# Rezultate lab5

Enunt

Aceeasi problema ca si la Laborator 4 dar cu urmatoarele modificari:

- Dintre cele p threaduri p1 sunt cititoare si p2 sunt de tip ‘worker’ (preiau din coada si adauga in lista rezultat) – p=p1+p2

- Coada in care se adauga monoamele citite are o capacitate maxima = MAX

- Implementarea pentru producator consumator (writer/reader) trebuie sa foloseasca mecanisme de tip wait notify (nu busy waiting)

- Sincronizarea operatiile cu lista inlantuita se fac la nivel de nod nu pentru intreaga lista!

o Sincronizarile se vor face folosind variabile de tip Lock asociate cu fiecare nod.

o Daca rezulta un nod cu coeficient nul acest nod se va sterge din lista!

Observatie: Este obligatoriu sa se foloseasca sablonul ‘producator-consumator’ pentru legatura dintre threadurile ‘reader’ si cele de tip ‘worker’ --- wait|notify.

Limbaj: la alegere intre Java si C++

Analiza timpului de executie pentru urmatoarele cazuri:

1) Rezolvare secventiala

2) 10 polinoame fiecare cu gradul maxim 1000 si cu maxim 50 monoame si MAX=20

a. p = 4, 6, 8 ; si p1=2 , p1=3

b. secvential ; si p1=2 , p1=3

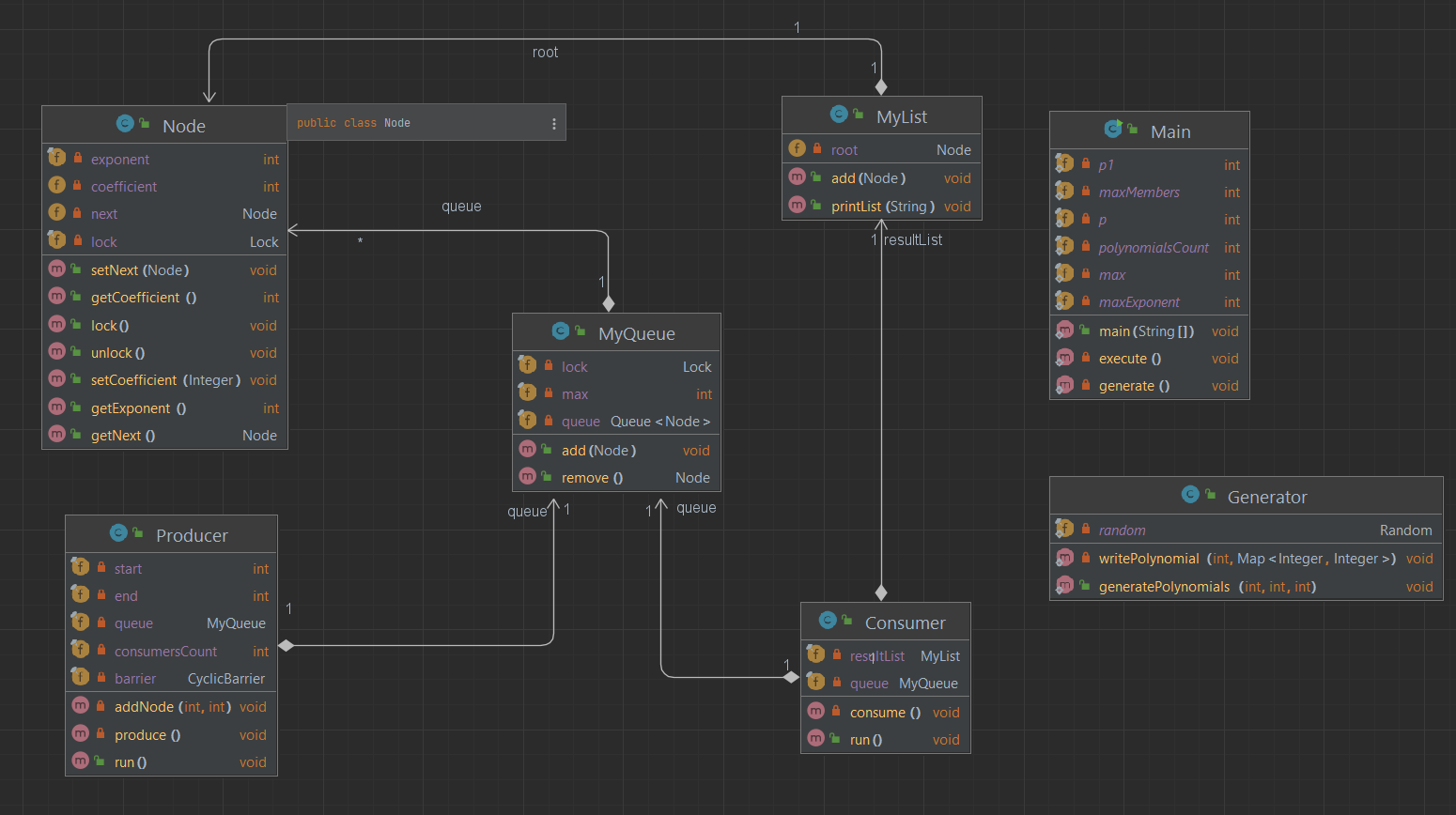
3) 5 polinoame fiecare cu gradul maxim 10000 si cu maxim 100 monoame si MAX=30

a. p = 4, 6, 8 ; si p1=2 , p1=3

b. secvential ; si p1=2 , p1=3

Analiza comparativa a performantei –Lab 4) versus Lab 5)

Diagrama de clase



Partitionarea datelor

* Cazul secvential
  + In acest caz, exista un singur thread(main) care citeste polinoamele din fisier, adauga monom cu monom in coada si la sfarsit null
  + Tot acest thread, ia element cu element din coada si il adauga in lista rezultat, pana cand intalneste null
* Cazul paralel
  + Avem doua categorii de thread-uri: producer(p1) si consumer(p – p1)
  + Producer-ii isi impart datele (cele polynomialsCount polinoame) dupa formula polynomialsCount / p1 +- 1, fiecare producer primind un start si end
  + Un producer citeste fisierele dintre start(inclusiv) si end(exclusiv) si adauga monoamele in coada; dup ace termina de procesat fisierele sala, producer-ul asteapta la o bariera; mai apoi, thread-ul cu start = 0 adauga in coada elemente cu valoarea null, atatea cat numarul de consumer-i
  + Consumer-ii iau din coada element cu element, fiecare cum prinde; in momentul in care un consumer are ca element null, executia acestuia se incheie; actiunea propriu-zisa de “consum” consta in adaugarea monomului in lista rezultat, cu calculele aferente
  + Sincronizarea in lista se face la nivel de nod; lista se parcurge cu elementul precedent si cel curent, acestea fiind locked

Cazuri de testare

* Datele au fost validate, comparand rezultatele cazului secvential(din lab 4) cu rezultatele de acum, in fiecare caz, accesand site-ul <https://text-compare.com/>

|  |  |  |  |
| --- | --- | --- | --- |
| **Caz** | **p** | **p1** | **Timp(milis)** |
| 10 polinoame  maxExponent = 1000  maxMembers = 50  MAX = 20 | 4 | 2 | 88 |
| 3 | 98 |
| 6 | 2 | 82 |
| 3 | 87 |
| 8 | 2 | 77 |
| 3 | 75 |
| secvential | 2 | 75 |
| 3 | 82 |
| 5 polinoame  maxExponent = 10000  maxMembers = 100  MAX = 30 | 4 | 2 | 78 |
| 3 | 72 |
| 6 | 2 | 67 |
| 3 | 83 |
| 8 | 2 | 72 |
| 3 | 70 |
| secvential | 2 | 85 |
| 3 | 89 |

Concluzii

* Timpii sunt putin mai mari fata de lab4, din cauza dimensiunii cozii
* Se observa ca cele mai bune cazuri se obtin cand situatia este echilibrata, adica 6 thread-uri din care 2 produceri, respectiv 8 si 3
* In cazul secvential (un singur consummator si mai multi produceri), fisierele sunt procesate mult prea repede si din coada de face remove mai rar decat se face add, ceea ce ingreuneaza procesarea