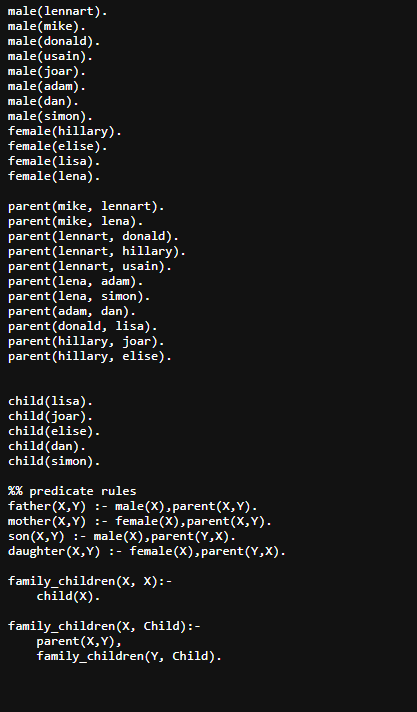
MUHAMMAD TARMIDZI BARIQ

51422161

3IA11

M8



* male(lennart). % Lennart adalah laki-laki
* female(hillary). % Hillary adalah perempuan

Ini menyatakan jenis kelamin seseorang. Fakta ini digunakan untuk mendefinisikan siapa yang laki-laki (male) dan siapa yang perempuan (female).

parent(mike, lennart).

parent(lennart, donald).

Fakta ini menyatakan hubungan orang tua → anak:

child(lisa).

Ini menyatakan bahwa lisa adalah seorang anak (tidak dijelaskan anak siapa — ini bisa digunakan sebagai simpul terminal/leaf dalam pohon keluarga).

father(X,Y) :- male(X), parent(X,Y).

Artinya: X adalah ayah dari Y jika:

* X adalah laki-laki, dan
* X adalah orang tua dari Y.

mother(X,Y) :- female(X), parent(X,Y).

Artinya: X adalah ibu dari Y jika:

* X adalah perempuan, dan
* X adalah orang tua dari Y.

son(X,Y) :- male(X), parent(Y,X).

Artinya: X adalah anak laki-laki dari Y jika:

* X adalah laki-laki, dan
* Y adalah orang tua dari X.

daughter(X,Y) :- female(X), parent(Y,X).

Artinya: X adalah anak perempuan dari Y jika:

* X adalah perempuan, dan
* Y adalah orang tua dari X.

family\_children(X, X) :- child(X).

Artinya: jika X adalah anak (terminal), maka X adalah salah satu family\_children.

family\_children(X, Child) :-

parent(X,Y),

family\_children(Y, Child).

Artinya:

Jika X adalah orang tua dari Y, dan Y memiliki anak cucu (descendant) bernama Child, maka Child adalah keturunan dari X.

?- family\_children(mike, dan).

Artinya: apakah dan adalah salah satu keturunan (anak/cucu) dari mike.



import copy

from heapq import heappush, heappop

* copy: digunakan untuk deep copy matriks puzzle tanpa mengubah aslinya.
* heapq: pustaka Python untuk Priority Queue, dipakai untuk memilih node dengan cost terkecil.

n = 3

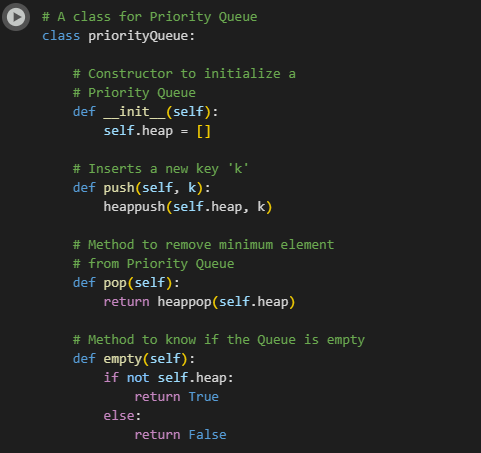
Menentukan ukuran puzzle (3×3 → 8-puzzle).

row = [1, 0, -1, 0]

col = [0, -1, 0, 1]

Menyatakan arah perpindahan tile kosong:

* row/col digunakan untuk menghitung koordinat baru.
* Kombinasi ini menghasilkan arah: bawah, kiri, atas, kanan.



def \_\_init\_\_(self):

self.heap = []

* Konstruktor yang menginisialisasi atribut heap sebagai list kosong, yang akan digunakan sebagai heap (priority queue).
* Secara internal, heapq memanipulasi list agar berperilaku seperti min-heap.

def push(self, k):

heappush(self.heap, k)

* Menambahkan elemen k ke heap.
* Elemen k harus dapat dibandingkan (memiliki urutan), biasanya tuple (cost, node).

def pop(self):

return heappop(self.heap)

* Menghapus dan mengembalikan elemen dengan prioritas tertinggi (nilai terkecil).
* Ini akan menjadi node dengan cost function terkecil dalam algoritma pencarian.

def empty(self):

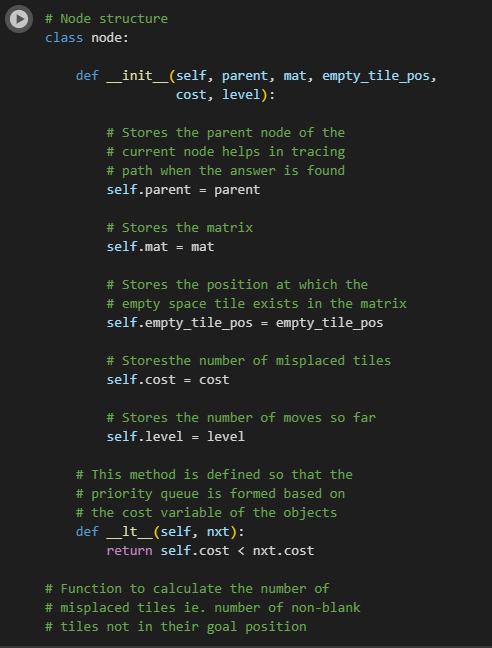
if not self.heap:

return True

else:

return False

* Mengembalikan True jika queue kosong.
* not self.heap adalah cara Pythonic untuk memeriksa list kosong.



def \_\_init\_\_(self, parent, mat, empty\_tile\_pos, cost, level):

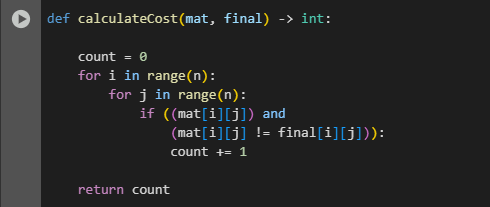
Parameter ini menyimpan semua informasi penting untuk satu simpul (state):

* parent – node sebelumnya (untuk merekonstruksi jalur solusi).
* mat – matrix puzzle saat ini (konfigurasi posisi ubin).
* cost – total biaya (misalnya jumlah ubin yang salah tempat).
* empty\_tile\_pos – posisi (baris, kolom) dari ubin kosong.
* level – jumlah langkah dari titik awal (depth atau level tree).

ef \_\_lt\_\_(self, nxt):

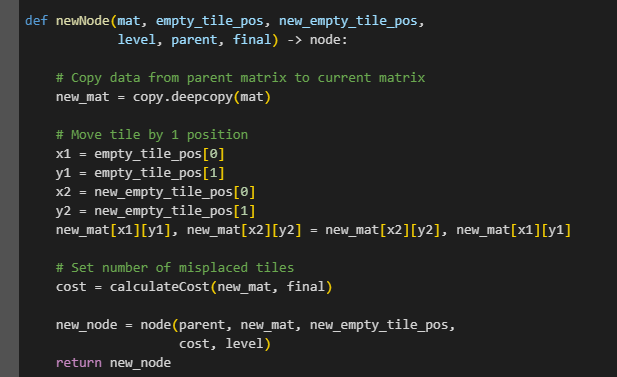
return self.cost < nxt.cost

* Mendefinisikan bagaimana dua node dibandingkan saat dimasukkan ke dalam priority queue.
* Priority Queue akan memilih node dengan cost terendah, sesuai dengan strategi pencarian heuristik.
* Metode \_\_lt\_\_ (less than) digunakan secara otomatis oleh heapq.



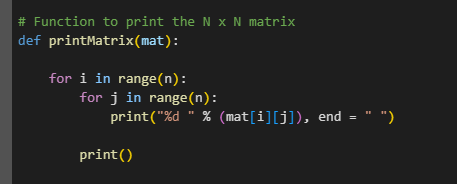
Fungsi ini menghitung heuristik berupa jumlah ubin yang salah tempat (misplaced tiles), yaitu:

* Tidak kosong (!= 0)
* Tidak sesuai posisi pada final.

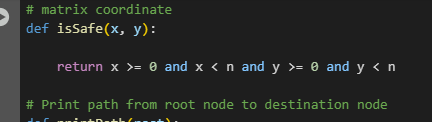


🔹 Fungsi ini membuat node baru dengan:

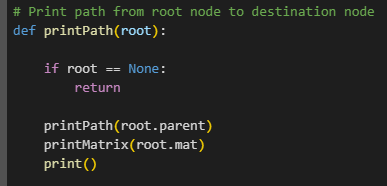
* Menduplikasi matrix dari parent.
* Menukar posisi ubin kosong ke arah gerakan baru.
* Menghitung cost baru (jumlah ubin salah tempat).
* Membuat objek node baru yang menyimpan konfigurasi ini.



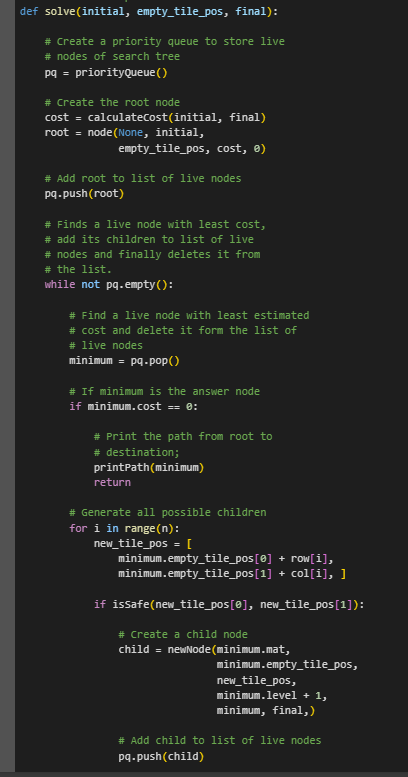
Fungsi sederhana untuk menampilkan puzzle dalam bentuk matriks 2D di terminal.



Mengecek apakah koordinat (x, y) valid dalam matriks puzzle ukuran n x n.



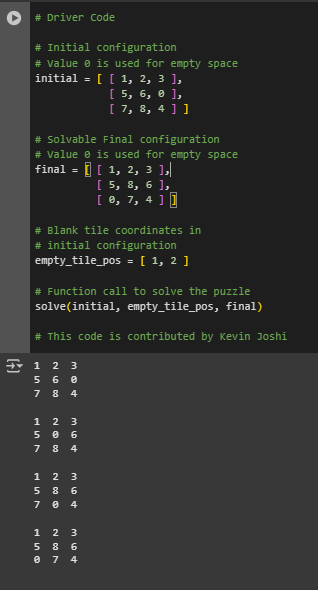
* Mencetak jalur solusi dari akar hingga simpul solusi:
* Menggunakan rekursi ke parent, lalu mencetak mat saat ini.
* Ini memastikan jalurnya dicetak berurutan dari awal ke akhir.



Fungsi utama yang menyelesaikan puzzle menggunakan strategi Branch and Bound dengan priority queue.

Alur Eksekusi:

* Inisialisasi Priority Queue dan buat node awal (root) dengan level 0 dan cost awal.
* Selama queue tidak kosong:
* Ambil node dengan cost terkecil.
* Jika cost-nya 0 → puzzle selesai → cetak jalur.
* Jika belum selesai:
* Cek semua kemungkinan gerakan (4 arah: atas, bawah, kiri, kanan).
* Jika posisi valid (isSafe) → buat newNode, hitung cost, dan push ke queue.



initial = [

[ 1, 2, 3 ],

[ 5, 6, 0 ],

[ 7, 8, 4 ]

]

* Ini adalah puzzle awal (3x3 atau 8-puzzle).
* 0 adalah ubin kosong (digunakan untuk tukar posisi).

final = [

[ 1, 2, 3 ],

[ 5, 8, 6 ],

[ 0, 7, 4 ]

]

Ini adalah konfigurasi yang ingin dicapai.

empty\_tile\_pos = [1, 2]

Menunjukkan bahwa ubin kosong berada di baris ke-1, kolom ke-2 pada initial.

solve(initial, empty\_tile\_pos, final)

* Ini akan memulai algoritma Branch and Bound untuk mencari solusi dari konfigurasi awal ke konfigurasi akhir.
* Jika solusi ditemukan, akan dicetak jalur langkah demi langkah dari awal sampai akhir.