PROBLEMA VASELOR CU APĂ
DOCUMENTAȚIE

Exemplu apel program

- `python3 problema_vaselor_cu_apa.py cale_folder_input cale_folder_output numar_solutii_cautate timeout`
- `python3 problema_vaselor_cu_apa.py “./InputFolder” “./OutputFolder” 1 2`
- `python3 problema_vaselor_cu_apa.py “./InputFolder” “./OutputFolder” 2 10`

Explicarea validărilor și optimizărilor

Funcția valid_arguments() este folosită imediat după apelul programului, pentru a semnala eventuale nereguli(inexistența folderului pentru input, timeout sau numărul de soluții căutate sunt numere negative).

Funcția has_scope() este apelată înainte de expandarea fiecărui nod, pentru a verifica dacă există un nod scop pe drumul care pleacă din nodul curent, astfel: se calculează cantitățile maxime care se pot obține pentru fiecare culoare din starea scop, iar dacă nu se poate obține destul lichid dintr-o anumită culoare, returnează False.

Pentru a stabili dacă graful nu are soluții din starea inițială, verificăm dacă avem destule vase cu capacități ce pot satisface cantitățile dorite în starea scop, după care verificăm și existența unui nod scop folosind funcția has_scope().

Explicarea euristicilor folosite

- **euristica banală**: dacă nodul curent este nod scop, h = 0, altfel h = 1

- **euristica admisibilă 1**: pentru fiecare culoare din starea scop care nu există în nodul curent sau nu este conținută într-un singur vas, sau nu are cantitatea dorită, h crește cu 1

Corectitudine:

- o culoare din starea scop nu există în starea actuală: pentru a obține această culoare, trebuie să amestecăm 2 culori, deci în cel mai bun caz, turnăm 1 litru de lichid cu cost 1 în alt vas => cost minim = 1 pentru această mutare;
- lichidul dintr-o anumită culoare nu este într-un singur vas: deoarece ne interesează o cantitate fixă într-un singur vas, pentru a ajunge la starea scop e nevoie să aducem conținutul într-un singur vas, cu costul minim 1, considerând cazul ideal în care turnăm 1 litru de culoare cu costul 1;
- lichidul dintr-o anumită culoare este într-un singur vas, dar nu este cantitatea dorită: atunci trebuie să mai vărsăm/adăugăm lichid pentru a obține cantitatea dorită, deci costul va fi 1 în cazul optim;

- **euristica admisibilă 2**: numărul de culori din starea finală care nu se găsesc în starea curentă

  Corectitudine:
  Fie h = numărul de culori din starea finală care nu se găsesc în starea curentă, deci trebuie să mai obținem încă h culori, ceea ce se poate face în cel mai bun caz prin h amestecuri de culoare, în care turnăm mereu câte 1 litru cu cost 1.

- **euristica neadmisibilă**: pentru fiecare culoare din starea scop, adaug cantitatea care mai trebuie obținută pentru a ajunge în starea scop

  Contraexemplu:
  Fie starea următoare, din care vrem să obținem în final 1 litru de lichid verde și 3 litri de lichid mov, știind amestecurile posibile și costurile:
  roșu + albastru = mov
  albastru + galben = verde

  roșu 1
  albastru 4
  galben 3
  mov 7
  verde 5

  0: 5 4 roșu
  1: 2 1 verde
  2: 3 2 albastru
  3: 4 0
  4: 1 1 galben

Euristica neadmisibilă va produce valoarea 3, pentru că mai este nevoie de 3 litri de lichid mov, însă h-ul optim se obține dacă turnăm 1 litru de lichid roșu din vasul 0 în vasul 2, cu costul 1 * 1 = 1.

Analizare algoritmi

input3.txt - fișierul care nu se blochează pe niciun algoritm, are drumuri scurte
input4.txt - fișierul pe care se blochează unii algoritmi, iar pentru euristica neadmisibilă nu oferă soluțiile crescător după cost

Algoritm	Euristica	input3.txt	input4.txt
UCS	banală	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 9 Numar total: 9	Timpul de executie a depasit timeout-ul!
	admisibilă 1	-	-
	admisibilă 2	-	-
	neadmisibilă	-	-
A*	banală	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 9 Numar total: 9	Lungime: 6 Cost: 20 Timp: 1.838 secunde Numar maxim: 6773 Numar total: 7602
	admisibilă 1	Lungime: 2 Cost: 4 Timp: 0.001 secunde Numar maxim: 9 Numar total: 9	Lungime: 6 Cost: 20 Timp: 1.439 secunde Numar maxim: 4384 Numar total: 4925
	admisibilă 2	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 9 Numar total: 9	Lungime: 6 Cost: 20 Timp: 1.792 secunde Numar maxim: 6730 Numar total: 7555
	neadmisibilă	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 9 Numar total: 9	Lungime: 6 Cost: 20 Timp: 1.78 secunde Numar maxim: 6748 Numar total: 7576

A* optimizat	banală	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 10 Numar total: 9	Lungime: 6 Cost: 20 Timp: 0.089 secunde Numar maxim: 662 Numar total: 1022
	admisibilă 1	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 10 Numar totale: 9	Lungime: 6 Cost: 20 Timp: 0.074 secunde Numar maxim: 580 Numar total: 868
	admisibilă 2	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 10 Numar total: 9	Lungime: 6 Cost: 20 Timp: 0.081 secunde Numar maxim: 648 Numar total: 973
	neadmisibilă	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 10 Numar total: 9	Lungime: 6 Cost: 20 Timp: 0.093 secunde Numar maxim: 692 Numar total: 1040
IDA*	banală	Lungime: 2 Cost: 4 Timp: 0.001 secunde Numar maxim: 9 Numar total: 18	Lungime: 6 Cost: 20 Timp: 1.199 secunde Numar maxim: 57 Numar total: 32132
	admisibilă 1	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 9 Numar total: 18	Lungime: 6 Cost: 20 Timp: 0.917 secunde Numar maxim: 57 Numar total: 24220
	admisibilă 2	Lungime: 2 Cost: 4 Timp: 0.001 secunde Numar maxim: 9 Numar total: 18	Lungime: 6 Cost: 20 Timp: 1.218 secunde Numar maxim: 57 Numar total: 31266
	neadmisibilă	Lungime: 2 Cost: 4 Timp: 0.0 secunde Numar maxim: 9 Numar total: 18	Lungime: 7 Cost: 22 Timp: 1.491 secunde Numar maxim: 67 Numar total: 40596

Algoritmii în ordinea descrescătoare a timpului de rulare: **A* optimizat, IDA*, A*, UCS.**

Algoritmii în ordinea crescătoare a numărului maxim de noduri reținute la un moment dat: **A* optimizat, IDA*, A*.**

Algoritmii în ordinea crescătoare a numărului total de noduri generate: **A* optimizat, A*, IDA*.**

Algoritmul **A* optimizat** este cel mai bun din punct de vedere al memoriei utilizate și al timpului de răspuns, însă oferă o singură soluție.

Algoritmul **A*** este mai lent și folosește mai multă memorie, însă furnizează mai multe soluții.

Algoritmul **IDA*** este mai rapid decât **A***, numărul de noduri reținute la un moment dat este mai mic, însă folosește mai multă memorie pentru a stoca nodurile generate.