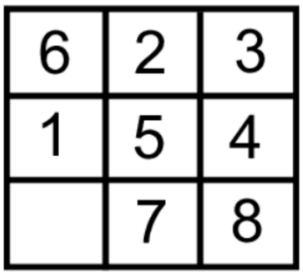
Nama : Abdan Hafidz
NRP : 5054231021

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_dee
pening_search], [soal])

print("Informed Search : ")
report([greedy_bfs, astar_search], [soal])
```

Didapatkan outputnya adalah :

```
Uninformed Search :
breadth_first_search:
  92,267 nodes |  92,268 goal |  20 cost |  33,832 actions | EightPuzzle((6, 2, 3, 1, 5, 4, 0, 7, 8),
  92,267 nodes |  92,268 goal |  20 cost |  33,832 actions | TOTAL

uniform_cost_search:
 114,310 nodes |  41,765 goal |  20 cost |  41,784 actions | EightPuzzle((6, 2, 3, 1, 5, 4, 0, 7, 8),
 114,310 nodes |  41,765 goal |  20 cost |  41,784 actions | TOTAL

iterative_deepening_search:
 583,725 nodes | 583,731 goal |  20 cost | 213,497 actions | EightPuzzle((6, 2, 3, 1, 5, 4, 0, 7, 8),
 583,725 nodes | 583,731 goal |  20 cost | 213,497 actions | TOTAL

Informed Search :
greedy_bfs:
     344 nodes |     127 goal |  34 cost |     160 actions | EightPuzzle((6, 2, 3, 1, 5, 4, 0, 7, 8),
     344 nodes |     127 goal |  34 cost |     160 actions | TOTAL

astar_search:
   1,273 nodes |     477 goal |  20 cost |     496 actions | EightPuzzle((6, 2, 3, 1, 5, 4, 0, 7, 8),
   1,273 nodes |     477 goal |  20 cost |     496 actions | TOTAL
```

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()
```

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     5    107.6 MiB    107.6 MiB           1   @profile
     6
     7                                         def calc():
     8    108.1 MiB      0.5 MiB           1       breadth_first_search(soal)
     9    107.7 MiB     -0.5 MiB           1       uniform_cost_search(soal)
    10    107.7 MiB      0.0 MiB           1       iterative_deepening_search(soal)
    11    107.7 MiB      0.0 MiB           1       greedy_bfs(soal)
    12    107.7 MiB      0.0 MiB           1       astar_search(soal)
```

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

```
BFS :  0.244065523147583
Uniform Cost Search : 0.5861105918884277
Iterative Deepening Search : 2.6352384090423584
Greedy BFS : 0.0011577606201171875
Astar Search : 0.004792451858520508
```

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 :

TABLE 1: Comparison of different search algorithms

Algorithm	**Time**	**Memory**	**Complete**	**Optimal**
Breadth First	$O(b^d)$	$O(b^d)$	Yes	Yes
Depth First	$O(b^d)$	O(d)	No	No
DF iterative deepening	O(bd)	O(d)	Yes	Yes
Bidirectional	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes
Hill climbing	$O(b^d)$	O(1)- $O(b^d)$	No	No
Best First	$O(b^d)$	$O(b^d)$	Yes	No
A*	$O(b^d)$	$O(b^d)$	Yes	Yes
Beam Search	$O(n^d)$	$O(n^d)$	No	No
Means End	$O(b^d)$	$O(b^d)$	No	No

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]
```