Минимална описана окръжност

Проект по Разпределени софтуерни архитектури

Изготвил: Петър Събев, факултетен №61558

Проверил:.....

/ас. Христо Христов/

Съдържание

Условие на задачата

Разглеждаме множество от точки в равнината. Точките имат целочислени координати в интервала [0, 10240]. Съществуват безброй много окръжности в равнината, съдържащи това множество от точки.

Вашата задача е да напишете програма намираща минималната, описана около множеството от точки окръжност. Програмата намира минималната описана окръжност използвайки паралелни процеси (нишки).

Преглед на алгоритмите за решение на задачата

Минималната, описана около множество от точки окръжност е най-малката затваряща окръжност за множеството от точките. Намирането и се свежда до математическата задача за намиране (изчисляване на центърът и радиусът) на най-малката окръжност, която съдържа всички точки от дадено множество от точки в равнината.

Съществуват различни геометрични подходи за намирането на минималната, описана около множество от точки окръжност, при които се разглеждат точките, лежащи на границата на най-малката окръжност. Тези подходи се базират на следните факти:

- Минималната затваряща окръжност е уникална;
- Минималната затваряща окръжност за дадено множество от точки S, може да бъде определена от три точки в S, като тези три точки лежат на границата на окръжността. Ако тя е определена само от две точки, тогава отсечката съединяваща тези две точки е диаметърът на минималната окръжност. Ако минималната затваряща окръжност е определена от три точки, тогава триъгълникът построен през тези три точки не е тъпоъгълен.

Някои от известните алгоритми за намиране на минималната, описана около множество от точки окръжност са:

- **Вероятностен алгоритъм (randomized algorithm) със сложност $O(n)$, предложен от Emo Welzl и основаващ се на алгоритъмът за линейно програмиране (linear programming) на Raimund Seidel.**

  Този алгоритъм е рекурсивен и приема като аргументи две множества от точки S и Q. Той изчислява най-малката затваряща окръжност на обединението на S и Q, като всяка точка от Q е една от граничните точки (точките лежащи върху границата на евентуалната най-малка затваряща окръжност) на евентуалната най-малка затваряща окръжност. По този начин първоначално поставената задача за намиране на минималната, описана около множество от точки окръжност може да бъде решена посредством изпълнение на алгоритъмът със S равно на множеството от точки, около които трябва да се опише минималната окръжност и Q равно на празното множество. Понеже алгоритъмът извиква себе си рекурсивно, той ще увеличава множеството Q, предавайки го в рекурсивните извиквания докато в Q влязат всички гранични точки на окръжността.

  Алгоритъмът обработва точките от множеството S в произволен ред, като поддържа множеството от обработени точки P и най-малката окръжност затваряща обединението на множествата P и Q. На всяка следваща стъпка, алгоритъмът проверява дали следващата (която трябва да бъде обработена) точка r принадлежи на тази окръжност; ако не принадлежи, тогава той заменя затварящата окръжност с резултатът от рекурсивното извикване на алгоритъмът върху множествата P и $Q + r$ (Q, съдържащо и точката r). Независимо от това дали окръжността е заменена или не, r е включена в

множеството P. Следователно обработването на всяка точка се състои от проверяване за константно време, дали точката принадлежи на една окръжност и евентуално извършване на рекурсивно извикване на алгоритъмът. Може да бъде показано, че i-тата точка, подлежаща на обработване е с вероятност $O(1/i)$ за генериране на рекурсивно извикване и следователно общото време е линейно.

- **Изброяване и тестване на всички комбинации от две и три точки за даденото множество от точки S.**

Този алгоритъм изброява всички комбинации от две и три точки за даденото множество от точки S, като използва тези комбинации за построяване на окръжност по две точки и построяване на окръжност по три точки. По този начин алгоритъмът построява всички окръжности, определени от две или три точки и намира най-малката измежду тези окръжности, която е описана около всички точки в множеството S.

Комбинациите на две точки от общо n са:

$$\binom{n}{2} = \frac{n.(n-1)}{2!}$$

Комбинациите на три точки от общо n са:

$$\binom{n}{3} = \frac{n.(n-1).(n-2)}{3!}$$

Следователно сложността на алгоритъмът за построяването на окръжност по две точки и по три точки е $O(n^3)$. Понеже за всяка такава, построена окръжност трябва да се провери, дали тя е описана около всичките n точки от множеството S, което се извършва със сложност на алгоритъма $O(n)$, за целият алгоритъм се получава сложност $O(n^4)$. Този алгоритъм е с много по-висока сложност в сравнение с линейния вероятностен алгоритъм (randomized algorithm) със сложност O(n), предложен от Emo Welzl и основаващ се на алгоритъмът за линейно програмиране (linear programming) на Raimund Seidel, но предимството му е, че подлежи на ефективна паралелизация, понеже изисква минимална синхронизация и следователно генерирането на комбинациите на две и три точки, построяването на окръжностите и проверката за принадлежност на точка в окръжност, могат ефективно да бъдат разпределени за паралелно изчисление, без да се изисква синхронизация между тях.

С цел постигане на по-висока ефективност на паралелната спрямо серийната реализация на програмата, решаваща поставената задача е избран алгоритъмът изброяване и тестване на всички комбинации от две и три точки за даденото множество от точки S.

Архитектура на приложението и реализация на избрания алгоритъм

Някои от основните използвани класове са:

```
/**
* Представя дадена точка в равнината. Точката се определя по зададени координати
* x и y. Предоставя възможност за определяне на: разстояние между две точки; точка
* намираща се в средата на отсечката, определена от две точки и други помощни
* действия.
*
*/
public class Point2D
```

```
/**
* Предоставя възможност за генериране на точки по зададени критерии, както поддържа
* възможност за запазване на генерираните точки в файлове и чете на файлове с
* вече генерирани точки.
*
*/
public class Point2DGenerator
```

```
/**
* Представя окръжност определена от център Point2D и радиус реално число.
* Предоставя възможност за построяване на окръжност по две точки от тип Point2D
* и построяване на окръжност по три точки от тип Point2D, както и други помощни
* действия.
*
*/
public class Circle
```

```
/**
* Представя минималната, описана около множеството от точки окръжност.
*
*/
public class SmallestEnclosingCircle
```

```
/**
* Представя задача за построение на окръжност по две точки.
* По зададени точки от множеството от точки се генерират всички комбинации, в
* които участват зададените точки и за тези комбинации се построява
* окръжност по две точки и се проверява, дали тя е минималната описана окръжност.
*
*/
public class CircleBuilderAndTesterByTwoPoints implements Runnable
```

```
/**
* Представя задача за построение на окръжност по три точки.
* По зададени точки от множеството от точки се генерират всички комбинации, в
* които участват зададените точки и за тези комбинации се построява
* окръжност по три точки и се проверява, дали тя е минималната описана окръжност.
*
*/
public class CircleBuilderAndTesterByThreePoints implements Runnable
```

В приложението е реализиран алгоритъмът за изброяване и тестване на всички комбинации от две и три точки за даденото множество от точки S. Основата от която се започва в този алгоритъм е намирането на всички комбинации от две и три точки измежду дадените n точки в множеството S. Изреждането на тези комбинации се извършва паралелно, като за целта работата по намирането на комбинациите е разделена на самостоятелни и независими единици работа (задачи), не изискващи синхронизация по между си. Тези задачи са обекти от тип **CircleBuilderAndTesterByTwoPoints** и **CircleBuilderAndTesterByThreePoints**. Изпълнението на тези задачи се осъществява, посредством използването на услугата за изпълнение **ExecutorService** в Java, която представлява подинтерфейс на интерфейсът **Executor**, подпомагащ стартирането на задачи. Конкретната имплементация на услугата за изпълнение,

която е използвана е **ThreadPoolExecutor**, построена посредством статичния метод **Executors.newFixedThreadPool(numberOfThreads)**. По този начин се създава басейн от нишки (thread pool), който преизползва фиксиран брой нишки (аргументът **numberOfThreads**, който се чете от стандартния вход), работещи върху споделена неограничена опашка (shared unbounded queue) от задачи. Това означава, че във всеки един момент най-много **numberOfThreads** нишки ще бъдат активни и ще обработват задачи. Когато се добавят допълнителни задачи и всички нишки са активни, задачите ще чакат в опашката, докато не се освободи нишка. Посредством използването на този басейн от нишки (thread pool), състоящ се от нишки работници (worker threads) се минимизират допълнителните разходи по създаване на нишки. При стартиране на приложението главната нишка (main thread) прочита стойностите на аргументите (брой точки или файл с вече генерирани точки и брой нишки, които да се използват), подадени през командния ред и въз основа на подадения брой нишки, създава описаният басейн от нишки (thread pool), имплементиращ услугата за изпълнение **ExecutorService**. След това, създава обект от тип **SmallestEnclosingCircle**, който ще съхранява минималната, описана около множеството от точки окръжност. В зависимост от подадените стойности на аргументът за точки (брой точки или прочитане на точките от файл) се генерира множеството от точки, като се използва **Point2DGenerator.generateRandomPointsInRange(0, 10240, numberOfPoints)** или **Point2DGenerator.readRandomPoints(filePath)** за прочитане на точки от файлът **filePath**. След това главната нишка (main thread) започва да генерира и поставя задачи на нишките работници (worker threads) в басейна от нишки (thread pool). Всяка задача представлява обект от тип **CircleBuilderAndTesterByTwoPoints** и обект от тип **CircleBuilderAndTesterByThreePoints**, чиито конструктори приемат като аргументи: идентификатор на точките, комбинациите на които ще изчисляват; масив, представящ цялото множество от точки; текущата минимална, описана около множеството от точки окръжност. Идентификаторът на точките представлява масив от позициите на които се намират точките в масива, съхраняващ всички точки (използва се за представяне на множеството от точки S). Всяка задача, когато изчислява комбинациите, в зависимост от типа си се опитва да построи окръжност по две или три точки: за **CircleBuilderAndTesterByTwoPoints**, това става, като се извика конструкторът на **Circle** и му се подадат двете точки; за **CircleBuilderAndTesterByThreePoints** това става като се извика статичния метод **Circle.constructCircleByThreePoints(Point2D one, Point2D two, Point2D three)**, който проверява и построява окръжност по зададените три точки, ако тя може да бъде построена. След, като построят окръжността, задачите проверяват, дали тази окръжност е затваряща за даденото множество от точки, което означава, че всички точки от множеството трябва да са или вътре в окръжността, или на границата ѝ. Тази проверка е осъществена, чрез извикването на методът **contains(Point2D point)** на **Circle**, който връща, дали подадената точка лежи вътре в окръжността (**Point2D.PositionToCircle.insideCircle**), на границата на окръжността (**Point2D.PositionToCircle.onCircle**) или извън окръжността (**Point2D.PositionToCircle.outsideCircle**). Ако получената окръжност е затваряща, тогава се преминава към проверка, дали тази окръжност е най-малката затваряща окръжност. Проверката се осъществява, като се използва синхронизирания метод **setCircle(Circle circle)** на обектът от тип **SmallestEnclosingCircle** и се направи сравнение между текущо построената затваряща окръжност и текущата минимална описана, около множеството от точките окръжност.

След, като всички задачи бъдат поставени за изпълнение главната нишка изключва услугата за изпълнение, като извиква методът **shutdown()** на използваната имплементацията на **ExecutorService** и изчаква изпълнението на поставените задачи от нишките работници (worker threads), като извиква методът **awaitTermination(long timeout, TimeUnit unit)** на използваната имплементацията на **ExecutorService**.

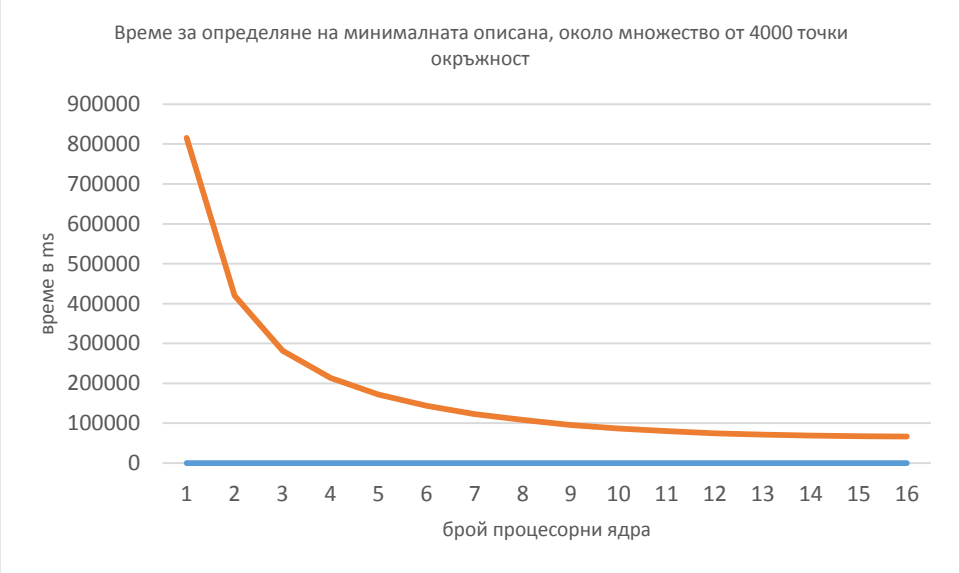
Проведени тестове и измервания

На предоставена машина за извършване на тестовете са проведени тестове за намиране на минималната, описана около множество от 4000 точки окръжност. Данните за точките са прочетени от входния файл points-data.in. В следната таблица са представени данните получени при извършване на тестовете:

Брой процесорни ядра	***Време в ms***	***Ускорение***	***Ефективност (ефикасност)***
1	815833	1	1
2	419716	1,943774	0,971887
3	282065	2,892358	0,964119
4	213470	3,821769	0,955442
5	172050	4,741837	0,948367
6	143839	5,671848	0,945308
7	122991	6,633274	0,947611
8	108023	7,552401	0,94405
9	95740	8,521339	0,946815
10	86734	9,40615	0,940615
11	80049	10,19167	0,926515
12	74813	10,90496	0,908747
13	71543	11,40339	0,877184
14	68960	11,83052	0,845037
15	67284	12,12522	0,808348
16	66108	12,34091	0,771307

Данните в горната таблица са представени графично, посредством следните диаграми:

- На диаграмата е показано времето, необходимо за определяне на минималната описана, около множеството от 4000 точки окръжност, при употребата на различен брой процесорни ядра, започвайки от 1 ядро (серийната версия на програмата) до всичките 16 ядра на предоставената машина:



- На диаграмата е показано постигнатото ускорение при определяне на минималната описана, около множеството от 4000 точки окръжност, при употребата на различен брой процесорни ядра, започвайки от 1 ядро (серийната версия на програмата) до всичките 16 ядра на предоставената машина:

- На диаграмата е показана постигнатата ефективност (ефикасност) при определяне на минималната описана, около множество от 4000 точки окръжност, при употребата на различен брой процесорни ядра, започвайки от 1 ядро (серийната версия на програмата) до всичките 16 ядра на предоставената машина: