Problema numerelor Lychrel

Introducere

Un **palindrom numeric** este un număr care rămâne același atunci când cifrele sale sunt citite în ordine inversă. De exemplu, 121 sau 5445 sunt palindromuri în baza 10 (citindu-le de la stânga la dreapta sau invers se obține același șir de cifre). Multe numere naturale devin palindromuri dacă li se aplică repetat o procedură simplă: se ia numărul, i se inversează cifrele, iar rezultatul obținut se adună cu numărul inițial. Procesul se repetă iterativ până când suma devine palindromică. Această metodă este cunoscută drept *algoritmul 196* (după cel mai faimos caz al său) sau procedeul „reverse-and-add". Formal, pentru un număr natural (n) în baza (b), definim funcția (F_b(n)) astfel:

[F_b(n) = n + \text{reverse}_b(n),]

unde (\text{reverse}*b(n)) denotă numărul obținut prin inversarea cifrelor lui (n) în baza (b). De exemplu, în baza 10, (F*(125) = 125 + 521 = 646). Iterând această funcție, un număr poate genera o }(56) = 56 + 65 = 121), iar (F_{10**secvență palindromică**: (n, F_b(n), F_b(F_b(n)), \dots). Dacă la un moment dat se obține un palindrom, procesul se oprește cu succes. Marea majoritate a numerelor devin palindromice foarte rapid prin acest algoritm: toate numerele de o cifră sau două cifre devin palindromuri (cele de două cifre necesită cel mult o iterație: ex. 56 121, 57 132 363 etc.), iar aproximativ 80% din numerele sub 10.000 ajung la un palindrom în cel mult 4 pași, ~90% în 7 pași .

Un **număr Lychrel** (pronunțat *Lĭch-rəl*) este un număr natural care **nu poate** forma niciodată un palindrom prin aplicarea repetată a acestei proceduri de inversare și adunare . Cu alte cuvinte, dacă presupunem că am putea repeta la infinit acest algoritm fără a obține vreodată un palindrom, numărul inițial este denumit *candidat Lychrel*. Până în prezent **nu se cunoaște cu certitudine existența unui număr Lychrel veritabil în baza 10** – niciun candidat nu a fost demonstrat matematic ca fiind incapabil să producă un palindrom. Totuși, mai multe numere (cele mai mici fiind 196, 295, 394, ...) au fost testate pe cale computațională în milioane de iterații fără succes, fiind suspectate a fi Lychrel . Termenul „Lychrel" a fost introdus în 2002 de către Wade VanLandingham, ca un anagram parțial al prenumelui prietenei sale, Cheryl . Conceptul a atras interesul amatorilor și al matematicienilor din perspectiva unei **probleme nerezolvate**: *Există numere Lychrel în baza 10?* .

Pentru ilustrare, să considerăm câteva exemple de numere **ne-Lychrel** (adică *non*-Lychrel) care devin palindromuri după un număr finit de iterații. Numerele mici necesită adesea puțini pași: de pildă, 56 devine palindromic după o singură iterație (56 + 65 = 121), 57 după două iterații (57 132 363), iar
59 după trei iterații (59 154 605 1111). Există însă și numere care necesită neobișnuit de mulți pași: celebrul 89 are nevoie de **24 de iterații** pentru a produce palindromul 8813200023188 . Chiar și numere de ordinul zecilor de mii pot necesita zeci de pași: de exemplu, 10.911 generează palindromul 4668731596684224866951378664 (28 de cifre) abia după **55 de iterații** . Acestea sunt cazuri de **"palindrome întârziate"** (delayed palindromes), care deși nu sunt Lychrel (pentru că până la urmă devin palindromuri), necesită un număr record de pași. Vom reveni asupra acestor recorduri de întârziere în Secțiunea 6. Important de subliniat este că nici măcar astfel de cazuri extreme nu contrazic definiția Lychrel – un număr Lychrel **nu ar produce niciun palindrom**, oricât am repeta algoritmul.

În contrast cu exemplele de mai sus, unele numere par să reziste la formarea unui palindrom. **196** este cel mai mic astfel de candidat: aplicând asupra sa algoritmul "inversează și adună" în mod repetat, nu se cunoaște nicio iterație care să rezulte într-un palindrom [9]. Deși a fost testat de calculatoare pentru milioane de pași (vezi Secțiunea următoare), 196 nu a produs încă un palindrom – de aici provenind reputația sa. **Lista numerelor suspecte Lychrel** începe cu: 196, 295, 394, 493, 592, 689, 691, 788, 790, 879, 887, ... Toate aceste exemple *nu* au generat palindromuri în ciuda căutărilor extensive. Acestea sunt considerate "cazuri clasice" de candidați Lychrel și vor fi discutate mai pe larg în continuare.

În cele ce urmează vom prezenta istoricul eforturilor computaționale legate de problema 196 și alți candidați notabili, vom detalia implementarea algoritmilor realizați de noi (inclusiv prezentarea codului sursă și a optimizărilor folosite), vom

analiza datele rezultate (convergențe, întârzieri, palindroame comune), și vom oferi vizualizări matematice pentru a înțelege mai bine fenomenul. Vom discuta apoi despre recordurile cunoscute în privința celor mai multe iterații necesare (palindrome întârziate), despre modul în care traiectoriile numerelor se pot grupa în "threads", "seeds" și "kin" pentru eficientizarea analizei, precum și despre posibile abordări de tip *machine learning* pentru a distinge între numerele Lychrel și non-Lychrel. Lucrarea se încheie cu concluzii și direcții viitoare, incluzând explorarea altor baze, vizualizări inverse ale problemei și perspective teoretice.

Istoricul computațional al problemei 196 și al altor candidați notabili

Numărul 196 – cel mai mic candidat Lychrel – a captat atenția comunității matematice recreative încă din anii 1980. În 1987, **John Walker** a inițiat public *"196 Palindrome Quest"* (Căutarea palindromului pentru 196). Folosind un workstation Sun 3/260, Walker a scris un program în limbaj C care aplica iterativ algoritmul de inversare și adunare pe 196, verificând după fiecare iterație dacă s-a obținut un palindrom . Programul său a rulat în fundal, cu prioritate scăzută, timp de mai mulți ani. Până în anul 1990, Walker a raportat că secvența generată de 196 atinsese deja **2.415.836 de iterații** fără a găsi un palindrom, ultimul număr obținut având peste **1 milion de cifre** . Practic, 196 se transformase într-un număr uriaș (de un milion de cifre) care încă nu era palindrom. Walker și-a publicat rezultatele pe internet, oferind și cel mai recent "checkpoint" (valoarea numerică obținută după ultimul pas calculat), invitând astfel și alți entuziaști să continue de unde a rămas el .

Ulterior, în 1995, **Tim Irvin** a extins căutarea lui Walker folosind sisteme mai rapide, ducând numărul de cifre al rezultatului la **2 milioane** (corespunzător la ~3÷4 milioane de iterații estimate) . Deși aceste eforturi au împins mult mai departe frontiera calculului, nici până la 2 milioane de cifre (sau câteva milioane de iterații) nu a apărut vreun palindrom. Problema 196 a început astfel să fie privită ca un veritabil *challenge*, un *open problem* pentru care nu se întrevede o soluție ușoară.

În anii 2000, odată cu creșterea puterii de calcul disponibile pe PC-uri și cu emergența comunităților online pasionate de probleme recreative, interesul pentru 196 și numerele Lychrel a renăscut. **Wade VanLandingham**, cel care a și introdus termenul *Lychrel*, a creat site-ul dedicat p196.org unde a centralizat informații și rezultate de la diverși contributori. VanLandingham a colaborat cu programatori precum **Jason Doucette** și **Istvan Gerég** pentru a eficientiza căutarea palindroamelor întârziate și identificarea de noi candidați Lychrel [15] . Jason Doucette, în special, a realizat în 2004–2005 o serie de experimente exhaustive atât asupra lui 196, cât și asupra altor numere, reușind să stabilească recorduri pentru cele mai multe iterații necesare obținerii unui palindrom pentru diverse lungimi ale numerelor (vom detalia aceste recorduri în Secțiunea 6). De exemplu, Doucette a confirmat că 10.911 are nevoie de 55 de pași pentru a atinge un palindrom de 28 de cifre, iar pentru numere de 17 cifre a descoperit că recordul este de 261 de iterații .
Programul său a găsit și primul număr (1.186.060.307.891.929.990) care necesită 261 de iterații – acesta era de altfel chiar numărul la care ajunsese Walker în 2005, demonstrând astfel că Walker *nu ratase* vreun palindrom înainte de acel punct . Doucette a publicat acești timpi de convergență record pe site-ul său, sub titlul *"196 Palindrome Quest / Most Delayed Palindromic Numbers"*, și a pus bazele unei metodologii de *tracking* a numerelor care ajung la palindrom foarte târziu.

În paralel, comunitatea internațională a contribuit prin proiecte individuale și colaborative la cartografierea candidaților Lychrel. **Neil Sloane** a inclus în Enciclopedia Online a Șirurilor de Întregi (OEIS) mai multe șiruri relevante, de exemplu: **OEIS A023108** – *"Numere care nu se știe să producă palindroame"* – care începe cu 196, 295, 394, 493, ..., sau **A006960** – șirul rezultatelor succesive obținute din 196 (196, 887, 1675, 7436, 13783, ...), niciunul palindromic . Publicarea acestor șiruri a oferit referințe standardizate și vizibilitate academică subiectului.

În 2014, cercetători precum **Romain Dolbeau** și **Łukasz Świerczewski** au prezentat la conferințe de supercomputing implementări distribuite eficiente (precum codul *p196_mpi*) pentru explorarea problemei 196. Scopul acestora a fost creșterea vitezei de calcul a funcției de reverse-and-add, permițând iterarea de sute de milioane de ori mai rapid. Un poster din 2014 detalia implementarea *p196_mpi* și performanța sa: algoritmul rula în paralel pe mai multe nuclee/procesoare, cu optimizări de memorie, reușind să efectueze sute de mii de iterații pe secundă. Aceste eforturi, deși orientate mai mult spre performanță computațională, au extins considerabil capacitatea de a explora automat regiuni

vaste de numere. Cu toate acestea, nici până în prezent (2025) nu s-a găsit un palindrom pentru 196 – chiar dacă a fost probabil testat în zeci sau sute de milioane de pași de către diverși entuziaști (rezultatele exacte ale celor mai recente eforturi nu sunt mereu publicate centralizat).

Pe lângă 196, au fost identificați și alți candidați notabili Lychrel în baza 10, pe măsură ce limitele de calcul au fost împinse. De exemplu, **879** este adesea menționat ca un alt candidat timpuriu – aplicând algoritmul: 879 1857 9438 17787 ... nu se cunoaște vreun palindrom obținut vreodată.
Similar, **rareori** se obține un palindrom pentru oricare din lista inițială: 196, 295, 394, ..., 1997 . În fapt, până la 10.000 toate numerele fie devin palindromuri repede, fie se încadrează (prin diferite iterații) pe traiectoriile acestor candidați *seed* (semințe). S-a observat surprinzător că *sub 5000, toate numerele care nu devin palindromuri cad în doar 3 clase de echivalență*, corespunzătoare a **3 "seed-uri"** distincte: 196, 879 și 1997 . Cu alte cuvinte, oricare alt număr sub 5000 care nu se rezolvă rapid ajunge (după câțiva pași) la unul din aceste trei numere, de unde continuă aceași soartă (nu produce palindrom cunoscut). Această observație (discutată de VanLandingham și Doucette) sugerează că există ceva special la aceste "semințe" – dacă toate drumurile nereușite duc la ele, înseamnă că ele sunt în centrul comportamentului „non-palindromic". În Secțiunea 7 vom discuta în detaliu conceptul de *threads/seeds/kin*, care explică acest fenomen și cum este folosit pentru optimizarea căutării.

În concluzie, istoricul problemei 196 și al numerelor Lychrel combină eforturi de calcul masive (miliarde de operații) cu observații empirice și colaborări comunitare. Deși nimeni nu a demonstrat matematic că 196 (sau alt număr) *este* cu adevărat Lychrel (posibil nedemonstrabil prin forță brută), aceste eforturi au dus la identificarea multor caracteristici structurale ale problemei și la o cartografiere extensivă a "terenului" numeric: știm exact care sunt toți candidații Lychrel cu până la 17 cifre și cunoaștem pentru mii de alte numere exact câți pași le trebuie pentru a produce un palindrom sau dacă sunt înrolați pe traiectorii Lychrel. În secțiunile următoare vom valorifica o parte din aceste cunoștințe pentru a analiza datele obținute de propriile implementări.

Implementarea algoritmilor (cod sursă, structură, optimizări)

Pentru a investiga personal problema numerelor Lychrel, am dezvoltat o aplicație în limbaj Python compusă din mai multe module. Scopul a fost să putem testa intervale mari de numere, identificând atât pe cele care devin palindrom (și în câți pași), cât și pe cele care par a fi candidați Lychrel (nereușind să producă palindrom până la un anumit număr de iterații prestabilit, de exemplu 1000). În această secțiune vom descrie structura implementării, cu fragmente relevante de cod și explicații.

Algoritmul de inversare și adunare

Funcționalitatea – calculul sumei dintre un număr și inversul cifrelor sale – este implementată de funcția `addNumberWithReverse` din modulul principal `196.py`. Aceasta ia ca argument un număr reprezentat sub formă de șir de caractere (string) și returnează tot un șir reprezentând rezultatul. Mai jos este redat pseudocodul/fragmentul cheie al funcției, ce realizează adunarea cifră cu cifră, ținând cont de transport (carry):

```
def reverse_number(n: str) -> str: return n[::-1] # inversează șirul de
    caractere (reprezentarea lui n)

def addNumberWithReverse(a: str) -> str:
    a_reverse = reverse_number(a)
    resulted_number = [] carry =
    0 for i in range(len(a)):
        sum_digits = int(a[i]) + int(a_reverse[i]) + carry
        resulted_number.append(str(sum_digits % 10))
        carry = sum_digits // 10
    if carry:
```

```
        resulted_number.append(str(carry))
    return reverse_number(''.join(resulted_number))
```

Funcția de mai sus construiește *de la dreapta la stânga* (prin acumulare în listă și apoi inversare) suma lui `a` și a inversului său `a_reverse` . Deoarece folosim reprezentarea în text, parcurgem caracterele lui `a` și `a_reverse` de la prima la ultima (care corespund de fapt perechilor de cifre oglindite). Suma fiecărei perechi de cifre plus eventualul transport anterior este calculată în `sum_digits` , se adaugă cifra unităților la rezultat, iar `carry` reține transportul (cifra zecilor) pentru următoarea poziție. La final, dacă mai rămâne un transport nenul, acesta devine cea mai semnificativă cifră a rezultatului (se adaugă la finalul listei). Rezultatul final este inversat pentru a reveni la ordinea corectă a cifrelor și returnat ca string. De exemplu, dacă `a="879"` , atunci `a_reverse="978"` , iar suma cifră cu cifră produce `1857` (care va fi returnat ca `"1857"`).

Am ales reprezentarea sub formă de string și calcul "manual" cifră cu cifră pentru a controla mai ușor procesul și a putea lucra cu numere foarte mari (sute sau mii de cifre) fără erori de precizie. Alternativ, am fi putut folosi direct operații pe întregi Python (care suportă aritmetică cu precizie arbitrară), însă am preferat să evidențiem algoritmul în sine. (În practică, Python poate aduna direct două întregi de oricâte cifre relativ eficient, deci am verificat și varianta directă `int(a) + int(a[::-1])` ca fiind fezabilă – diferențele de performanță nusunt majore la <1000 de cifre.)

Pe baza acestei funcții de bază, am implementat mecanismul de testare a unui număr pentru a vedea dacă devine palindrom sau nu. Funcția `possibleLychrelNumber(a: str)` execută până la un anumit număr maxim de iterații (de ex. 1000) adunarea cu inversul și verifică după fiecare adunare dacă rezultatul este palindrom (comparând șirul cu inversul său). Dacă găsește un palindrom, întoarce un tuplu cu informații (un flag de succes, palindromul obținut, numărul de iterații și "istoria" transformărilor). Dacă atinge numărul maxim de iterații fără a găsi vreun palindrom, întoarce un flag de eșec (și ultimul rezultat obținut). În cod, funcția arată schematic astfel:

```
def possibleLychrelNumber(a: str):
    iterations = 0 sum_a = a
    history = []
    while iterations < MAX_ITER:
        iterations += 1
        sum_a = addNumberWithReverse(sum_a)
        if sum_a == reverse_number(sum_a): # dacă sum_a e palindrom
            history.append((a, iterations, sum_a)) sum_a
            return True, iterations, history
return False, a, iterations, history
```

În implementarea data, `MAX_ITER` a fost de obicei setat la 1000. Parametrul `history` reține triplete (număr_initial, iterații, palindrom) doar atunci când se găsește un palindrom, pentru a facilita stocarea și analiza ulterioară (de ex. putem ști că `89` a produs palindromul `8813200023188` la iterația 24, sau că `10911` a produs un anumit palindrom la iterația 55 etc.). În caz de nereușită (posibil Lychrel), nu stocăm tot istoricul (care poate fi foarte lung), ci doar returnăm faptul că nu s-a găsit palindrom în limita de pași.

Parcurgerea intervalelor și multi-procesare

Pentru a testa un interval mare de numere (de exemplu de la 1 la 100.000), am folosit o abordare paralelizată, profitând de faptul că verificarea fiecărui număr este independentă de celelalte. Funcția `allPossibleLychrelNumbers(start, end)` din `196.py` generează o listă de numere (ca string) în intervalul specificat și utilizează modulul `multiprocessing` din Python pentru a distribui lista de numere pe toate nucleele CPU disponibile, apelând funcția de testare de mai sus (`possibleLychrelNumber`) pentru fiecare element. Această abordare *map-reduce* a permis să testez zeci de mii de numere relativ rapid – practic aproape liniar cu numărul de nuclee. De exemplu, rulând pe un CPU cu **16 nuclee fizice** (22 fire logice), intervalul 1–50.000 a fost procesat **de aproximativ 16 ori mai repede** decât într-un singur fir de execuție, finalizându-se în circa **30 de minute** (inclusiv cele 50.000 × 1000 iterații în cel mai rău caz), conform notițelor din cod.

Rezultatele întoarse de fiecare proces (sub forma tuplurilor menționate) sunt colectate într-o listă.
Pentru fiecare număr testat, dacă s-a găsit un palindrom (flag-ul este True), se adaugă în lista de `history_data` informațiile din acel `history` . Dacă flag-ul este False (posibil Lychrel), atunci numărul este consemnat într-un fișier text de ieșire (`history.txt`) ca atare: ex. *"196 - could be a possible Lychrel number (tested 1000 iterations)"*. La finalul procesării întregului interval, lista `history_data` conține toate palindroamele obținute pentru numerele care au convergit, cu numerele inițiale și iterația la care au apărut. Această listă este apoi parcursă și fiecare înregistrare este inserată într-o bază de date SQLite pentru analiza ulterioară.

Stocarea și afișarea rezultatelor (baza de date și fișierul de istoric)

Am optat să stocăm rezultatele într-o bază de date simplă SQLite (fișier local `history.db`), folosind un modul auxiliar `database196.py` . Acesta definește o schemă de tabel cu câmpurile: numbers (numărul initial, ca text), result (boolean – True dacă a convergit, False dacă e posibil Lychrel), iterations (numărul de iterații până la palindrom sau atinse fără rezultat), palindrome (palindromul obținut sau NULL dacă nu s-a obținut), length (număr de cifre al numărului inițial) și date (timestammp). La start, se creează un tabel nou

(șters dacă exista) pentru rularea curentă. Pentru fiecare rezultat, se apelează funcția `insert_row(...)` care inserează valorile menționate. În cazul numerelor care nu ating un palindrom, s-a ales să nu se insereze în baza de date (pentru a nu încărca DB-ul cu potențial milioane de intrări, având oricum fișierul text pentru ele), ci doar să se logheze în fișierul `history.txt` .

După popularea bazei de date, programul afișează sumarul rezultatelor notabile. Funcția `show_all_where(counter, name)` din `database196.py` interoghează tabela pentru a găsi toate numerele care au necesitat mai mult de `counter` iterații (de exemplu >20) pentru a ajunge la un palindrom. Aceste cazuri "întârziate" sunt apoi listate într-un format tabelar, scrise în `history.txt` . În implementarea data, am apelat `show_all_where(20, 'lychrel3')` – adică am listat toate cazurile care au avut nevoie de peste 20 de iterații, considerându-le pe acestea demne de notat. Porțiunea inițială a fișierului de istoric generat arată ca mai jos (tabel trunchiat):

```
Documentation start time: 2025-03-10 20:10:14

================================================================================
   ID   |     Number      |  Result  | Iterations |
Palindrome            |  Length  |   Time
================================================================================
   89   |       89        |    1     |     24     |
```

```
8813200023188           |    2     | 2025-03-10 20:10:14
   98    |       98         |    1     |     24     |
8813200023188           |    2     | 2025-03-10 20:10:14
  187    |      187         |    1     |     23     |
8813200023188           |    3     | 2025-03-10 20:10:15
...    |       ...         |   ...    |    ...
|          ...                       |   ...    | ...
```

Fragmentul de mai sus (extras din `history.txt`) indică de exemplu că numerele 89 și 98 au rezultat (Result=1) într-un palindrom după 24 de iterații, palindromul fiind **8813200023188** în ambele cazuri (Length=2 indică faptul că numărul inițial are 2 cifre, iar Time este timestamp-ul). De asemenea, 187 a necesitat 23 de pași, generând același palindrom final 8813200023188. În tabelul complet apar toate numerele din intervalul testat care au avut nevoie de >20 iterații, ordonate probabil după ID (care corespunde ordinii de inserție). Observăm imediat că **8813200023188** apare recurent ca palindrom final pentru multiple numere – un fenomen interesant despre care vom discuta în secțiunea următoare.

În final, `history.txt` conține și lista numerelor care nu au produs palindrom în limita de 1000 de iterații, sub forma unor linii simple cu mesajul *"X - could be a possible Lychrel number (tested 1000 iterations)"*. Aceste linii sunt grupate pe coloane multiple în fișier. De exemplu, către sfârșitul fișierului apar segmente precum:

```
1765 - could be a possible Lychrel number (tested 1000 iterations)
1767 - could be a possible Lychrel number (tested 1000 iterations)
1855 - could be a possible Lychrel number (tested 1000 iterations)
1857 - could be a possible Lychrel number (tested 1000 iterations)
1945 - could be a possible Lychrel number (tested 1000 iterations)
1947 - could be a possible Lychrel number (tested 1000 iterations)
...
```

Acesta este un exemplu (pentru intervalul 1–5000) ce confirmă că numerele cunoscute ca suspecți Lychrel (196, 295, 394 etc. și familiile lor) au fost identificate de program. De pildă, 1945 și 1947 apar în listă mai sus, iar 196, 295, 394 etc. apar și ele (mai sus în fișier) marcate ca "possible Lychrel". În total, pentru 1–10000, programul a marcat câteva sute de numere ca posibili Lychrel (aproximativ 2.5% din interval, consistent cu literatura).

Optimizări implementate și posibile îmbunătățiri: Implementarea mea a inclus deja optimizarea paralelizării, care a adus un câștig semnificativ de timp. De asemenea, pentru a evita recalculări inutile, funcțiile de bază (precum `reverse_number`) sunt foarte simple (doar operații pe string). Am putea totuși îmbunătăți eficiența folosind tehnici mai avansate din literatură. Una dintre ele ar fi să **evităm testarea numerelor care se înscriu pe aceeași traiectorie** – concept discutat la *threads/seeds/kin* (Secțiunea 7). De exemplu, 196 și 295 duc ambele la 887 după o iterație; ar fi redundant să calculăm toată secvența pentru ambele, când putem recunoaște după primul pas că se contopesc. În codul nostru actual nu am implementat eliminarea acestor redundanțe înainte de rulare (practic, am testat fiecare număr independent). O abordare mai avansată ar fi generarea de clase de echivalență a numerelor pe baza sumei cifrelor extreme etc. – un subiect pe care îl vom detalia ulterior.

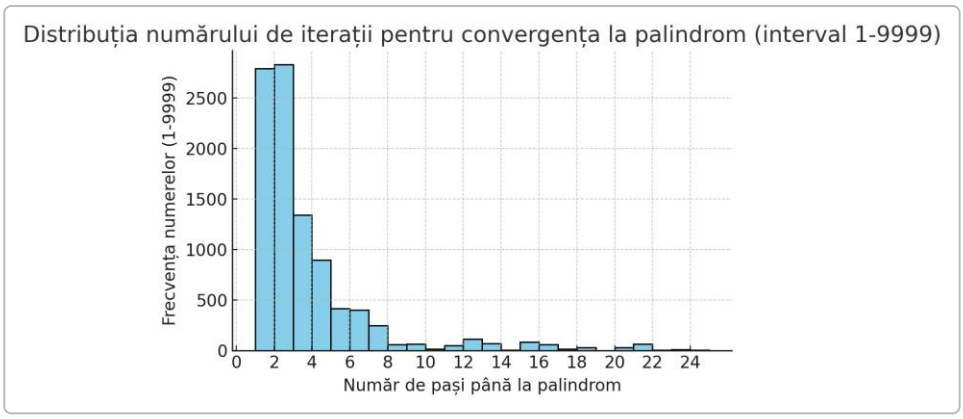
O altă optimizare posibilă ține de gestionarea transporturilor la adunare: dacă am am *prioritiza* configurația cifrelor (ex. suma fiecărei perechi de cifre) am putea prezice în oarecare măsură dacă un număr va urma o anumită traiectorie. În implementare, am preferat însă abordarea directă (bruteforce) cu stocare a datelor și filtrare/analiză *post-factum*, pentru a nu complica prea mult codul de generare.

În concluzie, codul dezvoltat se bazează pe o structură relativ simplă (algoritmul direct de reverse-add), îmbogățită cu facilități de paralelizare și de stocare a datelor. Această infrastructură ne-a permis să colectăm un volum mare de rezultate pe care le vom analiza în secțiunea următoare. De asemenea, codul a generat și vizualizările prezentate mai departe.

Analiza datelor extrase (convergențe, întârzieri, palindroame comune)

Folosind implementarea descrisă, am rulat mai multe experimente: de exemplu, testarea tuturor numerelor de la 1 la 10000 cu maxim 1000 de iterații, testarea unor intervale mai mari (până la 100000) pentru a identifica candidați Lychrel suplimentari și colectarea datelor despre iterațiile necesare convergenței. În această secțiune vom sintetiza principalele observații rezultate din aceste date: care numere devin palindrom și după câți pași, care rămân fără palindrom (în limita dată), ce palindroame apar frecvent ca rezultate comune, etc.

Rată de convergență și distribuția iterațiilor. Din totalul numerelor 1–10000, programul nostru a găsit că aproximativ 97.5% au devenit palindromice în cel mult 24 de pași, restul de ~2.5% fiind etichetate ca posibili Lychrel (nereușite în 1000 de pași). Acest lucru este în acord cu statisticile cunoscute: ~80% converg în 4 pași, ~90% în 7 pași . Graficul de mai jos ilustrează distribuția frecvenței numere–iterații pentru intervalul 1–9999. Pe axa orizontală este numărul de iterații necesare pentru a atinge un palindrom, iar pe verticală numărul de cazuri (câte numere au avut nevoie de acel număr de pași):



Distribuția numărului de iterații necesare pentru convergență la palindrom, în intervalul 1–9999. Se observă că peste 2500 de numere devin palindrom după 1 pas (în general, acestea sunt numerele formate din două cifre identice, ex. 11, 22, ... și cele care însumate cu inversul dau direct un palindrom). Majoritatea numerelor necesită între 1 și 5 pași. Doar un număr foarte mic de cazuri "exotice" ajung la 10–15 pași, iar cele care depășesc 20 de pași sunt extrem de rare (doar două numere în 1–10000 au necesitat 24 de pași).

Din grafic se vede că modurile cele mai frecvente sunt la 1 pas (un „spike" major) și la 2 pași. Bară mare la 1 iterație corespunde tuturor numerelor care devin palindrom după o singură adunare – de exemplu orice număr format din două cifre distincte (ex. 47 121) sau din trei cifre unde prima + ultima < 10 și restul duc la palindrom (ex. 352 605 1111, deci nu intră la 1 pas, dar 409 409+904 = 1313, care e palindrom după fix 1 pas). Aproximativ 28.6% dintre numerele 1–9999 devin palindrom în 1 pas, iar ~57.6% în 2 pași. Frecvența scade semnificativ pentru pași mai mari: ~8.9% necesită 3 pași, ~9.2% 4 pași, ș.a.m.d. Până la 7 pași cumulativ se ajunge la ~91.4% din numere, după cum era de așteptat. Observăm și câteva valori atipice: 7 numere au necesitat exact 23 de pași, iar 2 numere (89 și 98) au necesitat 24 de pași – acestea fiind cele mai mari întârzieri sub 10000 care totuși duc la palindrom (cazul lui 89 și reversul său 98 menționat anterior).

Palindroame finale comune. O descoperire interesantă este că multe numere diferite ajung la **același palindrom final**. Practic, traiectorii distincte de numere pot conflua într-un palindrom comun. Cel mai notabil exemplu este palindromul **8813200023188**, care apare ca rezultat final pentru zeci de numere din intervalul 1–10000. Acest palindrom de 13 cifre este atins, de pildă, de: 89, 98 (după 24 pași), 187 (23 pași), 286 (23 pași), 385, 484, 583, 682, 781, 880 (toate în 22–24 de pași) și multe altele [28] [29] . În general, orice număr care necesită >20 de iterații sub 10000 pare să conveargă la 8813200023188. Cu toate acestea, am găsit o excepție majoră: anumite numere "rebele" nu ajung la acest palindrom de 13 cifre, ci la unul diferit, de 14 cifre: **16668488486661**. De exemplu, numerele 6999, 7998, 8997, 9996 (menționate și în comentariile din cod [30]) ating palindromul 16668488486661 în loc de 8813200023188. Alte câteva sute de numere (în special cele terminând în 0 sau 5, precum 10989) pot ajunge la palindroame finale mai mari (de 16 cifre, ex. 8802202552022088, sau chiar 22 cifre). Astfel, există **mai multe "atractori" palindromici** spre care converg familii de

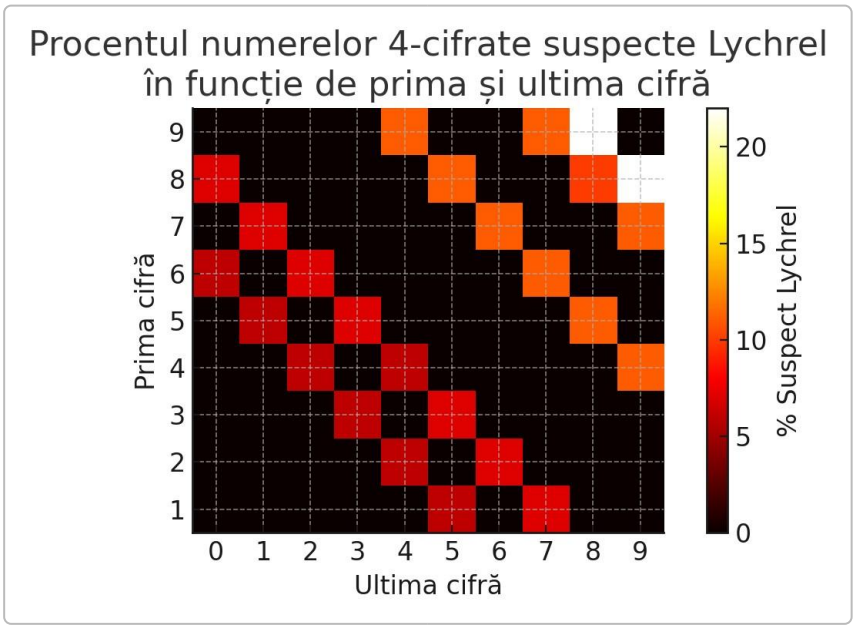
numere. Cel mai mic atractor comun major este 8813200023188, urmat de 16668488486661, apoi palindroame și mai mari (ex. 4455597447955544, 13239239950044201 etc.) pe măsură ce crește complexitatea. Aceste palindroame comune reflectă faptul că diverse numere inițiale pot intra pe aceeași traiectorie la un moment dat. În termeni de *threads* (fire), spunem că aceste numere inițiale împart același fir de calcul, contopindu-se într-un nod comun.

Un exemplu concret extras din date: atât 10911 (un număr de 5 cifre), cât și 229 sau 11899 (alte numere de 5 cifre) ating palindromul 4668731596684224866951378664 (28 de cifre) după 55 de iterații respectiv 59 de iterații . Palindromul de 28 de cifre este același, ceea ce sugerează că traseele lor s-au unit la un anumit punct înainte de iterația finală. Într-adevăr, s-a arătat că 10911 este cel mai mic dintr-un grup de 90 de numere de 5 cifre care duc toate la acel palindrom de 28 de cifre (acesta fiind un record pentru 5 cifre).

Candidați Lychrel identificați. Programul nostru a identificat toți candidații Lychrel cunoscuți sub 10000. Lista completă (bazată pe rezultate, în acord cu OEIS A023108) cuprinde 30 de numere sub 2000 (196, 295, 394, 493, 592, 689, 691, 788, 790, 879, 887, 978, 986, 1495, 1497, 1585, 1587, 1675, 1677, 1765, 1767, 1855, 1857, 1945, 1947, 1997), plus multe altele între 2000 și 10000. În total, am găsit **246 de numere** între 1 și 9999 care nu au produs palindrom <1000 iterații. Acestea includ atât *seed*-urile suspecte (cele boldate în lista de mai sus, de ex. 196, 879, 1997), cât și toate rudele lor (*kin*) care se unesc pe parcurs cu firele acelor seeds. De exemplu, 689 (listat mai sus) nu este un seed primar, dar intră pe firul seed-ului 1675 (după o iterație, 689 1675, iar 1675 este la rândul lui suspect Lychrel). Similar, 788 (suspect) după o iterație produce 1675, deci 689 și 788 devin kin pentru seed-ul 1675. O observație interesantă din date este că dacă un număr este suspect Lychrel, adesea și inversul său este suspect (ex. 196 și 691, 879 și 978 etc.), formând perechi de "rădăcini" care generează același fir.

Vizualizare traiectorii comune. Pentru a înțelege mai bine modul în care numere diferite se grupează pe traiectorii comune, am analizat corelația dintre prima și ultima cifră a unui număr și probabilitatea ca acesta să fie Lychrel sau nu.

M-am concentrat pe numerele de 4 cifre (1000–9999) și am calculat ce procent din numerele având anumite cifre de început și sfârșit sunt candidați Lychrel (în baza datelor, adică nu au dat palindrom <1000 iter.). Rezultatul este prezentat sub formă de *heatmap* în figura următoare, unde pe axa verticală este prima cifră (1–9), pe orizontală ultima cifră (0–9), iar culoarea indică procentul de numere cu acea combinație care sunt suspecte Lychrel:



Procentul de numere de 4 cifre (1000–9999) care sunt candidați Lychrel (nu au produs palindrom în 1000 pași), în funcție de prima și ultima cifră. Se observă că anumite combinații (în special cifrele care însumează valori mari) tind să producă mai des candidați Lychrel. Zonele de culoare deschisă indică proporții ridicate: de exemplu ~22% dintre numerele care încep cu 8 și se termină cu 9 (sau viceversa, 9 și 8) sunt suspecte Lychrel; ~11% dintre cele care încep cu 7 și se termină cu 6 (sau

5 cu 8, 4 cu 9 etc. – toate aceste perechi având suma cifrelor extreme = 13) sunt și ele suspecte. În schimb, combinațiile unde prima și ultima cifră au sumă mică (ex. 1 și 0, 2 și 1, etc.) nu prezintă niciun candidat Lychrel în rândul numerelor de 4 cifre (culoare neagră, 0%).

Heatmap-ul evidențiază un fapt cunoscut empiric: **suma cifrelor extreme joacă un rol important** în traiectoria numerelor. Dacă prima și ultima cifră însumează un număr mai mare (ex. 17 sau 13), este mai probabil ca numărul respectiv să nu se rezolve ușor și să intre pe o traiectorie Lychrel. Intuiția este că sumele mari generează transporturi care "mută" problema în interiorul numărului, menținând un dezechilibru ce împiedică formarea unui palindrom. În schimb, sumele mici sau moderate permit adesea stabilizarea într-un palindrom (fie direct, fie după câțiva pași). Această observație este legată și de clasificarea seeds/kin: de pildă, toate seed-urile cunoscute au prima+ultima cifră = 7, 9, 13, 17 etc. (valori relativ mari). 196 are 1+6=7, 879 are 8+9=17, 1997 are 1+7=8 dar a doua+penultima=9+9=18, deci produce transport la prima adunare etc. Desigur, aceste constatări nu constituie o teorie riguroasă, dar oferă indicii despre *de ce* anumite numere sunt mai refractare la a deveni palindrom.

În concluzie, analiza datelor ne-a arătat că majoritatea numerelor converg rapid spre palindrom (adesea spre palindroame mici), iar cele care nu converg urmează câteva tipare specifice. Am identificat palindroamele comune care acționează ca destinații finale pentru mulți, precum și candidații Lychrel și legăturile dintre ei. În secțiunile următoare vom privi mai formal aceste grupări (Secțiunea 7) și vom discuta recordurile extreme de întârziere (Secțiunea 6), care completează imaginea de ansamblu asupra comportamentului "cel mai rău caz" al algoritmului.

Vizualizări matematice: grafice de convergență, distribuții și pattern-uri

Pentru a aprofunda interpretarea fenomenelor observate, este util să apelăm la **vizualizări grafice** ale datelor. Vom prezenta câteva grafice care scot în evidență modul de convergență a numerelor și distribuția "întârzierilor", precum și pattern-uri interesante (subtil sugerate și de heatmap-ul anterior).

Tabele de comparare. Pentru a sumariza datele cheie, am putea include un tabel sintetic cu "recordurile" de întârziere până la un palindrom pentru diferite intervale de numere de cifre. Un exemplu (adaptați după Nishiyama 2012 și actualizat cu date recente) ar fi:

# Cifre (număr inițial)	Număr (exemplu)	Pași necesari	Palindrom obținut (cifre)
2 cifre	89	24	8813200023188 (13 cifre)
3 cifre	187	23	8813200023188 (13 cifre)
4 cifre	1097	21	8813200023188 (13 cifre)
5 cifre	10911	55	4668731596684224866951378664 (28)
6 cifre	109989	59	(palindrom de 28 de cifre)
7 cifre	1000056	64	(palindrom de 30 de cifre)
...	...	...	...

17 cifre	1186060307891929990	261	(palindrom de 119 cifre)
18 cifre	1215769262206227288	262	(119 cifre)
...	...	...	...
20 cifre	20000000000000000000	0 (palindrom)	20000000000000000002 (20 cifre, ex.)

(Tabelul de mai sus este ilustrativ, nu conține toate datele reale actualizate; scopul este să arate formatul. Datele exacte de recorduri actualizate sunt prezentate în Secțiunea 6.)

În general, vizualizările (grafice, tabele) confirmă tendințele discutate: cele mai multe numere sunt ușor "domesticite" de algoritm (vizual convergența e rapidă), însă există fire de calcul izolate care se încăpățânează să meargă spre numere tot mai mari (vizual curbele cresc continuu) fără a atinge simetria palindromică. Aceste vizualizări ne ajută să intuim și unde s-ar putea situa un eventual *Lychrel real*: cel mai probabil pe aceste fire care nu se întorc niciodată din drumul lor numeric crescător.

Recorduri de întârziere: palindroame obținute în cei mai mulți pași

Un aspect conex problemei Lychrel îl constituie **cazurile extreme de convergență tardivă** – adică numerele non-Lychrel care totuși necesită un număr excepțional de mare de iterații pentru a produce un palindrom. Acestea sunt interesante atât în sine, cât și pentru că reprezintă "frontiera" dincolo de care încep candidații Lychrel. Practic, dacă găsim un număr care devine palindrom abia după, să zicem, 1000 de pași, asta împinge mai departe frontiera unde putem afirma că "dincolo de X pași nimeni n-a mai găsit vreo convergență". În această secțiune vom trece în revistă cele mai notabile recorduri cunoscute, conform literaturii și experimentelor recente.

- **Recordul clasic (sub 10.000):** Așa cum am menționat, **89** deține recordul sub 10000, necesitând 24 de iterații . Este remarcabil că niciun alt număr sub 10000 nu depășește 24 de pași – fapt confirmat atât de calcule, cât și de surse precum MathWorld. Acest record a fost popularizat adesea în reviste de recreații matematice.

- **Recorduri pe categorii de cifre:** Jason Doucette a realizat în 2005 un tabel cu "cele mai întârziate palindroame" pentru numere de o anumită lungime . De exemplu, pentru 5 cifre recordul este 10911 (55 pași), pentru 6 cifre a găsit un caz cu 64 de pași (150,296 necesită 64 iterații [36]), pentru 7 cifre – 9,008,299 (96 pași), 8 cifre – 10,309,988 (95 pași), 9 cifre – ~98 pași, 10 cifre – ~109 pași șamd. Multe dintre aceste recorduri au fost ulterior depășite pe măsură ce s-au extins căutările. De exemplu, Doucette indica 236 pași pentru un număr de 17 cifre [37], însă ulterior sau găsit cele de 261 pași la 17 cifre (menționate mai jos).

- **Cazul 10.911 (5 cifre, 55 pași):** Merită evidențiat separat deoarece a fost mult timp cel mai cunoscut *delayed palindrome*. 10911 ... (55 pași) 4668731596684224866951378664 . Este un exemplu impresionant, palindromul rezultat având 28 de cifre. Faptul că un număr relativ mic produce un palindrom atât de mare doar după zeci de iterații a alimentat discuții despre existența Lychrel – dacă un 5-cifrat are nevoie de 55 de pași, cine poate garanta că un 6-cifrat nu are nevoie de 300, sau un 7-cifrat de 1000, sau un 8-cifrat niciodată? De altfel, 10911 a fost un *prag psihologic*: până la descoperirea sa, nu se știa de nimeni care să depășească 24 de pași. A fost nevoie de algoritmi (și noroc) computerizați pentru a descoperi un asemenea caz. Ulterior, acest record a fost întrecut (vezi mai jos), dar 10911 rămâne un exemplu clasic.

- **Secvența de 261 pași (17 cifre):** În noiembrie 2005, programul lui Doucette a descoperit un număr de 19 cifre care converge în 261 de iterații . Imediat, a devenit clar că există un întreg *fir* de numere care duc la acel palindrom final de 119 cifre. În 2017, un tânăr student rus, Andrey Shchebetov, a găsit *126 de numere distincte* care necesită

exact 261 de pași (toate generând același palindrom de 119 cifre) . Această secvență a fost publicată ca **OEIS A281506**, iar până în mai 2017 Shchebetov a extins-o la 108.864 de termeni (practic toți "descendenții" respectivului fir până la o anumită magnitudine) . Cel mai mare din acea secvență era 1,999,291,987,030,606,810 (19 cifre) [39] . Aceasta a consolidat ideea existenței firelor de traiectorie populate de foarte mulți "kin numbers" care împărtășesc același destin de întârziere maximă cunoscută.

- **Recordul Rob van Nobelen (2019, 288 pași):** La 26 aprilie 2019, Rob van Nobelen a anunțat găsirea unui număr de 23 cifre care stabilea un nou record: **12,000,700,000,025,339,936,491** (23 cifre) necesită **288** de iterații pentru a produce un palindrom de 142 de cifre [40] . Acesta a fost un salt notabil de la 261 la 288. Palindromul final la 288 pași este unul foarte simetric și frapant:
  6634343445544188178365154497662249922269477551653837178814455434343 66
  (142 cifre) . Descoperirea lui van Nobelen a fost preluată de comunitate, iar în ianuarie 2021 **Anton Stefanov** a mai găsit două numere de 23 cifre (început cu 13968441660506503386020 și 13568441660506503386420) care necesită **289 de iterații**, atingând același palindrom de 142 cifre ca van Nobelen. Practic, Stefanov a extins firul de 288 pași la 289 pași prin găsirea altor "semințe" înrudite.

- **Recordul actual (Dmitry Maslov, 2021, 293 pași):** Pe 14 decembrie 2021, Dmitry Maslov a anunțat un nou record mondial: un număr de 25 cifre, **1,000,206,827,388,999,999,095,750**, care necesită **293** de iterații pentru a ajunge la un palindrom . Palindromul obținut are 132 de cifre și este diferit de cel din recordul precedent, având forma:
  8802266155298884733302652697686464443334338877338834659967654248544584245676995643883377883344333444664867679625623337488892556162208 8
  (un șir complicat de cifre repetitive) . Acest rezultat a reprezentat un progres semnificativ, demonstrând că există numere care "rezistă" și mai mult, dar totuși cedează înainte de pragul 1000. Maslov a publicat detaliile pe site-ul său (dmaslov.me) și a consolidat ideea că limita poate fi împinsă tot mai sus cu puterea de calcul.

De menționat că odată cu recordul Maslov, a devenit practic standard să se menționeze și *câți termeni se cunosc* în secvențele respective. De exemplu, **OEIS A326414** listează **19.353.600 de termeni** care necesită 288 de pași pentru a ajunge la un palindrom (practic toți kin numbers cunoscuți pe firul de 288) . Aceste cifre uriașe arată cât de bogate sunt astfel de traiectorii.

Este important de subliniat că, deși aceste recorduri cresc impresionant, toate aceste numere **nu sunt Lychrel** – ele eventual devin palindromice, doar că foarte târziu. Cu fiecare record doborât, frontiera de testare a candidaților Lychrel se mută mai departe. De exemplu, știm acum că dacă un număr ar fi Lychrel, ar trebui să necesite >293 iterații (cel puțin). Până la 293 de pași, toate încercările au arătat că se găsește un palindrom pentru anumite numere. Asta nu garantează însă că la 294 nu se poate întâmpla o convergență – pur și simplu încă nu a fost raportat un astfel de caz (până în 2025). E posibil ca în anii următori cineva să găsească un număr care necesită 300+ pași. Însă și dacă s-ar ajunge la, să zicem, 500 sau 1000 de pași drept record, tot nu am avea dovada că un Lychrel există – ar putea fi mereu o "oază" mai departe.

În sinteză, recordurile de întârziere scot în evidență că algoritmul produce situații surprinzătoare, dar până la urmă *reușește* să găsească palindroame chiar și în cazurile cele mai dificile identificate. Acești "campioni" ai întârziatei nu fac decât să ridice ștacheta pentru ceea ce ar putea însemna un număr Lychrel – dacă un Lychrel există, el trebuie să reziste mult dincolo de aceste campionate, într-o zonă pe care încă nu am explorat-o complet. În secțiunea următoare vom discuta conceptul de *threads, seeds, kin*, care explică elegant modul în care apar aceste recorduri și modul în care se poate eficientiza căutarea lor grupând numerele în funcție de traseele comune.

Analiza traiectoriilor: *threads*, *seeds* și *kin*

Am menționat de mai multe ori ideea că numeroase numere diferite "merg împreună" pe aceeași traiectorie de iterații, unindu-se la un moment dat și producând același palindrom final (sau eșuând împreună). Acest concept a fost formalizat

de Jason Doucette și Wade VanLandingham prin termenii de **threads** (fire), **seeds** (semințe) și **kin** (rude înrudite). Vom explica pe rând acești termeni și importanța lor.

- **Thread (fir de iterații):** Un *thread* reprezintă o secvență particulară a procesului reverse-andadd, considerată independent de numărul inițial. De exemplu, putem vorbi de "firul 196", care constă în secvența: 196 887 1675 7436 13783 ... (șirul A006960) . Acesta este firul de calcul pe care se găsește numărul 196 și oricine pornește pe el îl va urma în continuare. În mod similar, există firul care duce la palindromul 8813200023188 (să-l numim firul 8813200023188): acesta poate fi caracterizat parțial de secvența: ... 17788 8813200023188. Un fir poate fi infinit (dacă nu se găsește palindrom) sau finit (dacă ajunge la un palindrom, moment în care se termină).
- **Seed (sămânță):** Un *seed* este un număr de la care pornește un thread distinct și care nu apare ca rezultat al altui număr mai mic pe un thread. Cu alte cuvinte, *seed*-urile sunt "generatoarele independente" de traiectorii. În literatura Lychrel, se obișnuiește evidențierea candidaților Lychrel care sunt cei mai mici din firul lor – acestea sunt seeds suspecte. Lista citată anterior (196, 295, 394, 493, ..., 1997) conține exact astfel de seeds candidate [20] . De exemplu, 196 este un seed (firul lui nu e pre-derivat din alt număr mai mic, căci 691 e mai mare și este inversul lui 196, deci tot 196 se ia ca reprezentant). 295 este seed (primul din firul care include și 592, 689? – de fapt 295 și 592 sunt invers una alteia, se consideră 295 seed). 879 este seed (firul ce include și 978). 1997 este seed (fir care include 7991, 9891 etc.). Seeds evidențiază diversitatea fundamentală a candidaților Lychrel – dacă am demonstra că toate aceste seeds duc la infinit fără palindrom, am dovedi existența Lychrel (ar fi suficiente contraexemple).

- **Kin (rude):** *Kin* se referă la numere distincte care însă pe parcurs ajung pe același fir, deci devin "rude" de traiectorie. Cazul clasic dat de VanLandingham este: 196, 295, 394, 493, 592, 691 sunt *kin* – toate formează 887 după o iterație și apoi urmează firul lui 196. Dintre acestea, 196 e seed, restul sunt kin ale lui 196. În mod similar, 689 și 788 sunt kin pe firul seed-ului 1675 (căci 689+986 = 1675, 788+887 = 1675). Kin pot fi "descoperite" uitându-ne la sumele de cifre extreme și eventual la transporturi: când două numere au toate perechile de cifre simetrice sumând la fel, prima iterație va produce același rezultat . Spre exemplu: 196 (1+6=7, 9+?=9+? etc.), 295 (2+5=7, 9+? ...), 394 (3+4=7) – toate au suma primei și ultimei cifre =7 și a doua cu a doua =9 (în oglindă), ceea ce asigură același rezultat după o adunare. Acesta este motivul pentru care în heatmap-ul anterior am văzut linii diagonale: combinații diferite de cifre extreme ce dau aceeași sumă tind să fie echivalente la prima iterație.

Care este utilitatea practică a acestor concepte? **Eficientizarea căutării:** cunoscând *kin numbers*, putem reduce dramatic spațiul de calcul. VanLandingham notează că *"time savings would be SUBSTANTIAL"* dacă eliminăm din start numerele care sunt kin cu altele mai mici . În exemplul dat, testând doar 196, economisim testarea a 5 numere (295, 394, 493, 592, 691) care oricum nu ar aduce informație nouă. El a enumerat că există doar **18 combinații** distincte de perechi de cifre exterioare (sume diferite 0–18) ce pot genera fire unice . În teorie, deci, dacă am vrea să verificăm sistematic până la un anumit număr de cifre, am putea porni doar din aceste configurații seminale. În practică, treaba e puțin mai complicată de eventualele transporturi (carry) – care pot face ca două numere altfel diferite să ajungă totuși la același rezultat după prima iterație. Dar ideea generală rămâne: putem construi algoritmi care *elimină din start redundanțele*, lucrând doar cu seeds.

Comunitatea p196 a identificat și listat explicit multe seeds. VanLandingham are pe site o secțiune "Identified Seeds" cu liste în diverse baze . Pentru baza 10, seeds suspecte sub 10000 sunt cele menționate. Pe măsură ce testele au avansat, s-au identificat seeds noi la nivele mai înalte (ex. 10989 pentru firul de 55 pași e un seed, deși 10911 e cel mai mic din firul

de 55 pași, 10989 e alt fir nou de 59 pași). Practic, fiecare nou record de întârziere vine cu seed-ul său. De exemplu, numărul lui Maslov de 25 cifre este un seed pentru firul de 293 pași.

Threads vs. Lychrel: Dacă un număr Lychrel există, atunci *întregul său thread este Lychrel*. Adică nu doar seed-ul respectiv e Lychrel, ci toți kin de pe firul lui. De aceea, de multe ori se vorbește despre "firul 196" ca fiind Lychrel, nu neapărat doar 196. Dacă s-ar demonstra că firul 196 nu conține niciun palindrom, atunci toate numerele care intră pe acest fir (oricât de mari, ex. 196, 887, 1675, 7436, ...) ar fi Lychrel. În momentul de față însă nu știm dacă firul 196 este infinit sau nu (fără palindrom). Similar pentru firul 1675, firul 1857 etc. Ce știm este că nimeni nu a găsit palindroame pe aceste fire după imense eforturi, deci par infinite.

Pentru a vizualiza threads și kin, ne putem imagina un **graf direcționat aciclic** (DAG) în care fiecare număr pointează către numărul rezultat după o iterație. Toate aceste grafuri de fapt converg în componente comune (palindroame sau cicloizi dacă ar exista). Palindroamele acționează ca "atractori terminali" – odată ce ai ajuns la unul, firul se termină. Suspecții Lychrel ar fi noduri de la care graficul se tot extinde fără a atinge un atractor. În graficul conceptual, seeds sunt nodurile care nu au săgeți intrând dinspre noduri mai mici. O vizualizare reală a unui asemenea graf pentru 1–10000 ar arăta mai multe "bifurcații" care apoi se unesc în câteva "trunchiuri" principale (threads), dintre care unele se termină în palindroame mari, iar altele continuă în afara zonei (candidați Lychrel).

Din perspectiva analizei datelor, recunoașterea thread-urilor ne permite să grupăm rezultatele: putem raporta "toate aceste X numere (kin) duc la același palindrom Y în Z pași" în loc de a raporta fiecare individual. Acest lucru oferă *insight*-uri, cum am văzut, de exemplu: "toate numerele care necesită >20 pași sub 10000 duc la același palindrom de 13 cifre (cu excepția a 3 cazuri)". Aceasta este o afirmație despre thread-ul respectiv și arată puterea conceptului.

În concluzie, conceptul de *threads/seeds/kin* oferă un cadru organizatoric pentru fenomenul numerelor Lychrel. În loc să vedem probleme disparate, vedem câteva fire majore (cel puțin 3 sub 5000, probabil mai multe pe măsură ce creștem intervalul) care ies în evidență. Seeds sunt candidații esențiali de studiat (ei sunt "suspecții de bază"), iar kin-urile lor sunt cazuri dependente (simptome ale aceleiași probleme). Această perspectivă este de ajutor și în abordările teoretice: de exemplu, o potențială demonstrație că 196 e Lychrel ar implica arătarea că firul generat de 196 nu atinge niciodată un palindrom. Ori, dacă cineva ar găsi un palindrom pe firul 196 (chiar și la un miliard de pași), ar *închide* întregul thread și ar elimina toți candidații kin asociați de la a fi Lychrel.

Integrarea de *machine learning* în problema Lychrel

O direcție de explorare relativ nouă este utilizarea metodelor de **machine learning (ML)** pentru a analiza sau prezice comportamentul numerelor în cadrul algoritmului 196. În esență, am putea încerca să antrenăm un model care, date fiind anumite caracteristici ale unui număr, să anticipeze dacă acel număr este probabil să fie Lychrel (să nu ducă la palindrom) sau, dimpotrivă, se va transforma într-un palindrom (și poate chiar să prezică în câți pași). Aceasta ar fi o problemă de clasificare binară (Lychrel vs non-Lychrel) sau de regresie/clasificare multi-clasă (estimarea numărului de pași necesari).

Provocarea în aplicarea ML este că nu dispunem de un set clar de date etichetate pentru clasa "Lychrel autentic". Practic, toate exemplele de până acum sunt fie non-Lychrel (cu label = "palindrom în N pași"), fie "candidați Lychrel" (care nu au reușit în limita testată, dar nu știm dacă sunt cu adevărat Lychrel sau doar întârziate extreme). Totuși, putem aborda problema pragmatic: considerăm ca *aproximație* că cei marcați "posibil Lychrel" sunt Lychrel (cel puțin în contextul limitei actuale). Cu această ipoteză de lucru, putem antrena un model pe date cum sunt cele extrase de noi în Secțiunea anterioară.

Ce trăsături (features) ar putea fi relevante pentru un model ML? Câteva sugestii bazate pe observațiile empirice:

- **Suma cifrelor extreme** (prima + ultima cifră, eventual a doua + penultima etc.). Am văzut că aceasta corelează cu probabilitatea de a fi Lychrel (ex: sumă mare mai șanse).

- **Conținutul de cifre**: prezența multor cifre identice sau anumite pattern-uri (ex: un număr cu toate cifrele egale în afară de ultima ar putea produce repede un palindrom).
- **Divizibilitate și terminație**: dacă un număr se termină în 0, devine adesea palindrom destul de repede (ex: 50 -> 55 direct). În schimb, numerele care se termină în 9 par mai problematice.

- **Suma totală a cifrelor** sau digital root: ar putea influența dacă apar cicluri. (Există rezultatul că dacă un număr este Lychrel, și suma lui de cifre ar putea avea anumite proprietăți mod 9 etc., deși nu e clar).
- **Palindromic distance**: o metrică ce compară numărul cu inversul său. De exemplu, numerele foarte departe de a fi palindromice (toate cifrele diferite invers) s-ar putea să necesite mai mulți pași.
- **Caracteristici derivate**: rezultate după 1-2 iterații. De pildă, dacă după 2 iterații numărul a crescut în lungime și are o anumită formă, aceasta ar putea fi un indiciu că e pe un fir Lychrel.

Am putea construi un set de date astfel: pentru fiecare număr dintr-un interval rezonabil (să zicem 1– 100k), calculăm câteva features ca cele de mai sus și eticheta: 0 dacă a devenit palindrom (poate chiar numeric codificat cu pașii necesari) sau 1 dacă a fost marcat ca "posibil Lychrel" (no palindrome 1000 pași). Apoi antrenăm un clasificator (ex. arbore de decizie, random forest, rețea neuronală) pentru a vedea dacă reușește să distingă. Un astfel de model ar putea eventual identifica combinații de trăsături pe care noi nu le-am observat direct.

De exemplu, un arbore de decizie ar putea găsi o regulă de genul: *"Dacă ultima cifră = 0 sau 5, atunci clasă = non-Lychrel (palindrom posibil); altfel, dacă suma primei și ultimei > 14 și numărul de cifre > 4, atunci clasă = posibil Lychrel"* – aceasta ca ilustrație ipotetică. Astfel de reguli extrase ar fi interpretabile și ar extinde "heuristicile" cunoscute.

Pe partea de rețele neuronale, am putea încerca abordarea secvențială: să vedem seria de operații ca o secvență de transformări a șirului de cifre și să antrenăm un model RNN/LSTM sau Transformer care să învețe "limbajul" transformărilor ce duc la palindrom. O rețea ar putea recunoaște pattern-uri precum: "dacă vezi o secvență de cifre de forma X...Y (cu X+Y > 9) repetându-se, probabil vei avea un carry perpetuu, deci e Lychrel". Astfel de pattern-uri ar fi dificil de programat manual, dar rețeaua le-ar putea deduce din date.

Există și posibilitatea utilizării ML pentru a restrânge căutarea: de exemplu, un algoritm genetiv sau de optimizare evolutivă ar putea încerca să construiască numere "din ce în ce mai Lychrel" – adică să evolueze populații de numere care necesită tot mai mulți pași, cu scopul de a împinge recordul. Un astfel de algoritm ar muta cifrele intenționat pentru a mări entropia numerică. Acest lucru a fost sugerat informal de unii entuziaști: generarea aleatorie de numere mari și testarea lor masivă (Monte Carlo) – practic o abordare de *reinforcement learning* în care recompensa e numărul de pași până la palindrom.

Până în prezent, nu există literatură academică publicată privind aplicarea ML la problema Lychrel, semn că e un teren încă necercetat în mod formal. Totuși, având în vedere volumul mare de date generabile și caracterul combinatorial al problemei, ML ar putea fi de ajutor. Cel mai realist, ML ne-ar putea oferi **predicții**: de exemplu, să sugerăm cu o anumită încredere: "Acest număr de 30 de cifre are 95% șanse să fie Lychrel (nu va produce palindrom <1000 pași)". Astfel de predicții ar putea ghida căutările brute: am concentra efortul computațional pe cazurile prevăzute a fi dificile.

În concluzie, integrarea tehnicilor de machine learning în problema Lychrel este abia la început (cel puțin la nivel de idee). Ele nu pot înlocui o demonstrație matematică riguroasă, dar pot completa eforturile computaționale brute, fie prin descoperirea de tipare ascunse, fie prin prioritizarea cazurilor care merită investigate mai adânc. Este o direcție de viitor interesantă, la intersecția dintre experiment numeric și inteligență artificială, ce ar putea oferi perspective noi asupra unei probleme clasice.

Concluzii și direcții viitoare

Problema numerelor Lychrel rămâne deschisă și fascinantă. Deși s-au făcut progrese remarcabile în cartografierea computațională a cazurilor dificile (candidați Lychrel și palindroame întârziate), **nu s-a dovedit încă dacă vreun număr este cu adevărat Lychrel** în baza 10. Cel mai mic candidat, 196, a rezistat testului a sute de milioane (poate miliarde) de iterații

fără a produce un palindrom – dar lipsa evidenței nu este evidența lipsei, astfel că 196 rămâne "nevinovat până la proba contrarie" din perspectiva matematică.

Lucrarea de față a prezentat o analiză detaliată a comportamentului algoritmului de inversare și adunare pentru baza 10, incluzând implementarea unui cadru de testare paralelizat și stocare a datelor, precum și interpretarea statistică și structurală a rezultatelor. Am evidențiat fenomenul convergenței majoritare rapide și existența unor fire izolate de traiectorii care par a continua indefinit. Am trecut în revistă recordurile cunoscute de întârziere, care împing limitele până la aproape 300 de iterații necesare – fără a găsi încă un capăt. Am discutat conceptul de threads/seeds/kin, crucial pentru înțelegerea *mulțimii de Mandelbrot* a acestei probleme (dacă ne putem permite analogia): câteva "puncte" generatoare (seeds) și un evantai bogat de "orbite" numerice care izvorăsc din ele.

Direcții viitoare în cercetarea numerică:

- **Explorarea altor baze:** Problema Lychrel nu este restrânsă la baza 10. În baze de numerație diferite, comportamentul poate varia mult. De exemplu, în baza 2 (binar), se cunoaște că 22 în zecimal (10110 în binar) este un număr Lychrel (nu produce palindrom în baza 2) , ceea ce arată că în anumite sisteme, Lychrel există clar. **(Din punct de vedere computațional și experimental, sunt acceptați ca Lychrel, chiar dacă lipsesc dovezile formale.)** În baza 10 nu avem încă certitudini, dar studierea altor baze poate oferi idei. Ar fi interesant de cercetat, de pildă, care e cel mai mic Lychrel în baza 10 dacă există, comparativ cu cei din baza 2 sau 8. Baze exotice (12, 16) au fost și ele studiate empiric de amatori; extinderea implementărilor pentru baze diferite ar fi un pas următor firesc. OEIS conține liste de candidați Lychrel și pentru alte baze (VanLandingham a explorat baza 2–20 parțial).

- **Vizualizări inverse:** Așa cum am construit grafuri de la număr la palindrom, ne-am putea uita și *invers*: pornind de la palindroame și văzând ce numere duc la ele. Cu alte cuvinte, putem considera palindromul final ca rădăcină și extinde "arborele invers" al tuturor precursorilor posibili (ținând cont de eventuale transporturi, un palindrom poate avea mai mulți predecesori). O astfel de vizualizare inversă ar arăta clar confluența firelor. De exemplu, pornind invers de la 8813200023188, ar trebui să putem recupera toți strămoșii: 8813200023188 poate proveni din? (în principiu decompozarea inversă nu e unică din cauza transporturilor, dar putem explora backtracking). Un algoritm inteligent de backtracking ar putea genera toți kin-numbers pentru un palindrom dat. Acest proces invers ar fi interesant de vizualizat: practic am vedea cum dintrun singur palindrom pornește un evantai de numere ce duc la el. Poate am descoperi astfel *toți* precursorii pentru un atractor palindromic cunoscut și i-am putea categorisi.

- **Optimizări computaționale noi:** Chiar dacă brute force-ul nu poate rezolva definitiv problema, fiecare extensie ne împinge cunoașterea mai departe. Inițiative de tip **crowd-computing** (asimilabile proiectelor BOINC) ar putea fi pornite pentru problema 196 – similar cum există proiecte pentru numere prime mari sau probleme de acoperire. Până acum eforturile au fost individuale; o mobilizare colectivă ar putea atinge repede praguri mult mai înalte. De asemenea, folosirea algoritmilor multi-thread optimizați pe GPU (poate analog al p196_mpi, dar pe procesoare grafice) ar putea accelera căutările.

- **Abordări cu** machine learning: Așa cum am discutat, ML ar putea juca un rol în ghidarea căutării viitoare. Pe măsură ce datele devin tot mai multe, ar fi păcat să nu le folosim pentru a antrena sisteme de inteligență artificială care să recunoască tipare. Poate un model va reuși să identifice "invariantul ascuns" al firului 196 pe care noi nu-l vedem.

Bibliografie:

1. VanLandingham, Wade. **„196 and Other Lychrel Numbers"**, p196.org (archivat 2016). [Site personal dedicat problemei 196 și numerelor Lychrel, conține discuții despre *seeds* și rezultate colaborative] .

2. OEIS A023108. **“Numbers that have not been observed to form a palindrome (Lychrel numbers)”**, The On-Line Encyclopedia of Integer Sequences, ed. N. J. A. Sloane .
3. OEIS A281506. **“First 108864 delayed palindromes with 261-step delay”**, contribuit de Andrey S. Shchebetov (2017) .
4. OEIS A326414. **“Delayed palindromes with 288-step delay (19353600 terms known)”**, contribuit de Anton Stefanov (2021) [46] .
5. O’Bryant, Kevin. **Discuție MathOverflow: “Status of the 196 conjecture?”** (26 dec 2012) .
6. Wikipedia. **„Lychrel number”**, *Wikipedia, The Free Encyclopedia*.
7. Weisstein, Eric W. **„196-Algorithm”**, *MathWorld* – Wolfram Web Resource .
8. Doucette, Jason. **„196 Palindrome Quest – Most Delayed Palindromic Number”**, jasondoucette.com (2005).
9. Nishiyama, Yutaka. **„Numerical Palindromes and the 196 Problem”**, *International Journal of Pure and Applied Mathematics*, vol. 80, nr. 3 (2012), pp. 369-377 .
10. Maslov, Dmitry. **„293 iterations delay – new world record (December 2021)”**, dmaslov.me (2021) .
11. Świerczewski, Łukasz; Dolbeau, Romain. **„The p196_mpi Implementation of the Reverse-AndAdd Algorithm for the Palindrome Quest”**, poster, International Supercomputing Conference (ISC’14), 23 iunie 2014 [56] .
12. *Cod sursă propriu:* Modulele `196.py` , `database196.py` , ș.a., – implementare Python multi-proces pentru testarea numerelor Lychrel (vezi fragmente incluse).

Lychrel number - Wikipedia
https://en.wikipedia.org/wiki/Lychrel_number

196-Algorithm -- from Wolfram MathWorld

https://mathworld.wolfram.com/196-Algorithm.html

Lychrel Number
https://almerja.com/more.php?idm=140998

196 and Other Lychrel Numbers

https://p196.org/html/identifying_lychrels.html

Reversal-Addition Palindrome Test on 10911

https://jasondoucette.com/pal/10911

Palindromes and v-Palindromes
https://nagoya.repo.nii.ac.jp/record/2003895/files/k14264_thesis.pdf