

Documentație - Inteligență Artificială

- Problema de căutare (un mesaj...) -

Popescu Paullo Robertto Karloss

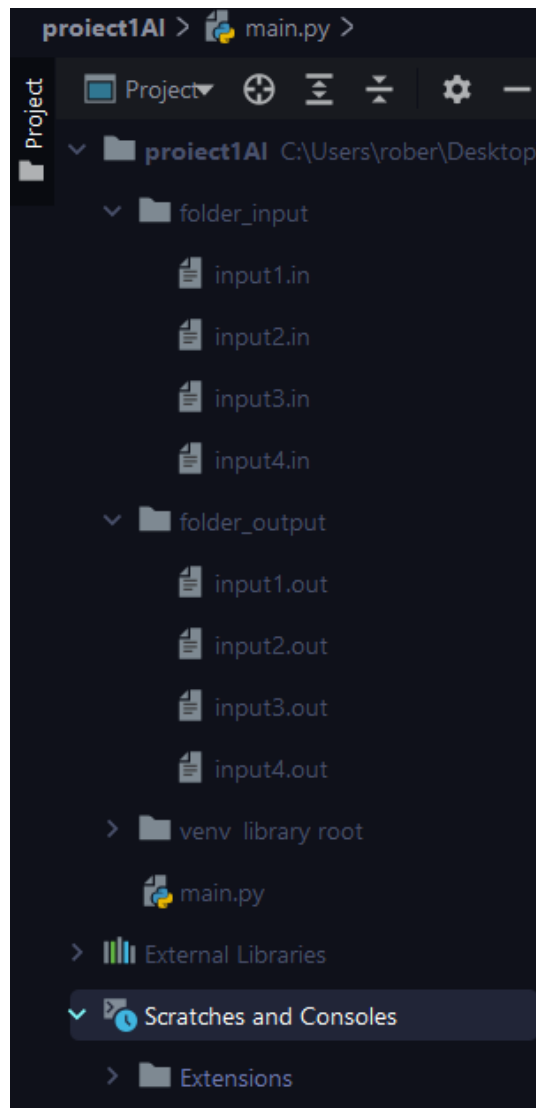
Grupa 231

2022 Aprilie

Cuprins:

1. Structura fișierelor.
2. Modul de rulare al proiectului.
 - 2.1 Exemplu de rulare.
3. Euristici.
 - 3.1 Euristică banală.
 - 3.2 Euristică admisibilă 1.
 - 3.3 Euristică admisibilă 2.
 - 3.4 Euristică neadmisibilă.
 - 3.5 Exemplu pentru euristica neadmisibilă, unde h -ul estimate este mai mare decât h -ul real.
4. Validări și optimizări.
 - 4.1 Optimizări.
 - 4.2 Validări.
5. Compararea algoritmilor.
6. Concluzie.

1. Structura fișierelor



Fișierele de input:

- input1.in -> nu are soluții
- input2.in -> stare inițială = stare finală
- input3.in -> nu se blochează pe niciun algoritm
- input4.in -> se blochează pentru anumiți algoritmi (depășește timeout-ul setat de noi)

2. Modul de rulare al proiectului

Argumentele primite din linia de comandă:

- -i sau --input pentru calea către folderul care conține fișierele de input
- -o sau --output pentru calea către folderul care conține fișierele de output
- -n sau --nrsolutii pentru numărul de soluții care trebuie generate
- -t sau --timeout pentru timpul în secunde după care ne dorim ca programul să se oprească pentru un anumit algoritm din generarea soluțiilor (de exemplu, pentru timeout = 60s, dacă algoritmul DFI depășește acest timp alocat, se va opri și va trece la următorul algoritm etc)

2.1 Exemplu de rulare.

```
PS C:\Users\rober\Desktop\proiect1AI> python main.py -i folder_input -o folder_output -n 5 -t 30
```

Dacă nu vom folosi linia de comandă argumentele vor avea valorile implicite

- Folder cu fișierele de input: folder_input
- Folder cu fișierele de output: folder_output
- Număr de soluții căutate 1
- Timeout 100s

3. Euristici

3.1 Euristica banală.

- Returnează 1 dacă suntem în stare finală (dacă biletul este pe poziția copilului destinație), 0 altfel.

3.2 Euristica admisibilă 1.

- Returnează distanța Manhattan de la poziția curentă la poziția copilului destinație.
- Observație! Este admisibilă întrucât pentru a ajunge la copilul destinație trebuie să parcurgi măcar distanța Manhattan (numărul de copii; diferența copiilor pe linie + diferența copiilor pe coloană), deci nu poate să meargă la un pas decât la un copil alăturat.

3.3 Euristica admisibilă 2.

- Returnează distanța euclidiană de la poziția curentă la poziția copilului destinație (se consideră distanța 1 între 2 copii aflați pe linii sau (exclusiv) coloane consecutive).
- Observație! Este admisibilă întrucât este mai mică întotdeauna decât distanța Manhattan. Din păcate, se observă că pe această problemă particulară, pe unele exemple de input se comportă mai lent, iar pe altele cel mai rapid.

3.4 Euristica neadmisibilă.

- Returnează $50 * \text{numărul de linii din matrice} * \text{numărul de coloane din matrice} + \text{linia curentă} + \text{coloana curentă}$, la această valoare adăgându-se $2^{(\text{linia curentă} + \text{coloana curentă})}$ doar dacă biletul stă pe loc.
- Observație! O mutare are costul maxim 2, iar biletul stă pe loc foarte rar, așa că factorul 50 de înmulțit cu numărul de copii din clasă face ca această euristică pe anumite cazuri să fie neadmisibilă. Am observat că pe euristica neadmisibilă avea tendința să rămână mai mult în același loc, așa că am adăugat și termenul $2^{(x+y)}$, pentru a îl forța să se deplaseze și să nu dea timeout excesiv.

- **Exemplu pentru euristica neadmisibilă** (cost real = 4, costul dat de euristica neadmisibilă = 8):



```
main.py X input1.out X input2.out X input3.out X input3.in X input4.out X
0
1  alin gigel liber liber cornel andrei
2  andreea miruna liber amalia alina teodora
3  sergiu george silviu laura nicu vasile
4  costel florin simona liber paul luca
5  suparati
6  alina teodora
7  ascultati:
8  5
9  teodora
10 silviu
11 simona
12 mesaj: alin → laura
```

Euristica neadmisibilă:

```
1001 Solution:
1002 Step 1:
1003 alin
1004
1005 Step 2:
1006 gigel
1007
1008 Step 3:
1009 miruna
1010
1011 Step 4:
1012 gigel
1013
1014 Step 5:
1015 miruna
1016
1017 Step 6:
1018 andreea
1019
1020 Step 7:
1021 andreea
1022
1023 Step 8:
1024 andreea
1025
1026 Step 9:
1027 andreea
1028
1029 Step 10:
1030 andreea
1031
1032 Step 11:
1033 alin
1034
1035 Step 12:
1036 gigel
1037
1038 Step 13:
1039 miruna
1040
1041 Step 14:
1042 andreea
1043
1044 Step 15:
1045 sergiu
1046
1047 Step 16:
1048 george
1049
1050 Step 17:
1051 silviu
1052
1053 Step 18:
1054 laura
1055
1056 The route of the ticket:
1057 alin > gigel V miruna ^ gigel V miruna < andreea # andreea # andreea # andreea # andreea ^ alin > gigel V miruna < andreea V sergiu > george >> silviu > laura
1058
1059 The cost of the road (f = g + h): 1213
1060 Real Cost (g): 8
1061 Estimate Cost (h): 1205
1062 Length of the road: 18
1063 Time: 8.243s
```

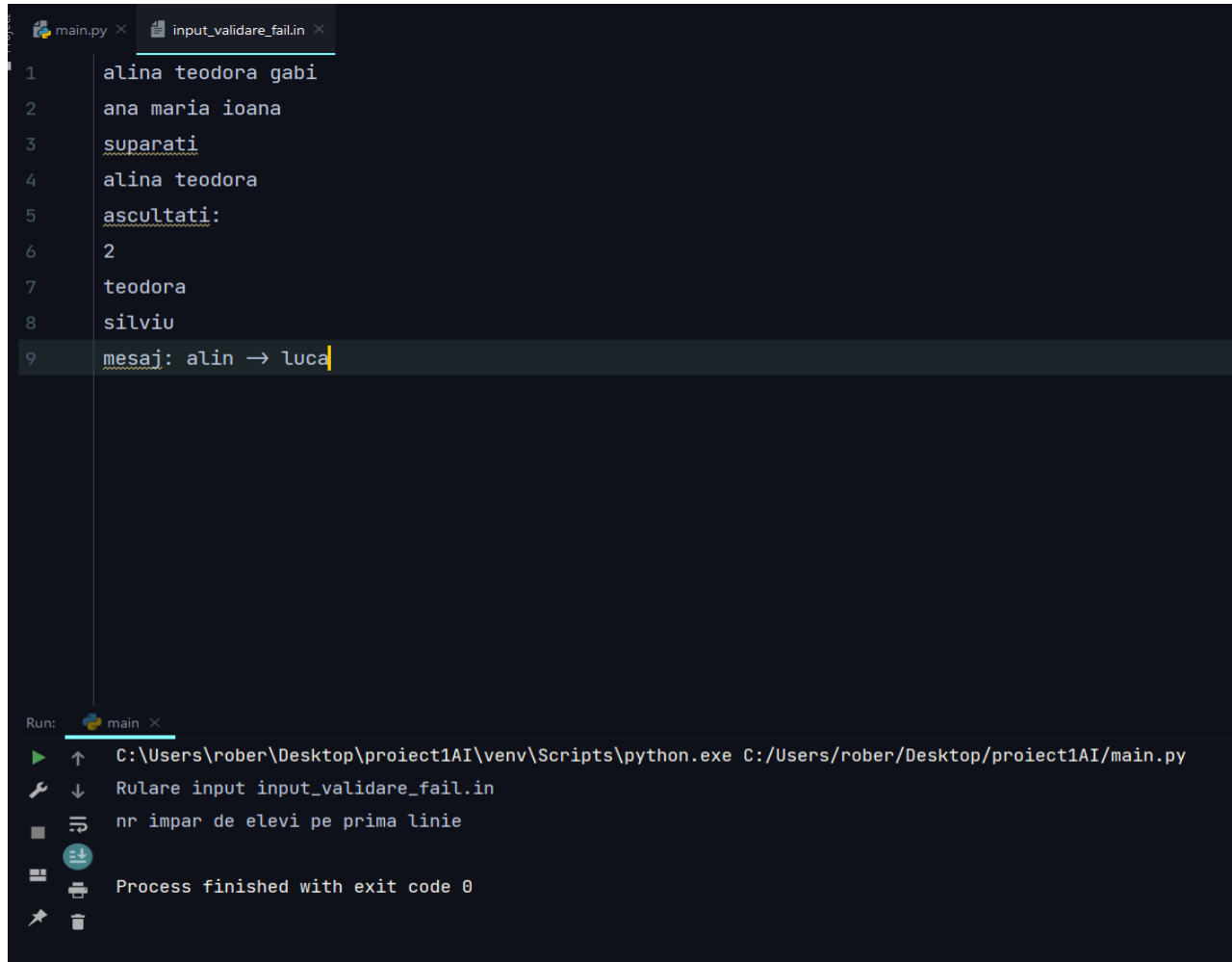
Euristica admisibilă 1 (distanța Manhattan):

```
859 Solution:
860 Step 1:
861 alin
862
863 Step 2:
864 gigel
865
866 Step 3:
867 miruna
868
869 Step 4:
870 andreea
871
872 Step 5:
873 sergiu
874
875 Step 6:
876 sergiu
877
878 Step 7:
879 sergiu
880
881 Step 8:
882 sergiu
883
884 Step 9:
885 sergiu
886
887 Step 10:
888 sergiu
889
890 Step 11:
891 sergiu
892
893 Step 12:
894 sergiu
895
896 Step 13:
897 sergiu
898
899 Step 14:
900 sergiu
901
902 Step 15:
903 sergiu
904
905 Step 16:
906 george
907
908 Step 17:
909 silviu
910
911 Step 18:
912 laura
913
914 The route of the ticket:
915 alin > gigel V miruna < andreea V sergiu # sergiu # sergiu # sergiu # sergiu # sergiu # sergiu # sergiu # sergiu # sergiu # sergiu > george >> silviu > laura
916
917 The cost of the road (f = g + h): 4
918 Real Cost (g): 4
919 Estimate Cost (h): 0
920 Length of the road: 18
921 Time: 0.001s
```


4. Validări și optimizări

4.1 Validări.

➤ *Număr par de elevi:*



```
1  alina teodora gabi
2  ana maria ioana
3  suparati
4  alina teodora
5  ascultati:
6  2
7  teodora
8  silviu
9  mesaj: alin → luca
```

Run: main

▶ C:\Users\rober\Desktop\proiect1AI\venv\Scripts\python.exe C:/Users/rober/Desktop/proiect1AI/main.py

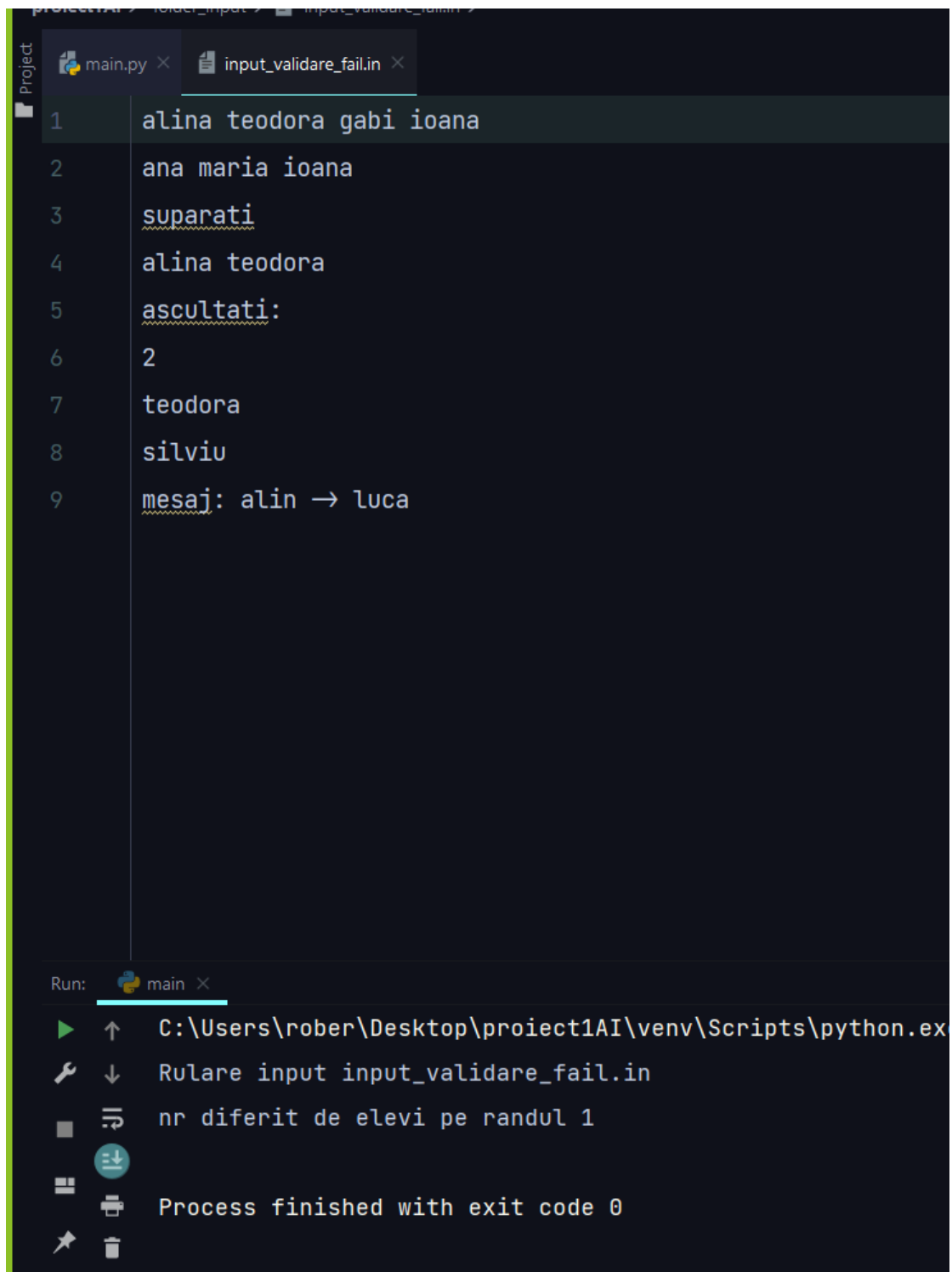
⚙ Rulare input input_validare_fail.in

■ nr impar de elevi pe prima linie

🔄 Process finished with exit code 0

```
114 while "suparati" not in linie:
115     linie = linie.split()
116     if i == 0:
117         assert len(linie) % 2 == 0, "nr impar de elevi pe prima linie"
118     else:
119         assert len(linie) == len(self.matrice[-1]), "nr diferit de elevi pe randul {}".format(i)
120     self.matrice.append(linie)
```

➤ **Număr diferit de elevi:**



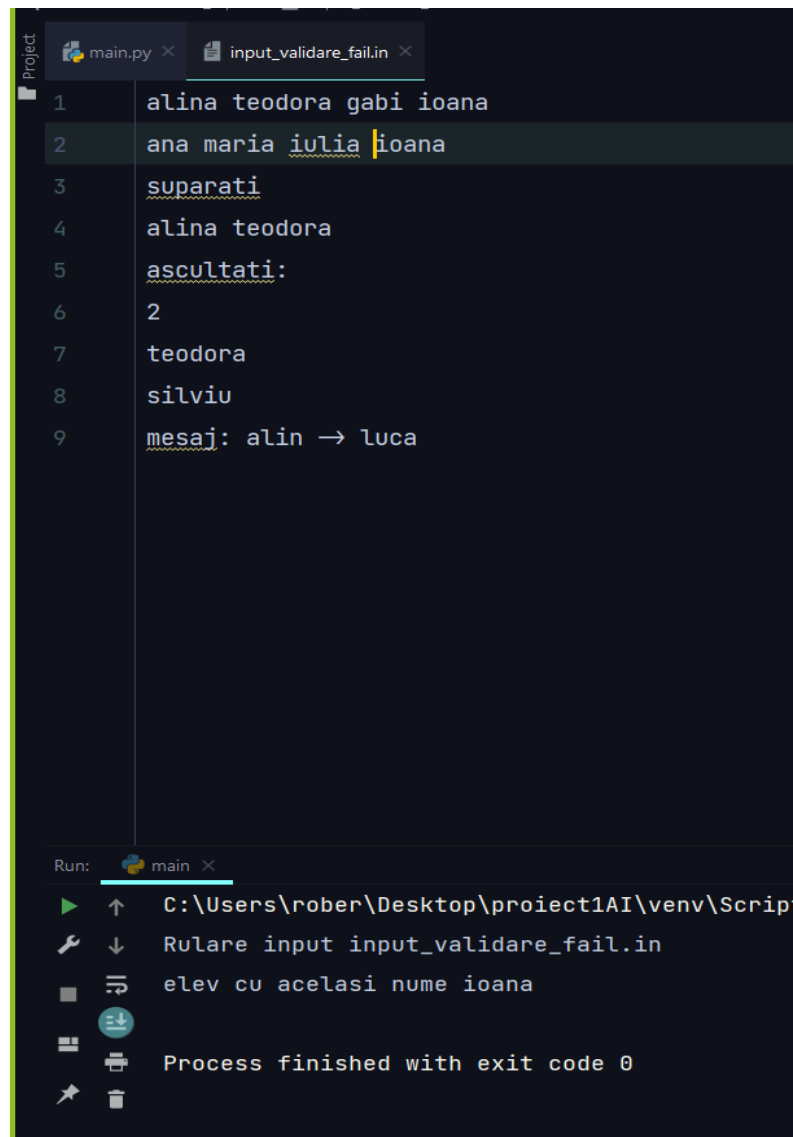
The screenshot shows a Python IDE with two tabs: `main.py` and `input_validare_fail.in`. The `input_validare_fail.in` tab is active, displaying the following input data:

```
1 alina teodora gabi ioana
2 ana maria ioana
3 suparati
4 alina teodora
5 ascultati:
6 2
7 teodora
8 silviu
9 mesaj: alin → luca
```

Below the editor, the Run console shows the execution details for `main`:

- Command: `C:\Users\rober\Desktop\proiect1AI\venv\Scripts\python.exe`
- Argument: `Rulare input input_validare_fail.in`
- Output: `nr diferit de elevi pe randul 1`
- Status: `Process finished with exit code 0`

➤ *Elev cu același nume:*



The screenshot shows a Python IDE with two tabs: `main.py` and `input_validare_fail.in`. The `input_validare_fail.in` tab is active, displaying the following input:

```
1 alina teodora gabi ioana
2 ana maria iulia ioana
3 suparati
4 alina teodora
5 ascultati:
6 2
7 teodora
8 silviu
9 mesaj: alin → luca
```

Below the editor, the `Run` panel shows the execution of `main.py` with the following output:

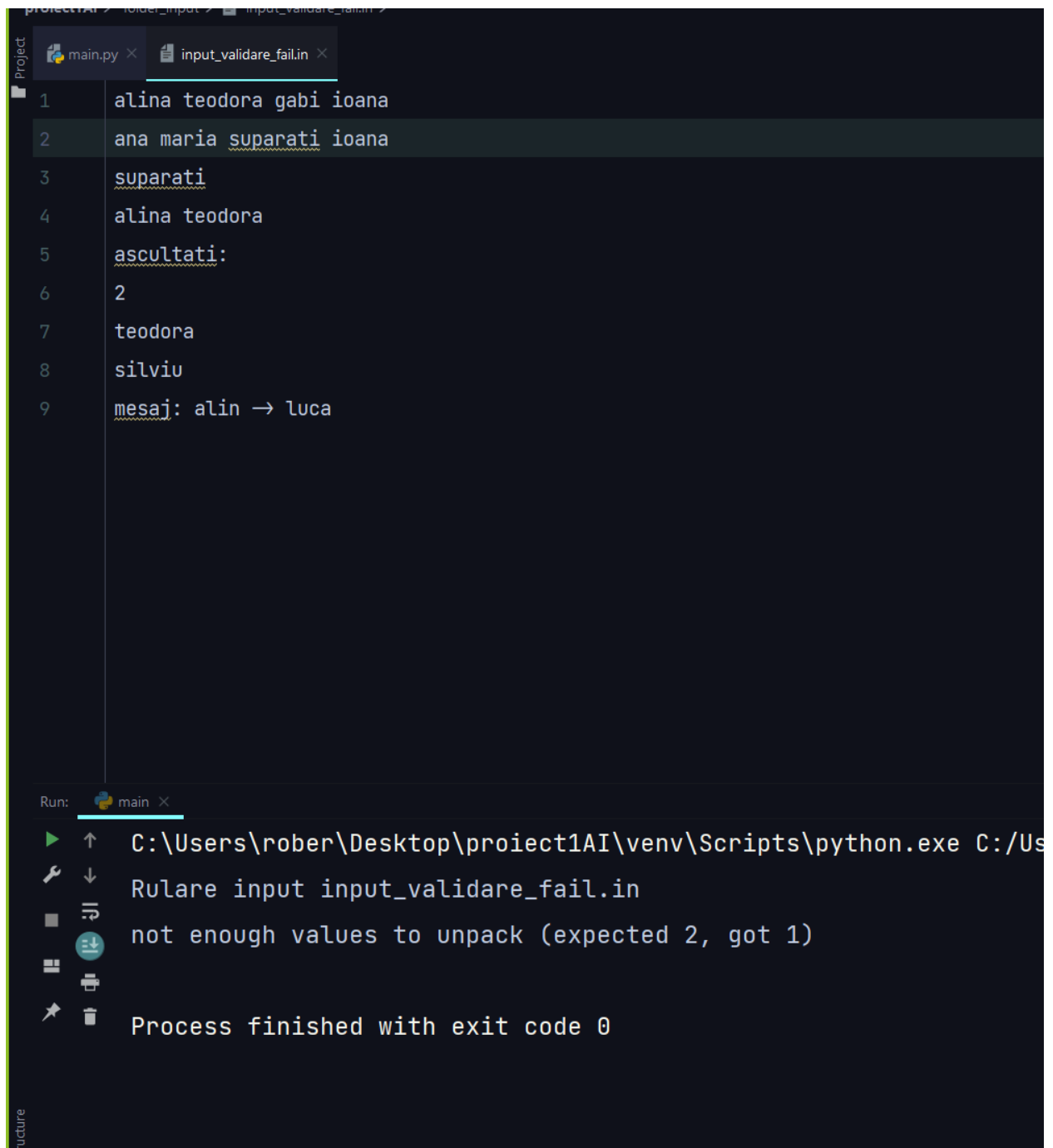
```
↑ C:\Users\rober\Desktop\proiect1AI\venv\Scripts\python.exe
↓ Rulare input input_validare_fail.in
elev cu același nume ioana
Process finished with exit code 0
```



The screenshot shows a snippet of a Python script with line numbers 121 to 130. The code is as follows:

```
121 j = 0
122 for elev in self.matrice[-1]:
123     if elev != "liber":
124         assert elev not in self.pozitie_elev, "elev cu același nume {}".format(elev)
125         self.suparati[elev] = []
126         self.pozitie_elev[elev] = (i, j)
127     j += 1
128     linie = fin.readline()
129     i += 1
130
```

➤ *Elev care se numește supărați:*



```
Project
main.py x input_validare_fail.in x
1 alina teodora gabi ioana
2 ana maria suparati ioana
3 suparati
4 alina teodora
5 ascultati:
6 2
7 teodora
8 silviu
9 mesaj: alin → luca

Run: main x
C:\Users\rober\Desktop\proiect1AI\venv\Scripts\python.exe C:/Us
Rulare input input_validare_fail.in
not enough values to unpack (expected 2, got 1)
Process finished with exit code 0
```

- În rest, se mai aruncă excepții de la transformări (exemplu lipsește timpul de ascultare pe elev):

```
1 alina teodora gabi ioana
2 ana maria iulia daria
3 suparati
4 alina teodora
5 ascultati:
6 ana
7 teodora
8 silviu
9 mesaj: alin -> luca
```

Run: main

C:\Users\rober\Desktop\proiect1AI\venv\Scripts\python.exe C:/U

Rulare input input_validare_fail.in

invalid literal for int() with base 10: 'ana\n'

Process finished with exit code 0

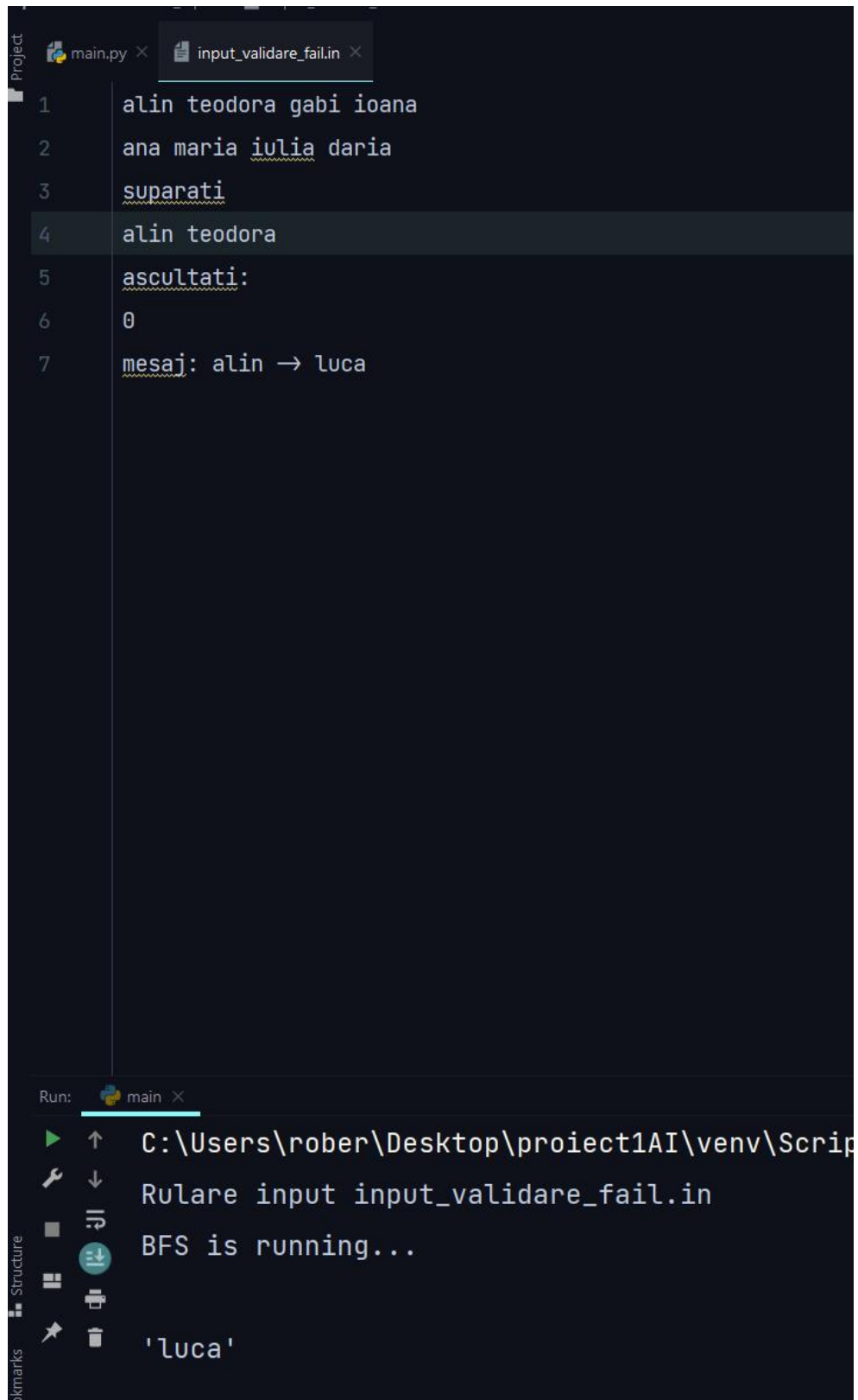
- *Dacă nu avem niciun elev ascultat*



```
Project
main.py x input_validare_fail.in x
1 alina teodora gabi ioana
2 ana maria iulia daria
3 suparati
4 alina teodora
5 mesaj: alin → luca

Run: main x
C:\Users\rober\Desktop\proiect1AI\venv\Script
Rulare input input_validare_fail.in
too many values to unpack (expected 2)
Process finished with exit code 0
```

- Dacă este ascultat un elev care nu există sau biletul se deplasează de la/către un elev care nu există se aruncă excepție cu acel nume.



The screenshot shows a Python IDE with two tabs: `main.py` and `input_validare_fail.in`. The `input_validare_fail.in` tab is active, displaying the following code:

```
1 alin teodora gabi ioana
2 ana maria iulia daria
3 suparati
4 alin teodora
5 ascultati:
6 0
7 mesaj: alin → luca
```

The code is executed, and the output is shown in the Run console at the bottom. The output indicates that the script is running and that the message is 'luca'.

```
Run: main ×
C:\Users\rober\Desktop\proiect1AI\venv\Scripts\python.exe
Rulare input input_validare_fail.in
BFS is running...
'luca'
```

- Numărul de soluții și timeout-ul primite în linia de comandă trebuie să fie mai mari > 0.

```
619         assert nSol > 0, "nSol must be > 0"
620         assert timeout > 0, "timeout must be > 0"
621
```

```
619         assert nSol > 0, "nSol must be > 0"
620         assert timeout > 0, "timeout must be > 0"
621
622         # creez folder-ul de output daca acesta nu exista
623         if not os.path.exists(outputPath):
624             os.mkdir(outputPath)
625
626         # sys.setrecursionlimit(1000000)
627         # index = 0
628
629         for inputFile in listdir('./' + inputPath):
            if __name__ == "__main__":
                Terminal: Local X + v
                Windows PowerShell
                Copyright (C) Microsoft Corporation. All rights reserved.
                Try the new cross-platform PowerShell https://aka.ms/pscore6
                PS C:\Users\rober\Desktop\proiect1AI> python main.py -i -o -n 0 -t 0
                usage: C:\Users\rober\Desktop\proiect1AI\main.py -i/--input in-o/--output out-n/--nr-solutii n
                main.py: error: argument -i/--input: expected one argument
                PS C:\Users\rober\Desktop\proiect1AI> python main.py -n 0 -t 0
                Traceback (most recent call last):
                  File "C:\Users\rober\Desktop\proiect1AI\main.py", line 619, in <module>
                    assert nSol > 0, "nSol must be > 0"
                AssertionError: nSol must be > 0
                PS C:\Users\rober\Desktop\proiect1AI>
```

```
if __name__ == "__main__":
    Terminal: Local X + v
    AssertionError: nSol must be > 0
    PS C:\Users\rober\Desktop\proiect1AI> python main.py -t 0
    Traceback (most recent call last):
      File "C:\Users\rober\Desktop\proiect1AI\main.py", line 620, in <module>
        assert timeout > 0, "timeout must be > 0"
    AssertionError: timeout must be > 0
    PS C:\Users\rober\Desktop\proiect1AI>
```


4.2 Optimizări.

- Găsirea unui mod de reprezentare a stării, cât mai eficient.
 - ✓ Starea curentă = nod
- Verificarea că nodul scope este diferit de nodul inițial, prin intermediul funcției `testeaza_scop_initial`.

```
269
270 def testeaza_scop_initial(self, start: NodParcure, nrSolutiiCautate):
271     if self.testeaza_scop(start):
272         end_time = time.time()
273         fout.write("Solutie:\n")
274         start.afisDrum(True, True)
275         fout.write("Timp: " + str(round(end_time - self.start_time)) + "s\n")
276         fout.write("\n-----\n")
277         nrSolutiiCautate -= 1
278         if nrSolutiiCautate == 0:
279             fout.write("Numarul total de noduri calculate: " + str(self.total) + "\n")
280             fout.write("Numarul maxim de noduri din memorie: " + str(self.maxim) + "\n")
281         return nrSolutiiCautate
282
```

- Implementarea cât mai eficientă a algoritmilor cu care se rulează programul, folosind eventual module care oferă structuri de date performante.
 - ✓ Algoritmul Breadth First este implementat cu `Queue`.
 - ✓ Algoritmul A* este implementat cu `PriorityQueue`.
 - ✓ Algoritmul A* Optimizat este implementat cu `Heapq`, am definit și operatorii `__lt__` și `__eq__` pentru a putea compara nodurile mai eficient. Ordonarea este făcută crescător după `f`, iar în cazul în care `f`-urile sunt egale, descrescător după `g`.

5. Compararea algoritmilor

Vom căuta câte o soluție pentru fiecare algoritm.

Pentru input3:

Algoritmul utilizat	Tipul de euristică folosit	Lungimea drumului	Costul drumului	Număr total de noduri generate	Număr maxim de noduri din memorie	Timp găsim soluție
BFS	-	18	12	1458232	872483	19.3483s
DFS	-	20	6	565	19	0.004s
DFI	-	18	12	2453996	17	19.2805s
A*	Banală	18	4	1553	875	0.018s
A*	Admisibilă 1	18	4	131	83	0.002s
A*	Admisibilă 2	18	4	135	85	0.001s
A*	Neadmisibilă	18	8	1658762	799972	31.2023s
A* OPT	Banală	18	4	255	50	0.0039894s
A* OPT	Admisibilă 1	18	4	70	27	0.0009987s
A* OPT	Admisibilă 2	18	4	41	41	0.0009973s
A* OPT	Neadmisibilă	18	14	111	22	0.005984s
IDA*	Banală	18	4	4912	20	0.0369s
IDA*	Admisibilă 1	18	4	69	17	0.001s
IDA*	Admisibilă 2	18	4	133	17	0.002s
IDA*	Neadmisibilă	18	8	808687	28	8.243s

Pentru input4 (cu timeout setat default 100s):

Algoritmul utilizat	Tipul de euristică folosit	Lungimea drumului	Costul drumului	Număr total de noduri generate	Număr maxim de noduri din memorie	Țimp
BFS	-	21	17	1795887	1096865	20.96s
DFS	-	39	35	101	38	0.001s
DFI	-	21	17	2979438	20	26.2698s
A*	Banală	-	-	5785486	3364216	TLE
A*	Admisibilă 1	23	17	80622	40524	1.3643s
A*	Admisibilă 2	25	17	102865	56417	1.7952s
A*	Neadmisibilă	-	-	4263386	1434622	TLE
A* OPT	Banală	23	17	773	196	0.015826s
A* OPT	Admisibilă 1	23	17	63	57	0.000997s
A* OPT	Admisibilă 2	21	17	215	119	0.003989s
A* OPT	Neadmisibilă	23	17	452	247	0.015957s
IDA*	Banală	-	-	9617440	32	TLE
IDA*	Admisibilă 1	21	17	116657	24	1.0712s
IDA*	Admisibilă 2	21	17	1518234	25	14.6498s
IDA*	Neadmisibilă	-	-	7844680	35	TLE

5. Concluzie

A* optimizat este cea mai bună alegere. Se mai observă că acesta se comportă cel mai bine cu euristica Manhattan sau Euclidiană. Acest fapt se datorează întrucât problema noastră ne impune să mergem doar în elevi vecini, la fiecare pas.

Observăm pe problema noastră că IDA* rulează cel mai greu (repetă calculări), dar are cele mai puține noduri ținute în memorie (număr maxim 35). Deci ocupă puțină memorie, dar timpul de găsim al unei soluții este mare.