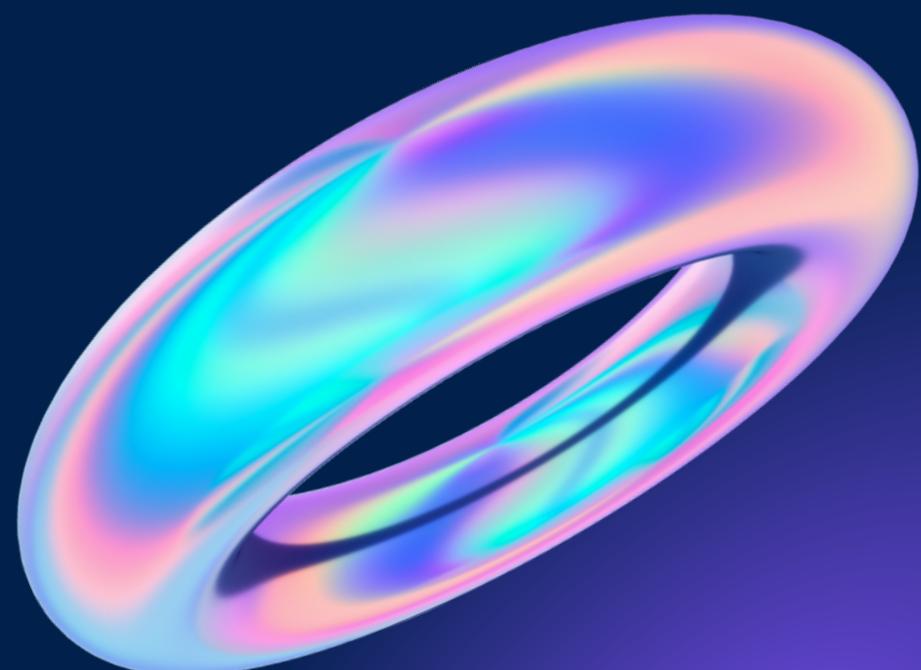




Tugas 1 Kecerdasan Buatan



Team Member

- Timothy Hosia (5025211098)
- Yohanes Teguh (5025211179)
- Shazia Ingeyla (5025211203)

Daftar Isi

Q1

Penyelesaian 8-
Puzzle
Menggunakan
DFS

Q2

Penyelesaian 8-
Puzzle
Menggunakan
BFS

Q3

Penyelesaian 8-
Queen
Menggunakan
DFS

Q4

Penyelesaian 8-
Queen
Menggunakan
BFS

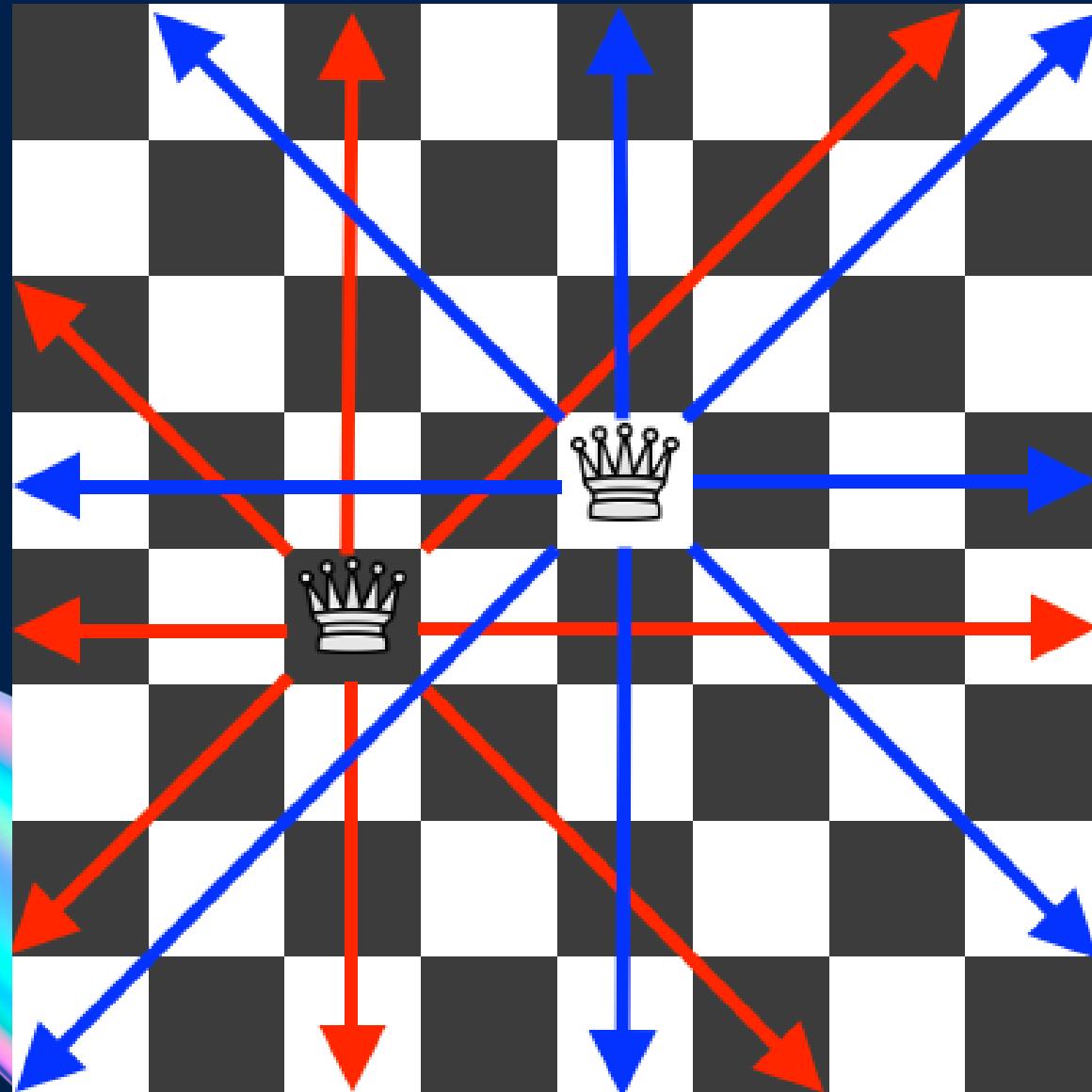
BFS AND DFS

Breadth First Search adalah algoritma pencarian dengan teknik yang berbasis simpul untuk menemukan penyelesaian terpendek dalam grafik. Dimana semua simpul yang berada pada level yang sama dieksplorasi terlebih dahulu baru pindah ke level berikutnya.

Depth First Search adalah salah satu algoritma penelusuran struktur graf. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya. Menjelajahi simpul tersebut sejauh mungkin sampai mencapai simpul yang tidak memiliki simpul berdekatan yang belum dikunjungi.

Blind Search	Breadth First Search	Depth First Search
Data Structure	Queue	Stack
Backtracking	BFS does not use the backtracking concept.	DFS uses backtracking to traverse all the unvisited nodes.
Speed	Slower	Faster
Memory efficient	Not memory efficient	Memory efficient

8-queens intro



Masalah 8 Ratu adalah masalah klasik dalam ilmu komputer di mana tujuannya adalah untuk menempatkan delapan ratu di papan catur sedemikian rupa sehingga tidak ada dua ratu yang saling menyerang.

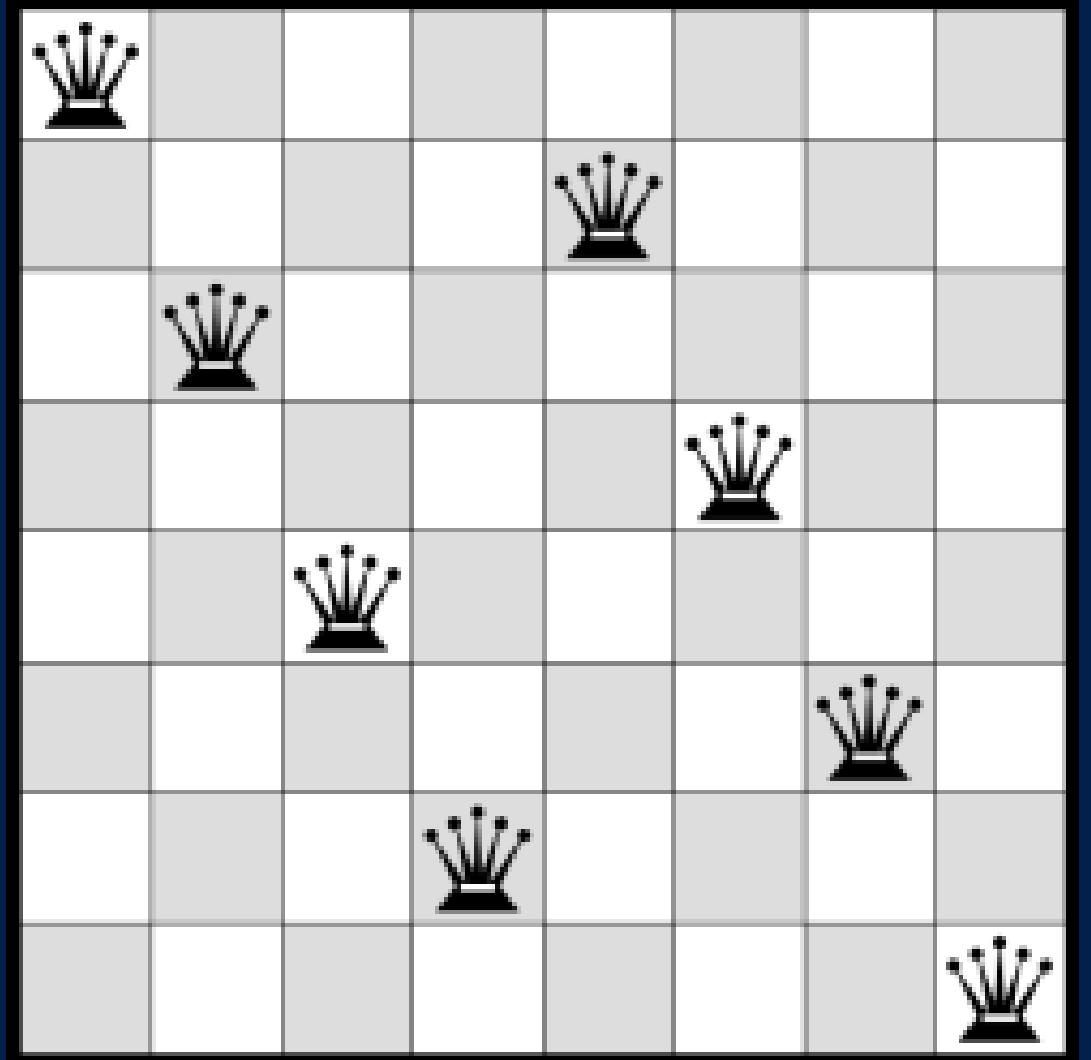
Masalah ini dapat diselesaikan dengan metode brute force dimana kita akan mencoba semua possible placement. Tetapi hal itu membutuhkan banyak memori waktu, di papan 8×8 sendiri akan memiliki kompleksitas sebagai berikut.

$$\left(\frac{(8+8)!}{(8+(8-1))!} \right) = 1.784629876 \times 10^{14}$$

DIKURANG SYARAT 8 QUEENS MENJADI

$$= 16777216$$

8-queens intro

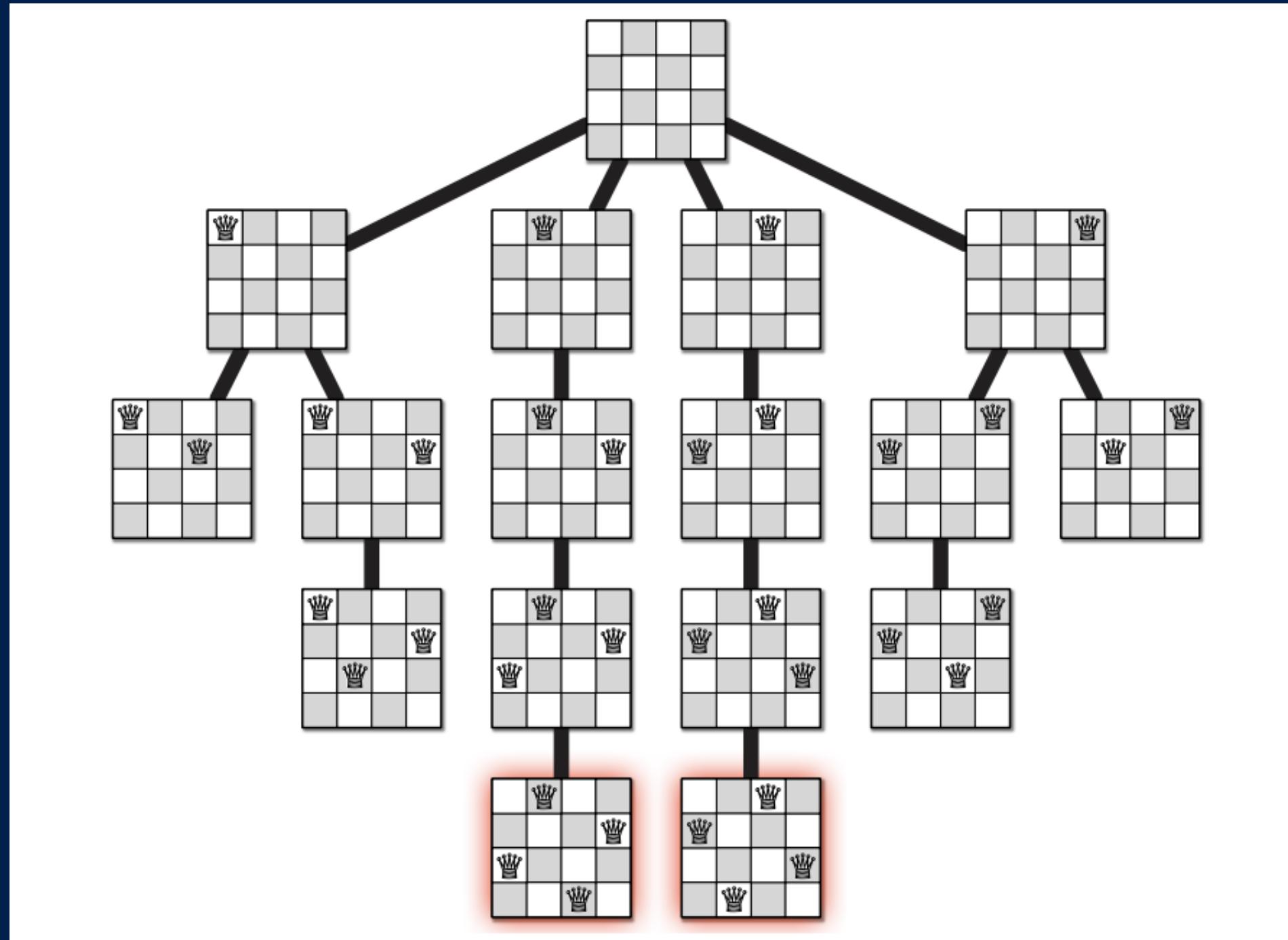


Permasalahan ini harus mengikuti peraturan berikut yaitu ada paling banyak satu ratu di setiap kolom, baris, dan diagonal.

Dalam penyelesaiannya nanti juga akan memiliki 4 elemen penyelesaian yaitu

1. Initial State : yaitu keadaan dimana tidak ada ratu di papan
2. successor function : yaitu menambahkan queen kedalam tempat yang kosong
3. goal test : yaitu memastikan tidak ada ratu yang saling menyerang

8-queens using DFS



Gambar disamping adalah ilustrasi dari decision tree / kemungkinan langkah yang bisa diambil saat menyelesaikan persoalan 8 queens. Bisa dilihat akan muncul berapa banyak kemungkinan langkah. Oleh karena itu kita akan menggunakan sebuah konsep penyelesaian yaitu metode dfs.

DFS bekerja dengan menjelajahi satu jalur di pohon pencarian sejauh mungkin sebelum melakukan backtracking untuk menjelajahi jalur lainnya. Dalam konteks masalah 8 Ratu, DFS akan mencoba menempatkan satu ratu di setiap kolom papan, dimulai dari kiri dan terus ke kanan. Jika seorang ratu tidak dapat ditempatkan di kolom tertentu tanpa mengancam ratu lain, DFS akan mundur dan mencoba posisi yang berbeda untuk ratu sebelumnya.

8-queens using DFS

Metode dfs akan menggunakan metode backtracking yang langkah langkahnya adalah sebagai berikut :

- 1) Mulai di kolom paling kiri
- 2) Jika semua ratu ditempatkan return nilai benar
- 3) Coba semua baris di kolom saat ini.

Lakukan langkah berikut untuk setiap baris yang dicoba.

- a) Jika ratu dapat ditempatkan dengan aman di baris ini tandai [baris, kolom] ini sebagai bagian dari solusi lalu secara rekursif memeriksa apakah menempatkan ratu di sini mengarah ke solusi.
 - b) Jika menempatkan ratu di [baris, kolom] mengarah ke solusi kemudian kembali benar.
 - c) Jika menempatkan ratu tidak menghasilkan solusi, maka hapus tanda [baris, kolom] ini (Mundur) dan pergi ke langkah (a) untuk mencoba baris lain.
-
- 4) Jika semua baris sudah dicoba dan tidak ada yang berhasil, return nilai false untuk memicu backtracking.

8-queens using DFS implementation

```
bool dfs(vector<int>& board, int row) { //passing parameter is the size of board and current row
    // Base case: If all queens are placed, return true
    if (row == 8) {
        return true;
    }

    // Try placing queen in each column of the current row
    for (int col = 0; col < 8; col++) {
        if (isSafe(board, row, col)) { // check if placement is in accordance with the requirements
            board[row] = col; // Place the queen

            // Recur to place the rest of the queens
            if (dfs(board, row + 1)) {
                return true;
            }

            // If placing the queen in this column didn't work,
            // backtrack and try placing the queen in the next column
            board[row] = -1;
        }
    }

    return false; // If no solution is found, return false
}
```

8-queens using DFS implementation

```
bool isSafe(vector<int>& board, int row, int col) {
    // Check if there is a queen in the same column
    for (int i = 0; i < row; i++) {
        if (board[i] == col)
            return false;
    }

    // Check if there is a queen in the diagonal to the left
    for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {
        if (board[i] == j)
            return false;
    }

    // Check if there is a queen in the diagonal to the right
    for (int i = row - 1, j = col + 1; i >= 0 && j < 8; i--, j++) {
        if (board[i] == j)
            return false;
    }

    return true;
}
```

8-queens using DFS implementation

```
void printSolution(vector<int>& board) {  
    for (int i = 0; i < 8; i++) {  
        for (int j = 0; j < 8; j++) {  
            if (board[i] == j) {  
                cout << "Q ";  
            } else {  
                cout << ". ";  
            }  
        }  
        cout << endl;  
    }  
    cout << endl;  
}
```

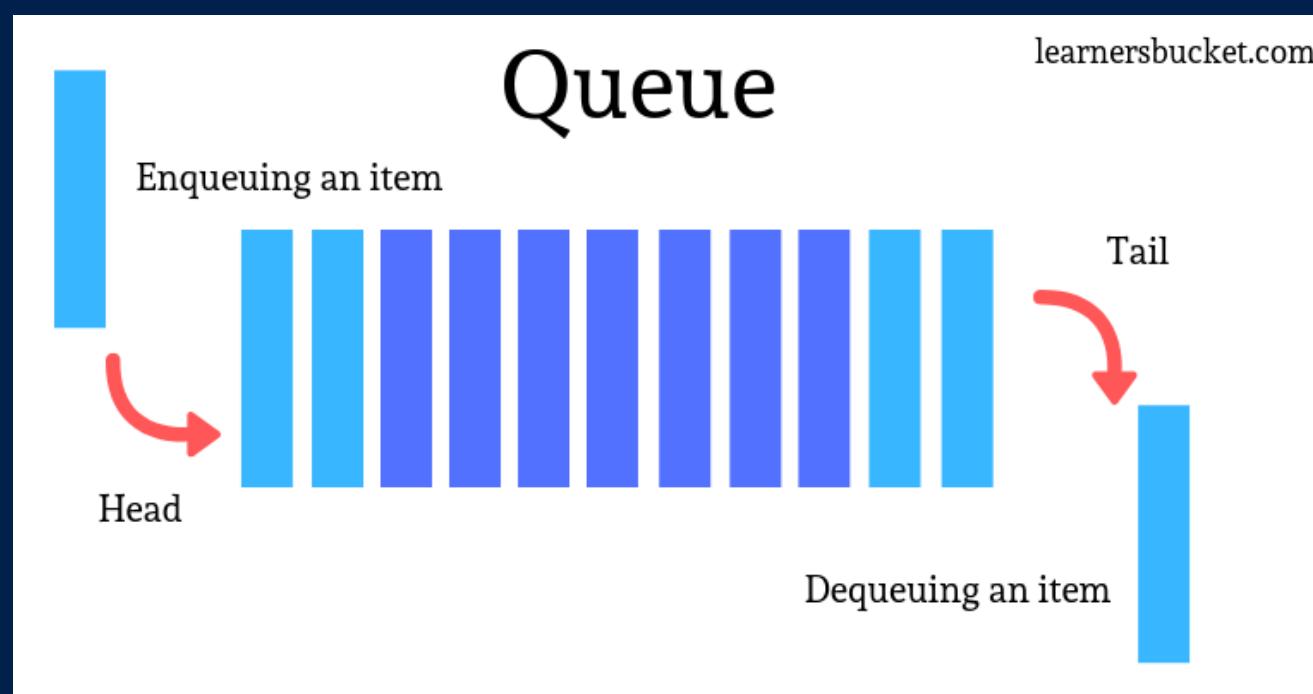
Output :

Q
.	.	.	.	Q	.	.	.
.	Q	.
.	Q	.	.
.	Q	.
.	Q
.	Q
.	.	Q

8-queens using BFS

Masalah 8 Queen juga dapat diselesaikan dengan menggunakan algoritma Breadth-First Search (BFS). Di BFS, kami menjelajahi semua kemungkinan konfigurasi papan, level demi level, mulai dari konfigurasi awal.

Untuk mengimplementasikan BFS untuk masalah 8 Queen, kami menggunakan struktur data queue. Kita mulai dengan memasukkan papan kosong awal ke queue. Kemudian, di setiap iterasi algoritma, kami mengeluarkan papan dari queue depan dan menghasilkan semua kemungkinan konfigurasi papan yang dapat diperoleh dengan menambahkan ratu ke baris papan berikutnya. Kami memasukan queue semua konfigurasi papan yang valid ke belakang antrean.



8-queens using BFS

Kami melanjutkan proses ini hingga kami menemukan konfigurasi papan dengan delapan ratu yang memenuhi semua batasan masalah, yang akan menjadi solusinya. Jika kami menjelajahi semua kemungkinan konfigurasi papan tanpa menemukan solusi, maka kami menyimpulkan bahwa masalahnya tidak dapat diselesaikan.

BFS memastikan bahwa kami menjelajahi semua kemungkinan konfigurasi papan pada level tertentu sebelum melanjutkan ke level berikutnya. Artinya, kami menjamin untuk menemukan solusi optimal dengan jumlah langkah minimum, karena solusi pertama yang ditemukan adalah solusi dengan jumlah langkah minimum. Namun, BFS bisa lebih lambat daripada DFS, karena memerlukan lebih banyak memori untuk menyimpan semua konfigurasi papan perantara.

8-queens using BFS

```
void bfs() {
    queue<State> q;

    // Add initial state to the queue
    vector<int> initial_board(8, -1);
    q.push(State(initial_board, 0));

    // BFS loop
    while (!q.empty()) {
        State current = q.front();
        q.pop();

        // Base case: If all queens are placed, print the board and return
        if (current.row == 8) {
            for (int i = 0; i < 8; i++) {
                for (int j = 0; j < 8; j++) {
                    if (current.board[i] == j)
                        cout << "Q ";
                    else
                        cout << "- ";
                }
                cout << endl;
            }
            return;
        }
    }
}
```

```

    }

    // Try placing queen in each column of the current row
    for (int col = 0; col < 8; col++) {
        if (isSafe(current.board, current.row, col)) {
            vector<int> new_board = current.board;
            new_board[current.row] = col; // Place the queen

            // Add the new state to the queue
            q.push(State(new_board, current.row + 1));
        }
    }

    cout << "No solution found." << endl;
}
```

Q	-	-	-	-	-	-	-
-	-	-	-	-	-	-	Q
-	-	-	-	-	-	-	Q
-	-	-	-	-	-	Q	-
-	-	-	-	-	Q	-	-
-	-	-	-	Q	-	-	-
-	-	-	-	-	-	Q	-
-	-	-	Q	-	-	-	-

8-queens Differences

DFS dan BFS adalah dua algoritma pencarian populer yang digunakan untuk mengeksplorasi semua kemungkinan solusi untuk suatu masalah. Perbedaan utama antara DFS dan BFS adalah urutan mereka mengunjungi node di pohon pencarian.

DFS bekerja dengan menjelajahi satu jalur di pohon pencarian sejauh mungkin sebelum melakukan backtracking untuk menjelajahi jalur lainnya. Dalam konteks masalah 8 Ratu, DFS akan mencoba menempatkan satu ratu di setiap kolom papan, dimulai dari kiri dan terus ke kanan. Jika seorang ratu tidak dapat ditempatkan di kolom tertentu tanpa mengancam ratu lain, DFS akan mundur dan mencoba posisi yang berbeda untuk ratu sebelumnya.

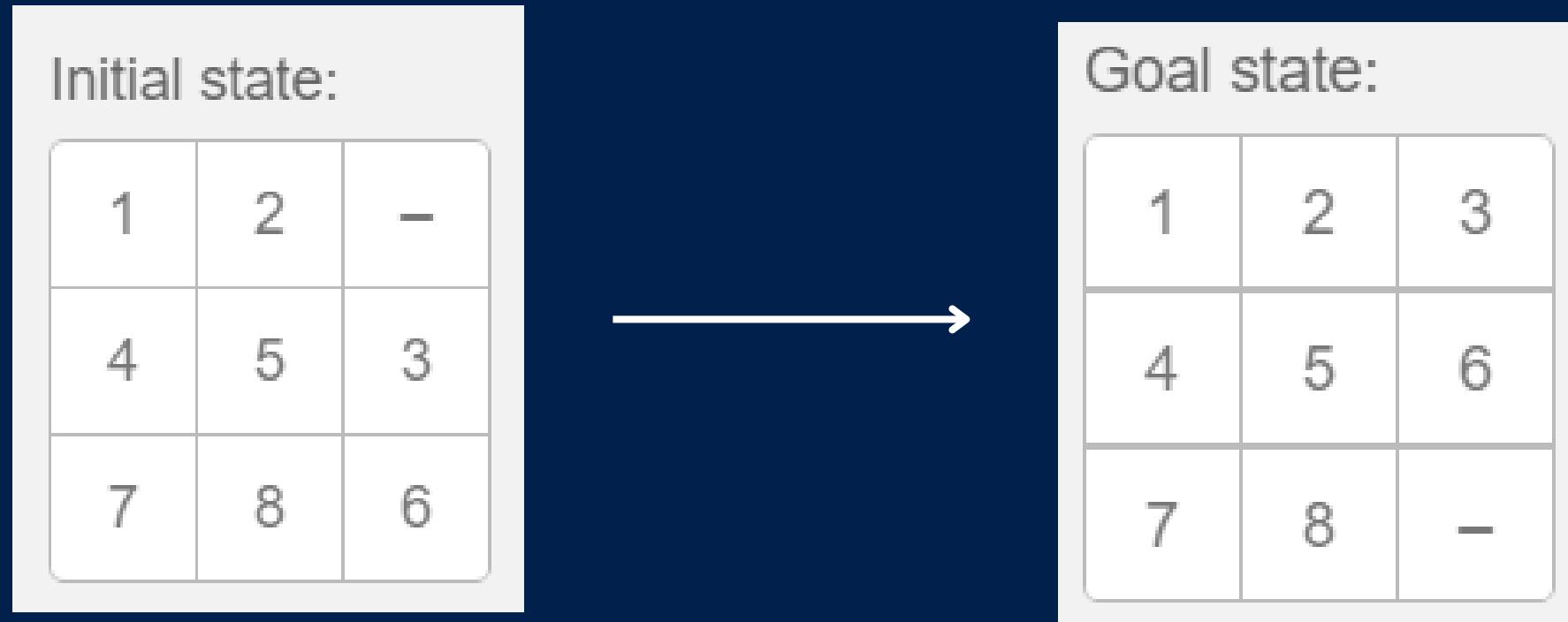
BFS, di sisi lain, mengeksplorasi semua kemungkinan jalur di setiap level pohon pencarian sebelum melanjutkan ke level berikutnya. Dalam konteks masalah 8 Ratu, BFS akan mencoba menempatkan satu ratu di setiap baris papan, mulai dari atas dan terus ke bawah. Jika seorang ratu tidak dapat ditempatkan di baris tertentu tanpa mengancam ratu lainnya, BFS akan berpindah ke baris berikutnya dan mencoba lagi.

8-queens conclusion

Dalam hal efisiensi, DFS umumnya lebih cepat daripada BFS untuk masalah 8 Queens, karena mencapai solusi lebih cepat dengan menjelajahi jalur yang lebih dalam terlebih dahulu. Namun, DFS mungkin tidak selalu menemukan solusi optimal, sedangkan BFS menjamin menemukan solusi terpendek dalam hal jumlah gerakan.

Secara keseluruhan, DFS dan BFS adalah pendekatan yang valid untuk memecahkan masalah 8 Queens, dan pilihan algoritma bergantung pada persyaratan khusus dari masalah dan preferensi programmer.

8-puzzle intro

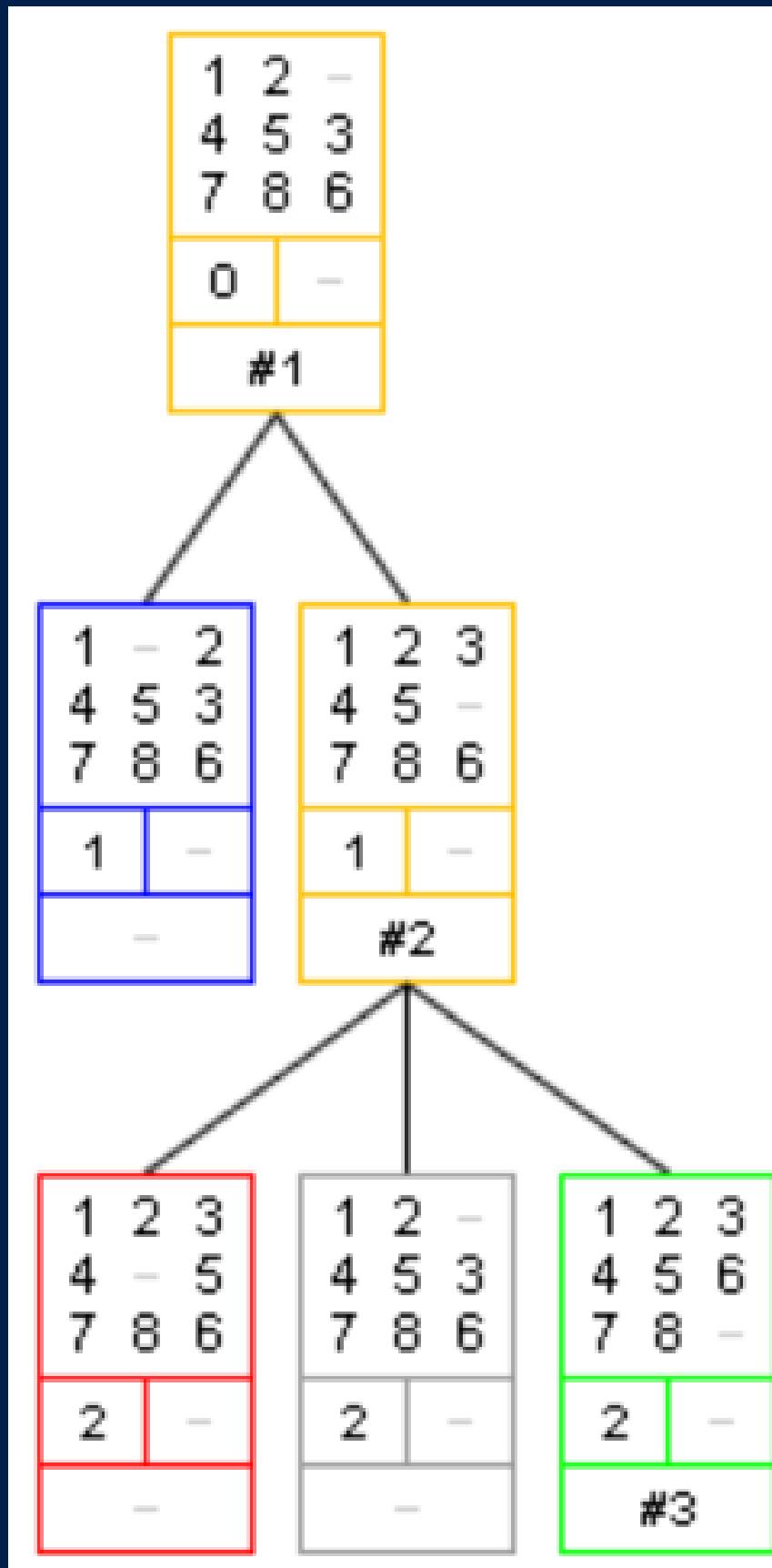


OPERATOR

- (1) Geser kotak kosong ke kanan
- (2) Geser kotak kosong ke kiri
- (3) Geser kotak kosong ke atas
- (4) Geser kotak kosong ke bawah

Terdapat puzzle berukuran 3x3 dengan 8 bilah dimana 1 bilah dikosongkan sehingga puzzle tersebut dapat digerakan ke atas, bawah, kanan dan kiri. Posisi awal yang tidak beraturan menjadi misi awal untuk kita membuat goal state dimana setiap bilah akan berubah menjadi urutan angka.

8-puzzle using DFS



Gambar disamping merupakan ilustrasi dari solusi penyelesaian kasus 8 puzzle dimana terdapat kemungkinan langkah yang akan diambil. seperti pada percabangan level satu. Bilah 2 digerakkan ke kanan dan bilah 3 digerakkan ke atas. Akan dilihat setiap kemungkinan dari pergerakkan setiap bilah, jika sekiranya masih belum mencapai goal state yang sudah ditentukan maka akan dibuka percabangan baru.

8-puzzle using DFS

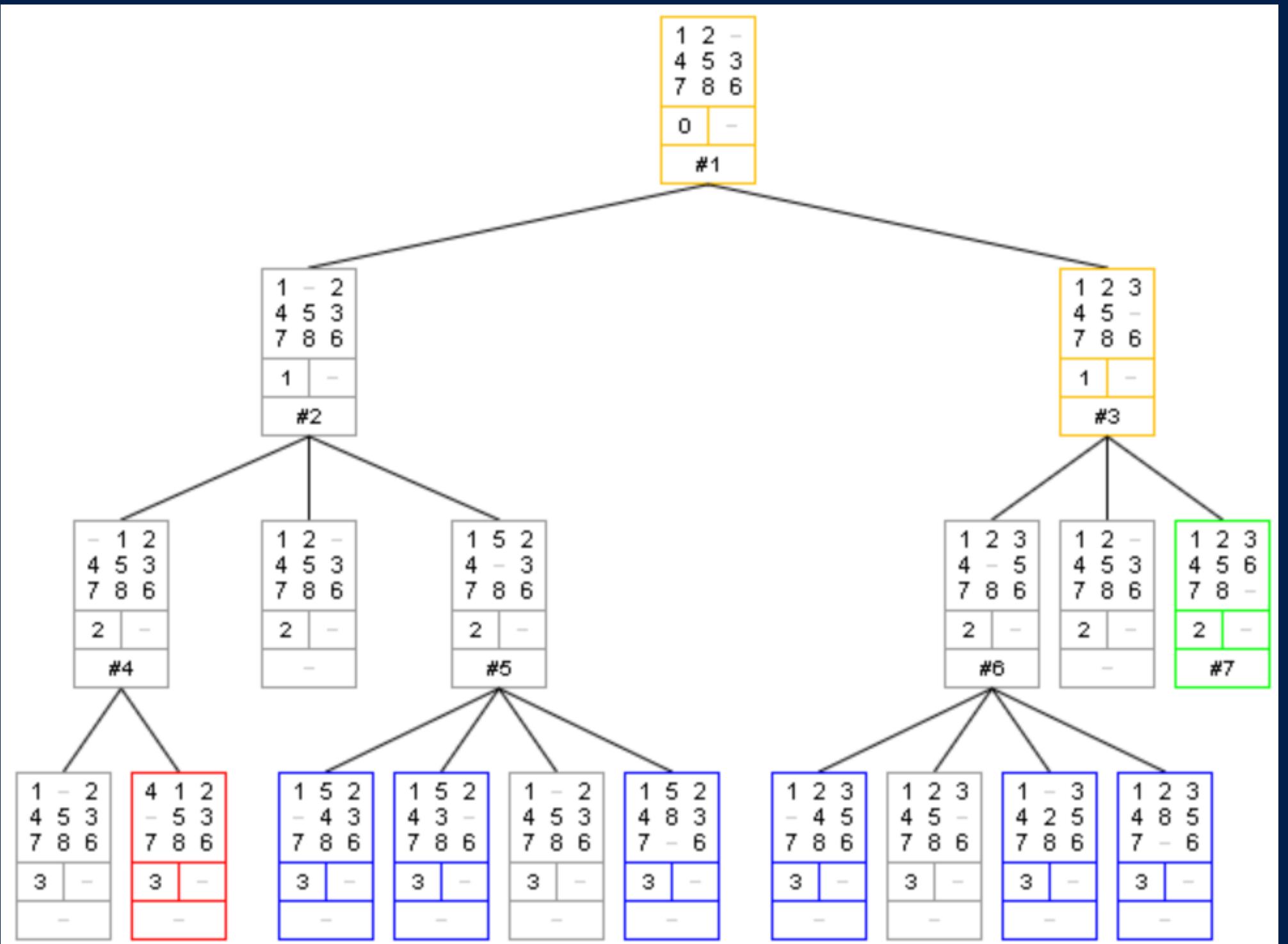
Metode dfs akan menggunakan metode backtracking yang langkah langkahnya adalah sebagai berikut :

- 1) Lakukan pemindahan bilah yang memungkinkan
- 2) Setelah bilah dipindahkan akan terdapat kemungkinan pemindahan yang baru
- 3) Pemindahan tersebut terus dilakukan tetapi dengan batasan agar tidak melakukan state yang sama yang sudah di lakukan
- 4) Selama state baru dibuat di cek apakah letak bilah sudah benar dengan bilah tujuan akhir yang diinginkan
- 5) Apabila letak bilah sudah benar maka bilah yang belum benar akan terus di ubah
- 6) Penyelesaian dengan DFS memungkinkan perulangan yang bisa sampai tidak terbatas apabila tidak diberikan batasan yang menyebabkan terjadinya looping

8-puzzle using DFS implementation

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer:** Shows two files: "8-Puzzle-BFS.py" and "8-puzzle-dfs.py".
- Editor:** The active file is "8-puzzle-dfs.py". The code implements a Depth-First Search (DFS) algorithm for an 8-puzzle. It defines a "Node" class, initializes start and goal states using NumPy arrays, and creates a "Puzzle" object. The "solve" method is called, followed by "print".
- Terminal:** The terminal output shows the command "python -u "c:\Users\gienawaty\tanudin\Downloads\8-puzzle-dfs.py"" being run. The output displays the initial state ([[1, 2, 3], [4, 5, 6], [7, 8, 0]]) and the message "Goal Reached!!".
- Status Bar:** Shows the current line (Ln 72), column (Col 48), and other settings like spaces (Spaces: 4), encoding (UTF-8), and file type (Python 3.11.0 64-bit).



Pada solusi menggunakan BFS setiap kemungkinan akan di fokuskan pada satu level terlebih dahulu. Dimulai dengan melakukan pengaplikasian keempat operator dari membuka percabangan node yang paling kiri. jika masih belum memenuhi goal state yang ditentukan maka lakukan perulangan. dimana akan melakukan pengecekan pada setiap node dari paling kiri dan membuka percabangan baru lagi.

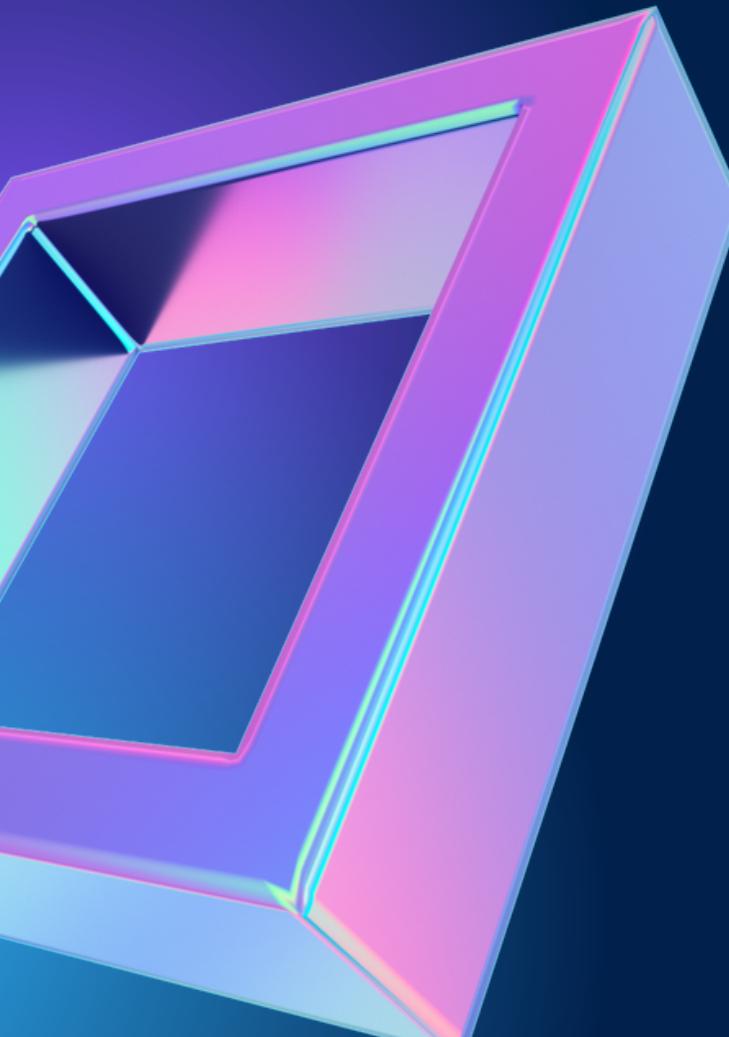
8-puzzle using BFS

8-puzzle using BFS

Metode dfs akan menggunakan metode backtracking yang langkah langkahnya adalah sebagai berikut :

- 1) Mulai memindahkan bilah yang bisa dipindah yaitu yang berada disekitar bilah kosong
- 2) Pada 1 bilah kosong memungkin terjadinya 2 - 4 kemungkinan bilah yang di gerakkan
- 3) Pada level subtree yang sama akan selalu di cek node mana yang memiliki posisi yang benar
- 4) Apabila sudah berada di posisi yang benar maka node tersebut yang akan dikembangkan dan memiliki subtree yang baru sehingga tidak seluruh kemungkinan akan dicoba satu persatu

8-puzzle using DFS implementation



Thank You