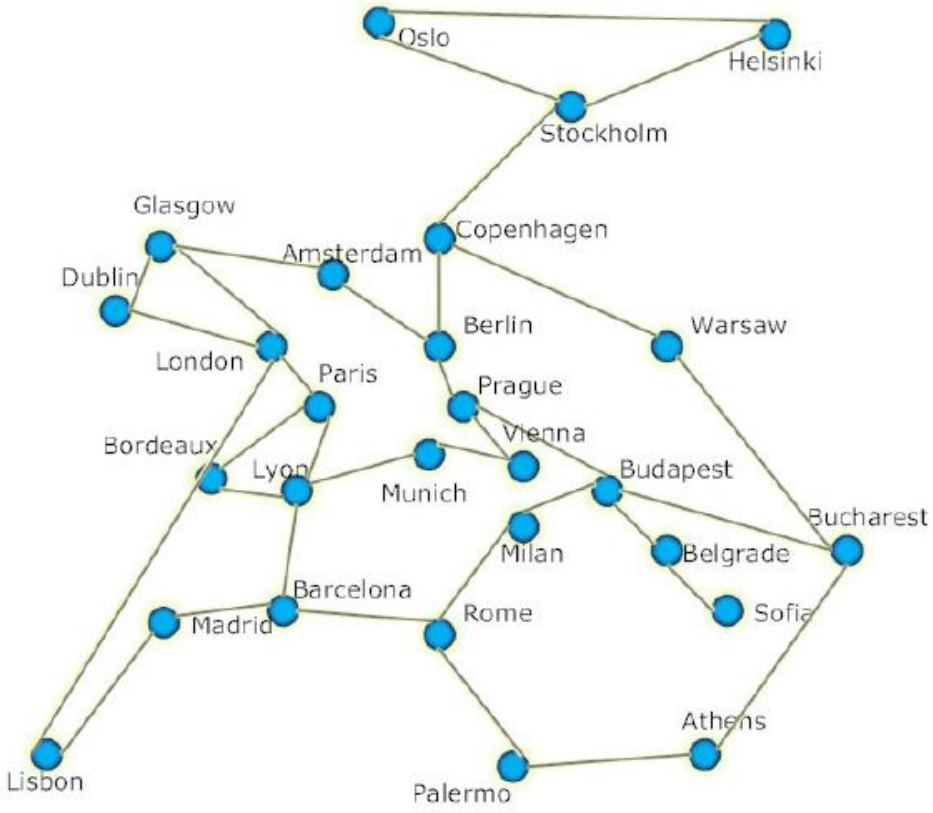
Proiect 2

Problema gasirii unui drum

Enuntul problemei:

Dându-se o hartă a oraşelor din Europa şi a distanţelor între ele, să se găsească cel mai scurt drum de la Bucureşti la London ştiind că distanţele între oraşe sunt date mai jos.



Distantele intre orasele de pe harta:

Olso - Helsinki:	970	Rome: Milan:	681	Madrid - Barcelona:	628
Helsinki - Stockholm:	400	Milan - Budapest:	789	Madrid - Lisbon:	638
Oslo - Stockholm:	570	Vienna - Budapest:	217	Lisbon - London:	2210
Stockholm - Copenhagen:	522	Vienna - Munich:	458	Barcelona - Lyon:	644
Copenhagen - Warsaw:	668	Prague - Vienna:	312	Paris - London:	414
Warsaw - Bucharest:	946	Prague - Berlin:	354	London - Dublin:	463
Bucharest - Athens:	1300	Berlin - Copenhagen:	743	London - Glasgow:	667
Budapest - Bucharest:	900	Berlin - Amsterdam:	648	Glasgow - Amsterdam:	711
Budapest - Belgrade:	316	Munich - Lyon:	753	Budapest - Prague:	443
Belgrade - Sofia:	330	Lyon - Paris:	481	Barcelona - Rome:	1471
Rome - Palermo:	1043	Lyon - Bordeaux:	542	Paris - Bordeaux:	579
Palermo - Athens:	907			Glasgow - Dublin:	306

Distantele in linie dreapta de la Bucuresti la fiecare oras de pe harta:

Amsterdam	2280	Copenhagen	2250	Madrid	3300	Rome	1140
Athens	1300	Dublin	2530	Milan	1750	Sofia	390
Barcelona	2670	Glasgow	2470	Munich	1600	Stockholm	2890
Belgrade	630	Helsinki	2820	Oslo	2870	Vienna	1150
Berlin	1800	Lisbon	3950	Palermo	1280	Warsaw	946
Bordeaux	2100	London	2590	Paris	2970		
Budapest	900	Lyon	1660	Prague	1490		

Citirea datelor de intrare

In cadrul programului, se apeleaza fisiere in care se afla starile initiale si finale ale unei probleme.

Exemplu de continut fisier:

```
StareInitiala: Prague
StareFinala: Bucharest

```

Apelarea functiei de citire din fisier se face in functia main() direct in parametrii problemei.

```
problema = ProblemaDrum(harta, distante, generare_input('input_1'))
```

Functia generare_input() arata astfel:

```
def generare_input(txt_fisier):
    stare_initiala = ''
    stare_finala = ''
    with open(txt_fisier, 'r') as file:
        for line in file:
            if line.startswith("StareInitiala:"):
                stare_initiala = line.split(':')[1].strip()
            if line.startswith("StareFinala:"):
                stare_finala = line.split(':')[1].strip()
    return stare_initiala, stare_finala
```

Sunt folosite 3 tipuri de fisiere de input, primul pentru o problema mai usoara, al doilea pentru o problema fara solutii, si al treilea pentru o problema cu o solutie mai dificila.

In cazul 2, problema fara solutii este construita astfel incat sa se caute un nod fara legatura catre celelalte noduri.

Clasa Nod

Un obiect de tip nod are urmatoarea structura:

```
1 usage
class Nod:
    def __init__(self, oras, cost, distanta, parinte):
        self.oras = str(oras)
        self.distanta = str(distanta)
        self.cost = cost
        self.parinte = parinte
```

Generarea succesorilor

Aceasta functie este definita in cadrul tipului de problema.

```
3 usages (3 dynamic)
def aplica_tranzitii(self, stare_curenta):
    succesori_oras = []
    if self.este_stare_valida(stare_curenta):
        if stare_curenta in self.harta:
            succesori_oras = list(self.harta[stare_curenta].keys())
    return succesori_oras
```

Testarea starii finale

Functia este de forma:

```
def is_stare_finala(self, stare):
    return stare == self.stare_finala
```

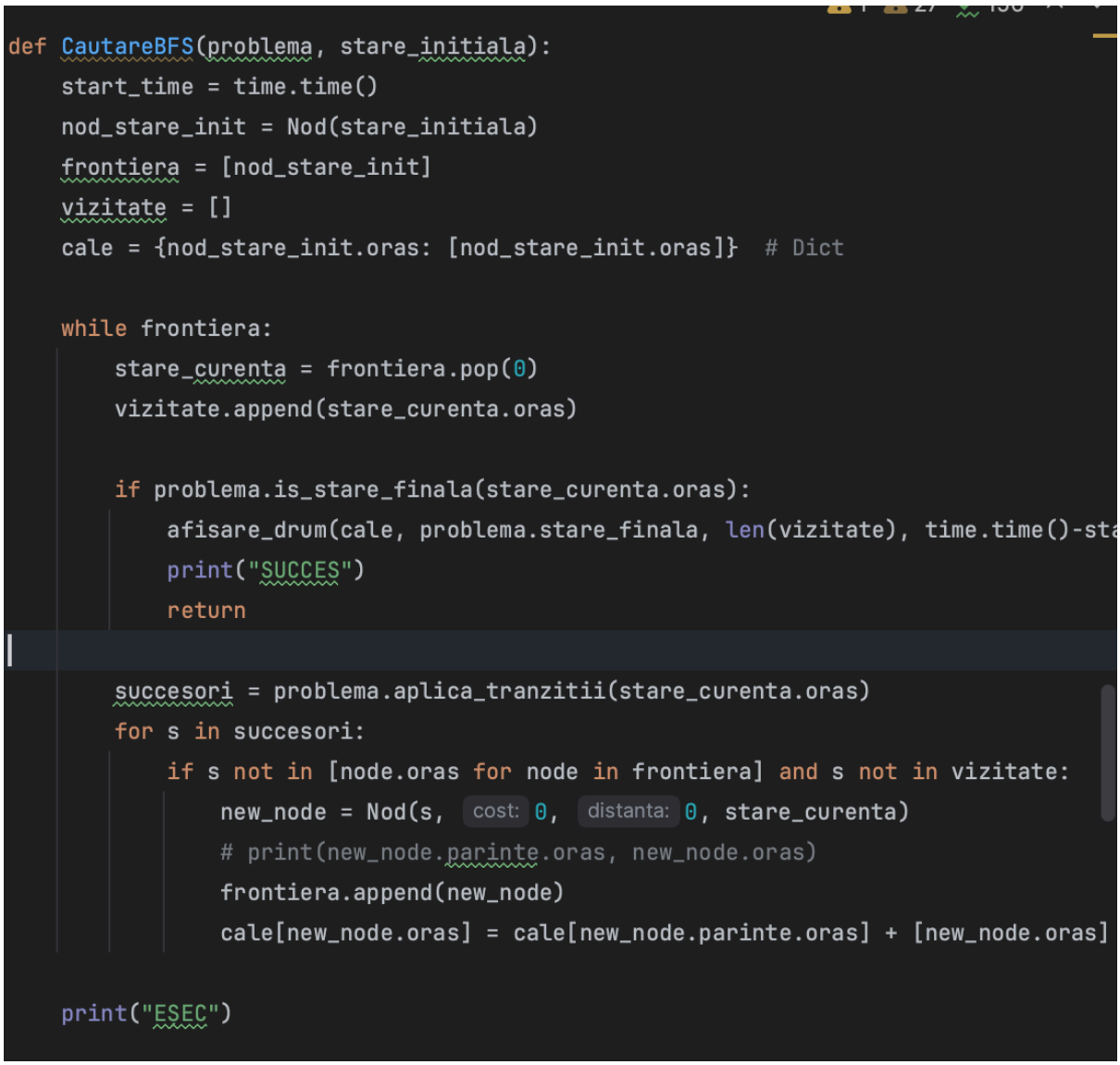
Este apelata in cadrul functiilor de cautare astfel:

```
if problema.is_stare_finala(stare_curenta):
    print(problema.afiseaza_drum(cale))
    print("SUCCES")
    return
```

Implementare algoritmi de cautare

Neinformati:

BFS



```
def CautareBFS(problema, stare_initiala):
    start_time = time.time()
    nod_stare_init = Nod(stare_initiala)
    frontiera = [nod_stare_init]
    vizitate = []
    cale = {nod_stare_init.oras: [nod_stare_init.oras]}  # Dict

    while frontiera:
        stare_curenta = frontiera.pop(0)
        vizitate.append(stare_curenta.oras)

        if problema.is_stare_finala(stare_curenta.oras):
            afisare_drum(cale, problema.stare_finala, len(vizitate), time.time()-sta
            print("SUCCES")
            return

        succesori = problema.aplica_tranzitii(stare_curenta.oras)
        for s in succesori:
            if s not in [node.oras for node in frontiera] and s not in vizitate:
                new_node = Nod(s, cost: 0, distanta: 0, stare_curenta)
                # print(new_node.parinte.oras, new_node.oras)
                frontiera.append(new_node)
                cale[new_node.oras] = cale[new_node.parinte.oras] + [new_node.oras]

    print("ESEC")
```

Informatii

AStar

```
def astar(problema, stare_initiala):
    start_time = time.time()
    frontiera = priorityQueue()
    frontiera.push(stare_initiala,  cost: 0)
    vizitate = []
    cale = {}
    distante = {}
    distante[stare_initiala] = 0
    cale[stare_initiala] = None
    while not frontiera.isEmpty():
        stareCurenta = frontiera.pop()
        vizitate.append(stareCurenta)
        if problema.is_stare_finala(stareCurenta):
            # afisare_drum(cale, problema.stare_finala, len(vizitate), time.time()-start_time)
            afiseazaDrum(problema, cale, distante, len(vizitate), time.time()-start_time)
            print("SUCCES")
            return

        succesori = problema.aplica_tranzitii(stareCurenta)
        for s in succesori:
            g_cost = distante[stareCurenta] + int(problema.harta[stareCurenta][s])
            # print(s, int(problema.harta[stareCurenta][s]), "now : " + str(distante[stareCurenta]), g_cost)

            if (s not in distante or g_cost < distante[s]):...
    print("ESEC")
```

Afisarea Rezultatelor

```
Drumul de la London la Bucharest se realizeaza astfel:
=====================================================
De la London la Glasgow cu costul de 667
De la Glasgow la Amsterdam cu costul de 1378
De la Amsterdam la Berlin cu costul de 2026
De la Berlin la Prague cu costul de 2380
De la Prague la Budapest cu costul de 2823
De la Budapest la Bucharest cu costul de 3723
Adancimea la care s-a gasit solutia este 6
Numarul total de noduri explorate este 13
Durata totala a algoritmului pentru problema abordata este de 4.1961669921875e-05
SUCCES
```