**Laborator 4**

1. **Analiza Cerintelor**

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!)

1. **Proiectare si implementare**

public class Node {  
 public int coefficient; **// coeficientul unui monom**   
 public int exponent; **// exponentul unui monom**   
  
 public Node next;  
 public Node() {  
 }  
  
 public Node(int coefficient, int exponent) {  
 this.coefficient = coefficient;  
 this.exponent = exponent;  
 }  
}

public class MyQueue {  
 Queue<Node> q**; // coada in care sunt pastrate monoamele citite de catre threadul reader**

final Lock lock = new ReentrantLock();   
 final Condition notEmpty = lock.newCondition();  
  
 public MyQueue(){  
 q = new LinkedList<>();  
 }  
 public void add(Node n){  
 lock.lock();  
 try{  
 q.add(n); **//se adauga un nou monom**   
 notEmpty.signalAll(); **// se notifica threadurile care asteptau**  
 }  
 finally {  
 lock.unlock();  
 }  
 }  
  
 public Node poll() throws InterruptedException {  
 lock.lock();  
 try {  
 while (q.isEmpty()) { **// cat timp coada este vida**   
 notEmpty.await(); **// threadurile asteapta**   
 }  
 Node node = q.remove();  
 return node;  
 }  
 finally{  
 lock.unlock();  
 }  
  
 }  
  
 public void stop(Integer noOfWorkers){  
 for (int i = 0; i < noOfWorkers; ++i) {  
 this.add(null); **// se adauga monoame null pentru a anunta threadurile workers ca s-au epuizat toate monoamele**   
  
 }  
  
 }  
  
}

public class MyList {  
 public Node sentinelHead;  
  
 MyList(){  
 sentinelHead = new Node(0,0); **// dummy node**   
 }

**// synchroniezed – lock pe metoda = lock pe obiectul de tipul MyList**  
 public synchronized void addNode(Node node){  
 boolean foundSameExponent = false;  
 Node current;  
 Node prev = sentinelHead;  
 for(current = sentinelHead.next ; current != null && node.exponent >= current.exponent; current = current.next) {  
 if(node.exponent == current.exponent){  
 foundSameExponent = true;  
 current.coefficient += node.coefficient;  
 if(current.coefficient == 0){  
 // delete the node  
 deleteNode(current);  
 break;  
 }  
 }  
 prev = current;  
 }  
 if(!foundSameExponent) {  
 node.next = prev.next;  
 prev.next = node;  
 }  
 }  
  
 private synchronized void deleteNode(Node toDelete) {  
 Node prev = sentinelHead;  
 Node current = prev;  
 while(current != toDelete){  
 prev = current;  
 current = current.next;  
 }  
 prev.next = current.next;  
 }  
  
 public List<Node> getResultSum() {  
 List<Node> result = new ArrayList<>();  
 for (Node current = sentinelHead.next; current != null; current = current.next) {  
 result.add(current);  
 }  
 return result;  
 }  
}

1. **Executie**
2. static class Producer implements Runnable{  
     
    int nrOfWorkers;  
    MyQueue myQueue;  
    int polynomialNumber;  
    public Producer(MyQueue myQueue, int nrOfWorkers, int polynomialNumber) {  
    this.myQueue = myQueue;  
    this.nrOfWorkers = nrOfWorkers;  
    this.polynomialNumber = polynomialNumber;  
    }  
     
    @Override  
    public void run() {  
    for(int nr = 0; nr < polynomialNumber; nr++) {  
    String filename = "Lab4/resources/polynom[" + nr + "].in";  
    try (BufferedReader bufferedReader = new BufferedReader(new FileReader(filename))) {  
    String line;  
    while ((line = bufferedReader.readLine()) != null) {  
    if (line.equals(""))  
    continue;  
    String[] numbers = line.split(" ");  
    Integer coef = Integer.*valueOf*(numbers[0]);  
    Integer exp = Integer.*valueOf*(numbers[1]);  
    Node node = new Node(coef, exp);  
    myQueue.add(node);  
    }  
    } catch (FileNotFoundException e) {  
    throw new RuntimeException(e);  
    } catch (IOException e) {  
    throw new RuntimeException(e);  
    }  
    }  
    myQueue.stop(nrOfWorkers); **// se adauga noduri null pentru a opri workers**  }  
    }  
   }

static class Worker implements Runnable{  
  
 MyQueue myQueue;  
 MyList result;  
  
 public Worker(MyQueue myQueue, MyList result) {  
 this.myQueue = myQueue;  
 this.result = result;  
 }  
  
 @Override  
 public void run() {  
 Node node = null;  
 try {  
 node = myQueue.poll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 while (node != null) {  
 result.addNode(node);  
 try {  
 node = myQueue.poll();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 }  
}

reader.start();  
for(int i = 0 ; i < *nrOfThreads*-1; i++) {  
 workers[i].start();  
}  
  
try {  
 reader.join();  
 for(int i = 0 ; i < *nrOfThreads*-1; i++) {  
 workers[i].join();  
 }  
} catch (InterruptedException e) {  
 throw new RuntimeException(e);  
}

1. **Cazuri de testare**

|  |  |  |
| --- | --- | --- |
| **Numar Polinoame** | **Nr Threaduri** | **Timp de executie** |
| 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame | 1 | 13.34758 |
|  | 4 | **17.35755** |
|  | 6 | **19.00652** |
|  | 8 | **19.17593** |
| 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame | 1 | **14.12514** |
|  | 4 | **15.07311** |
|  | 6 | **15.65924** |
|  | 8 | **23.04057** |

1. **Concluzii**
   1. Varianta secventiala are timp mai bun din cauza ca fiecare worker blocheaza intraga lista.