 1))
            queens.print(solution)
    print('Total BFS solutions: %d' % len(bfs_solutions))


if __name__ == '__main__':
    main()
```

```
BFS Solution 1:
| Q |   |   |   |   |   |   |   |
|   |   |   |   | Q |   |   |   |
|   |   |   |   |   |   |   | Q |
|   |   |   |   |   | Q |   |   |
|   |   | Q |   |   |   |   |   |
|   |   |   |   |   |   | Q |   |
|   | Q |   |   |   |   |   |   |
|   |   |   | Q |   |   |   |   |
BFS Solution 2:
| Q |   |   |   |   |   |   |   |
|   |   |   |   |   | Q |   |   |
|   |   |   |   |   |   |   | Q |
|   |   | Q |   |   |   |   |   |
|   |   |   |   |   |   | Q |   |
|   |   |   | Q |   |   |   |   |
|   | Q |   |   |   |   |   |   |
|   |   |   |   | Q |   |   |   |
BFS Solution 3:
```

```
BFS Solution 92:
|   |   |   |   |   |   |   | Q |
|   |   |   | Q |   |   |   |   |
| Q |   |   |   |   |   |   |   |
|   |   | Q |   |   |   |   |   |
|   |   |   |   |   | Q |   |   |
|   | Q |   |   |   |   |   |   |
|   |   |   |   |   |   | Q |   |
|   |   |   |   | Q |   |   |   |
Total BFS solutions: 92
```

# TERIMA KASIH