Sintaxa:
python main.py -i path_to_input_folder -o path_to_output_folder -n NSOL -t TIMEOUT

Unde NSOL = numarul de solutii, TIMEOUT = in secunde.

Exemplu:
python main.py -i input -o output -n 1 -t 10

Format fisier intrare:
Un grid care contine litere si butoane, sub matrice sirul constant "__cuvinte__"
si sub acest șir, vor urma câte unul pe rând cuvintele de obținut în grid.

Exemplu:

@ b o z q
r a s w z
i r u ^ >
p i q i #
__cuvinte__
bip
roz
zar
pisic
ari

Validari si Optimizari:

Reprezentarea cat mai eficienta a starii:
Fiecare stare este reprezentata sub forma unei liste de liste (matrice).

Testare scop:

Pentru a verifica daca o stare este scop am format un sir cu cuvintele din matrice
de pe linii (de la stanga la dreapta) si de pe coloane (de sus in jos). Cuvintele din sir
au fost separate prin spatiu. Am cuvant daca fiecare cuvant de obtinut se afla in acel sir.

```
  def testeaza_scop(self, nodCurent):
    lista_orizontala_verticala = [] #lista cu sirurile de pe linii si coloane
    for i in range(len(nodCurent)):
      lista_orizontala_verticala.append("".join(nodCurent[i]))
    for j in range(len(nodCurent[0])):
      cuvant = ""
      for i in range(len(nodCurent)):
        cuvant = cuvant + str(nodCurent[i][j])
      lista_orizontala_verticala.append(cuvant)
    sir_cuvinte_matrice = "" #sir cu sirurile de pe linii si coloane separate prin spatiu
    for i in range(len(lista_orizontala_verticala)):
      sir_cuvinte_matrice = sir_cuvinte_matrice + " " + lista_orizontala_verticala[i]

    for elem in self.cuvinte:
      if elem not in sir_cuvinte_matrice:
        return 0
    return 1
```

Verificarea corectitudinii datelor de intrare:

- liniile fisierului pana la linia "__cuvinte__" trebuie sa aiba aceeasi lungime
- liniile fisierului pana la linia "__cuvinte__" trebuie sa contina doar litere si simbolurile pt. butoane
- caracterele din input care vor constitui starea initiala trebuie sa fie separate prin spatiu
- sa existe cel putin un cuvant de cautat

```
lungime_linie = len(l[0])
   for linie in l:
      if len(linie) != lungime_linie:
         print("Input error in fisierul {0}! Linii cu lungime diferita. ".format(inp.name))
         sys.exit(-1)

   for linie in l:
      for i in range(1, len(linie), 2):
         if str(linie[i]) != " ":
            print("Input error in fisierul {0}! Caracterele din input care vor constitui starea initiala nu
sunt separate prin spatiu. ".format(inp.name))
            sys.exit(-1)

   for linie in l:
      for i in range(0, len(linie), 2):
         if not(str(linie[i]).islower()) or linie[i] not in ['@', '^', '#', '>']:
            print("Input error in fisierul {0}! Caracter necunoscut in input. ".format(inp.name))
            sys.exit(-1)

   if len(cuvinte) == 0:
      print("Input error in fisierul {0}! Nu exista cuvinte de cautat. ".format(inp.name))
      sys.exit(-1)
```

- sa existe linia "__cuvinte__", dar nu pe primul rand in fisier:

```
l = []
   linie = inp.readline().strip()
   while linie != "__cuvinte__" or linie != "":
      l.append(linie)
      linie = inp.readline().strip()
   #verificarea corectitudinii datelor de intrare:  sa existe linia "__cuvinte__", dar nu pe primul rand in
fisier
   if len(l) == 0:
      print("Input error in fisierul {0}! Fisierul de input nu contine starea initiala. ".format(inp.name))
      sys.exit(-1)
   if linie == "":
      print("Input error in fisierul {0}! Fisierul de input nu contine sirul constant : __cuvinte__.
".format(inp.name))
      sys.exit(-1)
```

Găsirea unui mod de a realiza din starea initială că problema nu are soluții:

- daca lungimea unuia dintre cuvinte este mai mare decat max(nr linii, nr coloane)
- daca vreunul dintre cuvinte contine o litera care nu se gaseste in matrice

```
set_matrice = {y for linie in mat for y in linie}  # formeaza un set cu literele din matrice
   set_cuvinte = {x for cuv in cuvinte for x in cuv}  # formeaza un set cu literele cuvintelor
   cel_mai_lung_cuv = max(cuvinte, key=len)
   # testeaza conditiile de validare
   if len(cel_mai_lung_cuv) > max(len(mat), len(mat[0])) or len(set_cuvinte - set_matrice) > 0:
      for out in [out_bf_df_dfi, out_astar, out_astar_optim, out_idastar]:
         out.write("Input fara solutii!")
         out.close
      return
```

Euristici folosite:

Banala: estimez costul ca fiind 1 daca nu am ajuns in starea scop, si 0 daca am ajuns.

```
      if self.euristica == "banala":
         if self.testeaza_scop(nod):
            return 0
         return 1
```

Admisibila_1: estimez costul ca fiind: (numarul total de cuvinte de obtinut - numarul de cuvinte obtinute) / numarul total de cuvinte de obtinut

```
if self.euristica == "admisibila_1":
   return (len(self.cuvinte) - self.nr_cuv_formate(mat)) / len(self.cuvinte)
```

Admisibila_2: estimez costul ca fiind: (numarul total de cuvinte de obtinut - numarul de cuvinte obtinute complet si incomplet) / numarul total de cuvinte de obtinut

Neadmisibila: indiferent de butonul activat, costul estimat al activarii este mai mare
decat costul real al activarii butonului respectiv

```
      mat = nod.info
      nr_butoane = 0
      for i in range(mat):
         for j in range(mat[0]):
            if mat[i][j] in ['@', '#', '^', '>']:
               nr_butoane = nr_butoane + 1
      cost_maxim_urcator = len(mat) - nr_butoane - 1
      cost_maxim_dreptator = len(mat[0]) - nr_butoane - 1
      cost_maxim_interschimbator = 2
      cost_maxim_invartitor = 8
      if self.euristica == "neadmisibila":
         return max(cost_maxim_urcator, cost_maxim_dreptator, cost_maxim_interschimbator,
cost_maxim_invartitor) + 1
      return 0
```

Comparatii: tabel

Input: c_nu_blocheaza_nimic

Algoritm	BF
Timp (ms)	293
Lungime drum	3
Stari max. existente	4103
Nr. stari totale	4506

Algoritm	A*				A* optim				IDA*			
Euristica	Banala	Adm1	Adm2	Neadm	Banala	Adm1	Adm2	Neadm	Banala	Adm1	Adm2	Neadm
Timp (ms)	1912	3019		1904	121	152		106	111	304		95
Lungime drum	3	3		3	3	3		3	3	3		3
Stari max. existente	7799	9361		7799	805	823		805	58	59		58
Nr. stari totale	8561	10272		8561	1875	1996		1875	3375	8997		3375

Input: d_blocheaza_unele

Algoritm	BF
Timp (ms)	Timeout
Lungime drum	
Stari max. existente	
Nr. stari totale	

Algoritm	A*				A* optim				IDA*			
Euristica	Banala	Adm1	Adm2	Neadm	Banala	Adm1	Adm2	Neadm	Banala	Adm1	Adm2	Neadm
Timp (ms)	Timeout	Timeout		Timeout	3903	1778		5318	1643	3617		1349
Lungime drum					7	7		7	5	5		5
Stari max. existente					4397	3243		4397	81	81		81
Nr. stari totale					13334	9912		13334	44768	116386		44768