Nama: Dewa Bagus Putu Arya Dhananjaya

NPM: 10122362

Kelas: 3KA21

```
# Program Python3 untuk mencetak jalur dari simpul awal
# ke simpul tujuan untuk puzzle N*N-1 (misal: 8-puzzle)
# menggunakan algoritma Branch and Bound
# Program ini mengasumsikan bahwa puzzle dapat diselesaikan

# Mengimpor modul copy untuk fungsi deepcopy
import copy

# Mengimpor fungsi heap dari pustaka Python
# untuk digunakan sebagai antrian prioritas
from heapq import heappush, heappop

# Variabel ini bisa diubah untuk menentukan ukuran puzzle
# 3 untuk 8-puzzle (3x3), 4 untuk 15-puzzle (4x4), dst.
ukuran = 3

# Arah gerak: bawah, kiri, atas, kanan
baris = [1, 0, -1, 0]
kolom = [0, -1, 0, 1]
```

```
# Kelas untuk Antrian Prioritas
class AntrianPrioritas:

    # Konstruktor untuk menginisialisasi Antrian Prioritas
    def __init__(self):
        self.tumpukan = []

    # Menambahkan elemen baru 'k' ke dalam antrian
    def tambah(self, k):
        heappush(self.tumpukan, k)

    # Menghapus dan mengembalikan elemen dengan prioritas tertinggi (nilai terkecil)
    def ambil(self):
        return heappop(self.tumpukan)

    # Mengecek apakah antrian kosong
    def kosong(self):
        if not self.tumpukan:
            return True
        else:
            return False
```

```
# Struktur simpul (node)
class Simpul:

    def __init__(self, induk, matriks, posisi_kosong, biaya, tingkat):
        # Menyimpan node induk dari node saat ini,
```

```
        # berguna untuk menelusuri jalur saat solusi ditemukan
        self.induk = induk

        # Menyimpan matriks puzzle
        self.matriks = matriks

        # Menyimpan posisi ubin kosong dalam matriks
        self.posisi_kosong = posisi_kosong

        # Menyimpan jumlah ubin yang salah tempat
        self.biaya = biaya

        # Menyimpan jumlah langkah (pindahan) sejauh ini
        self.tingkat = tingkat

    # Metode ini didefinisikan agar antrian prioritas
    # dapat mengurutkan berdasarkan nilai biaya (cost)
    def __lt__(self, berikutnya):
        return self.biaya < berikutnya.biaya


# Fungsi untuk menghitung jumlah ubin yang salah tempat,
# yaitu jumlah ubin (selain kosong) yang belum berada di posisi akhir yang benar
def hitung_ubin_salah(temp_matriks, matriks_tujuan):
    salah = 0
    for i in range(len(temp_matriks)):
        for j in range(len(temp_matriks[i])):
            if temp_matriks[i][j] != 0 and temp_matriks[i][j] != matriks_tujuan[i][j]:
                salah += 1
    return salah
```

```
# Fungsi untuk menghitung jumlah ubin yang salah tempat
def hitungBiaya(matriks, matriks_tujuan) -> int:
    hitung = 0
    for i in range(n):
        for j in range(n):
            if ((matriks[i][j]) and (matriks[i][j] != matriks_tujuan[i][j])):
                hitung += 1
    return hitung

# Fungsi untuk membuat simpul (node) baru
def simpulBaru(matriks, posisi_kosong, posisi_baru_kosong,
               tingkat, induk, matriks_tujuan) -> Simpul:

    # Salin data dari matriks induk ke matriks baru
    matriks_baru = copy.deepcopy(matriks)

    # Pindahkan ubin satu posisi
    x1, y1 = posisi_kosong
    x2, y2 = posisi_baru_kosong
    matriks_baru[x1][y1], matriks_baru[x2][y2] = matriks_baru[x2][y2], matriks_baru[x1][y1]

    # Hitung jumlah ubin yang salah tempat
    biaya = hitungBiaya(matriks_baru, matriks_tujuan)

    simpul_baru = Simpul(induk, matriks_baru, posisi_baru_kosong, biaya, tingkat)
    return simpul_baru

# Fungsi untuk mencetak matriks NxN
def cetakMatriks(matriks):
```

```
    for i in range(n):
        for j in range(n):
            print("%d " % (matriks[i][j]), end=" ")
        print()

# Fungsi untuk mengecek apakah koordinat matriks (x, y) valid
def aman(x, y):
    return x >= 0 and x < n and y >= 0 and y < n

# Fungsi untuk mencetak jalur dari simpul akar ke simpul tujuan
def cetakJalur(akar):
    if akar == None:
        return
    cetakJalur(akar.induk)
    cetakMatriks(akar.matriks)
    print()

# Fungsi utama untuk menyelesaikan puzzle NxN-1
# menggunakan algoritma Branch and Bound
# posisi_kosong adalah posisi ubin kosong pada kondisi awal
def selesaikan(matriks_awal, posisi_kosong, matriks_tujuan):

    # Buat antrian prioritas untuk menyimpan simpul aktif
    antrian = AntrianPrioritas()

    # Buat simpul akar (root)
    biaya = hitungBiaya(matriks_awal, matriks_tujuan)
    akar = Simpul(None, matriks_awal, posisi_kosong, biaya, 0)

    # Tambahkan akar ke antrian
    antrian.tambah(akar)

    # Jalankan hingga ditemukan solusi
    while not antrian.kosong():

        # Ambil simpul dengan biaya terkecil
        minimum = antrian.ambil()

        # Jika simpul tersebut adalah tujuan
        if minimum.biaya == 0:
            cetakJalur(minimum)
            return

        # Bangkitkan semua anak dari simpul tersebut
        for i in range(n):
            posisi_baru = [
                minimum.posisi_kosong[0] + row[i],
                minimum.posisi_kosong[1] + col[i]
            ]

            if aman(posisi_baru[0], posisi_baru[1]):
                anak = simpulBaru(minimum.matriks,
                                  minimum.posisi_kosong,
                                  posisi_baru,
                                  minimum.tingkat + 1,
                                  minimum,
                                  matriks_tujuan)
                antrian.tambah(anak)
```

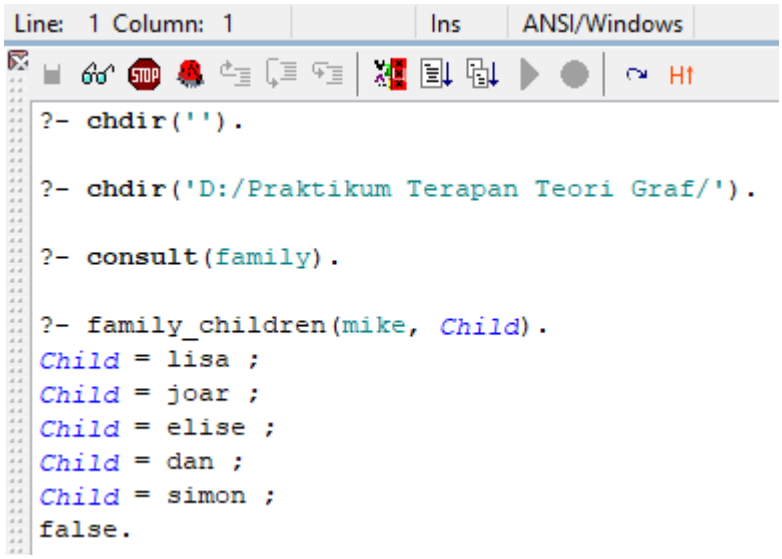
```
# Kode Utama (Driver Code)
```

```
# Konfigurasi awal
# Nilai 0 digunakan untuk ruang kosong
matriks_awal = [
    [ 1, 2, 3 ],
    [ 5, 6, 0 ],
    [ 7, 8, 4 ]
]

# Konfigurasi akhir (solusi yang dapat diselesaikan)
# Nilai 0 digunakan untuk ruang kosong
matriks_tujuan = [
    [ 1, 2, 3 ],
    [ 5, 8, 6 ],
    [ 0, 7, 4 ]
]

# Koordinat ubin kosong dalam konfigurasi awal
posisi_kosong = [1, 2]

# Pemanggilan fungsi untuk menyelesaikan puzzle
selesaikan(matriks_awal, posisi_kosong, matriks_tujuan)

# Kode ini disumbangkan oleh Kevin Joshi
```

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

1  2  3
5  0  6
7  8  4

1  2  3
5  8  6
7  0  4

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

family.pl

```
male(lennart).
male(mike).
male(donald).
male(usain).
male(joar).
male(adam).
male(dan).
male(simon).
female(hillary).
female(elise).
female(lisa).
female(lena).

parent(mike, lennart).
parent(mike, lena).
parent(lennart, donald).
parent(lennart, hillary).
parent(lennart, usain).
parent(lena, adam).
parent(lena, simon).
parent(adam, dan).
parent(donald, lisa).
parent(hillary, joar).
parent(hillary, elise).


child(lisa).
child(joar).
child(elise).
child(dan).
child(simon).

%% predicate rules
father(X,Y) :- male(X),parent(X,Y).
mother(X,Y) :- female(X),parent(X,Y).
son(X,Y) :- male(X),parent(Y,X).
daughter(X,Y) :- female(X),parent(Y,X).

family_children(X, X):-
    child(X).

family_children(X, Child):-
    parent(X,Y),
    family_children(Y, Child).
```



Penjelasan:

Kode program Prolog tersebut merepresentasikan sebuah silsilah keluarga melalui fakta dan aturan logika. Pertama-tama, program mendefinisikan fakta mengenai jenis kelamin dari setiap individu menggunakan predikat male(X) untuk laki-laki dan female(X) untuk perempuan. Misalnya, male(lennart) menyatakan bahwa Lennart adalah seorang laki-laki, sementara female(hillary) menyatakan bahwa Hillary adalah seorang perempuan. Selanjutnya, hubungan orang tua dan anak didefinisikan dengan predikat parent(X, Y), yang berarti X adalah orang tua dari Y. Sebagai contoh, parent(mike, lennart) menunjukkan bahwa Mike adalah ayah atau ibu dari Lennart, tergantung jenis kelaminnya.

Selain itu, program juga mencantumkan fakta child(X) untuk menunjukkan siapa saja yang merupakan anak dalam silsilah tersebut, meskipun tanpa menyebutkan siapa orang tuanya. Kemudian, aturan-aturan logis atau *predicate rules* digunakan untuk mendefinisikan hubungan lebih lanjut seperti father(X, Y) yang berarti X adalah ayah dari Y, dan berlaku jika X adalah laki-laki serta merupakan orang tua dari Y. Begitu juga, mother(X, Y) menunjukkan hubungan ibu, dan son(X, Y) atau daughter(X, Y) menggambarkan anak laki-laki atau perempuan dari Y berdasarkan jenis kelamin masing-masing.

Program ini juga menggunakan aturan rekursif family_children(X, Child) untuk melacak semua keturunan dari seseorang, termasuk anak, cucu, dan cicit. Aturan ini bekerja dengan cara mengecek apakah X adalah orang tua dari Y, kemudian mencari semua keturunan dari Y hingga menemukan individu yang diklasifikasikan sebagai child. Dengan pendekatan deklaratif ini, kode Prolog mampu membangun dan menjelajahi struktur keluarga secara sistematis, berdasarkan relasi antarindividu dan sifat logis yang diturunkan dari data yang ada.