Documentatie Laboratorul 4

Nita Dan-Lucian gr. 239

1. Cerinta:

Se considera **n** polinoame reprezentate prin lista de monoame.
Se cere adunarea polinoamelor folosind o implementare multithreading (p threaduri).

Consideratii generale:
- reprezentarea unui polinom in memorie: lista inlantuita (1 nod=1monom) ordonata dupa exponentii monoamelor cu urmatorul INVARIANT (predicat adevarat la orice moment al executiei) de reprezentare:
    -monoamele sunt ordonate dupa exponenti
    -nu se pasteaza in lista monoame cu coeficient 0;
    - nu exista doua noduri (monoame) cu acelasi exponent
- polinoamele se citesc din fisiere – cate un fisier pentru fiecare polinom;
- un fisier contine informatii de tip (coeficient, exponent) pentru fiecare monom al unui polinom,
- fisierele input se creeaza prin generare de numere aleatoare.

(Conditie: fisierele nu contin monoame cu coeficient egal cu 0!)

Rezolvare:
Se porneste prin crearea unei liste inlantuita - L corespunzatoare unui polinom nul.
In final aceasta lista va continue polinomul rezultat.

Metoda A) Implementare secventiala
- Se citeste pe rand din fiecare fisier cate un monom si se adauga in lista rezultat -L (atentie – invariantul trebuie sa ramana adevarat dupa fiecare adaugare de monom).

Metoda B) Implementare paralela – p threaduri

1. Primul thread citeste cate un monom si il adauga intr-o structura de date de tip coada. (conditie – pentru structura de tip coada NU se admite folosirea unei structuri de date pentru care partea de sincronizare este deja implementata!!!)
2. Celelalte threaduri preiau cate un monom din coada si il aduna la polinomul reprezentat in lista L.

➔ Se continua operatiile 1., 2. pana cand toate monoamele, din toate fisierele, sunt adunate la lista L.

3. Primul thread scrie rezultatul obtinut in lista L intr-un fisier rezultat (conditie: fisierul nu contine monoame cu coefficient egal cu 0)

2. Detalii de implementare

2.1 Generarea fișierelor cu numere random

Workflow-ul începe cu generarea fișierelor cu polinoame. Aceasta se face rulând clasa main din GenerateFile, care generează x fișiere cu y monoame, având gradul maxim z. Ele se vor scrie în folderul “data”. Cu ajutorul funcției de scriere de numere random, în cele x fișiere se vor scrie y linii de forma (A, B) unde a reprezintă coeficientul și B exponentul.

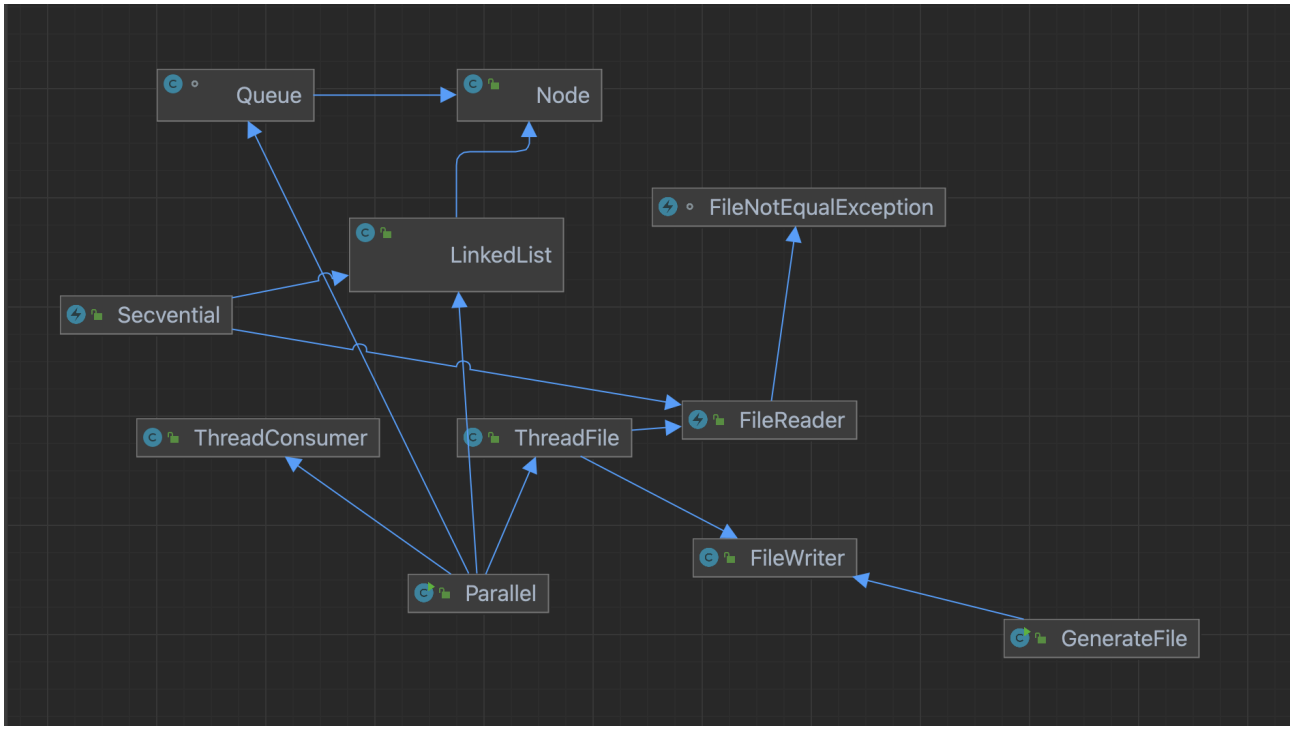
2.2 Varianta secventiala

In varianta secventiala, cream o lista simplu înlănțuită goala, folosindu-ne de clasa “LinkedList”. Pentru fiecare fișier, citim din fișier direct în lista noastră, cu ajutorul funcției readToList din clasa FileReader. Citim fiecare linie în parte și adăugăm perechea (A,B) citită în lista, folosindu-ne de funcția addNode. Aici, încercăm sa găsim un nod care are aceelasi exponent. Daca il gasim, doar actualizam coeficientul. In cazul în care nu îl găsim, găsim ultimul nod cu exponentul mai mic decat cel pe care îl căutăm, și introducem între el și nodul sau next, valoarea dorita. Astfel, unul câte unul, fiecare variabila din fiecare fișier este introdusă la locul potrivit în lista.

2.3 Varianta paralela

Spre deosebire de varianta secventiala, aici initializam o lista vida și o coada vida. Creăm apoi un array de thread-uri, primul setand-ul ca fiind threadFile, iar pe ceilalți threadConsumeri. Pornim toate thread-urile, apoi le dam join. ThreadFile, este producatorul nostru și citește numerele din fișier, apoi le adauga în coada folosind funcția readToQueue, asemănătoare cu readToList din varianta secventiala. Pentru coada, am implementat o clasa Queue, ce are doua funcții: push și pop, ce adauga, respectiv scot elemente din coada, coada blocându-se pe parcursul execuției funcțiilor cu ajutorul unui ReentrantLock. Thread-urile consumatoare încearcă sa scoată câte un element din coada. Daca reușesc, ele fac apoi Lock pe lista, încearcă sa adauge elementul la locul potrivit și apoi deblochează lista.

3. Diagrama claselor



4. Timpi de rulare

4.1. 10 polinoame cu grad max 1000 și 50 monoame

Numarul	Thread-uri	Timp
1.	s	36.4
2.	s	30.5
3.	s	30.6
4.	s	34.7
5.	s	24.8
6.	s	33.25
7.	s	26.5
8.	s	29.9
9.	s	27.9
10.	s	28.8
Timpul mediu de rulare pentru secvential este: 30.335		
1.	4	29.5
2.	4	35.9

3.	4	25.3
4.	4	30.8
5.	4	27.1
6.	4	29.6
7.	4	32.2
8.	4	33.9
9.	4	31.6
10.	4	25.4
Timpul mediu de rulare pentru 4 thread-uri este de: 30.13		
1.	6	27.95
2.	6	38.9
3.	6	34.7
4.	6	32.5
5.	6	29.5
6.	6	29.1
7.	6	26.9
8.	6	38.3
9.	6	32.1
10.	6	28.4
Timpul mediu de rulare pentru 6 thread-uri este de: 31.83		
1.	8	29.4
2.	8	28.4
3.	8	28.6
4.	8	28.9
5.	8	27.5
6.	8	27.6
7.	8	27.8
8.	8	28.9
9.	8	27.96
10.	8	31.0
Timpul mediu de rulare pentru 8 thread-uri este de: 28.6		

Concluzie pentru prima cerință: rulând pe 8 thread-uri am obținut cel mai bun rezultat, dar avem o anumită inconsistenta în timpi, rapiditatea nefiind proporțională cu nr de thread-uri ceea ce înseamnă ca am avea nevoie de un set mai mare de date și/sau rulare mai multe.

4.2. 5 polinoame cu grad max 10000 și 100 monoame

Numarul	Thread-uri	Timp
1.	s	29.6
2.	s	32.0
3.	s	44.4
4.	s	30.8
5.	s	31.6
6.	s	33.7
7.	s	30.4
8.	s	31.8
9.	s	33.5
10.	s	28.1
Timpul mediu de rulare pentru secvential este: 32.5		
1.	4	36.0
2.	4	37.7
3.	4	31.4
4.	4	34.2
5.	4	33.5
6.	4	27.7
7.	4	32.8
8.	4	34.9
9.	4	36.6
10.	4	32.2
Timpul mediu de rulare pentru 4 thread-uri este de: 33.7		
1.	6	42.5
2.	6	32.9
3.	6	48.2
4.	6	34.5

5.	6	41.1
6.	6	35.6
7.	6	31.7
8.	6	33.8
9.	6	29.8
10.	6	39.9
Timpul mediu de rulare pentru 6 thread-uri este de: 37.0		
1.	8	38.4
2.	8	31.6
3.	8	34.5
4.	8	36.9
5.	8	40.0
6.	8	38.4
7.	8	35.7
8.	8	36.3
9.	8	38.7
10.	8	38.8
Timpul mediu de rulare pentru 8 thread-uri este de: 36.9		

Concluzie pentru prima cerință: varianta secventiala a obținut cel mai bun timp. Exista posibilitatea ca implementarea sa nu fie destul de eficienta, sau ca numărul de date sa fie prea mic pentru a se vedea avantajul unei rulare paralele.