Para najbliższych punktów na płaszczyźnie

Mamy dany zbiór punktów na płaszczyźnie dwuwymiarowej. Zadanie polega na znalezieniu pary punktów, dla których odległość między nimi jest najmniejsza.

Etap 1, rozgrzewkowy (*0.5p*)

Uzupełnij metodę `FindClosestPointsBrute()` w pliku `SweepClosestPair.cs`. Algorytm ma działać w złożoności $O(n^2)$, gdzie n to liczba punktów w chmurze. Algorytm po prostu oblicza wszystkie możliwe odległości pomiędzy punktami i wybiera najmniejszy z nich.

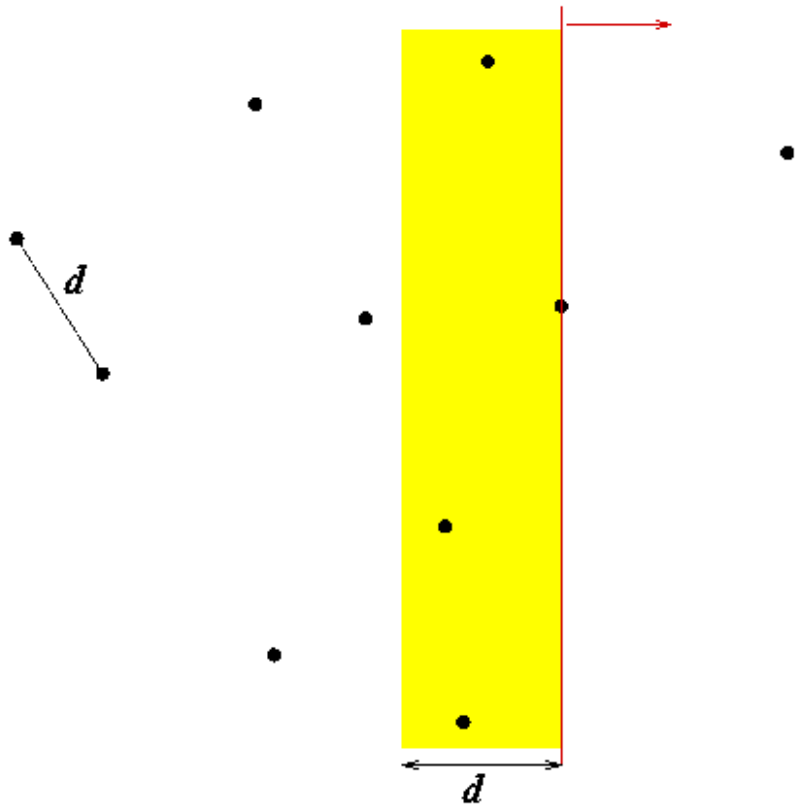
Aby zdobyć punkty z tego etapu, muszą przechodzić testy z grupy pierwszej.

Etap 2, właściwy (*1p*)

Uzupełnij metodę `FindClosestPoints()` w pliku `SweepClosestPair.cs`. Wymaganą złożonością jest $O(n \log n)$, gdzie n to liczba punktów w chmurze.

Zadanie należy rozwiązać znaną i lubianą techniką, jaką jest zamiatanie. Zamiatamy miotłą od lewej do prawej. Miotła zatrzymuje się na każdym z punktów. Podczas trwania algorytmu powinniśmy przechowywać:

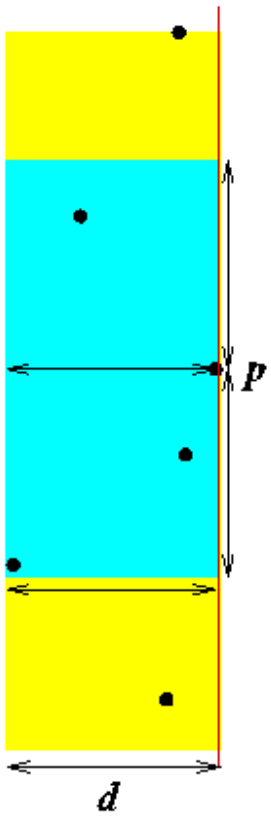
1. parę najbliższych punktów dotychczas znalezionych,
2. dystans d pomiędzy tą parą,
3. punkty znajdujące się od miotły w odległości co najwyżej d w lewo (po x-owej współrzędnej). Punkty te powinny znajdować się w strukturze posortowanego zbioru (np. drzewa) D, posortowane po y-owej współrzędnej.



Podczas każdego napotkania punktu p przez miotłę podczas zamiatania, należy wykonać następujące operacje:

1. usunąć z D wszystkie punkty, które po x-owej współrzędnej są w odległości większej niż d od p (te punkty są na lewo od p)
2. wyznaczyć najbliższy punkt na lewo od p w D
3. jeśli ten punkt jest bliżej niż dotychczasowe d, uaktualnić najbliższą parę i d

Największą zagwozdką w tym algorytmie jest punkt 2 (wyznaczenie najbliższego punktu na lewo od p). Oczywiście należy rozważać jedynie punkty, które po x-owej współrzędnej są nie dalej niż d (a więc są w D). Jednak to nie wszystko. Należy skorzystać z tego, że D jest posortowaną strukturą drzewiastą posortowaną po y współrzędnej i wybrać z niej jedynie te punkty, które znajdują się nie dalej niż d we współrzędnej y-owej. I dopiero od tych (i tylko od tych) punktów należy mierzyć dystans do p.



Parę słów na temat złożoności: najciekawszym elementem w tym algorytmie jest moment, gdy dla danego punktu porównujemy się z punktami zawartymi w prostokącie $d \times 2d$. Może się wydawać, że w takim prostokącie może być zawartych bardzo wiele punktów, w pesymistycznym przypadku rzędu $O(n)$. Gdyby tak było istotnie, algorytm oczywiście nie miałby złożoności $O(n \log n)$, ale gorszą ($O(n^2)$). Da się jednak pokazać, że w takim prostokącie zawsze, nawet w pesymistycznym przypadku, będzie maksymalnie pewna stała liczba punktów, zależna od przestrzeni, w której działamy (ale nie od liczby punktów, jaką liczy sobie nasza chmura). Jest ona rzędu 6 punktów dla przestrzeni Euklidesowej, choć w praktyce nie pojawia się więcej niż 1-3 punkty. Dlatego pętlę po punktach w prostokącie potraktuj jako operację stałą, gdy będziesz obliczał złożoność obliczeniową swojego algorytmu.

Aby otrzymać punkty za ten etap, muszą przechodzić testy z grupy drugiej, przy czym ocenianie nie jest binarne: można postawić częściową liczbę punktów za ten etap w przypadku częściowego rozwiązania (decyduje prowadzący).

Etap 3, wydajnościowy (*1.0p*)

Tutaj nadal testowana jest metoda `FindClosestPoints()`. Nie trzeba wprowadzać żadnych modyfikacji. Aby zdobyć punkty z tego etapu, muszą przechodzić testy z grupy trzeciej (testy z timeoutami). Poza tym powinny przechodzić wszystkie testy z grupy drugiej.

Uwagi

- Możemy założyć, że zawsze w chmurze będą przynajmniej dwa punkty
- W C# klasą, która może posłużyć za D, jest `SortedSet`. Aby posortować punkty po y-owej współrzędnej, należy wykorzystać odpowiedni komparator. Podczas samej implementacji i wyboru punktów z odpowiedniego przedziału należy użyć metody `GetViewBetween()` w tej klasie. Klasa `SortedSet` znajduje się w:

```
Namespace:   System.Collections.Generic
Assembly:  System (in System.dll)
```

- W podanej chmurze punktów nie ma duplikatów. Wszystkie punkty są unikatowe
- Wykorzystaj strukturę `Point` do przechowywania punktów i obliczania odległości między punktami
- Wykorzystaj metodę `Distance()` ze struktury `Point`, aby liczyć dystans. Sam napiszesz tę metodę
- Kolejność zwracanych punktów (który jest pierwszy,a który drugi) nie ma znaczenia
- Inne sposoby rozwiązania zadania, które nie będą wykorzystywać metody zamiatania, **nawet jeśli będą miały odpowiednią złożoność**, nie będą uznawane