Documentatie Tema 1

1. Pentru a apela programul vom folosi comanda python in terminal. Avem urmatoarele exemple de apel:

python main.py input output 4 10

python main.py input output 5 100

De asemenea putem apela si cu mai putini parametrii din terminal sau de la butonul de run.

python main.py input output 2

python main.py input

6.

Euristica admisibila 1 a fost aleasa, deoarece nestiind exact ce stive se vor goli nu vom putea spune care blocuri nu sunt la locul lor. Astfel, am ordonat listele dupa lungime in ordine crescatoare si le-am adunat pe primele K si la aceasta suma am adunat costul minim al unei mutari, adica 1.

Euristica admisibila 2 a fost aleasa ca o imbunatatire a primei euristici, deoarece tinem cont de forma unui bloc. Astfel, vom aduna la suma respectiva costul real unei mutari.

Euristica neadmisibila se bazeaza pe un rationament asemanator, dar pentru a nu indeplini conditia de admisibiliatate am ales sa adun lungimea celor mai mici K stive, deoarece nu stim pe care vrem sa le facem vide, inmultite cu 3. Apoi, la acestea am adunat lungimea restului de stive inmultite cu 5, si pentru a depinde de forma blocului, am adunat costul mutarii blocului respectiv si apoi am inmultit cu 5 pentru piramida, 10 pentru cub si 8 pentru sfera. Astfel, aceasta va fi mai mare decat costul real al drumului.

8. Inainte de optimizarile pentru BF, UCS si A*.

Input 3(are conditiile c si e)

	Lungime	**Cost**	**Timp(milisecunde)**	**Numar total de noduri**
BF	3	4	21	259
DF	6	10	4	54
DFI	3	4	3	45
UCS	3	4	136	1156
A* ADM1	3	4	2	22
A* ADM2	3	4	2	22
A* NEADM	4	6	11	93
Greedy ADM1	3	4	2	22
Greedy ADM2	3	4	2	22
Greedy NEADM	7	10	11	93

Desi, BF, DFI, UCS, A* ADM1, A*ADM2, Greedy ADM1 si Greedy ADM2, returneaza toti un drum de cost si lungime minima, cei mai eficienti sunt A* si Greedy cu euristicile admisibile, deoarece parcurg mai putine noduri pentru ca au euristici si de asemenea sorteaza optiunile, fie dupa f pentru A*, fie dupa h pentru Greedy nodurile. Se mai poate observa ca pentru euristica neadmisibila se obtine o performanta mai buna, atat pentru A*, cat si pentru Greedy, fata de BF sau UCS. BF, obtine o performanta

mai slaba deoarece el genereaza toti succesorii pe niveluri. UCS, va obtine cea mai slaba performanta, cel mai mare numar de noduri si cel mai mare timp, deoarece el parcurge foare mult, pentru ca n-are euristica, arborele construit pe baza costurilor. De asemenea, arborele de la UCS nu e construit pe nivel, ci in functie de costul minim. DF, desi nu returneaza drumul de cost minim, deoarece da drumul cel mai din stanga, putem observa ca este mediu ca performanta. Acesta completeaza arborele de la stanga la dreapta, si din aceasta cauza el daca primeste un graf infinit va rula la infinit sau daca drumul corect se afla cel mai in dreapta va cauta foarte mult. Spre deosebire, de acesta DFI, obtine drumul cost minim si lungime minima, fiindca el face DF pana la o adancime maxima, dar cu o performanta mai slaba decat A* sau Greedy.

Input 4(are conditia d)

	Lungime	**Cost**	**Timp(milisecunde)**	**Numar total de noduri**
BF	7	11	29591	195293
DF	23	15	10	100
DFI	7	11	5014	37653
UCS	-	-	Timeout	-
A* ADM1	7	11	6371	13662
A* ADM2	7	11	5913	13099
A* NEADM	-	-	Timeout	-
Greedy ADM1	-	-	Timeout	-
Greedy ADM2	-	-	Timeout	-
Greedy NEADM	-	-	Timeout	-

Pe baza acestui exemplu, putem observa ca UCS, A* NEADM si Greedy dau timeout. Fiind aceleasi motive ca mai sus, este de inteles pentru UCS. Insa, pentru Greedy, el o sa dea timeout, deoarece incearca sa obtina cea mai buna varianta pe cazul respectiv. Pentru, A* NEADM acesta o sa dea timeout, fiindca avem o euristica neadmisibila, care pe input-ul dat nu va face fata. Se poate observa, ca A* ADM1, A* ADM2, BF si DFI returneaza drumul de cost minim si lungime minima, din aceleasi motive ca mai sus, iar DF returneaza drumul cel mai din stanga.

10. Pentru a optimiza algoritmul am validat datele de intrare, astfel:

- Am verificat numarul de parametri
- Pentru parametrii care trebuiau transformati in numere, deoarece sys.argv e o lista de string-uri, am folosit re.sub pentru a verifica validitatea acestora. Adica, am comparat string-ul initial cu string-ul dupa eliminarea literelor.
- Pentru generarea succesorilor i-am generat doar pe aceia care indeplinesc conditiile din cerinta.
- Daca o stare initiala este invalida, am considerat sa nu mai apelam algoritmii de cautare.
- In cazul in care avem, mai multe piramide decat stivele goale si cele care nu au piramide, atunci nu vom mai face generarea, deoarece nu avem solutii.