Problema vaselor cu apa

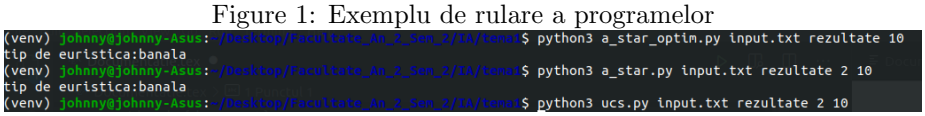
Savu Ioan-Daniel

April 18, 2021

Punctul 1

Cerinta: Fișierele de input vor fi într-un folder a cărui cale va fi dată în linia de comanda. În linia de comandă se va da și calea pentru un folder de output în care programul va crea pentru fiecare fișier de input, fișierul sau fișierele cu rezultatele. Tot în linia de comandă se va da ca parametru și numărul de soluții de calculat (de exemplu, vrem primele NSOL=4 soluții returnate de fiecare algoritm). Ultimul parametru va fi timpul de timeout. Se va descrie în documentație forma în care se apelează programul, plus 1-2 exemple de apel.
Implementare:
Am organizat structura fisierelor la fel ca in cerinta, astfel fisierele de input se gasesc intr un folder numit input_files. De asemenea, pentru fiecare solutie vom avea un fisier cu rezolvarea iar acestea se vor gasi in folderul numit rezultate sau intr-un folder specificat de utilizator. Prin urmare, pentru a se rula scriptul de python este necesar sa se dea ca argumente calea catre fisierul de input, calea catre fisierul in care se vor scrie solutiile, numarul de solutii cautate (daca este cazul pentru algoritm) si timpul de timeout pentru algoritm. Ca observatie, pentru algoritmii a_star si ida_star se va cere la rularea programului si tipul de euristica ce se doreste a fi folosit. Euristicile sunt : banala, adm_1, adm_2, inad. (in ordine banala, admisibila 1, admisibila 2, inadmisibila).

Figure 1: Exemplu de rulare a programelor



Punctul 2

Cerinta: Citirea din fisier + memorarea starii. Parsarea fișierului de input care respectă formatul cerut în enunț
Implementare:
In cadrul fiecarui algoritm am folosit urmatoarea secventa de cod ce parseaza fisierul de input

```
1
2
3
4 #functie de citire din fisierul de input
```

```
5  #returneaza trasnformariile posibile, lista initiala de vase si
       starea finala
6  #primeste ca input path-ul catre fisierul txt
7  def readInput(inputPath):
8      #modul in care retinem starea initiala
9      transformations = dict()
10     vessels = []
11     #modul in care retinem starea finala
12     finalState = dict()
13
14     file = open(inputPath, "r")
15     lines = [x.strip() for x in file.readlines()]
16     inputType = 0
17
18     #citim din fisierul initial starea initiala, transformariile si
           starea finala
19     for i in range(len(lines)):
20         if lines[i] == 'stare_initiala':
21             inputType = 1
22             continue
23         if lines[i] == 'stare_finala':
24             inputType = 2
25             continue
26         #daca citim transformariile
27         if inputType == 0:
28             colors = lines[i].split()
29             if (len(colors) != 3):
30                 print("Input invalid\n")
31                 return
32             transformations[(colors[0], colors[1])] = colors[2]
33             transformations[(colors[1], colors[0])] = colors[2]
34         #daca citim starea initiala
35         if inputType == 1:
36             dim = lines[i].split()
37             maxSize = int(dim[0])
38             actualSize = int(dim[1])
39             if actualSize == 0 and len(dim) == 3:
40                 print("Input invalid")
41                 return
42             if actualSize != 0 and len(dim) == 2:
43                 print("Input invalid")
44                 return
45
46             col = ""
47             if actualSize == 0:
48                 col = ""
49             else:
50                 col = dim[2]
51             vessels.append((maxSize, actualSize, col))
52         #daca citim starea finala
53         if inputType == 2:
54             ammount = int(lines[i].split()[0])
55             col = lines[i].split()[1]
56             finalState[col] = ammount
57
58     return (transformations, vessels, finalState)
```

De asemenea, in cadrul acestei functii fac si validarea fisierului de input (daca datele date nu sunt valide atunci inchid programul). Transformariile intre culori le retin sub forma unui dictionar a carei chei este compusa dintr-un tuplu format din primele doua culori iar valoarea este data de rezultatul combinarii. Starea initiala o retin sub forma unui vector de tupluri, iar tuplul este format din 3

componente : capacitatea maxima a vasului, volumul ocupat, culaorea din vas. Starea finala o retin sub forma de dictionar in care cheia este o culoare iar valoarea este target-ul respectiv pentru acea culoare.

Punctul 3

Cerinta: Functia de generare a succesorilor
Implementare:
Codul corespunzator functiei de generare a succesorilor este cel de mai jos

```
1  #va genera succesorii sub forma de noduri in arborele de parcurgere
2  def genereazaSuccesori(self, nodCurent, tip_euristica = 'banala'):
3      listaSuccesori=[]
4      n = len(nodCurent.info)
5      current_vessels = nodCurent.info
6
7      for i in range(n):
8          #daca nu e gol vasul din care turnam
9          if current_vessels[i][1] > 0:
10             #incercam sa facem toate combinatiile
11             for j in range(n):
12                 if i != j:
13                     #daca nu e gol, putem pune si obtine o noua
                           stare
14                     if current_vessels[j][1] != current_vessels[j
                           ][0]:
15                         #facem mutari
16                         new_vessels = copy.deepcopy(current_vessels
                               )
17                         quantity = min(new_vessels[i][1],
18                         new_vessels[j][0] \- new_vessels[j][1])
19                         #noile cantitati si culori ce urmeaza sa
                               fie puse
20                         quantityi = new_vessels[i][1] - quantity
21                         quantityj = new_vessels[j][1] + quantity
22                         colorj = new_vessels[j][2]
23                         colori = new_vessels[i][2]
24                         #vedem ce culaore punem
25                         if (new_vessels[i][2], new_vessels[j][2])
                               in
26                                         self.transformations.keys()
                                               :
27                             colorj = self.transformations
28                             [(new_vessels[i][2], new_vessels[j][2])
                                   ]
29                         else :
30                             #daca e gol pastreaza culoarea lui i
31                             if new_vessels[j][1] == 0:
32                                 colorj = new_vessels[i][2]
33                             else:
34                                 colorj = 'nedefinit'
35                         #ii spunem ca nu are culoare daca e gol
36                         if new_vessels[i][1] - quantity == 0:
37                             colori = ""
38                         new_vessels[i] = (new_vessels[i][0],
39                                           quantityi, colori)
40                         new_vessels[j] = (new_vessels[j][0],
41                                           quantityj, colorj)
42
```

```
43                          this_move = (i, quantity, current_vessels[i
                                ][2], j)
44
45                          #bagam nodul in coada fara duplicate, adica
                                daca
46                          #am ajuns intr-un nod, ne intereseaza doar
47                          #drumul cu costul cel mai mic pana la acel
                                nod.
48                          to_add = True
49
50                          h = self.calculeaza_h(new_vessels,
                                tip_euristica)
51
52                          #inseamna ca e ok sa il punem in lista de
                                succesori
53                          if to_add and not nodCurent.contineInDrum(
                                new_vessels)
54                          and self.isWorthExpanding(new_vessels):
55                              listaSuccesori.append(NodParcurgere
56                              (new_vessels, nodCurent.g + 1,
57                              nodCurent, h, move = this_move)
58                              )
59      return listaSuccesori
```

Pentru a genera succesorii am folosit urmatoarea idee: pentru fiecare vas din starea curenta care nu este gol punem pentru in fiecare vas care nu este plin. Atunci cand facem o astfel de mutare tinem cont de transformariile aduse culorilor. Pentru fiecare noua stare calculata verificam daca aceasta nu se afla deja in drumul de la nodul start la frunza curenta(deoarece ar insemna ca intram intr-un ciclu infinit). In plus, pentru fiecare noua stare creeata verificam daca indeplineste conditia ca suma culorilor care nu sunt "nedefinite" sa fie cel putin egala cu suma culorilor din target, deoarece in caz contrar nu se poate ajunge la rezultat. Aceasta verificare se face prin apelul functiei isWorthExpanding. Pe langa aceste aspecte pentru fiecare stare tinem minte si ce mutare am facut de la parinte la aceasta pentru a ne usura afisarea solutiei. O observatie referitoare la aceasta functie de generare a succesorilor este ca prezinta mici variatiuni in functie de algoritmul folosit deoarece spre ex. la UCS nu avem nevoie de functia h si in acelasi timp putem face optimizari precum : un nod cu aceeasi configuratia cu unul din coada dar cu un cost mai mare va fi ignorat.

Punctul 4

Cerinta: Calcularea costului pentru o mutare
Implementare:
Deoarece in problema nu se specifica costul unei mutari, conform baremului, o mutare are costul 1. Urmatoarea secventa de cod trateaza acest aspect

```
1  listaSuccesori.append(NodParcurgere(new_vessels, nodCurent.g + 1,
2                                      nodCurent, h, move = this_move)
                                           )
```

Punctul 5

Cerinta: Testarea ajungerii în starea scop (indicat ar fi printr-o funcție de testare a scopului)

Implementare:
Prin urmatoarea secventa de cod am implementat functia care testeaza daca un nod este si nod scop:

```
1       #primeste ca parametru un nod si verifica daca se afla in
            starea finala sau nu
2       #returneaza true / false
3       def testeaza_scop(self, nodCurent):
4           used = set()#sa nu numaram de 2 ori acelasi scop
5           nMatches = 0 # numar de culori atinse
6
7           for elem in nodCurent.info:
8               culoare = elem[2]
9               dimensiune = elem[1]
10              #daca mai descoperim o culoare in plus nefolosita
11              if culoare in self.scope.keys() and culoare not in used
                    and dimensiune == self.scope[culoare]:
12                  used.add(culoare)
13                  nMatches += 1
14          #daca am gasit toate culorile din scope
15          if len(self.scope.keys()) == nMatches:
16              return True
17          return False
```

Modul in care am facut aceasta testare a fost sa verific daca in lista de vase am un numar de target-uri atinse egal cu numarul de target-uri totale. Pentru a numara o singura data un target, de fiecare data cand gasim unul il bagam intr-un set pentru a nu-l gasi de mai multe ori.

Punctul 6

Cerinta: 4 euristici:
a. banala
b. doua euristici admisibile posibile (se va justifica la prezentare si in documentație de ce sunt admisibile)
c. o euristică neadmisibilă (se va da un exemplu prin care se demonstrează că nu e admisibilă). Atenție, euristica neadmisibilă trebuie să depindă de stare (să se calculeze în funcție de valori care descriu starea pentru care e calculată euristica).
Implementare:
a. Deoarece fiecare mutare are costul 1, euristica banala se refera la a atribui lui $\widehat{h}$(nod) valoarea 1 daca nu este final(deoarece mai aavem de facut cel putin 1 pas) si cu 0 daca este nod final(nu mai avem de facut niciun pas). Demonstratia pentru care aceasta euristica este admisibila este evidenta. Implementarea acesteia este urmatoarea:
b. - O prima euristica admisibila este urmatoarea: $\widehat{h}$(nod) = numarul de culori care ne lipsesc din starea finala
admisibilitate: Deoarece la o mutare obtinem in cel mai bun caz o noua culoare, iar in cel mai bun scenariu acea culaore este dintre cele care lipsesc din solutie, avem prin urmare ca $\widehat{h}$(nod) $\leq$ h(nod). Cu alte cuvinte, la fiecare mutare obtinem in cel mai bun caz inca o culoare din starea finala si in acelasi timp cantitatea corespunzatoare, deci h(nod) $\geq$ nr. culori lipsa = $\widehat{h}$(nod).
consistenta: Deoarece la o mutare avem costul de 1 si in cel mai bun caz dupa o mutare mai obtinem o culoare din cele dorite, avem ca $\widehat{h}$(nod1) $\leq 1 + \widehat{h}$(nod2)

si deci conditia de consistenta este respectata
Prin urmare, aceasta euristica este admisibila.
- O a doua euristica admisibla este urmatoarea : $\widehat{h}$(nod) = numarul de target-uri finale (adica cate culori de o anumita cantitate nu avem din starea finala) / 2.
admisibilitate: Deoarece la o mutare modificam continutul a doua vase, in cel mai bun caz dupa o astfel de mutare vom obtine 2 elemente din starea finala pe care nu le aveam. Prin urmare, trebuie sa facem cel putin (nr target neatinse) / 2 mutari pentru a ajunge intr-o stare finala. Prin urmare, relatia $\widehat{h}$(nod) $\leq$ h(nod) este respectata.
consistenta: Deoarece dupa o mutare avem ca $\widehat{h}$(n1) - $\widehat{h}$(n2) *leq* 1 si deoarece costul unei mutari este de 1, este repsectata conditia de consistenta.
c. O euristica neadmisibila: $\widehat{h}$(nod) = cate vase care nu sunt in starea finala. Aceasta euristica este inadmisibila deoarece nu respecta conditia de admisibilitate. Un exemplu in acest sens este urmatorul : pentru inputul

```
1       alb negru gri
2       stare_initiala
3       2 2 negru
4       3 0
5       100 100 alb
6       2 1 negru
7       2 1 negru
8       stare_finala
9       3 gri
```

h(nod) = 2 (punem din vasul 0 in vasul 1 si din vasul 2 in vasul 1), in schimb folosind aceasta euristica avem ca $\widehat{h}$(nod) = 5. Prin urmare $\widehat{h}$(nod) > h(nod) si euristica este inadmisibila.

Punctul 7

Cerinta: crearea a 4 fisiere de input cu urmatoarele proprietati:
a. un fisier de input care nu are solutii
b. un fisier de input care da o stare initiala care este si finala (daca acest lucru nu e realizabil pentru problema, aleasa, veti mentiona acest lucru, explicand si motivul).
c. un fisier de input care nu blochează pe niciun algoritm și să aibă ca soluții drumuri lungime micuță (ca să fie ușor de urmărit), să zicem de lungime maxim 20.
d. un fisier de input care să blocheze un algoritm la timeout, dar minim un alt algoritm să dea soluție (de exemplu se blochează DF-ul dacă soluțiile sunt cât mai "în dreapta" în arborele de parcurgere)
e. dintre ultimele doua fisiere, cel putin un fisier sa dea drumul de cost minim pentru euristicile admisibile si un drum care nu e de cost minim pentru cea euristica neadmisibila
Implementare:
a.Un fisier de input care nu are solutii:

```
1       rosu albastru mov
2       albastru galben verde
3       mov verde maro
4       stare_initiala
```

```
5       5 4 rosu
6       2 2 galben
7       3 0
8       7 3 albastru
9       1 0
10      9 2 galben
11      4 3 rosu
12      stare_finala
13      30 rosu
14      2 verde
```

b. Un fisier in care starea initiala este si finala

```
1       rosu albastru mov
2       albastru galben verde
3       mov verde maro
4       stare_initiala
5       5 4 rosu
6       2 2 galben
7       3 0
8       7 3 albastru
9       1 0
10      4 3 rosu
11      stare_finala
12      4 rosu
13      2 galben
```

c. Un input care sa nu blocheze niciun algoritm

```
1       alb negru gri
2       stare_initiala
3       2 2 negru
4       3 0
5       100 100 alb
6       2 1 negru
7       2 1 negru
8       stare_finala
9       3 gri
```

d. Un input care sa produca cel putin un timeout si cel putin un algortim sa treaca

```
1       rosu albastru mov
2       albastru galben verde
3       mov verde maro
4       stare_initiala
5       5 4 rosu
6       2 2 galben
7       3 0
8       7 3 albastru
9       1 0
10      4 3 rosu
11      stare_finala
12      3 mov
13      2 verde
```

e. Inputul de la d) respecta aceasta cerinta deoarece pe euristicile valide obtinem drumul urmator:

```
1       Duration: 0.028834104537963867
2       Cost: 2
3       Length: 3
4       Maximum nodes in memory: 146
5       Generated successors: 169
```

```
6       Path:
7       ----------------------------------------
8       1
9       0: 2 2 negru
10      1: 3 0
11      2: 100 100 alb
12      3: 2 1 negru
13      4: 2 1 negru
14      2
15      Din vasul 4 s-au turnat 1 litri de apa de culoare negru in
            vasul 1
16      0: 2 2 negru
17      1: 3 1 negru
18      2: 100 100 alb
19      3: 2 1 negru
20      4: 2 0
21      3
22      Din vasul 2 s-au turnat 2 litri de apa de culoare alb in vasul
            1
23      0: 2 2 negru
24      1: 3 3 gri
25      2: 100 98 alb
26      3: 2 1 negru
27      4: 2 0
```

iar pe euristica invalida obtinem drumul

```
1   Duration: 0.0035142898559570312
2   Cost: 4
3   Length: 5
4   Maximum nodes in memory: 30
5   Generated successors: 33
6   Path:
7   ----------------------------------------
8   1
9   0: 2 2 negru
10  1: 3 0
11  2: 100 100 alb
12  3: 2 1 negru
13  4: 2 1 negru
14  2
15  Din vasul 2 s-au turnat 1 litri de apa de culoare alb in vasul 4
16  0: 2 2 negru
17  1: 3 0
18  2: 100 99 alb
19  3: 2 1 negru
20  4: 2 2 gri
21  3
22  Din vasul 3 s-au turnat 1 litri de apa de culoare negru in vasul 2
23  0: 2 2 negru
24  1: 3 0
25  2: 100 100 gri
26  3: 2 0
27  4: 2 2 gri
28  4
29  Din vasul 2 s-au turnat 2 litri de apa de culoare gri in vasul 3
30  0: 2 2 negru
31  1: 3 0
32  2: 100 98 gri
33  3: 2 2 gri
34  4: 2 2 gri
35  5
36  Din vasul 2 s-au turnat 3 litri de apa de culoare gri in vasul 1
```

```
0: 2 2 negru
1: 3 3 gri
2: 100 95 gri
3: 2 2 gri
4: 2 2 gri
```

Punctul 8

Costul unei mutari fiind 1, lungimea drumului este cu 1 mai mare decat costul drumului (din cauza nodului initial). Timpul de gasire al unei solutii este determinat folosin libraria time. De asemenea, in cadrul algoritmilor am contorizat maximul numarului de noduri din memorie la un moment dat si in acelasi timp cate noduri am generat pentru a ajunge la solutie. Un exemplu in acest sens este urmatorul output: Mai multe outputuri se gasesc in fisierul rezultate atasat proiectului.

```
Duration: 0.3920137882232666
Cost: 4
Length: 5
Maximum nodes in memory: 1756
Generated successors: 1977
Path:
---------------------------------------
1
0: 5 4 rosu
1: 2 2 galben
2: 3 0
3: 7 3 albastru
4: 1 0
5: 4 3 rosu
2
Din vasul 1 s-au turnat 1 litri de apa de culoare galben in
    vasul 0
0: 5 5 nedefinit
1: 2 1 galben
2: 3 0
3: 7 3 albastru
4: 1 0
5: 4 3 rosu
3
Din vasul 3 s-au turnat 1 litri de apa de culoare albastru in
    vasul 1
0: 5 5 nedefinit
1: 2 2 verde
2: 3 0
3: 7 2 albastru
4: 1 0
5: 4 3 rosu
4
Din vasul 3 s-au turnat 1 litri de apa de culoare albastru in
    vasul 5
0: 5 5 nedefinit
1: 2 2 verde
2: 3 0
3: 7 1 albastru
4: 1 0
5: 4 4 mov
5
```

```
40      Din vasul 5 s-au turnat 3 litri de apa de culoare mov in vasul
            2
41      0: 5 5 nedefinit
42      1: 2 2 verde
43      2: 3 3 mov
44      3: 7 1 albastru
45      4: 1 0
46      5: 4 1 mov
```

Punctul 9

Cerinta: Afisarea in fisierele de output in formatul cerut
Implementare:
Outputul este in formatul cerut, asa cum se poate observa in exemplele de la subpunctele anterioare

Punctul 10

Cerinta: Validări și optimizari. Veți implementa elementele de mai jos care se potrivesc cu varianta de temă alocată vouă:
1.găsirea unui mod de reprezentare a stării, cât mai eficient
2.verificarea corectitudinii datelor de intrare
3.găsirea unor conditii din care sa reiasă că o stare nu are cum sa contina in subarborele de succesori o stare finala deci nu mai merita expandata (nu are cum să se ajungă prin starea respectivă la o stare scop)
4.găsirea unui mod de a realiza din starea initială că problema nu are soluții.
Validările și optimizările se vor descrie pe scurt în documentație.
Implementare:
1. Starea este reprezentata de o lista de tupluri, iar fiecare tuplu reprezinta un vas, pastrand ca informatii capacitatea acestuia, cat este ocupat si ce culoare are apa. Acest mod de reprezentare este foarte eficient deoarece nu prezinta redundanta.
2. Conform sectiunii anterioare, atunci cand se citeste inputul se fac verificari pentru ca acesta sa fie corespunzator.
3. Deoarece in urma transformariilor nu se pierde si respectiv nu apare lichid nou, stim cu siguranta ca daca suma cantitatilor de apa care sunt colorate in mod "definit" este mai mica decat suma cantitatilor de apa din target, atunci nu putem ajunge la o solutie. Aceasta verificare este facuta de fiecare data cand se creeaza un nod nou, iar codul corspunzator pentru aceasta verificare este urmatorul:

```
1  # returneaza true daca mai are sens sa expandam nodul
2  # primeste ca arugment nodul curent
3  # verifica daca suma culorilor care nu sunt nedefinite >= suma
       target-urilor
4  def isWorthExpanding(self, current_vessels):
5      suma_culori = 0
6      suma_target_culori = 0
7
8      for elem in self.scope.values():
9          suma_target_culori += elem
10
```

```
11      for elem in current_vessels:
12          if elem[2] != 'nedefinit':
13              suma_culori += elem[1]
14
15      if suma_culori < suma_target_culori:
16          return False
17
18      return True
```

4. In mod analog, pentru starea initiala facem verificarea de la 3, iar pe langa aceasta verificam ca vasul cel mai mare pe care il avem in starea initiala sa fie suficient de mare pentru a cuprinde cea mai mare cantitate de apa din target. Codul corespunzator acestor verificari este urmatorul:

```
1  # verifica daca poate avea solutie. modul in care face asta: Daca
       avem mai putine vase decat conditii
2  # de final sau suma initiala de culori < suma target-urilor sau nu
       avem un vas suficient de mare inseamna ca nu
3  # avem solutie
4  def initialCheck(self, nodInitial):
5      suma_culori = 0
6      suma_target_culori = 0
7      maxSize = 0
8      maxTargetSize = 0
9      #facem suma si maximul pentru scope
10     for elem in self.scope.values():
11         suma_target_culori += elem
12         maxTargetSize = max(maxTargetSize, elem)
13     #facem suma si maximul pentru array ul nostru
14     for elem in nodInitial.info:
15         suma_culori += elem[1]
16         maxSize = max(maxSize, elem[0])
17     if suma_culori < suma_target_culori or maxTargetSize > maxSize:
18         return False
19     #daca suma cantitatilor de culori < suma targe si daca avem loc
           sa punem pe cel mai voluminos, e ok
20     return True
```

Punctul 11

Cerinta: Comentarii pentru clasele și funcțiile adăugate de voi în program (dacă folosiți scheletul de cod dat la laborator, nu e nevoie sa comentați și clasele existente). Comentariile pentru funcții trebuie să respecte un stil consacrat prin care se precizează tipul și rolurile parametrilor, cât și valoarea returnată (de exemplu, reStructured text sau Google python docstrings).
Implementare:
Codul este comentat corespunzator in locurile in care nu a fost luat din laborator si aunmite exemple de cod comentat se pot gasi si in punctul anterior.

Punctul 12

Implementare: Pentru urmatorul input:

```
1  alb negru gri
2  stare_initiala
3  2 2 negru
4  3 0
```

```
5  100 100 alb
6  2 1 negru
7  2 1 negru
8  stare_finala
9  3 gri
```

Avem urmatorul tabel de rezultate.

Algoritm	Eurisitca	Cost	Timp executie	Noduri generate	Maximul noduri
A*	banala	2	0.00296	20	22
A*	admisibla 1	2	0.031793	135	160
A*	admisibla 2	2	0.01394	79	91
A*	neadmisibila	4	0.003571	30	33
A* optim	banala	2	0.00296	20	22
A* optim	admisibla 1	2	0.0283	146	169
A* optim	admisibla 2	2	0.0172	105	120
A* optim	neadmisibila	4	0.00350	30	33
IDA*	banala	2	0.0034	11	15
IDA*	admisibila 1	2	0.0059	31	45
IDA*	admisibila 2	2	0.0033	11	15
IDA*	neadmisibila	2	0.0058	31	45
UCS	N/A	2	0.0049	90	103

De asemenea, pentru urmatorul input

```
1  rosu albastru mov
2  albastru galben verde
3  mov verde maro
4  stare_initiala
5  5 4 rosu
6  2 2 galben
7  3 0
8  7 3 albastru
9  1 0
10 4 3 rosu
11 stare_finala
12 3 mov
13 2 verde
```

Avem urmatorul tabel

Algoritm	Eurisitca	Cost	Timp executie	Noduri generate	Maximul noduri
A*	banala	4	1.5020	2841	3292
A*	admisibla 1	4	0.9407	1980	2419
A*	admisibla 2	4	7.6141	6133	7450
A*	neadmisibila	5	0.04088	218	247
A* optim	banala	4	2.4790	4609	5226
A* optim	admisibla 1	4	0.4500	1756	1977
A* optim	admisibla 2	4	3.3710	4964	5684
A* optim	neadmisibila	5	0.0395	227	253
IDA*	banala	4	0.16331	3804	5300
IDA*	admisibila 1	4	0.10870	1878	3271
IDA*	admisibila 2	4	0.14475	3804	4656
IDA*	neadmisibila	4	0.076346	1241	1886
UCS	N/A	Timeout	Timeout	Timeout	Timeout

Analizand aceste tabele putem sa observam in primul rand ca algoritmul UCS este in general ineficient pentru cazurile in care solutiile se afla la o adancime mai mare in arbore. Deoarece acesta nu filtreaza nodurile intr-un mod eficient (precum A* cu o euristica) complexitatea devine exponentiala iar acest lucru este usor observabil pentru un drum cu cost mai mare. De asemenea, se poate observa ca o euristica admisibila este cu atat mai eficienta cu cat este mai apropiata de costul real al nodului respectiv. Astfel, o euristica admisibila dar care nu se pliaza pe input poate fi destul de ineficienta (cum se observa pentru varianta admisibila 2 a algorimtiilor A* si A* optim). Pe de alta parte se poate observa ca daca solutia se afla la adancime mica in arbore, atunci este mai eficient sa cautam cu algoritmi precum IDA* si UCS deoarece euristicile pot "pacali" algoritmii sa dezvolte mai intai noduri care nu duc atat de rapid la rezultat si sa ignore solutiile "shallow". In acelasi in anumite cazuri algoritmul A* s-a dovedit a fi mai rapid decat algorimtul A* optim, insa acest lucru se datoreaza in mare complexitatii in plus pe care acesta o prezinta datorita procesarii listelor open si closed, iar pentru valori mici in input acest impact este vizibil. Algortimul ida* s-a dovedit a fi rapid pentru solutiile "shallow" insa costisitor din punct de vedere al memoriei folosite.
In concluzie, fiecare algoritm are avantajele si dezavantajele sale, algoritmii precum IDA* si UCS sunt eficienti pentru solutiile shallow insa pentru alte tipuri de solutii pot fi foarte costisitori, in timp ce algorimtii precum A* si A* optim se comporta in general bine si depinzand de euristica folosita pot scoate rezutlate foarte rapide pentru solutii ce se gasesc mai adanc in arborele de solutii.