Зад. 6 (Пресмятане на Pi – Ramanujan 1)

Числото (стойността на) Pi може да бъде изчислено по различни начини. Използвайки сходящи редове, можем да сметнем стойността на Pi с произволно висока точност. Един от бързо сходящите към Pi редове е този, открит от индийския математик Srinivasa Ramanujan през 1910-1914 година. За стойността на Pi имаме:

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Реда може да бъде записан и като:

$$(1)\ Ramanujan\ 1,\ 1914$$
$$\frac{1}{\pi} = \frac{\sqrt{8}}{99^2} \sum_{n=0}^{\infty} \frac{(4n)!}{(4^n n!)^4} \frac{1103+26390n}{99^{4n}}$$

Вашата задача е да напишете програма за изчисление на числото Pi изпозлвайки цитирания ред, която използва паралелни процеси (нишки) и осигурява пресмятането на Pi със зададена от потребителя точност. Изискванията към програмата са следните:

(o) Команден параметър задава точността на пресмятанията. По Ваше желание, точността се изразява или в брой цифри след десетичната запетая или в брой членове на реда. Командният параметър задаващ точността има вида - **“-p 10240”;**

(o) Друг команден параметър задава максималния брой нишки (задачи) на които разделяме работата по пресмятането на Pi – например **“–t 1”** или **“–tasks 3”;**

(o) Програмата извежда подходящи съобщения на различните етапи от работата си, както и времето отделено за изчисление и резултата от изчислението;

Примери за подходящи съобщения:

„**Thread-<num> started.**“,

„**Thread-<num> stopped.**“,

„**Thread-<num> execution time was (millis): <num>**“,

„**Threads used in current run: <num>**“,

„**Total execution time for current run (millis): <num>**“ и т.н.;

(o) Записва резултата от работа си (стойността на Pi) във изходен файл, зададен с подходящ параметър, например **“-o result.txt”**. Ако този параметър е изпуснат, се избира име по подразбиране фиксирано отнапред във Вашата програма;

(o) Да се осигури възможност за **„quiet“** режим на работа на програмата, при който се извежда само времето отделено за изчисление на Pi, отново чрез подходящо избран друг команден параметър – например **“-q”;**

ЗАБЕЛЕЖКА:

(o) При желание за направата на подходящ графичен потребителски интерфейс (**GUI**) с помощта на класовете от пакета **javax.swing** задачата може да се изпълни от **двама души**; Разработването на графичен интерфейс не отменя изискването Вашата програма да поддържа изредените командни параметри. В този случай към функцията на параметъра параметъра **„-q“** се добавя изискването **да не пуска** графичният интерфейс. Причината за това е, че Вашата програма трябва да позволява отдалечено тестване, а то ще се извършва в **terminal**.

Уточнения (hints) към задачата:

(о) В условието на задачата се говори за разделянето на работата на две или повече нишки. Работата върху съответната задача, в случаят в който е зададен **„-t 1“** (т.е. цялата задача се решава от една нишка) ще служи за еталон, по който да измерваме евентуално ускорение (т.е. това е **T1**). В кода реализиращ решението на задачата трябва да се предвиди и тази възможност – задачата да бъде решавана от единствена нишка (процес); Пускайки програмата да работи върху задачата с помощта на единствена нишка, ще считаме че използваме серийното решение на задачата; Измервайки времето за работа на програмата при използването на **„p“** нишки – намираме **Tp** и съответно можем да изчислим **Sp**. Представените на защитата данни за работата на програмата, трябва да отразят и ефективността от работата и, тоест да се изчисли и покаже **Ep**.

Като обобщение - данните събрани при тестването на програмата Ви, трябва да отразяват **Tp**, **Sp** и **Ep**. Желателно е освен табличен вид, да добавите и графичен вид на **Tp**, **Sp**, **Ep**, в три отделни графики.

(о) Не се очаква от Вас да реализирате библиотека, осигуряваща математически операции със комплексни числа. Подходяща за тази цел е например **Apache Commons Math3 (http://commons.apache.org/proper/commons-math/userguide/complex.html).** При изчисленията, свързани с генерирането на множеството на Манделброт (задачите за фрактали), определено ще имате нужда от нея.

(о) Не се очаква от вас да реализирате библиотека, осигуряваща математически операции със голяма точност. Подходяща за тази цел библиотека е например **Apfloat (http://www.apfloat.org)**. Ако програмата Ви има нужда от работа с големи числа, можете да използвате нея.

Разбира се **BigInteger** и **BigDecimal** класовете в **java.math** са също възможно решение – въпрос на избор и вкус.

Преди да направите избора, проверете дали избраната библиотека не използва също нишки – това може да доведе до неочаквани и доста интересни резултати; ;)

(о) Не се очаква от Вас да търсите (пишете) библиотека за генериране на **.png** изображения. Java има прекрасна за нашите цели вградена библиотека, която може да се ползва. Примерен проект, показващ генерирането на чернобялата и цветната версия на фрактала на Манделброт /множество на Манделброт за формула (2)/, цитирани в задачите за фрактали, е качена на **http://rmi.yaht.net/docs/example.projects/ - pfg.zip**.

(о) Командните аргументи (параметри) на терминална Java програма, получаваме във масива **String args[]** на **main()** метода, намиращ се в стартовият клас. За „разбирането“ им (анализирането им) може да ползвате и външни библиотеки писани специално за тази цел . Един добър пример за това е: **Apache Commons CLI (http://commons.apache.org/cli/)**.

(о) Интересен е въпросът, кога достигаме зададената точност на изчисленията? Тоест кога сме сметнали Pi със зададените от потребителя брой цифри след десетичната точка. Едно добро ограничение за серийната (последователната) програма е разликата между две поредно изчислени стойности на Pi да е произволно малка.