Universitatea din Craiova
Facultatea de Automatică, Calculatoare şi Electronică

20 Mai 2020

Student : Nedianu Gabriel-Cătălin
Specializarea: Calculatoare Română
Anul I
Grupa: CR 1.2 B

Cuprins

1 Enunţul Problemei

Tortul Împăratului Roşu

Împăratul Roşu a comandat pentru nunta fiului său un tort împărătesc cu n straturi circulare. Bucătarii împăratului au pregătit tortul pe un platou de argint. Însă, împăratul doreşte ca tortul să fie mutat pe un platou de aur. Din cauza dimensiunii enorme a tortului, operaţia de transfer trebuie realizată cu mare atenţie, pentru a evita distrugerea accidentală a tortului; aşa că bucătarii nu ştiu cum să mute tortul de pe platoul de argint pe platoul de aur. Ei şi-au dat însa seama că, pentru mutarea tortului, pot folosi un al treilea platou de bronz. La fiecare pas este cel mai sigur să transfere un singur strat al tortului de pe un platou pe altul şi întotdeauna un strat al tortului va fi pus doar peste un strat de diametru mai mare, astfel încat să se evite deteriorarea tortului. Scrieţi un program care să ajute bucătarii să transfere tortul fără să îl strice, astfel încât tortul transferat să fie identic cu tortul iniţial. Se vor implementa doi algoritmi diferiţi.

2 Algoritmii

2.1 Algoritmul 1 (recursiv)

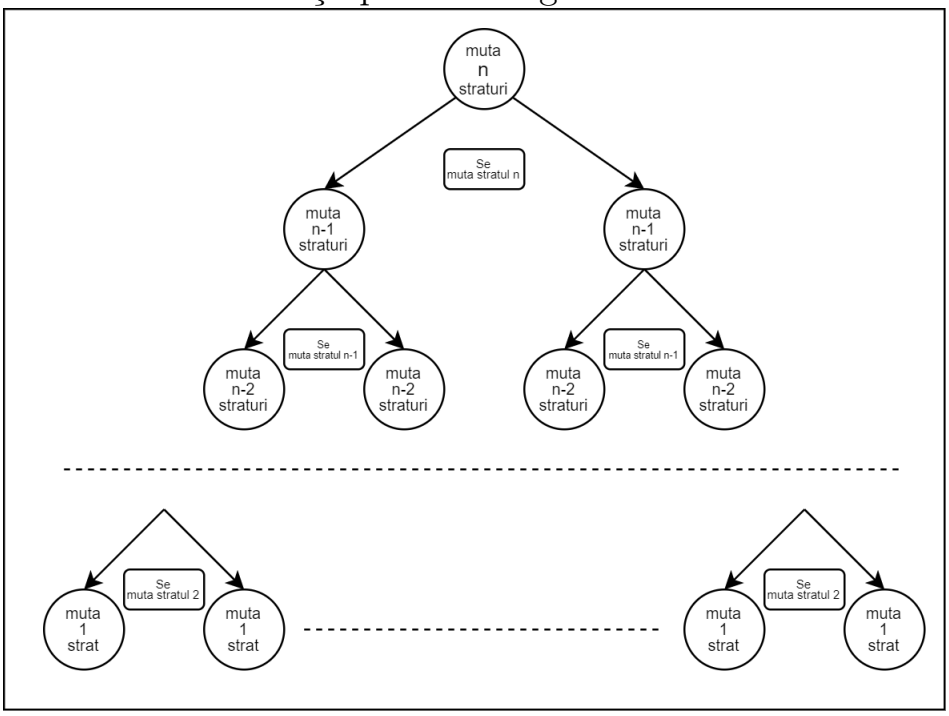
MUTA_TORTUL($numar\_felii, platou\_initial, platou\_final, platou\_ajutator$)
1.**daca** ($numar\_felii == 1$)
2. **afiseaza** Se muta stratul 1 al tortului de pe $platou\_initial$ pe $platou\_final$
3.**altfel**
4. MUTA_TORTUL($numar\_felii - 1, platou\_initial, platou\_ajutator, platou\_final$)
5. **afiseaza** Se muta stratul 1 al tortului de pe $platou\_initial$ pe $platou\_ajutator$
6. MUTA_TORTUL($numar\_felii - 1, platou\_ajutator, platou\_final, platou\_initial$)

Complexitatea de timp a algoritmului este O(2^n), deoarece fiecare funcţie se va reapela de alte două ori pana când se va ajunge ca numar_felii primit de funcţie să fie 1. În total vor fi: $2^1+2^2+...+2^{n-1}$ reapelări + apelarea iniţiala:2^0.

Complexitatea de memorie a algoritmului este O(n), deoarece recursivitatea nu va fi mai mare pe o ramură de n ori.

Algoritmul funcţionează astfel: se consideră că tortul format din n straturi va fi aşezat iniţial pe primul platou (cel de argint) şi se doreşte ca acesta să fie mutat pe platoul final (cel de aur) cu ajutorul platoului ajutător (cel de bronz). Considerăm că toate straturile tortului, fără ultimul, se vor muta pe platoul ajutător, apoi ultimul strat va fi mutat pe platoul final; la final, straturile aflate pe platoul ajutător se vor muta pe ultimul platoul peste ultimul strat.

Schema recursivităţii primului algoritm



2.2 Algoritmul 2 (iterativ)

Muta_Tortul_Iterativ($nr\_straturi$,$*platou\_initial$,$*platou\_final$,$*platou\_ajutator$)
1.**daca** ($nr\_straturi\%2 == 0$)
2. interschimba($platou\_final - nume, platou\_ajutator - nume$)
3.creeaza_tortul($*platou\_intial, nr\_straturi$)
4.$nr\_mutarii = 0$
4.**cat timp** (este_plin($*platou\_final$) ! = 1 && este_plin($*platou\_ajutator$) ! = 1)
5. $nr\_mutarii++$
6. **daca** ($numarul\_mutarii\%3 == 1$)
7. muta_strat_intre($*platou\_initial$,$*platou\_final$)
8. **daca** ($numarul\_mutarii\%3 == 2$)
7. muta_strat_intre($*platou\_initial$,$*platou\_ajutator$)
8. **altfel**
9. muta_strat_intre($*platou\_ajutator$,$*platou\_final$)

Pentru acest algoritm am folosit o structură, *platou, care va fi folosită ca o stivă pentru a stoca straturile aflate pe fiecare platou, în ordine.

Complexitatea de timp a acestui algoritm depinde de numărul de mutări efectuate, experimentând aceasta ajunge să fie $O(2^n)$.

Complexitatea de memorie este O(n), deoarece se vor utiliza doar trei vectori de capacitate maximă n pentru a stoca straturile şi se va lucra doar cu aceştia.

Algoritmul funcţionează astfel (pentru torturile cu numar de straturi impar): dacă se mută consecutiv câte un strat între două platouri, se va ajunge ca tortul să fie mutat complet pe un alt platou. Ordinea de mutare a straturilor este următoarea: se mută între primul şi ultimul platou stratul posibil, se mută între primul şi al doilea platou stratul posibil, apoi se mută între cel de-al doilea şi cel de-al treilea platou stratul posibil. Aceste mutări se vor repeta până când tortul va fi mutat complet pe al treilea platou (cel de aur).

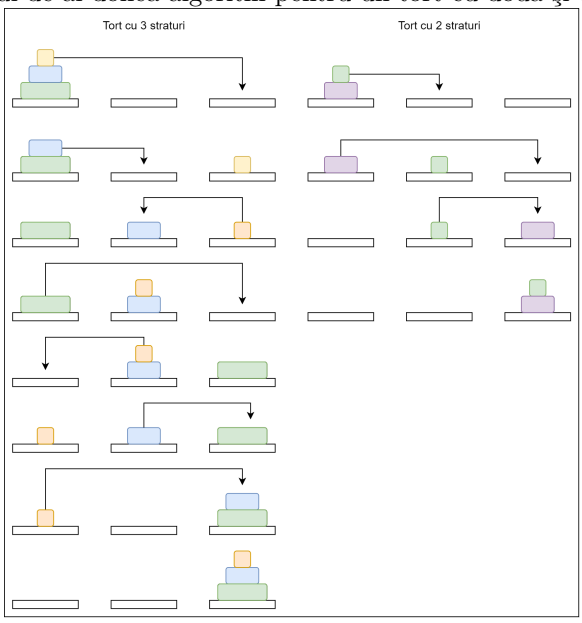
În cazul torturilor cu număr de straturi par, principiul este acelaşi, singura diferenţă fiind că se schimbă puţin ordinea mutărilor: prima mutare va fi între primul platou şi cel de-al doilea, a doua mutare între primul platou şi ultimul platou, iar cea de-a treia între cel de-al doilea şi cel de-al treilea platou. Astfel, va trebui să fie inversat platoul final cu cel ajutător ca să ramana mutările de la cazul cu număr de straturi impare (lucru ce se poate face schimbând doar denumirile acestora, având în vedere că stivele sunt goale).

Deci, se vor face serii de câte trei mutări consecutive, aceste mutări acoperind toate mişarile posibile între cele trei platouri.

Pentru a verifica corectitudinea numărului de paşi, am introdus variabila numarul_mutarii, care va arăta la final căţi paşi vor fi necesari pentru mutarea tortului.

Iteraţia va funcţiona până când tortul va fi mutat pe platoul final necesar, şi astfel se va afla şi numarul mutărilor necesare.

Schema celui de-al doilea algoritm pentru un tort cu două şi trei straturi



3 Date Experimentale

Ca date de intrare, problema are nevoie de un singur parametru, acela fiind numărul de straturi pe care îl va avea tortul, care urmează a fi mutat de pe platoul de argint pe cel de aur.

În prima etapă a creării programului, datele de intrare aparţineau intervalului $(1, 23)$, iar pentru salvarea paşilor mutării tortului în fişier, se ocupa un spaţiu mai mare de 500 Mb (pentru 23 de straturi) şi executarea programului durează peste 50 de secunde cu oricare dintre algoritmi.

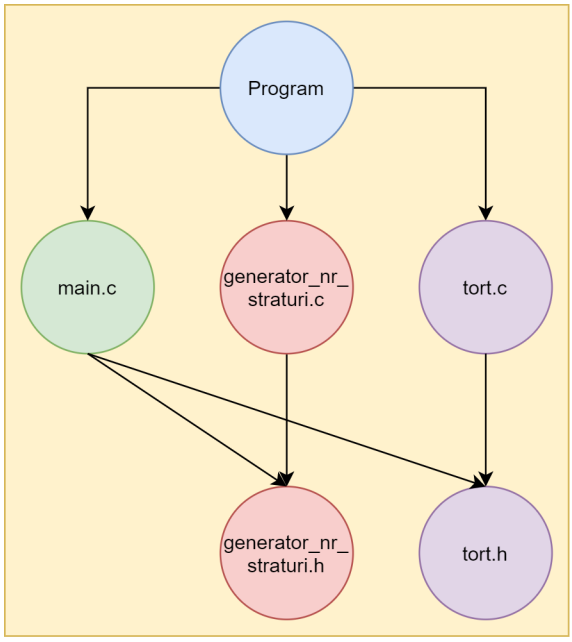
După ce am limitat numarul de paşi scrişi la 1000, am putut mări intervalul pana la 31, în acest caz execuţia durând maxim 50-60 secunde, în cel mai rău caz (31 de straturi), iar fişierul de output are doar 60 Kb.

Am creat un generator simplu care va genera un număr din intervalul $(1, 31)$ folosind, în limbajul C funcţia rand(), iar în limbajul Python o funcţie asemănătoare, randitnd().

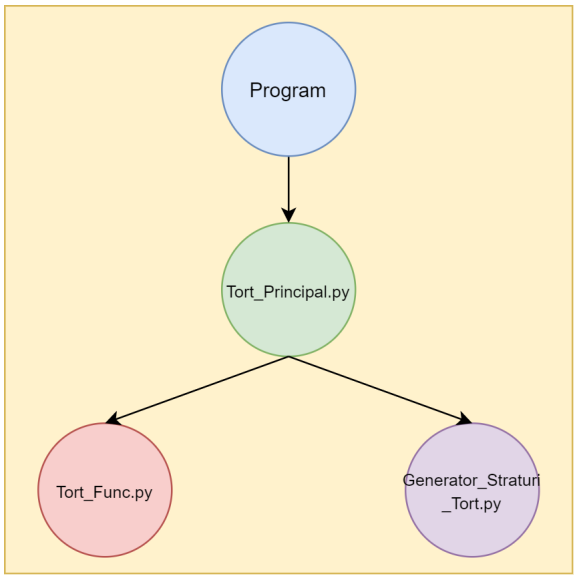
4 Proiectarea aplicaţiei experimentale

4.1 Structura de nivel înalt a aplicaţiei

Aplicaţia conţine următoarele fişiere în limbajul C: main.c, tort.c, torth.h, generator_nr_straturi.c, generator_nr_straturi.h conectate astfel:



Aplicaţia conţine următoarele fişiere în limbajul Python: Tort_Principal.py, Tort_Func.py si Generator_Straturi_Tort.py conectate astfel:



4.2 Descrierea Mulţimii Datelor de intrare

Singura dată de intrare pentru algoritm va fi un număr, care va simboliza numărul de straturi ale tortului ce urmează să fie mutat.

4.3 Descrierea Mulţimii Datelor de ieşire

Datele de ieşire ale problemei vor fi stocate în mod implicit într-un fisier .txt numit ajutor_bucatari (în timpul testelor, aceste date vor fi stocate în fişiere test_i, i fiind cuprinse între 1 şi 10).

Fişierul de ieşire va cuprinde pe primul rând un mesaj iniţial, care va arăta câte straturi are tortul, pe următoarele rânduri paşii ce trebuie făcuţi pentru a muta tortul de pe platoul iniţial pe cel final (paşii care vor fi afisaţi sunt limitaţi la 1000), pe penultimul rând va fi un mesaj ce va arăta numărul de mutări necesare pentru a muta tortul, iar pe ultimul rând va fi afişat timpul de execuţie (în secunde).

4.4 Lista modulelor aplicaţiei

In C , modulele sunt:

- main.c -modulul principal
- tort.c -modulul ce cuprinde majoritatea corpurilor funcţiilor utilizate
- generator_nr_straturi.c -modulul în care se află corpul funcţiei, care va genera automat numărul de straturi
- tort.h - modulul ce conţine headerele funcţiilor din tort.c
- generator_nr_straturi.h modulul ce conţine headerele funcţiei de generare

In Python , modulele sunt:

- Tort_Principal.py -modulul principal
- Tort_Func.py -modulul în care sunt scrise majoritatea funcţiilor
- Generator_Straturi_Tort.py -modulul care conţine generatorul

4.5 Funcţiile aplicaţiei

4.5.1 main.c

Pentru ambii algoritmi, acest modul conţine doar deschiderea fişierelor unde se vor salva datele, apeluri de funcţii, iniţializări de structuri necesare/variabile şi metoda prin care este cronometrat timpul de execuţie al algoritmului.

Pentru algoritmul recursiv, în main.c sunt apelate funcţiile genereaza_straturi_tort() şi muta_tortul().

Pentru algoritmul iterativ în main.c sunt apelate funcţiile genereaza_straturi_tort(), creeaza_platou() şi muta_tortul().

Pentru ambii algoritmi poate fi apelată şi funcţia test(), ce a fost folosită pentru a efectua testele.

Testarea timpului a fost efectuată utilizându-se librăria ¡time.h¿ şi funcţia clock().

4.5.2 generator_nr_straturi.h si tort.h

Aici se află prototipurile funcţiilor folosite şi o variabilă globală nr_straturi_tort.
Pentru algoritmul iterativ, tot aici (tort.h) se va găsi declaraţia structurii Platou, ce va fi o stivă în care se vor stoca straturile tortului aflate pe un anumit platou, dar şi numele platoului.
Pentru algoritmul recursiv, aici se va găsi o variabilă globală, în care se va salva numărul de mutări necesare.

4.5.3 generator_nr_straturi.c

În acest modul se găseşte corpul funcţiei ce generează aleatoriu un număr de straturi al torului din intervalul (1,31): genereaza_straturi_tort(). Această funcţie nu returnează nimic, deoarece schimbă automat valoarea variabilei globale nr_straturi. Sunt folosite funcţiile srand() şi rand().

4.5.4 tort.c

Aici se găsesc corpurile majorităţii funcţiilor folosite în programe.

Pentru algoritmul recursiv găsim doar funcţia muta_tortul(). Această funcţie primeşte ca parametrii: numărul de straturi ale tortului, numele celor trei platouri şi pointerul la fişierul în care se vor salva informaţiile. Aceasta este funcţia principală care creează recursivitatea prin care se mută
straturile tortului, recursivitate explicată în **Secţiunea 2.1** Funcţia nu returnează nimic, aceasta scrie în fişierul ajutor bucătari paşii/mutările necesare care vor fi limitate de variabila definită ca limita_afisari = 1000; dupa ce se trece de această limită, recursiunea se va realiza fără afişare.

Tot în acest modul, în ambele file ale ambilor algoritmi, vom folosi funcţia test(), ce va ajuta la efectuarea testelor pentru nişte valori prestabilite, prin care se va testa funcţionalitatea programului, timpul de execuţie şi numărul de paşi ce vor fi efectuaţi pentru fiecare test în parte. Algoritmul va fi testat pentru următoarele straturi de tort : 1, 3, 6, 14, 15, 20, 21, 27, 30, 31.

Pentru algoritmul iterativ, aici se găsesc mai multe funcţii folosite.
Majoritatea funcţiilor sunt folosite utilizând structura Platou creată
Structura Platou este o stivă ce va conţine numărul de straturi ale tortului în timpul mutării acestora. Aceasta are în componenţa sa nr_straturi_maxime, un vector de tip char ce va conţine numele platoului respectiv, variabila top ce va reprezenta câte straturi se află în acel moment pe platou (-1 reprezintă faptul că platoul este gol) şi un vector de tip int ce va stoca numărul straturilor aflate pe platou, în ordinea asezării lor.

Funcţia creeaza_platou() creează platoul propriu-zis.

Funcţia este_plin() primeşte ca parametru un Platou şi returnează dacă acesta este plin sau nu. Pentru a afla dacă platoul este plin, am verificat dacă straturile maxime admise de un platou sunt egale cu numărul de elemente aflate în el (deoarece numerotarea straturilor începe de la 0, am scăzut 1 din numărul de straturi maxime).

Funcţia nu_sunt_straturi() funcţionează asemănător cu funcţia anterioară, diferenţa fiind că aceasta verifică dacă nu sunt straturi ale tortului pe platou.

Funcţia adauga() primeşte ca parametrii un Platou şi o variabila de tip int ce reprezinta numărul stratului ce va fi adăugat pe platoul respectiv. Valoarea top a platoului este crescută cu 1.

Funcţia scoate_stratul() primeşte un Platou, returnează ultimul strat aflat pe acesta şi este scăzută valoarea top a platoului cu -1. Dacă nu sunt straturi pe acel platou, este returnată valoarea "-2".

Funcţia muta_strat_salvare() primeşte numărul stratului ce se va muta, numele platourilor între care se va face mutarea şi un pointer la fişierul în care va fi salvată această mutare.

Funcţia creaza_tortul() primeşte un Platou şi adaugă pe acesta straturile tortului de la cel mai mare la cel mai mic. Astfel, ultimul strat adăugat va fi scos primul. Această funcţie va fi folosită pentru a pune tortul pe platoul iniţial.

Functia muta_straturile_intre_2_platouri() primeşte două Platouri şi pointer la fişierul de output; această funcţie va muta ultimul strat mai mic dintre cele două platouri peste stratul mai mare.
Această funcţie scoate ultimul strat din fiecare platou şi compară aceste straturi, existând patru cazuri posibile. Primul caz este acela în care primul platou este gol; astfel, se va muta stratul de sus al celui de-al doilea platou peste acesta şi se va salva mutarea. Al doilea caz este cel în care cel de-al doilea platou are stratul superior gol. În ultimele două cazuri se compară straturile superioare ale celor două platouri, stratul mai mare se adaugă la loc pe platoul de unde provine, iar celălalt se asază peste el şi apoi se savlează mutarea in fişier.

Funcţia muta_straturile_intre_2_platouri_fara_afisare() funcţioneaza identic cu funcţia anterioară, doar că nu se mai salvează (sau afişează) mutările .

Funcţia muta_tortul() primeşte ca parametrii numărul de straturi ale tortului, cele trei structuri reprezentând fiecare platou şi fisierul de output.
Prima parte a funcţiei va schimba denumirea platoului ajutător cu cea al celui final, dacă avem un număr par de straturi.
În următoarea parte a funcţiei se va crea tortul pe primul platou (adaugând straturile de la cel mai mare la cel mai mic pe acesta).
În ultima parte a funcţiei va începe iteraţia prin care se mută strat cu strat tortul, dupa tiparul definit în **Secţiunea 2.2**. Avem o instrucţiune while, în care se vor număra mutările efectuate până când tortul va fi complet mutat.

4.5.5 Tort_Principal.py

Acesta este modulul principal al funcţiei în Phyton, pentru ambii algoritmi; aici doar vom apela funcţii şi vom iniţializa variabilele pentru testarea timpului.

Variabilele şi funcţiile au denumiri aproximativ asemănătoare cu cele din C, iar funcţionalitatea lor este aceeaşi.

Tot aici va fi apelată o funcţie nouă, afisare_nr_pasi(), care va salva în fişier numărul de paşi efectuaţi de bucătari pentru a muta tortul, această funcţie primind doar pointerul la fişier.

În acest modul vom avea şi modalitatea de testare (ce a fost comentată pentru a permite rularea o singură dată a programului, fără a rula şi testul).

4.5.6 Generator_Straturi_Tort.py

În acest modul avem funcţia care generează straturile tortului pentru rulări la întâmplare, fiind creată folosind funcţia random redenumită r, ce ne va returna un numar din intervalul (1, 31).
Aici am creat şi variabila nr_straturi_tort în care, la fiecare rulare, se va salva alt număr format de generator.

4.5.7 Tort_Func.py

În acest modul vom găsi funcţiile folosite pentru ambii algoritmi din ambele programe în Phyton, dar şi variabila limita_afisari ce va limita numărul de afişări în fiecare program la 1000.

Pentru algoritmul recursiv, avem functîile:

Funcţia muta_tortul(), funcţia principală, care va fi folosită pentru a salva în fişier mutările ce vor fi efectuate de bucătari (am folosit funcţia globals() pentru a salva printr-o iteraţie şi numărul de paşi ce va fi afişat la final şi va fi folosit ca o verificare), această funcţie primind numărul de straturi, denumirile celor trei platouri şi fişierul de output. Modul de funcţionare al algoritmului a fost explicat anterior.

Funcţia afisare_numar_pasi() funcţie ce primeşte fişierul de output şi salvează în el penultima linie, conţinând un mesaj cu numărul de paşi, iar apoi, după afişare, resetează numărul de paşi la 0, pentru a putea fi folosită pe parcursul testelor.

Aici mai avem şi variabila numarul_mutarii ce va fi iniţializată cu 0 la începerea fiecărui test/ rulari.

Pentru algoritmul iterativ avem mai multe funcţii; aceste funcţii au fost implementate pe un principiu asemănator cu cel din C, doar că în loc de structura creată, am folosit pe post de stive listele deja implementate în Phyton şi comenzile programului , precum pop(), push(). Astfel, am avut mai puţine funcţii de implementat.

Funcţia scoate_ultimul_strat() se foloseşte de comanda .pop() pentru a scoate un strat de pe platou şi a-l salva într-o variabilă, şi de funcţia len() pentru a verifica dacă un platou este gol sau nu.

Funcţia muta_discurile_intre_2_platouri() funcţioneaza la fel ca funcţia din C. Idem şi funcţia muta_discurile_intre_2_platouri_fara_salvare().

Funcţia muta_tortul are funcţionalitatea identică cu funcţia în C, singura diferenţă fiind că numele platoului este separat de listă (folosită ca o stivă) cu straturi, iar denumirile platourilor sunt iniţializate în interiorul funcţiei.

Anumite părţi din cod sunt comentate, ele reprezentând functionalităţi suplimentare ce pot fi folosite. (De exemplu: codul pentru teste este comentat, deoarece acestea au fost deja o dată folosite etc.)

5 Rezultate

Aici voi arăta rezultatele testelor şi anumite observaţii pe baza implementărilor şi ale executării codului în timpul testelor.

O prima observaţie ar fi că timpul în phyton este mai mare decât cel în C, acest lucru explicându-se prin faptul că limbajul Phyton este interpretat, ceea ce îi sporeşte timpul de execuţie faţă de un limbaj compilat. În acelaşi timp, implenetarea în phyton este mult mai uşoara decât cea în C.

5.1 Organizarea datelor de iesire

Pentru o simplă rulare, rezultatele sunt afişate în fisierul ajutor_bucatari.txt; pentru vizualizarea testelor, aceste rezultate sunt afişate în fişierele de tip test%i.txt, i$\in \{1, 10\}$.

Toate fişierele au acelaşi format, pe care îl voi mai explica o dată aici, şi voi explica şi de ce am ales să fie salvate astfel.

Prima linie dintr-un fişier de output arată pentru câte straturi ale tortului a fost executat programul.

Următoarele maxim 1000 de linii conţin primele mutări, în ordine.

Ultimele două linii sunt cele mai importante şi vor fi folosite pentru a prezenta rezultatele în ansamblu.

Penultima linie conţine numărul de mutări necesare pentru a muta tortul, iar ultima linie conţine timpul de execuţie.

De asemenea, după ce se va termina de editat fişierul output (denumit sugestiv fisier_ajutor_bucatari pentru o simplă rulare sau test pentru rulările de tip test) se va afişa în consolă un mesaj în care se va menţiona pentru câte straturi a fost efectuat algoritmul.

Atunci când se vor rula testele, vor fi afişate următoarele mesaje sugestive:

```
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 1 straturi in Teste_Efectuate\test1.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 3 straturi in Teste_Efectuate\test2.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 6 straturi in Teste_Efectuate\test3.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 14 straturi in Teste_Efectuate\test4.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 15 straturi in Teste_Efectuate\test5.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 20 straturi in Teste_Efectuate\test6.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 21 straturi in Teste_Efectuate\test7.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 27 straturi in Teste_Efectuate\test8.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 30 straturi in Teste_Efectuate\test9.txt.
S-au generat miscarile care vor ajuta bucatarii sa mute tortul de 31 straturi in Teste_Efectuate\test10.txt.

Process returned 0 (0x0)   execution time : 78.703 s
```

Aceste mesaje vor fi afişate pentru oricare dintre cele patru programe.

5.2 Testarea datelor de ieşire

Pentru a testa dacă algoritmul salvează mutări corecte în fişierele de output am folosit mai multe metode:

- Pentru valorile mici ale numărului de straturi am verificat manual mutările şi toate s-au dovedit a fi corecte, iar numărul de mutări salvat de program era cel corect (corespunzând cu numărul de mutări citite manual).
- Pentru valori mai mari, doar am dat scroll prin fişier pentru a vedea că nu există erori ale mutărilor (exemplu: nu se muta un strat ce nu aparţinea intervalului de straturi posibile, acest interval fiind (1,nr_straturi_tort), dupa care am verificat numărul de mutari salvate şi acesta a fost întotdeauna $2^{nr\_straturi\_tort} - 1$, ceea ce înseamna că a fost calculat corect.

5.3 Timpii de execuţie pentru date mici si mari

Am organizat compararea datelor timpilor de execuţie în două tabele: primul tabel compară cele două programe recursive, iar al doilea pe cele două iterative.

Tabel Comparare Algoritmi Recursivi

Nr.Test	Numarul de Straturi	C	Python
1	1	0.000000	0.0
2	3	0.000000	0.0
3	6	0.000000	0.0
4	14	0.000000	0.0069799423217
5	15	0.000000	0.0119671821594
6	20	0.000000	0.3540530204772
7	21	0.000000	0.7011260986328
8	27	0.000000	45.6151285171508
9	30	2.000000	351.2672555446625
10	31	4.000000	749.6825110912323

Tabel Comparare Algoritmi Iterativi

Nr.Test	Numarul de Straturi	C	Python
1	1	0.000000	0.0
2	3	0.000000	0.0
3	6	0.000000	0.0
4	14	0.000000	0.019946813583
5	15	0.000000	0.039893150329
6	20	0.000000	1.222729682922
7	21	0.000000	2.431497335433
8	27	3.000000	153.349867105484
9	30	25.000000	1208.827067375183
10	31	51.000000	2403.487896680832

5.4 Observaţii si Concluzii

Observăm că algoritmii în C sunt cu mult mai rapizi decat cei în Python, dar observăm şi că, pentru amblele limbaje, cu cât numarul de straturi devine mai mare, cu atât timpul de execuţie creşte.

De asemenea, am observat că algoritmul iterativ este mult mai lent decât cel recursiv; acest lucru se poate explica prin faptul că, pentru algoritmul iterativ am utilizat structuri/liste pentru a crea stive.

În oricare dintre cele patru programe, pentru un set de date de intrare identic, se va obţine un set de date de ieşire identic (paşii pentru mutarea tortului), iar numărul de mutări este demonstrat că va fi $2^n - 1$, unde n reprezintă numărul de straturi ale tortului.

O parte interesantă a implementării algoritmilor a fost limitarea salvărilor în fişier; pe lângă faptul că fişierul de output căpăta dimensiuni enorme, execuţia creştea mult datorită acestor salvări. Am ales să limitez la 1000 de mutări afişate algoritmul.

O altă parte provocatoare a fost trecerea de la un limbaj compilat la unul interpretat; am studiat, astfel, multe despre limbajul Phyton şi până să ajung la versiunile finale ale codului, am avut multe alte programe "test"efectuate în care am aprofundat utilizarea lui.

Pe viitor, acest program poate fi dezvoltat, iar în loc să afişeze sau salveze mutările efectuate, ar putea să le mute grafic într-un program de editare 3D, dar acesta este un proiect de viitor, greu de realizat cu ajutorul cunoştiinţelor actuale.

Referinţe

[1] Thomas H. Cormen and Charles E. Leiserson and Ronald L. Rivest and Clifford Stein, *Introduction to Algorithms*. MIT Press, 3rd Edition, 2009.

[2] Dennis Ritchie and Brian Kernighan, *The C Programming Language. 2nd Edition*, Prentice Hall, 1978.

[3] `https://www.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi`

[4] `https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`

[5] `https://www.latex-project.org/`

[6] `https://www.reddit.com/r/Python/`