Documentatie Tema 4

# Cerinta

La un concurs de programare participa mai multi concurenti din diferite tari. Concursul presupune rezolvarea mai multor(10) probleme intr-un interval de timp dat. Fiecare participant este identificat printr-un ID si rezultatele obtinute pentru rezolvarea fiecarei probleme sunt salvate intr-un fisier cu rezultate in care pe fiecare linie sunt inregistrari de forma (ID, Punctaj). Punctajele concurentilor dintr-o tara C1 corespunzatoare fiecarei probleme se gasesc in fisiere separate (“RezultateC1\_P1.txt”, “RezultateC1\_P2.txt, …“RezultateC1\_P10.txt). Un punctaj negativ (-1) inseamna incercare de fraudare si va conduce la eliminarea concurentului din concurs. Daca un concurent nu rezolva o problema NU se adauga in fisier o pereche cu ID –ul concurentului si punctaj 0.

Pentru a se ajunge la clasamentul final se cere sa se formeze o lista inlantuita ordonata descrescator (dupa punctaj) care contine noduri cu valori de tip pereche (ID, Punctaj). Lista este o lista ordonata (*invariant: dupa orice operatie cu lista aceasta este ordonata – inserati in lista ordonata nu sortati dupa fiecare operație*).

Se porneste prin crearea unei liste inlantuita vida si se adauga noduri sau punctaje pe masura ce se citesc inregistrari din fisiere.

La citirea unei noi perechi (ID\_n, Punctaj\_n) se verifica daca exista deja in lista o pereche cu ID egal cu ID\_n:

* Daca exista atunci
  + Daca Punctaj\_n este pozitiv se aduna punctajul Punctaj\_n la punctajul existent in nod,
  + Daca Punctaj\_n = -1 atunci se sterge nodul gasit din lista si se adauga intr-o lista cu ID-uri de persona non-grata
    - Daca se va citi ulterior o pereche cu acelasi ID nu se va mai adauga in lista cu punctaje
* Daca nu exista atunci
  + se adauga un nou nod cu valoarea (ID\_n, punctaj\_n).

In final aceasta lista va conține clasamentul final si se va salva intr-un fisier „Clasament.txt” de catre main-thread.

Metoda A) Implementare secventiala

* Se citeste pe rand din fiecare fisier cate o pereche si se adauga in lista rezultat -L. La final se scriu elementele listei in fisierul rezultat.

Metoda B) Implementare paralela – p threaduri (nu se folosesc Executori!!!)

1. **p\_r** threaduri (readers) citesc din fisiere perechi (ID , Punctaj) si le adauga intr-o structura de date de tip coada care nu este marginita (nu are capacitate maxima). Politica de partajare a fisierelor intre readers trebuie sa asigura o incarcare de calcul balansata.

(conditie – pentru structura de tip coada NU se admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)

1. Celelalte threaduri p\_w= (p-p\_r) threaduri (workers) preiau cate o pereche din coada si o adauga in lista L.
   * Se continua operatiile 1., 2. pana cand toate perechile, din toate fisierele, sunt adaugate la lista L.
2. Dupa ce toate rezultatele au fost adaugate in lista L, primul thread (primul reader) scrie rezultatul obtinut in lista L in fisierul „Clasament.txt”.

# Structura codului

## Clasa Main

Aceasta este clasa principală, care include următoarele metode:

**generazaFisiere(int numarTari)**

* **Scop**: Creează fișiere text care conțin rezultatele concurenților pentru un număr specificat de țări.
* **Parametri**:
  + numarTari – numărul de țări pentru care se generează fișierele.
* **Funcționalitate**:
  + Creează un folder Rezultate dacă nu există.
  + Pentru fiecare țară, generează 10 fișiere, corespunzătoare problemelor unui concurs.
  + Fiecare concurent primește un ID unic și un punctaj pentru fiecare problemă, generat aleator.

**secvential(int numarTari)**

* **Scop**: Procesează fișierele generate într-un mod secvențial și creează un clasament al concurenților.
* **Parametri**:
  + numarTari – numărul de țări a căror date trebuie procesate.
* **Funcționalitate**:
  + Citește datele din fișiere și adaugă punctajele concurenților.
  + Identifică concurenții descalificați (în baza punctajului -1).
  + Creează un fișier ClasamentSecvential.csv cu rezultatele finale.

**paralel(int numarTari, int numarThreaduri, int numarReaderi)**

* **Scop**: Procesează fișierele generate în mod paralel folosind thread-uri.
* **Parametri**:
  + numarTari – numărul de țări a căror date trebuie procesate.
  + numarThreaduri – numărul total de thread-uri utilizate.
  + numarReaderi – numărul de thread-uri dedicate citirii datelor.
* **Funcționalitate**:
  + Cititorii citesc fișierele și adaugă datele într-o coadă sincronizată (ThreadQueue).
  + Consumatorii procesează datele din coadă și actualizează clasamentul folosind ThreadLinkedList.
  + Creează un fișier ClasamentParalel.csv cu rezultatele finale.

## Clasa ThreadLinkedList

* **Descriere**: Gestionează clasamentul concurenților într-un mod sincronizat.
* **Funcționalitate**:
  + Metoda add(Pair pair) – adaugă un punctaj pentru un concurent, marcându-l ca descalificat dacă punctajul este -1.
  + Metoda clasament() – returnează clasamentul sub forma unui set ordonat de perechi (ID și punctaj).

## Clasa ThreadQueue

* **Descriere**: Gestionează o coadă sincronizată de perechi (ID concurent și punctaj).
* **Funcționalitate**:
  + Metoda produce(Pair pair) – adaugă o pereche în coadă.
  + Metoda consume() – extrage o pereche din coadă.
  + Metoda hasValue() – verifică dacă mai există date de procesat.
  + Metoda finish() – marchează finalizarea citirii datelor.

## Clasa Pair

* **Descriere**: Reprezintă o pereche de valori (ID concurent și punctaj).
* **Funcționalitate**:
  + Implementă metoda compareTo() pentru a permite ordonarea descrescătoare după punctaj.
  + Override pentru metodele equals(), hashCode() și toString().

# Timpi

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Timp** | | | | |
| Secvential | 72 | | | | |
|  | **Numar threaduri totale** | | | | |
|  | **4** | **6** | **8** | **16** |  |
| **P\_r=1** | 76 | 79 | 82 | 112 |  |
| **P\_r=2** | 91 | 118 | 131 | 104 |  |