

Universitatea din Craiova
Facultatea de Automatică, Calculatoare și Electronică



Problema arcașilor

Inteligența Artificială

Studentă: Bucă-Ghebaură Elizabetha Alexandrina

Specializarea: Calculatoare Română

Grupa: CR 2.1A

An: II

Iunie 2021

Introducere cuprins

1	Enuntul problemei	2
2	Descrierea si intelegerea problemei	3
3	Algoritmii propusi	5
4	Date experimentale	7
5	Proiectarea aplicației experimentale	7
5.1	Structura de nivel inalt a aplicatiei	7
5.2	Specificarea datelor de intrare	8
5.3	Specificarea ieșirilor/rezultatelor	8
5.4	Lista tuturor modulelor aplicației și descrierea lor . .	11
5.5	Lista tuturor funcțiilor aplicației, grupate pe module	12
5.5.1	Descrierea scopului functiei	12
5.5.2	Descrierea fiecarui parametru	12
5.5.3	Semnificația valorii de return	12
6	Concluzii	13
7	Referinte bibliografice	14

Abstract

Acest raport este o introducere in obiectivele de dezvoltare a temei de casa la disciplina Inteligenta Artificiala. Documentul contine, de asemenea, o descriere a livrabilelor.

1 Enuntul problemei

Să presupunem grila $k \times k$ prezentată în Figura 2. Grila este configurată cu un model de pereți. Vi se cere să plasați n arcași pe această grilă astfel încât să nu se poată împușca reciproc. Un arcaș poate trage în sus, în jos, la stânga, la dreapta și, de asemenea, în diagonală, iar împușcătura sa poate ajunge la cel mult w locații în toate direcțiile, până la marginile grilei.

- Scrieți o formulare detaliată pentru această problemă de căutare.
- Identificați un algoritm de căutare pentru această sarcină și explicați alegerea dvs.

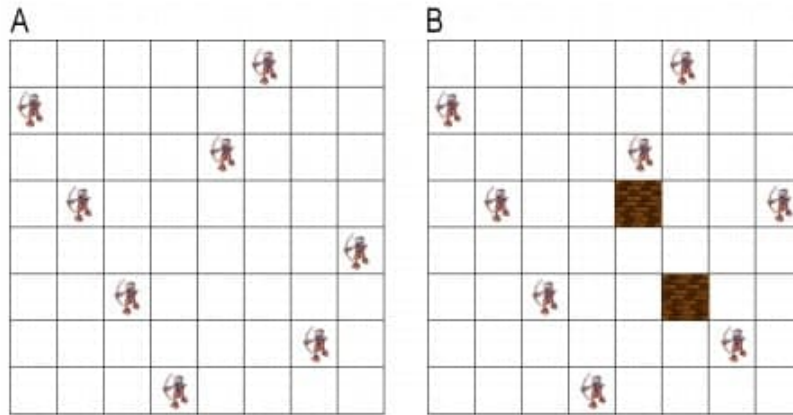
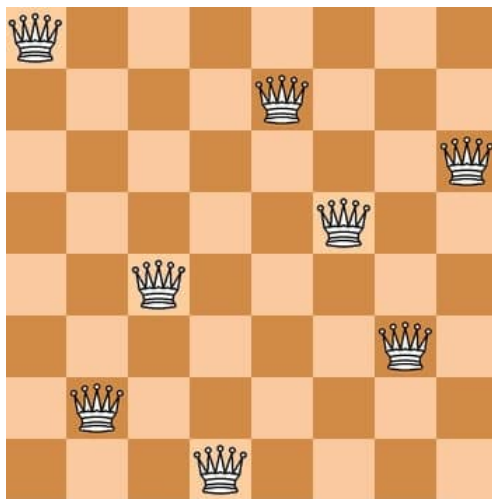


Figure 2: Valid arrangements of archers so that they cannot shoot each other. (A) no walls added. (B) two walls added such that the archer in the last column cannot shoot the archer in the second or fourth column. Note that these solutions do not contain any two archers on the same row, column and diagonal of the grid so they are valid for whatever value of w .

2 Descrierea si intelegerea problemei

Aceasta problema este asemanatoare cu cea a reginelor (the eight queens puzzle).



Problema damelor (sau problema reginelor) tratează plasarea a opt regine de șah pe o tablă de șah $n \times n$ astfel încât să nu existe două regine care se amenință reciproc. Astfel, se caută o soluție astfel încât nicio pereche de două regine să nu fie pe același rând, pe aceeași coloană, sau pe aceeași diagonală.

Similar, la problema actuala, trebuie sa plasam n arcasi pe o grila $k \times k$ astfel incat acestia sa nu se poata impusca reciproc; de asemenea, nu pot fi pe acelasi rand/coloana/diagonala mai multi arcasi, deci nu trebuie sa se suprapuna. Arcasii pot trage cu sageti in toate directiile (ca la problema cu reginele).

Personal, am gandit aceasta problema, pe scurt, astfel: sa folosesc algoritmul **DFS** pentru a vedea pozitia fiecarui arcas pe grila, sa verific apoi daca se suprapun doi sau mai multi arcasi in acelasi loc, lucru care trebuie facut imposibil si sa adaug un perete acolo unde se afla mai multi arcasi pe aceeasi linie/coloana/diagonala.

Starea initiala

La inceput avem un tabel gol, de dimensiune $\mathbf{N} = \mathbf{k} \times \mathbf{k}$ (in cazul de fata $N = 4$, in problema noastra - codul creat de mine), si un **pos** care reprezinta pozitiile arcasilor, pozitiile pe care le vom afla pe parcurs. Acesti arcasi pot trage cu arcul in toate directiile; acest fapt va fi rezolvat in stare finala.

Starea finala

Dupa ce vom trece prin toate conditiile, si anume acelea in care se afla mai multi arcasi pe aceeasi linie/coloana/diagonala, vom trece peretii acolo unde se intampla aceste conditii, iar in final vom afisa tot tabelul, cu arcasii si zidurile.

Actiunile arcasilor

Un arcas poate trage cu arcul in toate directiile, mai exact in sus, jos, stanga, dreapta si diagonala. Dar acesti arcasi nu se pot impusca unul pe celalalt, lucru ce, de asemenea, se va rezolva in starea finala.

Deci, ca un rezumat, problema ne cere sa positionam arcasii pe o grila/un tabel $\mathbf{k} \times \mathbf{k}$ astfel incat acestia sa nu traga reciproc in ei (unul in celalalt), iar acolo unde sunt mai multi arcasi pe aceeasi linie/coloana/diagonala, sa se adauge un perete, iar in final sa se afiseze intreg tabelul (matricea/grila), cu toti arcasii si toti peretii.

3 Algoritmii propusi

Un algoritm folosit este cel de afisare a matricei (grilei/tabelului) $k \times k$. Am adaugat si exceptia din figura urmatoare, in cazul in care apare o eroare si nu se poate deschide fisierul de iesire.

```
15 def printTable(): # Afisarea tabelului in fisierul de iesire
16     try:
17         f = open('output.txt', 'a')
18
19     except:
20         print("ERROR: Can't open the output file")
21
22     else:
23         for i in range(N):
24             for j in range(N):
25                 f.write(table[i][j] + ' ')
26             f.write('\n')
27         f.write('\n')
```

In pseudocod:

function **printTable()**

1. **for** i to n
2. **for** j to n
3. end **for**
4. end **for**
5. write table[i][j]

end *function*

Pentru aceasta problema am folosit algoritmul de parcurgere in adancime, si anume *DFS* (**depth-first search**).

Ca o definitie, **DFS** este un algoritm pentru parcurgerea sau căutarea într-o structură de date de tip arbore sau graf. Se începe de la rădăcină (sau alegând un nod arbitrar ca rădăcină în cazul unui graf) și se explorează cât mai mult posibil de-a lungul fiecărei ramuri înainte de a face pași înapoi.

În acest caz, rolul cautării în adâncime *DFS* este de a parcurge pozițiile arcașilor.

```
30 def searchDFS(index): # Cautare DFS pentru pozițiile arcașilor
31     global ok
32     global pos
33
34     if index >= N:
35         if check(index):
36             printNoWalls()
37             printWithWalls()
38             ok = 1
39
40     if ok == 0 and index < N:
41         for col in range(N):
42             pos[index] = col
43             searchDFS(index + 1)
```

Pentru codul meu, pseudocodul este următorul:

function **searchDFS(index)**

1. **if** $\text{index} \geq N$
2. **if** $\text{check}(\text{index})$
3. $\text{printNoWalls}()$
4. $\text{printWithWalls}()$
5. $\text{ok} = 1$
6. **end if**
7. **end if**
8. **if** $\text{ok} = 0$ and $\text{index} < N$ **then**
9. **for** col to N
10. $\text{pos}[\text{index}] = \text{col}$
11. $\text{searchDFS}(\text{index} + 1)$
12. **end for**
13. **end if**
- end function**

4 Date experimentale

Am folosit functia pentru generare automata de date aleatorii `randint`. Aceasta functie returneaza un numar aleator **X** astfel incat $a \leq X \leq b$; parametri: (**start**, **end**).

Acest **start** reprezinta *limita inferioară* - este punctul de plecare de la care se generează întregul aleator, iar acest **end** reprezinta *limita superioară* - este punctul de oprire până la care metoda ar returna întregul aleator.

Pentru a putea folosi aceasta functie, a trebuit sa importez modulul `random`, care oferă acces la diferite funcții utile și una dintre ele fiind capabilă să genereze numere aleatorii, care este **randint()**.

In programul meu, linia de cod unde este folosita aceasta functie este:

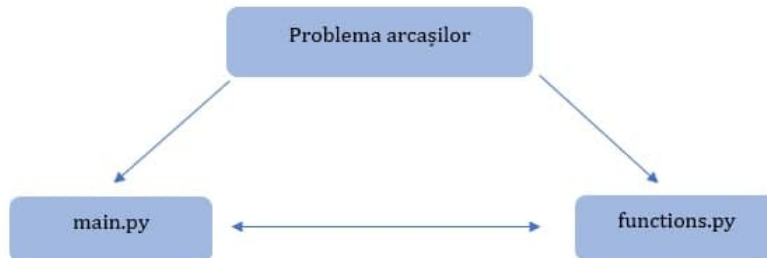
- ***wall_l = randint(0, N - 1)*** → ceea ce inseamna ca va returna un numar intreg aleatoriu astfel incat sa fie cuprins in intervalul (0, N - 1).

5 Proiectarea aplicației experimentale

5.1 Structura de nivel inalt a aplicatiei

Aceasta figura reprezinta programul meu cu toate sursele create si anume ***main.py*** si ***functions.py***.

In ***main.py*** sunt apelate functiile problemei, cu timpul de executie, iar in ***function.py*** sunt implementate toate functiile problemei.



5.2 Specificarea datelor de intrare

Ca date de intrare am ales cateva variabile de tip intreg si anume:

- * **N** - dimensiunea tabelului, care in cazul de fata este 4;
- * **pos** - pozitiile arcașilor care vor fi aflate pe parcurs;
- * **wall_l** - peretele de pe linie, luat aleatoriu;
- * **wall_c** - peretele de pe coloana, care de fapt este pozitia peretelui de pe linie.

Am ales sa folosesc si cele doua variabile globale si anume **ok** si **pos**. Aceste variabile globale pot fi folosite in afara sau in interiorul functiei.

Funcția **hash()**, in general, dar si in cazul nostru, este folosita pentru a accelera căutările în tabele (de exemplu in bazele de date mari sau comparările de date).

De asemenea, parcurgerea/cautarea in adancime (*DFS*) a pozitiilor arcașilor a contribuit la specificarea datelor de intrare.

5.3 Specificarea ieșirilor/rezultatelor

In **main.py** am importat modulul **timeit** si am folosit funcția **timeit.default_timer()** pentru a "masura" timpul de executie la fiecare rulare. Practic, biblioteca *Python* (modulul importat) ruleaza codul de foarte multe ori si ofera timpul minim preluat din setul dat de fragment de cod. Aceasta metoda este una utilă,



Figure 1: Graficele timpilor de executie

deoarece ajută la verificarea performanței codului.

Timpul de executie	
0.001670	0.003199
0.002155	0.001857
0.001358	0.003039
0.021012	0.473185

Acestui *timeit.default_timer()* i-am setat valoarea de start si de end, iar la final am afisat diferenta intre acesti doi parametri, adica *end - start* care va rezulta timpul de executie efectuat la fiecare rulare a codului.

Am atasat mai sus doua exemple de grafice, continand, pe rand, cate 6, respectiv 12 timpi de executie. In aceste grafice se pot observa cresterile si scaderile timpilor de executie, acestia calculati dupa cateva rulari ale codului.

Despre datele de iesire mai pot spune faptul ca in cod, spre finalul acestuia, am afisat intreg tabelul cu arcasii si zidurile. Pana sa fac acest lucru, a trebuit sa verific daca arcasul respectiv a putut fi pus in pozitia respectiva si sa pun pereti daca pe aceeasi linie/coloana/diagonala se aflau mai multi arcasii.

Acolo unde se pot pozitiona arcasii, in tabelul nostru va aparea litera "A", iar acolo unde vor fi ziduri/pereti, va aparea litera "W"

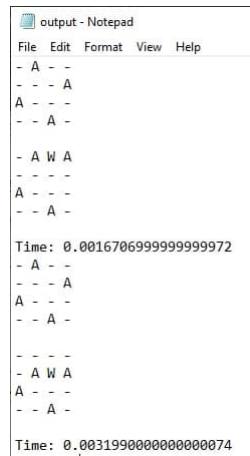
(acestia pusi in functie de pozitia arcasilor - unde se suprapun arcasi, acolo va fi zid intre ei).

Deci, in final, pentru datele de iesire vom avea:

- `table[i][pos[i]] = 'A'` - pentru afisarea fiecarui arcas
- `table[wall_l][wall_c] = '-'` - pentru afisarea initiala a peretilor; sunt 3 cazuri/conditii atunci cand trebuie sa punem pereti in locul arcasilor care se suprapun:
 - atunci cand **n** arcasi sunt plasati pe aceeasi linie, intre ei va fi pus un zid, astfel incat acestia sa nu traga unul in celalalt
 - atunci cand **n** arcasi sunt plasati pe aceeasi coloana, intre ei va fi pus un perete, astfel incat sa nu se impuse reciproc
 - atunci cand **n** arcasi sunt plasati pe aceeasi diagonala, se va introduce un zid intre ei, astfel incat acestia sa nu traga reciproc unul in celalalt

In cazul de fata, dupa ce se va trece prin fiecare conditie (atunci cand se afla mai multi arcasi pe aceeasi linie/coloana/diagonala), acesti pereti care vor fi pusi intre ei (pentru a nu se impusca intre ei arcasi) vor primi la iesire valoarea de "**W**".

Atunci cand vom rula codul, va aparea, pe rand, la fiecare rulare, tabelul cu arcasi si peretii, precum si timpul de executie. (*Figura 2*)



```
output - Notepad
File Edit Format View Help
- A -
- - A
A - -
- - A -

- A W A
- - -
A - -
- - A -

Time: 0.0016706999999999972
- A -
- - A
A - -
- - A -

- - -
- A W A
A - -
- - A -

Time: 0.0031990000000000074
```

Figure 2: Fisierul de output, dupa rulare

5.4 Lista tuturor modulelor aplicației și descrierea lor

Aici voi realiza o mica prezentare a tuturor modulelor aplicatiei si o descriere a acestora, cu ce scop au fost folosite, de ce si la ce ne ajuta. Am precizat la subparagraful 5.1 care este structura de nivel inalt a aplicatiei, urmand acum sa specific lista cu toate modulele cat si descrierea acestora.

Am ales ca pentru aceasta problema sa folosesc doar doua surse/module si anume cele enumerate mai jos:

- ***main.py*** - in acest modul sunt apelate functiile necesare ce se afla în *functions.py*; aici se calculeaza si timpul de executie a algoritmului, dupa fiecare rulare.
- ***functions.py*** - in acest modul se află multiplele funcții ale programului ce ajută la rezolvarea problemei

5.5 Lista tuturor funcțiilor aplicației, grupate pe module

5.5.1 Descrierea scopului funcției

- ***main.py*** :
 - * *timeit.default.timer()* - calculeaza timpul de executie al programului
- ***functions.py*** :
 - * *hash(name)* - am utilizat aceasta functie pentru a accelera cautarea in matrice
 - * *printTable()* - afiseaza tabelul in fisierul de iesire
 - * *searchDFS(index)* - cautarea in adancime pentru pozitiile arcasilor
 - * *check(index)* - verificarea fiecarei pozitii ale arcasului
 - * *threatens(line_i, col_i, line_j, col_j)* - aici se verifica daca se afla mai multi arcasi pe aceeasi linie/coloana/diagonala
 - * *randomWall()* - functie care returneaza un perete/zid, aleatoriu
 - * *printNoWalls()* - afisarea tabelului cu arcasii, fara pereti
 - * *printWithWalls()* - afisarea tabelului cu arcasii si peretii; aici verificam daca arcasul respectiv poate fi pus in acea pozitie si apoi punem pereti acolo unde sunt arcasi pe aceeasi linie/coloana/diagonala

5.5.2 Descrierea fiecarui parametru

- * *N* - dimensiunea tabelului NxN
- * *table* - tabelul gol
- * *pos* - pozitiile arcasilor
- * *ok/ok1* - contor
- * *index* - numarul arcasului respectiv
- * *lin_i* & *lin_j* - liniile tabelului in pozitiile i si j
- * *col_i* & *col_j* - coloanele tabelului in pozitiile i si j
- * *wall_c* & *wall_l* - perete de pe coloana/linie

5.5.3 Semnificația valorii de return

- * *return wall_l, wall_c* - returneaza zidurile pe linie si coloana, in mod aleatoriu

6 Concluzii

★ Observand inca de la inceput ca aceasta problema seamana intr-un oarecare fel cu problema reginelor, am pornit de la acest fapt si cu gandirea asupra modului de rezolvare.

★ A fost pentru prima data cand am folosit algoritmul de calculare a timpilor de executie, si mi s-a parut foarte interesant acest lucru: cum sunt de fiecare data acesti timpi diferiti.

★ De asemenea, un alt lucru interesant mi s-a parut algoritmul pe care l-am folosit pentru a afla pozitiile arcasilor, si anume cautarea/parcurgerea in adancime, *DFS*. Acesta are o complexitate, la fel ca si parcurgerea in latime, *BFS*, $O(|E|+|V|)$ (numarul de muchii plus numarul de noduri), care, in cazul cel mai nefavorabil, cand graful/tabelul este liniarizat, vor fi parcurse si explorate toate muchiile si toate nodurile.

★ Am ales sa rezolv aceasta problema (denumita de mine problema arcasilor) in limbajul de programare **Python**, deoarece, avand si anul trecut tema de casa in acest limbaj, mi s-a parut din nou un "challenge" si am dorit sa invat si mai multe despre el. Acest limbaj mi se pare intr-o oarecare masura mai "usor", prin simplul fapt ca este un limbaj de programare interpretat, mai exact ceea ce imi "surade" mie este ca nu mai trebuie sa implementam unele metode(*random*, *timeit*), ci doar le importam, iar **Python** isi face singur treaba; de asemenea, numele variabilelor nu are semnificatie sau concept pentru un tip de date, acestea fiind de fapt referiri la obiecte. Majoritatea limbajelor compilate/moderne necesita declararea unei variabile, inainte ca aceasta sa poata fi utilizata, iar cum **Python** este un limbaj interpretat, declararea variabilelor este optionala.

★ Pe viitor, voi incerca sa imi imbunatatesc experienta si performantele in acest limbaj de programare, **Python**, dar si in limbajele de programare actuale cunoscute, anume **C**, **C++**, **Java** si **C#** care, de asemenea, sunt foarte diferite intre ele, ceea ce le face foarte interesante, cel putin din perspectiva mea; ceea ce ma face sa fiu si mai curioasa in ceea ce le priveste.

7 Referinte bibliografice

Bibliografie

- [1] L^AT_EX project site
 - https://www.overleaf.com/learn/how-to/Creating_a_document_in_Overleaf
 - <https://www.caam.rice.edu/~heinken/latex/symbols.pdf>
- [2] Python, <https://www.w3schools.com/python/default.asp>
- [3] DFS, https://en.wikipedia.org/wiki/Depth-first_search
- [4] Generarea random a numerelor
 - <https://docs.python.org/3/library/random.html>
 - <https://www.geeksforgeeks.org/python-randint-function/>
- [5] Timpul de executie, <https://docs.python.org/3/library/timeit.html>
- [6] Problema Reginelor - N Queen Problem, <https://www.geeksforgeeks.org/n-queen-problem-backtracking-3/>
- [7] Artificial Intelligence: A Modern Approach, <http://aima.cs.berkeley.edu/>