Зад. 5 (Граф – обхождане в ширина)

Разглеждаме графа G(V,E). Графа може да бъде ориентиран или неориентиран, свързан или слабо свързан (виж изискванията към програмата). За представянето на графа ще използваме матрица на съседство. Да се напише програма, която реализира обхождане на графа G в **ширина**. Обхождането не извършваме спрямо определен връх. Като начална стъпка на обхождането имаме всички върхове на графа или група върхове (например съседите на връх условно избран за първи). По този начин реализираме покриващо дърво (ако графа е свързан) или гора (ако графа не е). Работата на алгоритъма да се раздели по подходящ начин на две или повече нишки (задачи).

Изискванията към програмата са следните:

(о) Чете размерността на графа (броя на върховете му) от подходящо избран команден параметър – например **“-n 10240”**; Ребрата на графа генерираме произволно с помощта на **Math.random()** (т.е. класа **java.util.Random)** или **java.util.concurrent.ThreadLocalRandom**;

Разликата между двата начина - **Math.random()** е достъпен във всички версии на Java и ужасно бавен. Не е проектиран за работа в много-нишкова (multi-threaded) среда; В Java 7 и 8, разполагаме с **ThreadLocalRandom**, който е специално проектиран за работа в рамките на отделен thread (респективно multi-threaded среда);

Ще бъде много интересно да реализирате програма, използваща и двата начина и съответно получим два вида резултати от работата на програмата;

(о) Команден параметър указващ входен текстов файл, съдържащ графът, който ще обхождаме – например **„-i graph-data.in“**. Параметрите **„-n“** и **„-i“** са взаимно-изключващи се; Ако все пак бъдат зададени и двата решението как да реагира програмата е Ваше.

Форматът на файла **graph-data.in** е следният:

=== цитат ===
n
0 1 1 0 1 1 0 … 0
1 0 1 0 1 0 0 … 1
…
1 0 0 0 0 0 0 … 1
=== цитат ===

Тоест:

1вият ред съдържа единствено число, указващо размерността на графа;

На оставащите **n** реда във файла са разположени редовете от матрицата на съседство описваща нашият граф. Елементите на всеки ред от матрицата са разделени със интервали.

(о) Команден параметър указващ изходен файл, съдържащ резултата от обхождането – например **„-o graph-data.out“**. Форматът на изходният файл можете да определите сами, стига файлът да е текстов; При липса на този команден параметър **не се** записва във файл резултата от обхождането на графа;

(о) Команден параметър задава максималния брой нишки (задачи) на които разделяме работата по обхождането на графа – например **“–t 1”** или **“–tasks 3”**;

(о) 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.;

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

ЗАБЕЛЕЖКА:

(о) 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/)**.