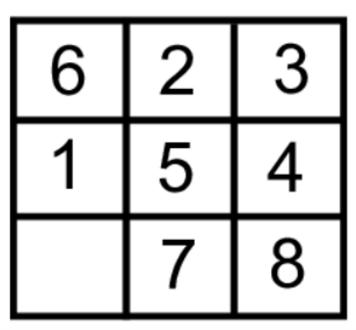
Nama : Abdan Hafidz

NRP : 5054231021

**Tugas Searching KKA**

**Deskripsi Soal**

Diberikan sebuah puzzle case 8-grid-puzzle sebagai berikut :



Kita diminta untuk menyelesaikan permasalahan tersebut menggunakan algoritma *informed search* dan *uninformed search* kemudian menyajikan perhitungan statistik jumlah node yang dihasilkan, nilai depth, runtime, dan memory usage untuk setiap algoritma yang digunakan dengan program python.

**Penyelesaian**

Di sini saya memilih untuk menggunakan algoritma *Uninformed Search : Breadth First Search, Uniform Cost Search,* dan *Iterative Deepening Search.*

Dengan memanfaatkan template fungsi yang telah disajikan melalui sumber <https://github.com/aimacode/aima-python/blob/master/search4e.ipynb> kita dapat melakukan pemanggilan fungsi report sebagai berikut :

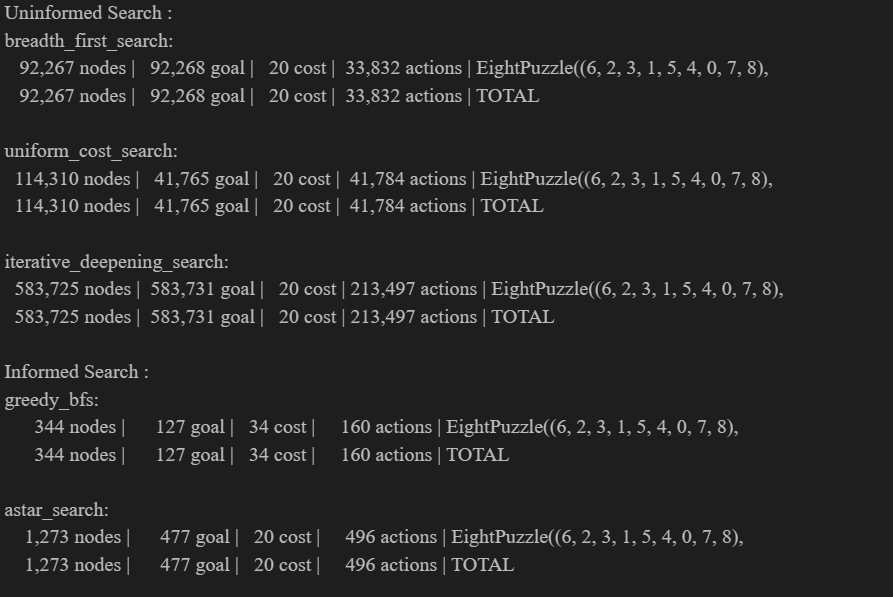
print("Uninformed Search : ")

report([breadth\_first\_search,uniform\_cost\_search,iterative\_deepening\_search], [soal])

print("Informed Search : ")

report([greedy\_bfs, astar\_search], [soal])

Didapatkan outputnya adalah :



Sekarang kita akan mencoba memperhitungkan alokasi memori dari algoritma di atas menggunakan bantuan library memory\_profiler :

# importing the library

from memory\_profiler import profile

# instantiating the decorator

@profile

def calc():

    breadth\_first\_search(soal)

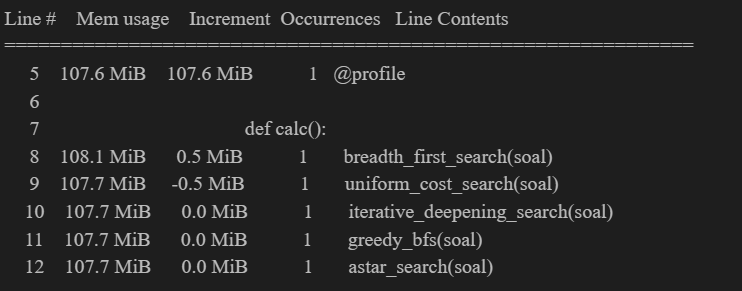
    uniform\_cost\_search(soal)

    iterative\_deepening\_search(soal)

    greedy\_bfs(soal)

    astar\_search(soal)

calc()



Selanjutnya kita akan menghitung run time dari masing – masing fungsi algoritma pencarian

import time

    start\_time = time.time()

    breadth\_first\_search(soal)

    end\_time = time.time()

    print("BFS : ",abs(start\_time - end\_time))

    start\_time = time.time()

    uniform\_cost\_search(soal)

    end\_time = time.time()

    print("Uniform Cost Search :",abs(start\_time - end\_time))

    start\_time = time.time()

    iterative\_deepening\_search(soal)

    end\_time = time.time()

    print("Iterative Deepening Search :",abs(start\_time - end\_time))

    start\_time = time.time()

    greedy\_bfs(soal)

    end\_time = time.time()

    print("Greedy BFS :",abs(start\_time - end\_time))

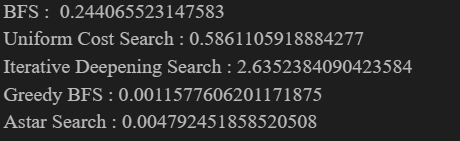
    start\_time = time.time()

    astar\_search(soal)

    end\_time = time.time()

    print("Astar Search :",abs(start\_time - end\_time))

Diperoleh hasil perhitungan



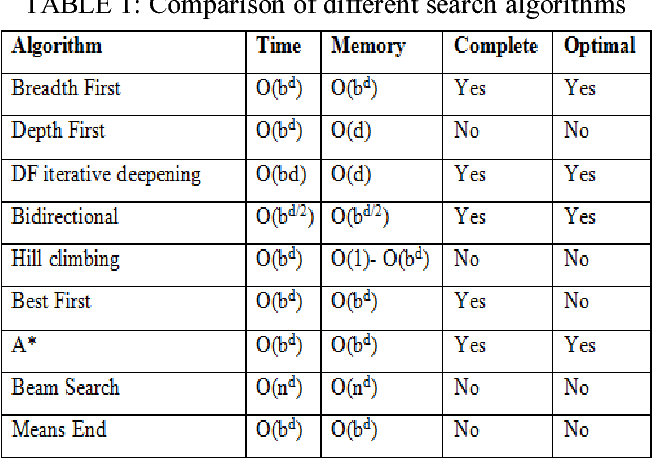
Berdasarkan hasil uji coba diperoleh bahwa urutan algoritma berdasarkan alokasi memori adalah :

***Greedy BFS < Astar Search < Iteratif Deepening Search < Unfiorm Cost Search < Breadth First Search***

Serta urutan algoritma berdasarkan kecepatan waktu eksekusi adalah :

***Greedy BFS < Astar Search < Breadth First Search < Uniform Cost Search Iteratif Deepening Search***

Ini membuktikan formula dari kompleksitas masing – masing algoritma :



**TUGAS TAMBAHAN**

Membuat interfaces menu yang menjalankan beberapa algoritma pencarian :

*Lebih lengkapnya anda bisa melihat pada file notebook* ***interface.ipynb.***

# Solve an 8 puzzle problem and print out each state

def search(puzzle,func):

    Puz = EightPuzzle((puzzle[0],puzzle[1],puzzle[2],puzzle[3],puzzle[4],puzzle[5],puzzle[6],puzzle[7],puzzle[8]))

    for s in path\_states(func(Puz)):

        print(board8(s))

print("--- SEARCHING ALGORITHM ---")

print("Silahkan masukkan 8-puzzlenya dengan format array dari kiri ke kanan!")

Puzzle = list(map(int,input().split()))

print("Pilih Menu Searching :")

print("\*Uninformed Search :\* ")

print("1.Breadth First Search")

print("2.Depth First Search")

print("3.Iterative Deepening Search")

print("\*Informed Search :\* ")

print("4.A Star Search")

print("5.Greedy Best First Search")

cmd = int(input("Masukkan angka [1/2/3/4/5] :"))

dict = {

    1:search(Puzzle,breadth\_first\_search),

    2:search(Puzzle,depth\_first\_bfs),

    3:search(Puzzle,iterative\_deepening\_search),

    4:search(Puzzle,astar\_search),

    5:search(Puzzle,greedy\_bfs)

}

dict[cmd]