Jakub Pawlewicz — Treść zadania, Opracowanie
Jakub Wojtaszczyk, Filip Wolski — Program

Dostępna pamięć: 64 MB. *OI, Etap I, 18.10–15.11.2010*

Wykres

Wykresem *nazywamy dowolny ciąg punktów na płaszczyźnie. Dany wykres* $(P_1, \ldots, P_n)$ *zamierzamy zastąpić innym wykresem, który będzie miał co najwyżej* m *punktów* ($m \leqslant n$), *ale w taki sposób, aby „przypominał" jak najbardziej oryginalny wykres.*

Nowy wykres tworzymy w ten sposób, że dzielimy ciąg punktów $P_1, \ldots, P_n$ *na* s ($s \leqslant m$) *spójnych podciągów:*

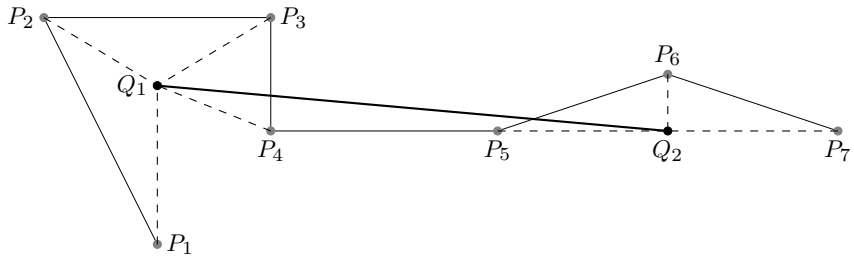
$$(P_{k_0+1}, \ldots, P_{k_1}), \quad (P_{k_1+1}, \ldots, P_{k_2}), \quad \ldots, \quad (P_{k_{s-1}+1}, \ldots, P_{k_s}),$$

przy czym $0 = k_0 < k_1 < k_2 < \ldots < k_s = n$, *a następnie każdy podciąg* $(P_{k_{i-1}+1}, \ldots, P_{k_i})$, *dla* $i = 1, \ldots, s$, *zastępujemy jednym nowym punktem* Q_i. *Mówimy wtedy, że każdy z punktów* $P_{k_{i-1}+1}, \ldots, P_{k_i}$ *został* **ściągnięty** *do punktu* Q_i. *W rezultacie otrzymujemy nowy wykres reprezentowany przez ciąg* $Q_1, \ldots, Q_s$. *Miarą podobieństwa tak utworzonego wykresu do oryginalnego jest maksimum odległości każdego z punktów* $P_1, \ldots, P_n$ *do punktu, do którego został on ściągnięty:*

$$\max_{i=1,\ldots,s} \left(\max_{j=k_{i-1}+1,\ldots,k_i} \left(d\left(P_j, Q_i\right) \right) \right),$$

przy czym $d(P_j, Q_i)$ *jest odległością między* P_j *i* Q_i *i wyraża się standardowym wzorem:*

$$d\left((x_1, y_1), (x_2, y_2)\right) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$



Rys. 1: Przykładowy wykres $(P_1, \ldots, P_7)$ i nowy wykres (Q_1, Q_2), gdzie $(P_1, \ldots, P_4)$ ściągamy do Q_1, natomiast (P_5, P_6, P_7) do Q_2.

Dla danego wykresu składającego się z n *punktów należy znaleźć wykres najbardziej go przypominający, który zawiera co najwyżej* m *punktów, przy czym podział wykresu na spójne podciągi jest dowolny. Ze względu na skończoną precyzję obliczeń, za poprawne będą uznawane wyniki, których podobieństwo do danego wykresu jest co najwyżej o* 0.000001 *większe od optymalnego wyniku.*

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n *oraz* m *oddzielone pojedynczym odstępem,* $1 \leqslant m \leqslant n \leqslant 100\,000$. *Każdy z następnych* n *wierszy*

zawiera po dwie liczby całkowite oddzielone pojedynczym odstępem. W $(i+1)$-szym wierszu znajdują się liczby x_i, y_i, $-1\,000\,000 \leqslant x_i, y_i \leqslant 1\,000\,000$, reprezentujące współrzędne (x_i, y_i) punktu P_i.

Wyjście

W pierwszym wierszu standardowego wyjścia należy wypisać jedną liczbę rzeczywistą d, będącą miarą podobieństwa znalezionego wykresu do wykresu oryginalnego. W drugim wierszu należy wypisać jedną liczbę całkowitą s, $1 \leqslant s \leqslant m$. Następnie, w kolejnych s wierszach powinny zostać wypisane współrzędne punktów $Q_1, \ldots, Q_s$, po jednym punkcie w wierszu. W $(i+2)$-gim wierszu powinny znaleźć się liczby rzeczywiste u_i i v_i oddzielone pojedynczym odstępem i określające współrzędne (u_i, v_i) punktu Q_i. Wszystkie liczby rzeczywiste na wyjściu należy wypisać z rozwinięciem do co najwyżej 15 cyfr po kropce dziesiętnej.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`7 2`	`3.00000000`
`2 0`	`2`
`0 4`	`2.00000000 1.76393202`
`4 4`	`11.00000000 1.99998199`
`4 2`	
`8 2`	
`11 3`	
`14 2`	

Rozwiązanie

Treść zadania sugeruje, że należy starać się minimalizować funkcję podobieństwa, mając ograniczoną liczbę punktów przybliżonego wykresu. Znacznie łatwiej natomiast myśleć o tym zadaniu pod kątem minimalizacji liczby punktów przybliżonego wykresu przy określonym maksimum na funkcję podobieństwa. Na tym spostrzeżeniu zasadza się rozwiązanie wzorcowe.

Klasycznym problemem, do którego odwołuje się rozwiązanie tego zadania, jest problem minimalnego koła pokrywającego zadany zbiór punktów na płaszczyźnie (ang. *smallest enclosing disc*). Jak się okazuje, możemy ten problem rozwiązać w *oczekiwanym* czasie liniowym. Taki randomizowany algorytm jest przedstawiony poniżej. Problem ten można nawet rozwiązać w *pesymistycznym* czasie liniowym, jednakże to rozwiązanie jest diametralnie bardziej skomplikowane, jak również ma dużą stałą w złożoności czasowej. Czytelników zainteresowanych takim algorytmem odsyłamy na stronę internetową [37], opis dostępny tylko w języku angielskim.

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe rozwiązuje, w pewnym sensie, problem dualny do tego zadania. Załóżmy, że mamy dane d — maksymalny dystans od punktów wykresu do punktów, do których zostały ściągnięte. Mamy dowolność w wyborze punktów ściągnięcia, a zatem pytanie, czy dany zbiór punktów możemy ściągnąć do jednego punktu, zachowując ograniczenie na d, jest równoważne pytaniu, czy najmniejsze koło pokrywające ten zbiór punktów ma promień co najwyżej d.

Załóżmy dalej, że mamy do dyspozycji funkcję stwierdzającą w czasie $O(k)$, czy dane k punktów leży w pewnym kole o promieniu d. Stosując wyszukiwanie binarne, potrafimy stwierdzić, ile, co najwyżej, może wynosić parametr k_1 (patrz treść zadania). W podobny sposób możemy obliczyć k_2, k_3 itd. — rzecz jasna, każdorazowo opłaca się wybrać największą możliwą wartość parametru. Złożoność takiego rozwiązania niestety nie szacuje się najlepiej. Każde pojedyncze wyszukiwanie binarne miałoby koszt czasowy $O(n \log n)$, co dałoby całkowitą złożoność czasową $O(nm \log n)$.

W zamian możemy zastosować alternatywną metodę wyszukiwania binarnego. Załóżmy, że znamy już maksymalną możliwą wartość k_{i-1} i teraz obliczamy wartość k_i. Niech $t_i = k_i - k_{i-1}$. Najpierw wyznaczamy takie całkowite e, że $2^{e-1} \leqslant t_i < 2^e$. Dopiero potem, już za pomocą „zwykłego" wyszukiwania binarnego, znajdujemy t_i w przedziale $[2^{e-1}, \min(n - k_{i-1}, 2^e - 1)]$. W ten sposób liczba wywołań algorytmu znajdującego minimalne koło pokrywające dany zbiór punktów nie przekroczy $2e$, a największy z tych zbiorów będzie miał nie więcej niż 2^e elementów. Wiemy, że $e = O(\log t_i)$ i $e \cdot 2^e = O(t_i \log t_i)$, a zatem złożoność czasowa całego rozwiązania to

$$\sum_{i=1}^{m} O(t_i \log t_i) \;\leqslant\; \sum_{i=1}^{m} O(t_i \log n) \;=\; O(n \log n).$$

Podsumowując — w złożoności czasowej $O(n \log n)$ potrafimy sprawdzić, ile minimalnie punktów przybliżonego wykresu potrzeba, żeby miara podobieństwa do oryginalnego wykresu wyniosła co najwyżej d. Możemy w takim razie użyć wyszukiwania binarnego, aby znaleźć najmniejsze takie d, dla którego przybliżony wykres ma co najwyżej zadane m punktów. Wymaga to $\lceil \log_2(10^{12}\sqrt{2}) \rceil$ iteracji opisanego algorytmu, jako że początkowy zakres wyszukiwania binarnego rozciąga się od zera do $10^6\sqrt{2}$ (bo koło o promieniu $10^6\sqrt{2}$ i środku w zerze na pewno pokrywa cały wykres), końcowy zakres nie może przekroczyć 10^{-6}, a logarytm dwójkowy ilorazu tych dwóch wartości określa liczbę kroków wyszukiwania binarnego.

Minimalne koło pokrywające zbiór punktów

W tym problemie szukamy najmniejszego koła, które pokrywa zbiór punktów $\{P_1, P_2, \ldots, P_n\}$. Zaprezentujemy klasyczny algorytm, który można znaleźć w książce [23]. Główny pomysł jest taki, aby budować to koło przyrostowo, tzn. kolejno dla $i = 2, \ldots, n$ tworzyć minimalne koło K_i pokrywające pierwsze i punktów $\{P_1, \ldots, P_i\}$.

Spostrzeżenie 1. *Niech $1 < i < n$. Jeśli $P_{i+1} \in K_i$, to $K_{i+1} = K_i$, a w przeciwnym razie P_{i+1} leży na brzegu koła K_{i+1}.*

Spostrzeżenie 1 mówi, że dodanie nowego punktu jest proste, jeżeli leży on wewnątrz dotychczas najmniejszego koła. W przeciwnym razie dostajemy tylko informację, że nowo dodany punkt leży na pewno na brzegu koła, które mamy utworzyć — w tym przypadku będziemy musieli zapewne wykonać więcej pracy. Aby ta sytuacja nie zachodziła zbyt często, wystarczy punkty rozważać w losowej kolejności. Samo spostrzeżenie udowodnimy później, a dotychczasowe rozważania prowadzą do następującego algorytmu. Zakładamy w nim, że punkty $P_1, \ldots, P_n$ są parami różne — jeśli tak nie jest, możemy posortować je po współrzędnych i wyrzucić powtórzenia, wszystko w czasie $O(n \log n)$.

```
1: function MinKoło({P_1, ..., P_n})
2: begin
3:   potasuj zbiór {P_1, ..., P_n}, wybierając losową permutację;
4:   K_2 := najmniejsze koło pokrywające P_1 i P_2;
5:   for i := 2 to n − 1 do
6:     if P_{i+1} ∈ K_i then
7:       K_{i+1} := K_i
8:     else
9:       K_{i+1} := MinKołoZPunktem(P_{i+1}, {P_1, ..., P_i});
10:  return K_n;
11: end
```

Wprowadziliśmy nową funkcję MinKołoZPunktem, którą trzeba zaimplementować. Jak skonstruować najmniejsze koło pokrywające zbiór punktów $\{P_1, \ldots, P_i\}$, wiedząc, że to koło musi mieć na brzegu punkt $Q = P_{i+1}$? Możemy zastosować analogiczne podejście jak poprzednio, a mianowicie, budować to koło przyrostowo. Tym razem kolejno dla $j = 1, \ldots, i$ tworzymy koło K'_j pokrywające pierwsze j punktów $\{P_1, \ldots, P_j\}$ oraz punkt Q, z dodatkowym w warunkiem, że Q musi leżeć na brzegu K'_j. Teraz dodanie kolejnego punktu P_{j+1} polega na sprawdzeniu, czy należy on do bieżącego najmniejszego utworzonego koła — jeśli tak, to koło to nie zmienia się, a w przeciwnym razie musi ono mieć na swym brzegu punkt P_{j+1}. To ostatnie oznacza, że szukane koło ma na brzegu punkty Q i P_{j+1}. Daje nam to taki oto pseudokod funkcji MinKołoZPunktem.

```
1: function MinKołoZPunktem(Q, {P_1, ..., P_i})
2: begin
3:   K'_1 := najmniejsze koło pokrywające P_1 i Q;
4:   for j := 1 to i − 1 do
5:     if P_{j+1} ∈ K'_j then
6:       K'_{j+1} := K'_j
7:     else
8:       K'_{j+1} := MinKołoZ2Punktami(P_{j+1}, Q, {P_1, ..., P_j});
9:   return K'_i;
10: end
```

Teraz z kolei stajemy przed problemem znalezienia najmniejszego koła pokrywającego punkty $\{P_1, \ldots, P_j\}$, które ma na brzegu zadane dwa punkty $Q_1 = P_{j+1}$ i $Q_2 = Q$. Ponownie stosujemy podejście przyrostowe. Jeżeli kolejny punkt P_{k+1} należy do bieżącego koła, to nic nie musimy robić. Jeżeli jest na zewnątrz, to nowo tworzone koło

musi mieć na brzegu P_{k+1}. Musi mieć ono zatem na brzegu punkty: Q_1, Q_2 i P_{k+1}. Koło, dla którego znamy trzy różne punkty z jego brzegu, jest wyznaczone jednoznacznie (jeśli tylko te trzy punkty nie są współliniowe — na szczęście, dzięki konstrukcji całego algorytmu, taka sytuacja nie wystąpi w naszym przypadku). Daje to poniższy pseudokod, który kończy rozwiązanie problemu.

```
1:  function MinKołoZ2Punktami(Q_1, Q_2, {P_1, ..., P_j})
2:  begin
3:    K''_0 := najmniejsze koło pokrywające Q_1 i Q_2;
4:    for k := 0 to j − 1 do
5:      if P_{k+1} ∈ K''_k then
6:        K''_{k+1} := K''_k
7:      else
8:        K''_{k+1} := koło mające na brzegu punkty Q_1, Q_2 i P_{k+1};
9:    return K''_j;
10: end
```

Zanim przejdziemy do analizy czasowej tego algorytmu, wypada udowodnić jego poprawność. Otóż w całym rozumowaniu notorycznie korzystaliśmy z faktu, że jeśli jakiś punkt jest poza najmniejszym kołem, to nowo tworzone koło musi mieć ten punkt na swoim brzegu. Dodatkowo, czasem znaliśmy jakieś punkty, które muszą być na brzegu szukanych kół. Poniższy lemat obejmuje wszystkie tego typu przypadki.

Lemat 2. *Rozważmy sytuację, w której mamy pewien zbiór punktów, nazwijmy go $\mathcal{P}$. Szukamy najmniejszego koła pokrywającego $\mathcal{P}$ z dodatkową informacją o zbiorze punktów, nazwijmy go $\mathcal{Q}$, które muszą należeć do brzegu tego koła (zbiór $\mathcal{Q}$ może być też pusty). Wtedy (i) takie koło jest wyznaczone jednoznacznie. Oznaczmy takie najmniejsze koło przez K.*

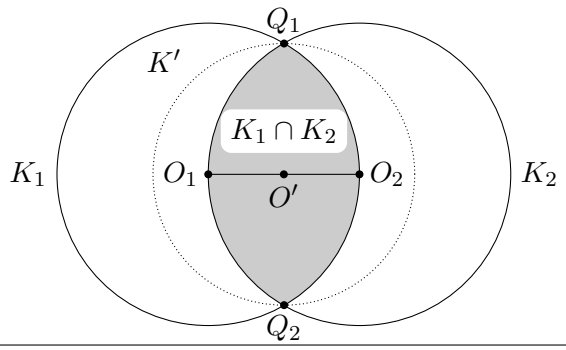
Teraz bierzemy kolejny punkt P. Szukamy najmniejszego koła K' pokrywającego zbiór $\mathcal{P} \cup \{P\}$, takiego że na brzegu tego koła na pewno są punkty ze zbioru $\mathcal{Q}$. Wtedy (ii) jeśli $P \in K$, to $K' = K$, a (iii) jeśli, przeciwnie, $P \notin K$, to K' musi być najmniejszym kołem pokrywającym zbiór $\mathcal{P}$, na którego brzegu muszą znaleźć się punkty ze zbioru $\mathcal{Q} \cup \{P\}$.

Dowód:

(i) Załóżmy przeciwnie, że istnieją dwa koła pokrywające $\mathcal{P}$ mające na brzegu wszystkie punkty z $\mathcal{Q}$. Ponieważ oba są najmniejsze, więc oba mają ten sam promień, tylko różne środki. Niech środki tych kół to O_1 i O_2. Oznaczmy te koła odpowiednio przez K_1 i K_2 (rysunek 1).

Ponieważ zbiór punktów $\mathcal{P}$ zawiera się zarówno w kole K_1, jak i w kole K_2, więc musi zawierać się w ich przecięciu: $\mathcal{P} \subseteq K_1 \cap K_2$. Podobnie jest z brzegiem — zbiór punktów $\mathcal{Q}$ musi należeć do przecięcia brzegów K_1 i K_2. Brzegi kół to okręgi. Okręgi te przecinają się w dwóch punktach, niech będą to Q_1 i Q_2. Stąd, w zbiorze $\mathcal{Q}$ mogą występować co najwyżej dwa punkty: $\mathcal{Q} \subseteq \{Q_1, Q_2\}$.

W związku z powyższym możemy stworzyć inne, mniejsze koło K', które będzie pokrywać $K_1 \cap K_2$ (a tym samym zbiór $\mathcal{P}$) oraz którego brzeg będzie przechodził przez punkty Q_1 i Q_2, co przeczy minimalności kół K_1 i K_2. Wystarczy obrać środek koła O' jako środek odcinka $\overline{O_1O_2}$, tak jak na rysunku 1.

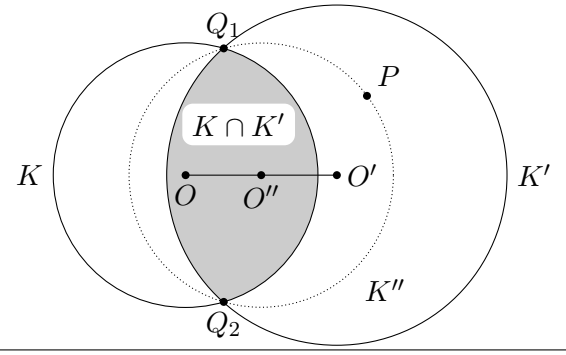


Rys. 1: Założenie o istnieniu dwóch minimalnych kół pokrywających $\mathcal{P}$ i przechodzących przez $\mathcal{Q}$ prowadzi do sprzeczności.

(ii) Szukane koło K' pokrywa również mniejszy zbiór $\mathcal{P}$, więc K' nie może być mniejsze niż K, gdyż K jest najmniejszym możliwym kołem pokrywającym $\mathcal{P}$. Ponieważ $P \in K$, więc K pokrywa zbiór $\mathcal{P} \cup \{P\}$. Z (i) wynika, że nie może istnieć drugie koło o tym samym promieniu, które również pokrywa $\mathcal{P} \cup \{P\}$, zatem $K' = K$.

(iii) Rozumowanie jest podobne jak w punkcie (i). Załóżmy, że szukane koło K' nie ma na brzegu punktu P. Ponieważ oba koła K i K' pokrywają zbiór $\mathcal{P}$, więc $\mathcal{P}$ zawiera się w ich przecięciu: $\mathcal{P} \subseteq K \cap K'$. Ponadto wiemy, że oba koła K i K' mają na brzegu zbiór punktów $\mathcal{Q}$. Oznacza to, że przecięcie brzegów kół K i K' zawiera $\mathcal{Q}$. Brzegi kół to okręgi, a przecięcie dwóch okręgów musi się w naszym przypadku składać z dwóch punktów Q_1 i Q_2. Zatem w skład zbioru $\mathcal{Q}$ mogą wchodzić jedynie te punkty: $\mathcal{Q} \subseteq \{Q_1, Q_2\}$.

Teraz możemy stworzyć inne koło K'', które będzie pokrywać zbiór $K \cap K'$ (a tym samym zbiór $\mathcal{P}$) oraz którego brzeg będzie przechodził przez punkty Q_1, Q_2 i dodatkowo przez punkt P, tak jak na rysunku 2.



Rys. 2: Założenie, że minimalne koło ma punkt P w swoim wnętrzu, prowadzi do utworzenia innego, mniejszego koła z P na brzegu.

Środek O'' takiego koła będzie leżał na odcinku łączącym środki kół K i K', oznaczone odpowiednio przez O i O'. Koło K'' ma promień mniejszy niż koło K', co przeczy minimalności K'.

■

Skoro jesteśmy pewni poprawności naszej metody, możemy przejść do szacowania złożoności czasowej. Zacznijmy od następującego spostrzeżenia, które wynika ze struktury całego algorytmu.

Spostrzeżenie 3. *Najmniejsze koło pokrywające zbiór punktów wyznaczone jest przez co najwyżej trzy punkty z tego zbioru.*

Dowód: Przyjrzyjmy się temu, w których miejscach pseudokodów naszych funkcji tworzymy konkretne koła. Są to: wyznaczanie najmniejszego koła pokrywającego dwa punkty (wiersz 4 funkcji MinKoło, wiersz 3 funkcji MinKołoZPunktem oraz wiersz 3 funkcji MinKołoZ2Punktami) oraz wyznaczanie koła przechodzącego przez trzy punkty (wiersz 8 funkcji MinKołoZ2Punktami). ■

Aby pokryć kołem zbiór punktów $\{P_1, \ldots, P_n\}$, wystarczy zatem znaleźć pewne trzy punkty P_i, P_j i P_k, dla których najmniejsze koło pokrywające jest również najmniejszym kołem pokrywającym cały zbiór. Zauważmy teraz, że sytuacja, w której nowo rozważany punkt P_n jest poza minimalnym kołem pokrywającym zbiór $\{P_1, \ldots, P_{n-1}\}$, może zajść tylko wtedy, gdy jest on jednym z punktów P_i, P_j, P_k.

Teraz możemy przejść do analizy czasowej. W funkcji MinKołoZ2Punktami mamy pętlę wykonującą j iteracji, z których każda zajmuje czas stały. Zatem złożoność czasowa tej funkcji to $O(j)$.

Funkcja MinKołoZPunktem zawiera pętlę wykonującą $i-1$ iteracji po $j = 1, \ldots, i-1$, przy czym każda iteracja wykonuje się w czasie stałym, chyba że punkt P_{j+1} nie należy do najmniejszego koła pokrywającego zbiór $\{Q, P_1, \ldots, P_j\}$. Ta sytuacja ma miejsce tylko wtedy, gdy punkt P_{j+1} jest jednym z trzech wyznaczających najmniejsze koło pokrywające zbiór $\{Q, P_1, \ldots, P_j, P_{j+1}\}$. Ponieważ Q jest jednym z tych trzech punktów, więc P_{j+1} musi być jednym z dwóch pozostałych punktów ze zbioru $\{Q, P_1, \ldots, P_{j+1}\}$, które leżą na obwodzie minimalnego koła pokrywającego ten zbiór punktów. Skoro punkty rozważamy w losowej kolejności, to prawdopodobieństwo ostatniego zdarzenia wynosi $2/(j+1)$ i w tym przypadku wołamy MinKołoZ2Punktami dla j punktów. Podsumowując, oczekiwany czas wykonania funkcji MinKołoZPunktem wynosi

$$\sum_{j=1}^{i-1}\Big(O(1) + \frac{2}{j+1}O(j)\Big) = \sum_{j=1}^{i-1} O(1) = O(i).$$

Podobnie analizujemy naszą główną funkcję MinKoło. Zawiera ona pętlę po $i = 2, \ldots, n-1$, w której każda iteracja wykonuje się w czasie stałym, chyba że punkt P_{i+1} nie należy do najmniejszego koła pokrywającego zbiór $\{P_1, \ldots, P_i\}$, a to jest możliwe tylko wtedy, gdy P_{i+1} jest jednym z trzech punktów wyznaczających najmniejsze koło pokrywające zbiór $\{P_1, \ldots, P_i, P_{i+1}\}$. Prawdopodobieństwo tego ostatniego zdarzenia wynosi $3/(i+1)$ i wtedy trzeba wywołać MinKołoZPunktem dla

i punktów, co zajmuje czas $O(i)$. W rezultacie otrzymujemy oczekiwaną złożoność czasową:

$$\sum_{i=2}^{n-1}\Big(O(1)+\frac{3}{i+1}O(i)\Big)=\sum_{i=1}^{n-1}O(1)=O(n).$$

Udowodniliśmy następujące twierdzenie.

Twierdzenie 4. *Oczekiwany czas wykonania funkcji* MinKoło *wynosi* $O(n)$.

Implementację rozwiązania wzorcowego można znaleźć w pliku `wyk.cpp`.

Rozwiązania wolniejsze

Rozwiązanie tego zadania składa się, *de facto*, z dwóch części — mamy mianowicie algorytm wyszukujący kolejne wartości k_i, wykorzystujący niejako osobną metodę znajdowania najmniejszego koła pokrywającego dany zbiór punktów. Poniżej opisujemy, w jaki sposób można każdą z tych części wykonać nieoptymalnie, a także podajemy, w których spośród rozwiązań nieoptymalnych `wyks[1-8].cpp` ta metoda została wykorzystana.

Kolejne wartości k_i

Łatwo spowolnić rozwiązanie zadania *Wykres*, jeżeli nie użyje się „sprytnej" metody wyszukiwania binarnego kolejnych wartości k_i, lecz zastosuje się zwykłe wyszukiwanie binarne (o takim rozwiązaniu wspominaliśmy już na początku opisu). Tę technikę wykorzystują rozwiązania powolne o numerach 1, 2, 3, 5, 7.

Jeszcze wolniej można szukać kolejnych k_i zwyczajnie przyrostowo, tj. zwiększając k_i o jeden, aż nie będzie możliwe pokrycie kolejnego zbioru punktów kołem o danym, ograniczonym promieniu. Implementacje wykorzystujące tę metodę znajdują się w rozwiązaniach powolnych numer 4, 6, 8.

Koło pokrywające zbiór punktów

Zauważmy, że w rozwiązaniu nie musimy tak naprawdę wyznaczać promieni minimalnych kół — wystarczy nam algorytm sprawdzania, czy istnieje koło o *zadanym* promieniu d, pokrywające dany zbiór punktów.

Ten problem można rozwiązać, na przykład, w czasie kwadratowym — na wszystkie sposoby wybieramy jeden punkt, a następnie, w czasie liniowym, poszukujemy koła o promieniu d z tym właśnie punktem na obwodzie, które pomieści wszystkie pozostałe punkty. Implementacje tej metody można znaleźć w rozwiązaniach powolnych numer 3 i 4.

Dalej, możemy to zaimplementować sześciennie — wybieramy dwa punkty, które jednoznacznie definiują koło o promieniu d (zawierające te punkty na obwodzie), i przeglądamy wszystkie pozostałe punkty, sprawdzając, czy znajdują się wewnątrz tego koła. Implementacje: rozwiązania powolne numer 5 i 6.

W końcu, najwolniejsza metoda działa w złożoności czasowej $O(n^4)$. Tutaj nie potrzebujemy już znać wartości parametru d: wybieramy trzy punkty, które definiują

koło, a następnie sprawdzamy, czy wszystkie pozostałe punkty znajdują się w tym kole. Spośród znalezionych w ten sposób kół wybieramy najmniejsze. Implementacje w rozwiązaniach powolnych numer 7 i 8.

Testy

Zadanie było sprawdzane na 13 zestawach danych testowych, różniących się przede wszystkim sposobem rozmieszczenia punktów na płaszczyźnie: losowe rozmieszczenie (typ 1), kwadratowa spirala (typ 2), zwykła spirala (typ 3), błądzenie losowe (typ 4), punkty gęsto zgrupowane w kilku miejscach płaszczyzny (typ 5), punkty w rogach (typ 6), wszystkie punkty w tym samym miejscu (typ 7). Poniżej tabelka opisująca podstawowe parametry testów oraz ich typy.

Nazwa	n	m	Opis
wyk1a.in	4	4	typ 1
wyk1b.in	4	1	typ 1
wyk2a.in	8	2	typ 1
wyk2b.in	20	10	typ 1
wyk3a.in	10	1	typ 3
wyk3b.in	20	5	typ 2
wyk4a.in	100	5	typ 4
wyk4b.in	100	99	typ 4
wyk5a.in	1 000	500	typ 6
wyk5b.in	1 000	50	typ 7
wyk6a.in	1 000	50	typ 1
wyk6b.in	1 000	500	typ 1
wyk7.in	1 000	20	typ 5
wyk8a.in	1 000	10	typ 3
wyk8b.in	1 000	100	typ 3

Nazwa	n	m	Opis
wyk9a.in	5 000	20	typ 4
wyk9b.in	5 000	100	typ 4
wyk9c.in	5 000	1 000	typ 4
wyk10a.in	50 000	4	typ 5
wyk10b.in	50 000	20	typ 1
wyk11a.in	100 000	100	typ 1
wyk11b.in	100 000	10 000	typ 3
wyk11c.in	100 000	100 000	typ 6
wyk12a.in	100 000	100	typ 3
wyk12b.in	100 000	20	typ 3
wyk12c.in	100 000	200	typ 5
wyk12d.in	100 000	1	typ 5
wyk13a.in	100 000	200	typ 7
wyk13b.in	100 000	80 000	typ 6
wyk13c.in	100 000	1	typ 4

Zawody II stopnia

opracowania zadań