ALGORITMI PARALELI ȘI DISTRIBUIȚI
Tema #1 Agregator de știri

Responsabili: Ion Dorinel FILIP, George Alexandru TUDOR,
Alexandra Ana-Maria ION

Termen de predare:
07 Decembrie 2025 - 23:59 (soft)
10 Decembrie 2025 - 23:59 (hard)

Cuprins

1 Cerință

Se cere implementarea unui program paralel în **Java**, care să simuleze funcționarea unui agregator de știri. Programul va procesa un volum mare de articole, le va organiza pe categorii și limbi, și va genera statistici și rapoarte agregate. Rezultatul final va fi un set de fișiere text, care să reflecte într-o manieră structurată și deterministă informațiile extrase din articolele de intrare.

2 Context, obiective și motivație

Volumul de informații disponibil online este foarte mare, fiind generat permanent de rețele sociale, site-uri de presă și diverse platforme digitale. Dacă accesul la aceste date este mai facil ca niciodată, abundența lor aduce și dificultăți: conținutul este adesea repetitiv, greu de organizat și dificil de urmărit într-o formă coerentă. **Agregatoarele de știri oferă o soluție la această provocare.** Ele colectează articole din surse variate și le prezintă într-un format structurat, adaptat nevoii utilizatorilor de a parcurge rapid subiectele relevante. Totodată, pot evidenția teme de interes major, cum ar fi sănătatea publică sau politica.

Tema urmărește reproducerea, la o scară mai mică, a funcționalității unui astfel de agregator. Scopul este de a ilustra cum poate fi procesat în paralel un volum mare de articole, cum pot fi grupate în funcție de categorii și cum pot fi generate rapoarte clare și consecvente, într-un mod similar cu aplicațiile reale care sprijină accesul organizat la informație.

În urma rezolvării acestei teme studenții:

- Vor exersa abilitățile de programare folosind Java Threads;
- Vor exersa descopunerea unei probleme descrise în limbaj natural în subprobleme care pot fi executate în paralel;
- Vor exersa procesul decizional pentru identificarea unei soluții paralele scalabile, prin abordarea problemei propuse.

3 Detalii tehnice

Pentru implementarea temei se va folosi un program scris în limbajul de programare **Java**. Scopul principal este procesarea în paralel a unui set de articole de știri aflate în fișiere `.json`.

Structura articolelor

Fiecare fișier conține **mai multe articole**, reprezentate sub formă JSON, cu mai multe câmpuri. Pentru această temă ne vom folosi de valorile cheilor **uuid**, **title**, **author**, **url**, **text**, **published**, **language** și **categories**. La parcurgerea fișierelor, thread-urile trebuie să extragă aceste câmpuri. Aceste informații vor sta la baza tuturor analizelor ulterioare.

Atenție! Abordarea aleasă trebuie să fie una suficient de eficientă în ceea ce privește utilizarea memoriei astfel încât să poată rula pe mediul de test.

Procesarea în paralel

Un obiectiv central al temei este utilizarea programării paralele pentru a lucra cu un volum mare de articole. Rămâne la latitudinea voastră să decideți ce părți ale procesării pot fi desfășurate în paralel și cum poate fi împărțită eficient munca între thread-uri. Rezultatele acestei alegeri vor fi evaluate în funcție de accelerația obținută.

Important este ca programul să folosească un **set fix de thread-uri, create o singură dată și toate în același moment**, la începutul execuției. Aceste thread-uri trebuie să colaboreze pentru atingerea scopului temei, iar orice alegere de paralelizare trebuie să respecte această regulă și să conducă la rezultate corecte.

Atenție! Încălcarea acestei reguli (pornirea mai multor grupuri succesive de thread-uri, de exemplu cu apeluri repetate de `thread.start()`) va duce la acordarea punctajului **0** pentru întreaga temă. Tot punctaj 0 se acordă și dacă nu este respectat numărul de thread-uri din enunț [1].

Eliminarea duplicatelor

Un pas necesar al procesării îl reprezintă eliminarea articolelor duplicate. Două articole sunt considerate duplicate dacă au același **uuid** sau același **title**. În astfel de cazuri, **nu se va păstra niciun articol care este un duplicat**.

Organizarea articolelor pe categorii

Articolele vor fi organizate pe categorii conform unei liste explicite furnizate în fișierul `categories.txt`. Doar categoriile prezente în acest fișier vor fi luate în considerare. Pentru fiecare categorie validă se va genera un fișier de ieșire care conține, pe câte o linie, identificatorul `uuid` al fiecărui articol încadrat în acea categorie. Dacă un articol aparține mai multor categorii valide, `uuid`-ul său va apărea în toate fișierele corespunzătoare. Articolele care au categorii în afara listei din `categories.txt` nu vor fi scrise în fișierele pe categorii.

Numele fișierelor de ieșire pentru categorii se vor construi pornind de la denumirea categoriei din fișierul `categories.txt`, aplicând următoarea normalizare: se elimină toate *virgulele* din nume, iar secvențele de spații se înlocuiesc cu caracterul de subliniere `_`. De exemplu, categoria `Economy, Business and Finance` va genera fișierul `Economy_Business_and_Finance.txt`.

`Uuid`-urile dintr-un fișier vor fi ordonate lexicografic, iar dacă în lista de articole nu apar articole pentru o categorie menționată în `categories.txt`, atunci nu va fi generat un fișier pentru categoria respectivă.

Organizarea articolelor pe limbi

În mod analog, articolele vor fi grupate după limbă pe baza unei liste explicite furnizate în `languages.txt`. Doar limbile prezente în `languages.txt` vor fi considerate la generarea ieșirilor, iar articolele cu valori ale câmpului `language` care nu se regăsesc în această listă vor fi ignorate în această etapă.

Pentru fiecare limbă validă se va crea un fișier de ieșire denumit exact după intrarea corespunzătoare din `languages.txt`, urmată de extensia `.txt` (de exemplu, `english.txt`). Fiecare astfel de fișier va conține lista `uuid`-urilor articolelor scrise în limba respectivă, câte unul pe linie, ordonate lexicografic după `uuid`. Dacă în lista de articole nu apar articole pentru o limbă menționată în `languages.txt`, atunci nu va fi generat un fișier pentru limba respectivă.

Generearea fișierului global cu articole

Pe lângă fișierele pentru clasificarea pe categorii și pe limbi, se va genera și un fișier `all_articles.txt`, care va conține toate articolele procesate. În acest fișier, fiecare linie va conține câmpurile `uuid` și câmpul `published` (data publicării).

Articolele trebuie ordonate *strict* cronologic **descrescător** după `published`. Pentru egalități perfecte de timp, ordinea se stabilește lexicografic după `uuid`. Fiecare linie din fișierul global are exact forma:

```
<uuid> <YYYY-MM-DDTHH:MM:SSZ>
```

[1] Se iau în calcul doar thread-urile create de programator, iar thread-ul main nu va executa instrucțiuni în paralel cu celelalte thread-uri.

```
f87cf602db8f1e57a688afac03635e40dd217637 2024-01-02T14:16:00Z
```

Formatul câmpului `published`

Câmpul `published` este furnizat **direct în UTC**, în format ISO 8601 **unic**:

`YYYY-MM-DDTHH:MM:SSZ`

adică dată și oră cu componente pe două cifre, separator `T`, fără fracțiuni de secundă, urmate de sufixul `Z` (UTC).

Astfel, nu este necesară nici o normalizare a datei (nu există offset-uri sau fracțiuni). Ordonarea cronologică poate fi realizată *direct lexicografic* pe șirurile `published` (deoarece ISO 8601 în UTC este compatibil cu ordinea temporală), iar la egalitate se folosește ordonarea lexicografică a `uuid`.

Cuvinte de interes

O altă analiză verifică în câte articole scrise în limba engleză (`english`) apar, **în textul articolului**, anumite cuvinte de interes. Din această analiză, vor fi excluse cuvintele menționate (câte un cuvânt pe linie) în fișierul `english_linking_words.txt`, luându-se în calcul toate celelalte.

Reguli de prelucrare

- Pentru fiecare cuvânt se determină **numărul de articole distincte** în care acesta apare măcar o dată.
- Rezultatele sunt sortate **descrescător după numărul de articole**, iar la egalitate **lexicografic** după cuvânt.

Se va crea un fișier `keywords_count.txt` ce conține lista cuvintelor din articolele scrise în limba `english` și numărul de articole în care apare fiecare, pe câte o linie în format `<cuvânt> <count>`, sortată conform regulilor de mai sus.

Procesarea textului urmează trei etape:

- Tot textul este convertit la **litere mici**.
- Textul se împarte în cuvinte folosind **spațiul** ca delimitator.
- Din fiecare cuvânt se elimină **orice caracter care nu este literă**.

De exemplu, cuvântul `CPU-bounded` devine `cpubounded`.

Rapoarte statistice

Alte rapoarte statistice se vor scrie într-un singur fișier numit `reports.txt`, câte o statistică pe linie, în format text simplu. Ordinea liniilor trebuie să fie **cea specificată mai jos**.

- `duplicates_found - <count>`
  Numărul total de articole eliminate ca duplicate (după `uuid` sau `title`).
- `unique_articles - <count>`
  Numărul de articole rămase după eliminarea duplicatelor (cele folosite în ieșirile finale).
- `best_author - <nume_autor> <count>`
  Autorul cu cele mai multe articole; la egalitate se alege alfabetic.
- `top_language - <limba> <count>`
  Limba în care au fost scrise cele mai multe articole; la egalitate se alege alfabetic.

- `top_category - <categorie_normalizata> <count>`
  Categoria cu cele mai multe articole; la egalitate se alege alfabetic.
- `most_recent_article - <YYYY-MM-DDTHH:MM:SSZ> <url>`
  Cea mai recentă publicare; la egalitate se alege articolul cu `uuid` mai mic (ordonare lexicografică).
- `top_keyword_en - <cuvant> <count>`
  Cuvântul de interes care apare în cele mai multe articole; la egalitate se alege lexicografic. Pentru această statistică se vor lua în considerare doar cuvintele din articolele scrise în limba engleză.

Structura programului

Programul va trebui să fie organizat într-o succesiune de etape, pornind de la citirea datelor de intrare și terminând cu generarea fișierelor de ieșire. Modul concret în care aceste etape sunt împărțite între faze secvențiale și faze paralele rămâne la latitudinea voastră. Este important ca designul ales să asigure atât corectitudinea rezultatelor, cât și un nivel bun de performanță.

Privind în ansamblu, execuția programului presupune: primirea argumentelor din linia de comandă, pornirea thread-urilor, ingestia informațiilor despre articole, aplicarea pașilor de analiză (eliminarea duplicatelor, organizarea pe categorii și limbi, generarea rapoartelor și a statisticilor suplimentare) și, în final, scrierea fișierelor de ieșire. Ordinea și modul exact de implementare al acestor pași trebuie proiectate de fiecare în parte, cu respectarea riguroasă a cerinței privind **folosirea unui set fix de thread-uri**.

4 Compilare și execuție

Arhiva temei va cuprinde fișierele sursă, fișierul `README.pdf` (alcătuit conform indicațiilor din Secțiunea 6 - Documentație) și un fișier `Makefile` care va expune:

- O regulă **build** care va compila sursele (dacă este cazul) va aduce rezultatele în folderul curent;
- O regulă **run** care va primi și parametrii în linie de comandă aferenți rulării și va executa programul.

Pentru compilare puteți folosi direct compilatorul de Java (**javac**) sau Maven [2] (folosind comanda **mvn**). În cazul în care folosiți Maven, în arhivă puteți include și un fișier **pom.xml**.

Soluția trimisă va fi compilată folosind OpenJDK-25[3] și testată de către echipa de asistenți într-un mediu Linux ce are alocate 4 CPU-uri și 8 GB de memorie RAM.

Atenție! Singurele biblioteci externe folosite vor fi cele pentru parsarea de JSON-uri (ex. Jackson[4]).

Main-ul va fi inclus în clasa publică `Tema1`, iar programul se va rula în felul următor:

```
java Tema1 <numar_threads> <fisier_articole> <fisier_suplimentar>
```

Primul argument reprezintă numărul de thread-uri ce vor fi create la începutul execuției. Al doilea argument este un fișier text ce descrie setul de articole care trebuie procesate. Al treilea argument este un fișier ce descrie toate fișierele auxiliare necesare pentru etapele de analiză.

Fișierul cu articole are următorul format:

```
numar_fisiere_de_procesat
fisier1
fisier2
...
fisierN
```

[2] https://maven.apache.org
[3] https://openjdk.org/projects/jdk/25/
[4] https://github.com/FasterXML/jackson

Prima linie conține un număr întreg N ce reprezintă câte fișiere cu articole trebuie procesate. Următoarele N linii conțin căile relative către fișierele individuale (`articles_1.json`, ..., `articles_N.json`). Fișierul de intrare cu fișiere auxiliare are următorul format:

```
3
./path/languages.txt
./path/categories.txt
./path/english_linking_words.txt
```

Cele 3 linii conțin căile către următoarele fișiere:

- `languages.txt` - lista completă a limbilor ce trebuie luate în considerare. Fiecare linie conține exact un nume de limbă (ex.: `english`, `french`, ...). Prima linie conține numărul de linii.
- `categories.txt` - lista completă a categoriilor de interes. Fiecare linie conține numele unei categorii. Prima linie conține numărul de categorii.
- `english_linking_words.txt` - lista de cuvinte care trebuie ignorate la căutarea cuvintelor de interes în textul articolelor ce au `language = english`. Fiecare linie conține un singur cuvânt. Prima linie conține numărul de linii.

5 Date de intrare

Pentru această temă, articolele de știri utilizate în fișierele de intrare provin din **Webhose / Webz.io Free News Datasets**, un set de date public și gratuit, disponibil online. Acest dataset conține colecții săptămânale de articole de știri în format JSON, împreună cu metadatele asociate (precum data publicării, autor, limbă și categorii).

Setul de date este întreținut de Webz.io și este distribuit în scopuri educaționale și de cercetare. Pentru detalii și descărcare, resursa completă poate fi accesată la:

- https://github.com/Webhose/free-news-datasets

Pentru această temă, un singur fișier de intrare poate conține mai multe articole.

6 Documentație

Pe lângă implementarea propriu-zisă, trebuie să furnizați un **document PDF** denumit `README.pdf`, care va conține atât descrierea tehnică a implementării, cât și o analiză completă a performanței și a scalabilității programului.

Acest fișier trebuie să fie structurat conform secțiunilor indicate mai jos.

1. Secțiunea 1 - Feedback

Această secțiune este destinată feedback-ului personal cu privire la cerința și implementarea temei. Spre exemplu ce a-ți fi îmbunătățit, ce v-a plăcut, cât a durat implementarea temei. *Secțiunea aceasta nu are o anumită lungime minimă, scrieți cât credeți de cuviință.*

2. Secțiunea 2 - Strategia de paralelizare

- Explicarea modului în care este împărțită munca între thread-uri (eventual pe etapele de procesare);
- Descrierea mecanismelor de sincronizare folosite (ex.: `synchronized`, `ConcurrentHashMap` etc.);
- Argumentați de ce designul vostru este corect și eficient pentru procesarea paralelă.

3. Secțiunea 3 - Analiza de performanță și scalabilitate

Această secțiune trebuie să demonstreze cum scalează programul odată cu creșterea numărului de thread-uri. Efectuați testele cu un set de date reprezentativ, suficient de mare pentru o analiză precisă.

- **Setup de testare:**
  - Configurația sistemului (CPU, număr de nuclee, RAM, sistem de operare) pe care s-a realizat testarea;
  - Versiunea de Java folosită;
  - Dimensiunea datasetului de test.
- **Rezultate:**
  - Tabel cu timpii de execuție (în secunde) pentru fiecare număr de thread-uri;
  - Calculați speedup-ul $S(p) = \frac{T(1)}{T(p)}$ și eficiența $E(p) = \frac{S(p)}{p}$;
  - Includeți cel puțin un grafic (timp execuție sau speedup).
- **Analiză și concluzii:**
  - Explicați comportamentul observat: unde crește performanța și unde se stabilizează;
  - Identificați cauzele posibile ale limitărilor (ex. overhead de sincronizare, dimensiunea datasetului);
  - Menționați numărul optim de thread-uri pentru sistemul vostru.

În generarea rezultatelor și concluziilor din această secțiune se va aplica **cel puțin** următoarea metolodogie:

- Se va alege un set de date de intrare relevant (care va fi și specificat în documentație);
- Se vor măsura timpii de execuție pentru `1, 2, 3, ...` fire de execuție (thread-uri);
- Se va repeta fiecare configurație de rulare de cel puțin 3 ori și se vor nota timpii de execuție obținuți și media acestora, pentru fiecare dintre configurații.

Predarea documentului `README.pdf` este **obligatorie**. Lipsa acestuia sau absența secțiunii de analiză a performanței va duce la **punctaj 0** pentru temă.

7 Exemplu de rulare

Input

În această secțiune este prezentat un exemplu simplu, denumit test_small, inclus în scheletul temei. Acest test nu contribuie la punctaj; el este oferit pentru a vă verifica rapid implementarea pe inputuri mici și pentru a depista eventualele erori înainte de rulările pe seturi mari de date. Pentru rularea automată, puteți rula: `./checker.sh test_small`.

Comandă:

```
java Tema1 4 aticles.txt inputs.txt
```

(`articles.txt`):

```
2
./articles/articles_small_0.json
./articles/articles_small_1.json
```

(inputs.txt):

```
3
./languages.txt
./categories.txt
./english_linking_words.txt
```

languages.txt:

```
3
english
french
portuguese
```

categories.txt:

```
6
Arts, Culture and Entertainment
Economy, Business and Finance
Health
Human Interest
Science and Technology
Weather
```

english_linking_words.txt:

```
140
a
about
accordingly
additionally
after
afterward
...
```

articles_small_0.json

```
[
  {
    "uuid": "a-001",
    "title": "Alpha",
    "author": "Alice",
    "url": "http://example.com/a-001",
    "text": "Apple dances with zebra today; they're swift, curious, playful,
    and genuinely surprising under bright circuits and soft melodies.",
    "published": "2025-01-01T10:00:00Z",
    "language": "english",
    "categories": ["Science and Technology"]
  },
  {
    "uuid": "a-002",
    "title": "Beta",
    "author": "Bob",
    "url": "http://example.com/a-002",
    "text": "Banana markets surge worldwide; we're noticing fresh trends,
    diverse flavors, and delightful morning smoothies everywhere today
    beyond headlines.",
```

```
    "published": "2025-01-02T10:00:00Z",
    "language": "english",
    "categories": ["Economy, Business and Finance"]
  },
  {
    "uuid": "a-003",
    "title": "Gamma",
    "author": "Carol",
    "url": "http://example.com/a-003",
    "text": "Cherry blends with banana and grapes; don't forget fiber.
    Read more at http://example.com/health for tips and science today.",
    "published": "2025-01-03T10:00:00Z",
    "language": "english",
    "categories": ["Health"]
  }
]
```

articles_small_1.json

```
[
  {
    "uuid": "a-004",
    "title": "Beta",
    "author": "Dave",
    "url": "http://example.com/a-004",
    "text": "Banana and apple collide on the field; they're quick,
    resilient, creative, and wildly entertaining to watch this
    evening under lights.",
    "published": "2025-01-04T10:00:00Z",
    "language": "english",
    "categories": ["Sport"]
  },
  {
    "uuid": "a-005",
    "title": "Lisboa Economia",
    "author": "Eva",
    "url": "http://example.com/a-005",
    "text": "Mercado portugues cresce; inovacao acelera, empresas estao
    confiantes e consumidores atentos, enquanto tendencias mudam
    rapidamente nesta semana dinamica.",
    "published": "2025-01-05T10:00:00Z",
    "language": "portuguese",
    "categories": ["Economy, Business and Finance"]
  }
]
```

Atenție! În exemplul de mai sus sunt prezentate doar câmpurile strict necesare pentru procesare.

Output

Eliminarea duplicatelor

Al doilea articol din primul fișier și primul articol din al doilea fișier au același titlu (`title`), astfel sunt considerate duplicate și se elimină ambele. Rămân **3** articole unice.

Fișierul global (`all_articles.txt`):

```
a-005 2025-01-05T10:00:00Z
a-003 2025-01-03T10:00:00Z
a-001 2025-01-01T10:00:00Z
```

Fișiere pe categorii

```
# Economy_Business_and_Finance.txt
a-006

# Health.txt
a-003

# Science_and_Technology.txt
a-001
```

Fișiere pe limbi

```
#english.txt
a-001
a-003

# portuguese.txt
a-005
```

Cuvinte de interes (`keywords_count.txt`):

```
today 2
apple 1
banana 1
blends 1
bright 1
cherry 1
circuits 1
...
```

Rapoarte (`reports.txt`):

```
duplicates_found - 2
unique_articles - 3
best_author - Alice 1
top_language - english 2
top_category - Economy_Business_and_Finance 1
most_recent_article - 2025-01-05T10:00:00Z http://example.com/a-005
top_keyword_en - today 2
```

8 Trimitere și notare

Tema se poate testa local, direct pe Linux/WSL, după cum se explică mai jos. Tema se va încărca pe Moodle, sub formă de arhivă ZIP care va conține **în rădăcina arhivei**:

- Rezolvarea temei sub forma de fișiere sursă Java;
- *Makefile* cu 3 directive:
  - **build**: compilează sursele proiectului și produce un fișier executabil/clasă principală (ex. `Tema1.class`).
  - **run**: rulează aplicația folosind artefactul generat la build (ex. `java Tema1.class $(ARGS)`) și acceptă argumente din linia de comandă (ex. `make run ARGS="..."`).
  - **clean**: șterge artefactele generate (ex. fișiere `.class`), readucând proiectul la o stare curată.
- *README* - **document PDF obligatoriu**.

Atenție! În arhivă **nu** va fi inclus niciun fișier compilat și nici părți din scriptul de testare.

Punctajul este divizat după cum urmează:

- **45p** - scalabilitatea soluției [5]
- **30p** - corectitudinea rezultatelor (la rulări multiple pe aceleași date de intrare, trebuie obținute aceleași rezultate)
- **25p** - claritatea codului și a explicațiilor din README.

Nerespectarea următoarelor cerințe va duce la depunctări:

- **-45p** – încetinirea intenționată a programului sau folosirea unor tehnici care afectează în mod artificial timpul de execuție, prin mijloace precum cele din **exemplele** următoare (**nu este o listă exhaustivă**):
  - **Parcurgerea fișierelor se va face o singură dată** - citirea sau procesarea repetată a acelorași fișiere nu este acceptată;
  - **Orice formă de cod care nu contribuie la rezolvarea efectivă a problemei, dar influențează timpul de execuție**;
  - Repetarea inutilă a aceleiași operații;
  - Detectarea contextului de execuție (verificări pe variabile de mediu, working directory sau dacă rulează în checker);
  - Exploatarea ordinii de rulare a checker-ului (secvențial → 2 threads → 4 threads) pentru a pre-popula fișiere temporare;
  - Folosirea unor structuri de date sau algoritmi diferiți în funcție de numărul de thread-uri (ex: HashMap doar la rulările paralele, liste la cea secvențială);
  - Hardcodarea sau copierea rezultatelor din fișierele de referință;
  - Citire în buffere mici sau sincronizare excesivă aplicată selectiv;
  - Alocări de memorie redundante sau logging excesiv doar în modul secvențial;
  - Adăugarea de cod care introduce delay-uri artificiale.
- **-100p** - pseudo-sincronizarea firelor de execuție prin funcții cum ar fi sleep sau **busy waiting**

[5] Acest punctaj este condiționat de corectitudine. O soluție paralelă, chiar dacă are speed-up bun, nu va fi punctată în cazul în care rezultatele sunt greșite.

- **-100p** - crearea și oprirea de thread-uri în mod repetat (pentru a nu lua această depunctare, trebuie să creați un singur set cu numărul de thread-uri primite ca input la început, așa cum este dat în argumentele programului)
- **-100p** - pornirea/utilizarea unui număr de thread-uri decât cel indicat în enunț;
- **-100p - Lipsa README-ului**, lipsa secțiunii de analiză de performanță și scalabilitate sau README superficial

Atenție! O temă care nu compilează sau care nu trece niciun test pe mediul de testare nu va fi notată.
Atenție! Checker-ul vă dă cele 75 de puncte pentru scalabilitate și corectitudinea rezultatelor, dar acela nu este punctajul final (dacă, de exemplu, temă vă scalează și dă rezultate corecte, dar nu ați respectat regulile impuse, veți avea nota finală 0).

9 Testare

Pentru a vă putea testa tema, găsiți pe site-ul cursului un set de fișiere de intrare de test, precum și un script Bash (numit *checker.sh*) pe care îl puteți rula pentru a vă verifica implementarea. Acest script va fi folosit și pentru testarea finală.
Pentru a putea rula scriptul trebuie să aveți următoarea structură de fișiere:

```
$ tree
.
├── checker
│   ├── input
│   │   ├── articles
│   │   │   ├── [...] (articolele folosite in teste)
│   │   ├── files
│   │   │   ├── categories.txt
│   │   │   ├── languages.txt
│   │   │   ├── english_linking_words.txt
│   │   ├── tests
│   │   │   ├── test_1
│   │   │   │   ├── articles.txt
│   │   │   │   ├── inputs.txt
│   │   │   ├── [...] (alte directoare cu teste diferite)
│   │   │   ├── test_small
│   │   │   │   ├── [...] (fisierele necesare unui test de mici dimensiuni)
│   ├── output
│   │   ├── [...] (fisiere folosite de checker pentru verif. corectitudinii)
│   ├── checker.sh
├── src
│   ├── [...] (sursele voastre - Makefile, .java, etc.)
```

La rulare, scriptul *checker.sh* din directorul *checker* execută următorii pași:

1. compilează programul
2. pentru testele test_k, unde 1≤ k ≤5 execută următoarele operații:
   (a) rulează variantele cu 1/2/4 thread-uri și verifică dacă rezultatele sunt corecte
   (b) rulează variantele cu 1/2/4 thread-uri și calculează accelerația și verifică dacă este peste niște limite așteptate
3. se calculează punctajul final din cele 75 de puncte alocate testelor automate (25 de puncte fiind rezervate pentru claritatea codului și a explicațiilor, așa cum se specifică mai sus).

Dacă doriți să rulați checker-ul pentru un singur test, rulați comanda `./checker.sh test_k`, unde `test_k` este numele testului dorit.

10 Recomandări de implementare și testare

Găsiți aici o serie de recomandări legate de implementarea și testarea temei:

1. folosiți un repository **privat** de Git și dați commit-uri frecvent, pentru a vedea clar diferențele dintre iterațiile de cod și pentru a avea o copie sigură pe care o puteți recupera dacă ați șters ceva din greșeală sau dacă mașina pe care lucrați are probleme;
2. citiți documentația oficială pentru **concurența în Java** și urmăriți atent erorile/excepțiile returnate ca să depistați ușor utilizările greșite; pentru parsarea JSON folosiți o bibliotecă consacrată precum **Jackson** sau **Gson**;
3. folosiți un IDE precum **IntelliJ IDEA** sau **Visual Studio Code** cu extensiile pentru Java; pe lângă completarea automată a codului, puteți rula ușor proiectul și face debug direct din IDE.