Fritz Raluca-Mihaela

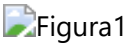
Artifical Intelligence KR Mini-Project

---

1. Problema de cautare (un mesaj...)

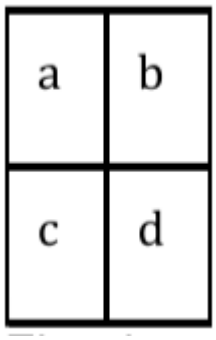
In drum spre scoala un copilas a auzit doi copii mai mari vorbind despre un coleg si prieten bun de-al lui. Cei doi voiau sa-i faca o farsa rautacioasa in prima pauza. Copilasul insa a ajuns dupa ce incepuse ora si a trebuit sa se aseze direct in banca, fara a-si putea avertiza colegul de primejdie. Din fericire i-a venit in minte sa-i trimita din coleg in coleg un bilet cu un mesaj in care sa-l atentioneze despre farsa.

Copii sunt asezati in banci de cate doua persoane. Bancile sunt dispuse in trei coloane astfel:



Un copil poate da fara probleme biletul catre colegul de banca, la fel neobservat de catre profesor poate da biletul catre colegii din fata sau din spatele lui, insa nu si in diagonala, deoarece, intinzanduse peste banca ar atrage privirea profesorului.

Considerand fragmentul de clasa de mai jos:



Elevul a poate trimite biletul doar catre b sau c.

Probleme apar din faptul ca unele locuri in banci pot fi libere, dar si din faptul ca o serie de colegi sunt suparati intre ei si nu-si vorbesc, deci nici nu ar trimite biletul (supararea este reciproca).

De asemenea, trecerea biletelului de pe un rand pe altul este mai anevoioasa, deoarece poate fi vazut foarte usor de catre profesor, de aceea singurele banci intre care se poate face transferul sunt penultimele si ultimele de pe fiecare rand.

Copilul vrea sa scrie pe bilet drumul pe care trebuie sa-l parcurga, de la un coleg la altul, pentru a fi sigur ca nu se rataceste prin clasa si nu mai ajunge la prietenul sau pana la inceputul pauzei.

Se considera prin conventie ca:

- fiecare copil este identificat unic prin numele sau.
- niciun copil nu se numeste 'suparati' sau 'liber'
- locurile libere sunt marcate prin identificatorul 'liber'

Formatul fisierului de intrare este urmatorul. Pe primele linii din fisier se precizeaza asezarea in banci. Fiecare linie cuprinde 6 nume (numele sunt formate doar din litere). Primii doi identificatori corespund primei coloane

de banci, urmatorii doi identificatori coloanei din mijloc, si utltimii doi, ultimei coloane de banci.

Dupa ce se termina liniile cu asezare in banci, apare un rand cu identificatorul suparati. Sub acest rand, sunt trecuti cate doi elevi (numele lor) care sunt suparati intre ei.

Exemplu de fisier de intrare:

```
ionel alina teo eliza carmen monica
george diana bob liber nadia mihai
liber costin anda bogdan dora marin
luiza simona dana cristian tamara dragos
mihnea razvan radu patricia gigel elena
liber andrei oana victor liber dorel
viorel alex ela nicoleta maria gabi
suparati
george ionel
ela nicoleta
victor oana
teo eliza
teo luiza
elena dragos
alina dragos
mesaj: ionel -> dragos
```

Exemplu de drum din fisierul de iesire:

```
ionel > alina v diana v costin v simona v razvan v andrei >> oana ^
radu > patricia v victor v nicoleta >> maria > gabi ^ dorel
^ elena < gigel ^ tamara > dragos
```

In urma rularii se va afisa drumul parcurs, in ordine cronologica. Daca biletelul merge in cadrul aceluiasi rand, deci catre un coleg de banca, in stanga, se va afisa <, daca merge in dreapta, se va afisa >.

Daca biletelul merge spre spatele clasei, intre copiii care transmit biletul se va afisa un v (pe post de sagetica in jos), iar daca merge spre fata clasei, se va afisa ^ pe post de sagetica in sus.

Cand biletul se deplaseaza spre stanga de pe un rand de banci pe altul, se va afisa <<, iar spre dreapta: >>.

Rezolvare:

Pentru incept, se citesc de la tastatura urmatoarele informatii:

- path-ul fisierului de input
- path-ul fisierului de output
- numarul de solutii necesare
- timput de timeout

Dupa acest pas, se citesc din fisierul de input, toate datele din acesta si se salveaza in liste / tupluri / seturi, in functie de modul in care vrem sa apelam aceste informattii. De asemenea, in functie de cate randuri de banci

si cate coloane de banci sunt intr-o sala de clasa, am salvat aceste date in 2 variabile globale.

Pentru usurinta apelarii datelor, am salvat in 2 dictionare `coordCopii` si `numeCopii`, coordonatele copiilor in sala, de forma (rand, coloana) in functie de numele acestora, respectiv numele copiilor in functie de coordonatele lor in sala.

Am initializat un dictionar de copii suparati, unde se adauga perechi de copii in momentul in care se citeste sectiunea in care sunt trecuti copii suparati.

Initial realizasem o clasa speciala numita `Nod` care retinea:

- numele copilului
- estimarea costului de la nodul curent la nodul scop

Pentru a evita confuzia in momentul realizarii acestui proiect, am preferat sa folosesc modelul dat la laborator, astfel am folosit:

CLASA NODPARCURGERE

Clasa `NodParcurgere` are proprietatile:

- nume copil
- parinte
- costul de la radacina arborelui pana la nodul curent
- costul estimat pentru drumul care porneste ede la radacina la nodul scop (g+h)
- directia catre un nod anume Pentru ca in cazul problemei noastre putem sa ne deplasam la
    - vecinul de sus `^`,
    - vecinul de jos `v`,
    - vecinul din dreapta `>`,
    - vecinul din stanga `<`,
    - vecinul din dreapta de pe celalalt rand `>>`
    - vecinul din stanga de pe celalalt rand `<<` Am ales sa salvez aceasta directie intr-o proprietate speciala.

In aceasta clasa exista functii ajutatoare care

- afiseaza drumul,
- obtin drumul,
- printeaza in fisier drumul,
- verifica daca un nod face parte din drum
- functie de generare a listei succesorilor
    - in cadrul acestei functii se ia pe rand fiecare caz de vecin(sus, jos, stanga, dreapta + in cazul ultimelor 2 banci si pentru bancile vecine separate) si se seteaza directia fiecaruia dintre ele apoi ii adaugam in lista succesorilor
- calcularea costului estimativ de la nodul curent la nodul scop
- verificare daca o banca este libera sau copilul din banca vecina si copilul curent sunt suparati
- testare daca nodul curent este nodul scop

CLASA GRAPHPROBLEMA

Clasa `GraphProblema` reprezinta o clasa care are doar 2 proprietati: nodStart si nodScop.

Initial am vrut sa implementez in aceasta clasa si listele specifice din care sa rezulte arborele de cautare `G`.

In cadrul acestei clase se regaseste si algoritmul `A*` prin functia `a_star()` in care se verifica in primul rand, cu ajutorul functiei ajutatoare `testSolutiePosibila`, care verifica daca in jurul `nodStart` si `nodScop` exista numai locuri libere, numai locuri cu copii suparati sau combinatia dintre locuri libere si copii suparati.

Apoi se aplica algoritmul `A*` propriu zis, si anume:

- se adauga o lista open care continue doar `nodStart`
- se adauga o lista vida numita `closed`
- se folosesc 2 variabile de tip boolean care sunt initializate cu `False` si vor fi folosite ulterior la verificare finalizarii programului cu succes(`end`) sau esec(`esteOpenVida`)
- la parcurgere a unui nod, daca lista open este vida, daca acesasta este vida => `esteOpenVida` se schimba in True si se termina programul cu esec
- daca open nu este vida, se continua rularea programului, elimindandu-se intr-o variabila locala, primul element din lista `open`, acesta adaugandu-se la lista `closed`
- se testeaza daca nodul curent este nodul scop, daca acesta este nodul scop => `end` se schimba in True` si se printeaza in fisier drumul obtinut
- daca nodul curent nu este nodul scop=> se continua programul prin preluarea succesorilor acestui nod in variabila numita `multimeaM`
- cu ajutorului unui `for`, vom parcurge aceasta lista de succesori si vom verifica daca acest succesor se afla in listele `open` sau `closed` si se va scoate din ambele -> ulterior fiecare succesor va fi adaugat in functie de valoarea costului estimativ de la nodul radacina la nodul scop
- in continuare se parcurge din nou `multimeaM` de succesori, pentru ca insertia in lista `open` sa fie realizata, iar lista `open` sa fie sortata
- daca la finalul acestui proces `end` este inca `False`, inseamna ca aceasta problema de cautare nu are o solutie.

In cadrul acestei clase am creat, dupa cum am mentionat mai sus si clasa ajutatoare `testSolutiePosibila()`, impreuna cu o alta clasa ajutatoare `esteInLista()` care, primind un copil si o lista, verifica daca intr-un lista data, se gaseste copilul dat.

Ulterior am introdus o serie de algoritmi precum:

- breadth first,
- depth first,
- depth first iterativ,
- ida_star, doar la nivel simplist si de baza, prezentat la laboratoare.

La finalul fisierului `main.py` se initializeaza `graphProblema` ca un obiect de tip `GraphProblema` cu nodul start si nodul scop din partea initiala a fisierului.

Urmatorii pasi inainte de aplicarea aloritmilor sunt selectarea tipului de euristica si de algoritm.

Initial am vrut sa folosesc 4 tipuri diferite de euristici: banala, admisibila 1, admisibila 2 si neadmisibila.

Euristica banala:

- este nodScop => cost 0

- nu este nodScop => cost 1