Praca domowa 03 - plane

Szlachetka Miłosz
Nr albumu. 114014

1.Cel zadania

Celem zadania była implementacja programu w języku Java z wykorzystaniem technologii servletów. Program ten odczytuje z SQL-owej bazy danych współrzędne x, y, z punktów. Ze zbioru punktów (x,y) wyznacza podobszary dla z = const i oblicza pola owych podobszarów i zwraca wartość największego z nich z dokładnością do 5 miejsc po przecinku.

2.Realizacja zadania

Program zawarty jest w jednym pliku Plain.java:

class Point reprezentuje punkt na płaszczyźnie

- atrybuty
  - **float x, y** współrzędne punktu
- metody
  - gettery i settery
  - konstruktor bezargumentowy oraz z argumentami “x”, “y”
  - **public float getTheta(Point p)** oblicza kąt theta między wektorem(wyznaczonym z punktów: “p” oraz aktualnego), a osią OX.
  - **public float getDistance(Point p)** oblicza odległość między dwoma punktami(aktualnym oraz “p”) na płaszczyźnie
  - **public boolean equals(Object obj) (@Override)** metoda, która sprawdza czy podany w parametrze obiekt jest jest równy aktualnemu obiektowi (Sprawdzane są współrzędne “x”, “y” oraz czy “obj” nie jest nullem i czy posiada tą samą klasę)
  - **public int hashCode() (@Override)** oblicza hashCode na podstawie współrzędnych “x”, “y”.

class PointsComparator implements Comparator<Point> klasa służąca do porównywania punktów

- atrybuty
  - **Point lowestPoint** przechowuje punkt, który posiada najmniejszą wartość parametru “y” lub “y” oraz “x” jeśli punkty posiadają tą samą wartość zmiennej “y”
- metody
  - **public int compare(Point p1, Point p2) (@Override)** porównuje punkty “p1” i “p2”. Korzystając z metody **getTheta** porównuje kąt nachylenia wektorów do osi OX. Jeśli wektor utworzony z “p1” i “lowestPoint” posiada większy kąt niż wektor utworzony z “p2” i “lowestPoint” to zwraca 1, jeśli mniejszy -1. Jeśli kąty są równe to sprawdzana jest odległość między punktami “p1” i

“lowestPoint” oraz “p2” i “lowestPoint” przy użyciu metody **getDistance**. Jeśli odległość odległość “p1” od “lowestPoint” jest większa niż odległość “p2” do “lowestPoint” to zwracana jest wartość 1, w przeciwnym wypadku -1.

class GrahamScan zawiera implementację algorytmu Grahama do tworzenia otoczki wypukłej na płaszczyźnie

- metody
  - **public Point getLowestPoint(List<Point> points)** z listy punktów “points” zwraca ten, który posiada najmniejszą wartość współrzędnej “y” lub najmniejszą wartość współrzędnej “y” oraz “x”.
  - **private Set<Point> getSortedPoints(List<Point> points)** dla listy punktów “points” tworzy posortowany zbiór punktów (TreeSet). Sortowanie jest realizowane przy użyciu metody **compare** z klasy **PointsComparator,** której opis znajduje się wyżej**.**
  - **private boolean areCollinear(List<Point> points)** sprawdza czy lista punktów “points” składa się tylko i wyłącznie z punktów leżących w jednej linii prostej. Jeśli rozmiar listy jest mniejszy lub równy jeden lub punkty leżą na jednej prostej to zwraca true, w przeciwnym wypadku false.
  - **private int turn(Point p1, Point p2, Point p3)** określa kierunek sprawdzania punktów 2D “p1”, “p2”, “p3”. Oblicza iloczyn wektorowy dla trzech punktów(wektorów) “p1”, “p2”, “p3”. Jeśli otrzymana wartość jest mniejsza od zera zwraca -1 co oznacza kierunek zgodnie ze wskazówkami zegara, jesli iloczyn wektorowy jest większy niż zero to zwracana jest wartość 1, co świadczy o tym, że kierunek jest przeciwny do ruchu wskazówek zegara. Jeśli iloczyn wektorowy wyniósł zero zwracana jest wartość 0 i oznacza to, że wektory leżą na tej samej prostej.
  - **public List<Point> getConvexHull(List<Point> points)** wyznacza otoczkę wypukłą dla podanej listy punktów (“points”). W metodzie tworzona jest lista posortowanych punktów “sortedPoints” (sortowanie z użyciem metody **getSortedPoints**, opisanej wyżej) . Jeśli posortowana lista zawiera mniej niż 3 elementy lub elementy leżą w jednej linii (sprawdzanie z użyciem metody **areCollinear** opisanej powyżej) to zwraca null. Następnie tworzony jest stos i umieszczane są w nim pierwsze dwa elementy z posortowanej listy punktów(“sortedPoints”). W pętli sprawdzanie jest położenie 3 ostatnich punktów (tzn. ostatnie 2 punkty na stosie i aktualnie sprawdzny punkt). Jeśli punkt “p2” leży na zewnątrz trójkąta p1lowestPointp3 to może należeć do otoczki wypukłej, jeśli natomiast leży wewnątrz to znaczy, że nie może należeć do otoczki i zostaje usunięty ze stosu. Jeśli punkty leżą na tej samej prostej to na stos dokładany jest aktualnie sprawdzany punkt. Po zakończeniu pętli tworzona jest lista uporządkowanych punktów, tworzących otoczkę wypukłą.

- **protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException** służy do obsługi żądań GET oraz POST. W tej metodzie ustawiane jest kodowanie UTF-8 na żądanie jak i odpowiedź. Następnie sprawdzany jest URL potrzebny do połączenia z bazą danych i pobierana jest

jego wartość. Później pobierane są dane z bazy danych. Z pobranych danych tworzone są punkty 2D i umieszczane w strukturze **Map<Float, List<Point>>.** Punkty te są pogrupowane ze względu na wartość zmiennej **"z"** (zmienna "z" jest kluczem w mapie). Następnie tworzona jest otoczka wypukła dla podanych zbiorów punktów, i dla każdej otoczki obliczane jest jej pole powierzchni. Na końcu zostaje wypisana wartość największego pola powierzchni z dokładnością do 5 miejsc po przecinku.

- private ResultSet getResultsFromDB(String db) throws SQLException, ClassNotFoundException zestawia połączenie z bazą danych dla podanego URL (parametr "db"). Następnie wykonywane jest zapytanie do bazy danych, a otrzymane wyniki zostają zwrócone.

- private Map<Float, List<Point>> createPoints(ResultSet rs)throws SQLException z danych pochodzących z zapytania do SQL-owej bazy danych tworzy mapę **Map<Float, List<Point>>**, której kluczem jest wartość zmiennej "z", natomiast wartościami są listy punktów posiadające tę samą wartość zmiennej "z". Zwraca utworzoną mapę.

- private float calculatePolygonArea(List<Point> points) oblicza pole dowolnego wielokąta wypukłego przy użyciu współrzędnych jego wierzchołków. Obliczanie pola powierzchni uzyskiwane jest zgodnie z podanym poniżej wzorem

- private float findMaxArea(Map<Float, List<Point>> pointsMap) z pomocą metody **getConvexHull** wyznacza otoczkę wypukłą dla listy punktów, a następnie przy użyciu metody **calculatePolygonArea** wyznacza jej polę. Operacja ta powtarzana jest dla każdej listy punktów. Zwraca największą uzyskaną wartość pola powierzchni.

3. Mechanizm działania programu

Początkowo z żądania pobierana jest wartość, która określa parametry połączenia z bazą danych. Jeśli pobrany parametr ma wartość null to następuje przerwanie programu. Jeśli wartość nie jest null, to jest ona kodowana zgodnie ze standardem UTF-8 po to aby poprawnie obsłużyć polskie znaki. Następnie zestawiane jest połączenie z bazą danych (przy użyciu pobranego parametru). Potem wykonywane jest zapytanie do bazy danych i otrzymywane są wyniki. Z otrzymanych wyników tworzone są listy obiektów typu **Point** (punkt posiadający współrzędne "x", "y"). Listy te są grupowane ze względu na wartość parametru "z" tzn. w jednej liście znajdują się tylko te punkty, któe mają tą samą wartość parametru "z". Do przechowywania danych w taki sposób użyto struktury **Map<Float, List<Point>>.** Następnie w pętli dla każdej listy punktów (pogrupowanych ze względu na wartość "z") wyznaczana jest otoczka wypukła przy pomocy **algorytmu Grahama**. Dla uzyskanej otoczki obliczane jest pole powierzchni zgodnie ze wzorem podanym poniżej

$$area = \left| \frac{(x_1\, y_2 - y_1\, x_2) + (x_2\, y_3 - y_2\, x_3) \ldots\ldots + (x_n\, y_1 - y_n\, x_1)}{2} \right|$$

Po zakończeniu wszystkich iteracji pętli zostaje znalezione maksymalne pole powierzchni otoczki wypukłej. Na końcu wartość ta jest wypisywana z dokładnością do 5 miejsc po przecinku.

Algorytm Grahama:

1. Z listy punktów wybierany jest ten, którego wartość współrzędnej y jest najmniejsza lub jeśli jest więcej niż 1 punkt o najniższej wartości współrzędnej y to znajdowany jest ten, który ma najmniejszą wartość współrzędnej y oraz x. Punkt ten oznaczany jest jako O.
2. Lista punktów jest sortowana ze względu na kąt nachylenia wektorów(punktów) do osi OX, przy czym za początek układu współrzędnych uznaje się punkt otrzymany z kroku nr.1. Jeśli punkty posiadają ten sam kąt nachylenia theta, to porównywana jest ich odległość od punktu otrzymany z kroku nr.1.
3. Dla posortowanej listy punktów przeglądane są od początku trójki punktów (A,B,C) przy czym przesuwanie następuje o jedną pozycję. W pętli:
    - jeśli punkt środkowy (B) jest na zewnątrz trójkąta AOC to może należeć do otoczki wypukłej
    - jeśli punkt środkowy (B) leży wewnątrz trójkąta AOC to nie należy do otoczki wypukłej. Punkt B jest usuwany z listy i następuje cofnięcie się o jedną pozycję w liście.