* **4** Oblicz metodą trapezów pole obszaru zamkniętego ograniczonego osią OX oraz funkcją $f(x) = x \cdot \sin x$ w przedziale $[0; \pi]$.

* **5** Napisz program znajdujący przybliżenie pierwiastka kwadratowego metodą Newtona–Raphsona z wykorzystaniem funkcji rekurencyjnej.

** **6** Przygotuj arkusz kalkulacyjny umożliwiający poszukiwanie miejsc zerowych funkcji $f(x) = x^3 - 2x^2 - 5x + 6$ metodą bisekcji.

** **7** Napisz program znajdujący jedno miejsce zerowe wielomianu stopnia n. Stopień wielomianu, jego współczynniki oraz przedział, w którym będziesz szukać miejsca zerowego, wczytaj z klawiatury. Do obliczania wartości wielomianu wykorzystaj schemat Hornera.

** **8** Napisz program znajdujący pierwiastek trzeciego stopnia z dodatniej liczby rzeczywistej.

** **9** Oblicz metodą trapezów pole obszaru zamkniętego ograniczonego osią OX oraz funkcją $f(x) = x \cdot \sin x$ w przedziale $[-2\pi; 2\pi]$.

*** **10** Przygotuj symulację ruchów Browna w arkuszu kalkulacyjnym. Przyjmij model, w którym cząstka pokonuje w jednostce czasu odległość równą 1 w dowolnym kierunku. Zilustruj symulację wykresem punktowym.

*** **11** Oblicz pole obszaru zamkniętego ograniczonego osią OX oraz funkcją $f(x) = -4x^2 + 4x$ w przedziale $[0; 1]$. Zastosuj metodę Monte Carlo.

*** **12** Napisz program znajdujący pierwiastek n-tego stopnia z dodatniej liczby rzeczywistej, gdzie $n \geq 2$.

9. Algorytmy badające własności geometryczne

Podczas projektowania gier komputerowych czy programów użytkowych czasami trzeba określić wzajemne położenie obiektów. Wykorzystuje się w tym celu pojęcia i zależności matematyczne. W tym temacie poznasz algorytmy badające własności geometryczne obiektów na płaszczyźnie.

Cele lekcji

- Zbadasz położenie punktów względem prostej i odcinka.
- Sprawdzisz, czy dwa odcinki się przecinają.
- Zbadasz przynależność punktu do wielokąta wypukłego.

9.1. Położenie punktów względem prostej i odcinka

Do badania położenia punktów wykorzystamy m.in. zagadnienia omawiane na lekcjach matematyki.

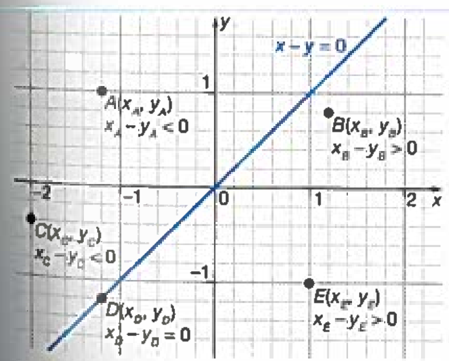
Położenie punktów względem prostej

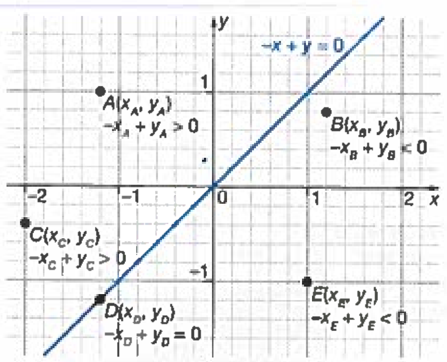
Zapewne znasz **równanie ogólne prostej**: $Ax + By + C = 0$, gdzie $A \neq 0$ lub $B \neq 0$. Na podstawie tego równania można łatwo określić **położenie punktu względem prostej**. Wszystkie punkty $P(x_P, y_P)$ spełniające nierówność $Ax_P + By_P + C > 0$ leżą po jednej stronie prostej, a punkty spełniające nierówność $Ax_P + By_P + C < 0$ – po drugiej stronie prostej. Jeśli $Ax_P + By_P + C = 0$, to punkt $P(x_P, y_P)$ leży na prostej.

Prostą można opisać różnymi równaniami ogólnymi. Rysunek 9.1 przedstawia przykładową prostą i położenie kilku punktów względem niej oraz informacje o wynikach otrzymanych po podstawieniu współrzędnych punktów do lewej strony równań ogólnych.

Równanie ogólne prostej

Położenie punktu względem prostej





Rys. 9.1. Położenie punktów względem prostej $y = x$ (w postaci ogólnej np. $x - y = 0$ lub $-x + y = 0$)

Warto wiedzieć
Za pomocą równania kierunkowego ($y = ax + b$) nie można opisać prostej prostopadłej do osi OX. Natomiast w postaci ogólnej można zapisać równanie każdej prostej.

Znak wartości otrzymanej po podstawieniu współrzędnych punktu do lewej strony równania ogólnego zależy od postaci równania. Dla punktów leżących po tej samej stronie prostej w przypadku jednego równania otrzymamy wartości dodatnie, a w przypadku innego ujemne.

Zbadamy, czy dwa punkty leżą po tej samej stronie prostej. Specyfikacja problemu jest następująca:

Specyfikacja

Dane: A, B, C – liczby rzeczywiste, współczynniki równania ogólnego prostej, A ≠ 0 lub B ≠ 0,
xP1, yP1, xP2, yP2 – liczby rzeczywiste, współrzędne punktów P_1 i P_2.

Wynik: wartość logiczna **prawda**, gdy punkty P_1 i P_2 leżą po tej samej stronie prostej, lub **fałsz** – w przeciwnym przypadku.

Warto wiedzieć
Jedną prostą można opisać nieskończenie wieloma równaniami ogólnymi – aby otrzymywać kolejne równania, wystarczy mnożyć obie strony równania przez liczby różne od zera. Natomiast każda prosta nieprostopadła do osi OX jest opisana dokładnie jednym równaniem kierunkowym.

Oto zapis algorytmu w pseudokodzie w postaci funkcji:

```
funkcja PktPoTejSamStr(A,B,C,xP1,yP1,xP2,yP2)
   jeśli (A*xP1+B*yP1+C)*(A*xP2+B*yP2+C)>0 to
         zwróć prawda i zakończ
   w przeciwnym przypadku
         zwróć fałsz i zakończ
```

Jeśli znaki wyrażeń A*xP1+B*yP1+C oraz A*xP2+B*yP2+C są takie same, czyli ich iloczyn jest liczbą dodatnią, to punkty leżą po tej samej stronie prostej. Zwróć uwagę, że jeśli któryś z punktów leży na prostej, czyli wartość wyrażenia dla tego punktu jest równa 0, to punkty nie leżą po tej samej stronie i wartością funkcji jest **fałsz**.

Przy implementacji powyższego algorytmu współrzędne punktu można pamiętać w **strukturze**. Oto jej definicja:

Struktura, podręcznik *Informatyka na czasie 2. Zakres rozszerzony*, s. 109

```
struct punkt
{
    float x, y;
};
```

W programie danymi będą współrzędne dwóch punktów. Dlatego warto zdefiniować pomocniczą funkcję CzytajPkt, którą wykorzystamy dwukrotnie: najpierw do wczytania współrzędnych pierwszego punktu, a potem do wczytania współrzędnych drugiego punktu. Kod źródłowy tej funkcji może być następujący:

Kod źródłowy funkcji wczytującej do programu współrzędne punktu

```
void CzytajPkt(string info, punkt &P)
{
    cout<<info<<endl;
    cout<<"x = "; cin>>P.x;
    cout<<"y = "; cin>>P.y;
}
```


Funkcja CzytajPkt ma dwa parametry: info typu string – napis wyświetlany na ekranie z informacją o tym, którego punktu współrzędne wczytujemy, oraz parametr P typu punkt przekazywany przez referencję – współrzędne wczytywanego punktu.

Oto kod źródłowy funkcji PktPoTejSamStr, sprawdzającej, czy dwa dane punkty leżą po tej samej stronie prostej:

Kod źródłowy funkcji sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej – z użyciem równania ogólnego prostej

```
1. bool PktPoTejSamStr(float A, float B, float C,
2.                     punkt P1, punkt P2)
3. {
4.     return ((A*P1.x+B*P1.y+C)*(A*P2.x+B*P2.y+C)>0);
5. }
```

Zwróć uwagę na linię 4. Nie trzeba używać instrukcji warunkowej – wartością funkcji jest wartość logiczna wyrażenia.

Kod źródłowy funkcji main programu badającego, czy dwa punkty leżą po tej samej stronie prostej, może być następujący:

Fragment kodu źródłowego programu badającego, czy dwa punkty leżą po tej samej stronie prostej – funkcja main

```
1. int main()
2. {
3.     float A, B, C;
4.     punkt P1, P2;
5.     cout<<"Podaj wartosci A, B i C ";
6.     cout<<"rownania ogolnego prostej. "<<endl;
7.     cout<<"A = "; cin>>A;
8.     cout<<"B = "; cin>>B;
9.     cout<<"C = "; cin>>C;
10.    CzytajPkt("Podaj wspolrzedne punktu P1. ",P1);
11.    CzytajPkt("Podaj wspolrzedne punktu P2. ",P2);
12.    if (PktPoTejSamStr(A,B,C,P1,P2))
13.        cout<<"Punkty leza po tej samej stronie prostej.";
14.    else
15.        cout<<"Punkty nie leza po tej samej stronie prostej.";
16.    return 0;
17. }
```

W liniach 7–9 wczytujemy dane prostej, a w liniach 10–11 za pomocą funkcji CzytajPkt – współrzędne punktów P_1 i P_2. Instrukcja warunkowa w liniach 12–15 wypisuje odpowiedni komentarz w zależności od wartości funkcji PktPoTejSamStr.

Ćwiczenie 1

Napisz program sprawdzający, czy dwa punkty leżą po tej samej stronie prostej, zgodnie ze specyfikacją podaną na s. 150. Wykorzystaj omówione w temacie funkcje CzytajPkt oraz PktPoTejSamStr.

Zapamiętaj

Do badania położenia punktu względem prostej można wykorzystać równanie ogólne prostej: $Ax + By + C = 0$, gdzie $A \neq 0$ lub $B \neq 0$.
Wartość wyrażenia $Ax + By + C$ dla punktów (x, y) znajdujących się po tej samej stronie prostej ma taki sam znak. Jeśli punkt (x, y) leży na prostej, to wartość wyrażenia $Ax + By + C$ jest równa 0.

Przynależność punktu do odcinka

Odcinek to fragment prostej ograniczony dwoma punktami leżącymi na tej prostej. Jeśli dany punkt należy do odcinka, to należy też do prostej przechodzącej przez końce tego odcinka.

Równanie prostej przechodzącej przez dwa punkty

Równanie prostej przechodzącej przez dwa punkty $A(x_A, y_A)$ oraz $B(x_B, y_B)$ ma postać:

$$(y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) = 0$$

Macierz

Do opisu prostej przechodzącej przez dwa punkty wykorzystuje się też **macierz**. Można ją zilustrować jako tablicę przechowującą określone elementy, np. liczby. W informatyce zwykle stosuje się macierze odpowiadające tablicom dwuwymiarowym. Rysunek 9.2 przedstawia przykład macierzy o trzech wierszach i trzech kolumnach. Liczba kolumn i liczba wierszy w macierzy może być różna.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Rys. 9.2. Przykład macierzy o trzech wierszach i trzech kolumnach, przechowującej liczby całkowite

Macierz kwadratowa

Wyznacznik macierzy

Jeśli macierz ma taką samą liczbę wierszy i kolumn, to nazywa się ją **macierzą kwadratową**. Takiej macierzy można przyporządkować liczbę nazywaną **wyznacznikiem macierzy**. Oznacza się go np. skrótem det (od ang. *determinant* – wyznacznik). Wyznacznik macierzy z rysunku 9.2 zapiszemy tak jak na rysunku 9.3.

Innym sposobem oznaczania wyznacznika jest umieszczenie elementów macierzy między pionowymi kreskami (rys. 9.4).

Warto wiedzieć
Macierze są wykorzystywane w grafice komputerowej. Za ich pomocą można opisać m.in. przekształcenia obiektów na płaszczyźnie, a także w przestrzeni trójwymiarowej, np. obroty.

$$\det\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Rys. 9.3. Oznaczenie wyznacznika macierzy z rysunku 9.2 – sposób 1

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix}$$

Rys. 9.4. Oznaczenie wyznacznika macierzy z rysunku 9.2 – sposób 2

Reguła Sarrusa

W tym temacie będziemy wykorzystywać macierze kwadratowe o trzech wierszach i trzech kolumnach. Dla macierzy o takiej liczbie wierszy i kolumn wyznacznik najwygodniej policzyć, korzystając z **reguły Sarrusa**. Przy obliczaniu wyznacznika wystarczy przepisać pod spodem dwa pierwsze wiersze. Iloczyny elementów znajdujących się na rysunku 9.5 na pomarańczowych liniach zapisujemy ze znakiem plus, a iloczyny elementów na niebieskich liniach – ze znakiem minus.

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} \begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} = 1 \cdot 5 \cdot 9 + 4 \cdot 8 \cdot 3 + 7 \cdot 2 \cdot 6 - 7 \cdot 5 \cdot 3 - 1 \cdot 8 \cdot 6 - 4 \cdot 2 \cdot 9 = 45 + 96 + 84 - 105 - 48 - 72 = 0$$

Rys. 9.5. Ilustracja reguły Sarrusa

Warto wiedzieć
Regułę Sarrusa można stosować tylko do obliczania wyznacznika macierzy o trzech wierszach i trzech kolumnach.

Równanie prostej przechodzącej przez dwa punkty – zapis z wykorzystaniem wyznacznika macierzy

Równanie prostej przechodzącej przez dwa punkty można zapisać **z wykorzystaniem wyznacznika macierzy.** Dla prostej przechodzącej przez punkty $A(x_A, y_A)$ oraz $B(x_B, y_B)$ wygląda ono następująco:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x & y & 1 \end{vmatrix} = 0$$

Obliczmy wyznacznik znajdujący się po lewej stronie równania:

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x & y & 1 \end{vmatrix} = x_A \cdot y_B \cdot 1 + x_B \cdot y \cdot 1 + x \cdot y_A \cdot 1 - x \cdot y_B \cdot 1 - x_A \cdot y \cdot 1 - x_B \cdot y_A \cdot 1 = x_A \cdot y_B + x_B \cdot y + x \cdot y_A - x \cdot y_B - x_A \cdot y - x_B \cdot y_A$$

Po przekształceniu lewej strony równania prostej z poprzedniej strony otrzymujemy wyliczony powyżej wyznacznik:

$(y - y_A)(x_B - x_A) - (y_B - y_A)(x - x_A) =$
$= x_B \cdot y - x_A \cdot y - x_B \cdot y_A + x_A \cdot y_A - x \cdot y_B + x_A \cdot y_B + x \cdot y_A - x_A \cdot y_A =$
$= x_A \cdot y_B + x_B \cdot y + x \cdot y_A - x \cdot y_B - x_A \cdot y - x_B \cdot y_A$

Przynależność punktu do odcinka

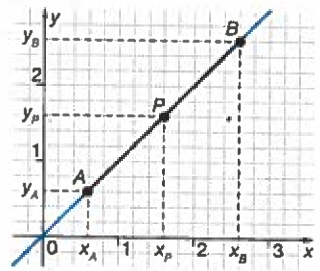
Podczas badania **przynależności punktu do odcinka**, np. punktu $P(x_P, y_P)$ do odcinka AB o końcach w punktach $A(x_A, y_A)$ i $B(x_B, y_B)$, sprawdzimy najpierw, czy należy on do prostej przechodzącej przez punkty A i B. W tym celu do równania prostej wykorzystującego wyznacznik macierzy podstawiamy w miejsce x i y współrzędne punktu P. Jeśli wyznacznik będzie równy 0, to punkty A, B, P są współliniowe. Wyznacznik macierzy dla punktów A, B, P oznaczamy $\det(A, B, P)$:

$$\det(A, B, P) = \begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_P & y_P & 1 \end{vmatrix}$$

Jeśli punkt P należy do prostej, to trzeba jeszcze sprawdzić, czy punkt ten znajduje się pomiędzy punktami A i B. Gdy tak jest, prawdziwe są poniższe nierówności (min oznacza minimum, a max – maksimum).

$\min(x_A, x_B) \leqslant x_P \leqslant \max(x_A, x_B)$
$\min(y_A, y_B) \leqslant y_P \leqslant \max(y_A, y_B)$

Zależność między współrzędnymi punktu P a współrzędnymi końców odcinka AB ilustruje rysunek 9.6.



Rys. 9.6. Zależność między współrzędnymi współliniowych punktów A, B, P, gdzie P – punkt należący do odcinka AB

Oto specyfikacja problemu sprawdzania, czy punkt należy do odcinka:

Specyfikacja

Dane: xA, yA, xB, yB – liczby rzeczywiste, współrzędne różnych punktów A i B tworzących odcinek AB,
xP, yP – liczby rzeczywiste, współrzędne punktu P.

Wynik: wartość logiczna **prawda**, jeżeli punkt P należy do odcinka AB, lub **fałsz** – w przeciwnym przypadku.

Algorytm sprawdzający, czy punkt należy do odcinka, można zapisać w pseudokodzie w postaci funkcji tak jak poniżej. Zakładamy, że mamy do dyspozycji funkcję det, obliczającą wyznacznik wykorzystywany w równaniu prostej dla trzech danych punktów, oraz funkcje min i max, wyznaczające odpowiednio minimum i maksimum z dwóch liczb.

```
funkcja PktWOdcinku(xA,yA,xB,yB,xP,yP)
   jeśli det(xA,yA,xB,yB,xP,yP) ≠ 0 to
      zwróć fałsz i zakończ
   jeśli xP ≥ min(xA,xB) oraz xP ≤ max(xA,xB) oraz
         yP ≥ min(yA,yB) oraz yP ≤ max(yA,yB) to
         zwróć prawda i zakończ
   w przeciwnym przypadku
         zwróć fałsz i zakończ
```

Oto kod źródłowy funkcji det oraz funkcji PktWOdcinku:

Kod źródłowy funkcji obliczającej wyznacznik macierzy dla współrzędnych trzech punktów oraz funkcji sprawdzającej, czy punkt P należy do odcinka AB

```
1.  float det(float x1, float y1,
2.            float x2, float y2,
3.            float x3, float y3)
4.  {
5.     return (y3-y1)*(x2-x1)-(y2-y1)*(x3-x1);
6.  }
7.
8.  bool PktWOdcinku(punkt A, punkt B, punkt P)
9.  {
10.    float w=det(A.x,A.y,B.x,B.y,P.x,P.y);
11.    if (abs(w)>EPS) return false;
12.    return (P.x>=min(A.x,B.x) && P.x<=max(A.x,B.x) &&
13.            P.y>=min(A.y,B.y) && P.y<=max(A.y,B.y));
14. }
```

Równanie prostej przechodzącej przez dwa punkty, s. 152

W funkcji det skorzystaliśmy z **równania prostej przechodzącej przez dwa punkty** bez wyznacznika macierzy, ponieważ w tym przypadku program wykona mniej działań mnożenia. Aby poprawić czytelność zapisu funkcji PktWOdcinku, zapamiętaliśmy wyznacznik w zmiennej pomocniczej w (linia 10). Zwróć uwagę, że nie porównujemy wyznacznika wprost z zerem, tylko z dokładnością do stałej EPS ze względu na przybliżoną reprezentację liczb rzeczywistych (linia 11). Kiedy badamy tylko znak wyrażenia, nie musimy używać stałej EPS. W liniach 12–13 wykorzystujemy predefiniowane **funkcje min** i **max**, których wartościami są odpowiednio mniejszy lub większy element z dwóch parametrów.

Funkcja min, funkcja max

Rysunek 9.7 przedstawia przykład wykonania programu sprawdzającego, czy punkt należy do odcinka.

```
Podaj wspolrzedne punktu A.
x = 0
y = 0
Podaj wspolrzedne punktu B.
x = 2
y = 2
Podaj wspolrzedne punktu P.
x = 1
y = 1
Punkt P nalezy do odcinka AB.
```

Rys. 9.7. Przykład wykonania programu sprawdzającego, czy punkt należy do odcinka

Dobra rada

Jeśli chcesz porównywać wartości z dokładnością do stałej, pamiętaj, aby ją zdefiniować, np.: `const float EPS=0.000001;`

Dobra rada

W funkcjach min i max możesz dodatkowo określić trzeci parametr funkcji, który będzie nazwą funkcji logicznej wskazującej sposób porównania dwóch pierwszych parametrów.

Ćwiczenie 2

Napisz program sprawdzający, czy punkt należy do odcinka, zgodnie ze specyfikacją podaną na s. 154. Wykorzystaj omówione w temacie funkcje CzytajPkt, det oraz PktWOdcinku.

Zapamiętaj

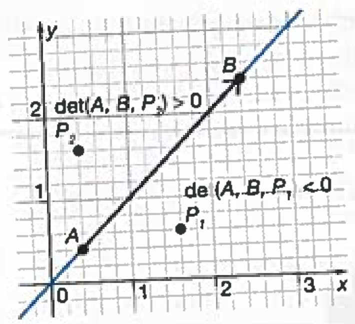
Do badania przynależności punktu do odcinka można wykorzystać równanie prostej przechodzącej przez dwa punkty. Badany punkt musi należeć do prostej, na której leży odcinek, a jego współrzędne muszą się zawierać pomiędzy współrzędnymi końców odcinka.

Położenie punktów względem prostej zawierającej wektor

Rozważmy **wektor** $\overrightarrow{AB}$ zawarty w prostej. W równaniu prostej przechodzącej przez punkty A i B, zapisanym z wykorzystaniem wyznacznika macierzy, kolejność punktów w dwóch pierwszych wierszach wyznacznika ma znaczenie. Współrzędne punktu A z pierwszego wiersza to współrzędne początku wektora $\overrightarrow{AB}$, a współrzędne punktu B z drugiego wiersza to współrzędne końca wektora.

Wektor to uporządkowana para punktów – jeden z tych punktów jest początkiem, a drugi końcem wektora. Każdy wektor ma kierunek, zwrot i wartość.

Na podstawie wyznacznika można określić położenie punktu P względem prostej zawierającej wektor $\overrightarrow{AB}$. Jeśli $\det(A, B, P) > 0$, to punkt P leży po lewej stronie prostej, a jeśli $\det(A, B, P) < 0$, to po prawej stronie. Ilustruje to rysunek 9.8 na s. 156.



Rys. 9.8. Położenie punktów P_1 i P_2 względem prostej zawierającej wektor $\overrightarrow{AB}$

Kod źródłowy funkcji sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej – z użyciem równania ogólnego prostej, s. 151

Wcześniej w zapisie **funkcji** PktPoTejSamStr, **sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej**, zastosowaliśmy równanie ogólne prostej. Teraz zapiszemy kod źródłowy tej funkcji, korzystając z równania prostej przechodzącej przez dwa punkty.

Kod źródłowy funkcji sprawdzającej, czy dwa punkty leżą po tej samej stronie prostej – z wykorzystaniem równania prostej przechodzącej przez dwa punkty

```
bool PktPoTejSamStr(punkt A, punkt B,
                    punkt P1, punkt P2)
{
    float w1, w2;
    w1=det(A.x,A.y,B.x,B.y,P1.x,P1.y);
    w2=det(A.x,A.y,B.x,B.y,P2.x,P2.y);
    return (w1*w2>0);
}
```

Ćwiczenie 3

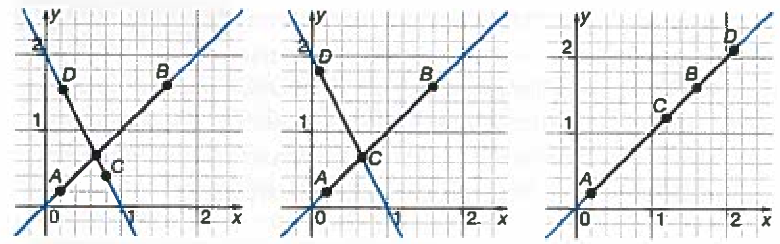
Napisz program sprawdzający, czy dwa punkty P_1 i P_2 leżą po tej samej stronie prostej przechodzącej przez dwa dane punkty A i B.

9.2. Przecinanie się dwóch odcinków

Dwa odcinki AB i CD się przecinają, jeśli mają co najmniej jeden punkt wspólny. Możemy wyróżnić następujące sytuacje.

1. Punkty A i B leżą po przeciwnych stronach prostej przechodzącej przez punkty C i D oraz punkty C i D leżą po przeciwnych stronach prostej przechodzącej przez punkty A i B. Wówczas istnieje dokładnie jeden punkt wspólny.
2. Początek lub koniec jednego odcinka należy do drugiego odcinka. Wówczas odcinki mają jeden punkt wspólny, jeśli leżą na prostych przecinających się, lub więcej niż jeden punkt wspólny – gdy leżą na tej samej prostej.

Możliwe wzajemne położenia przecinających się odcinków AB oraz CD pokazuje rysunek 9.9.



Rys. 9.9. Wzajemne położenia przecinających się odcinków AB oraz CD

Żeby rozpatrzeć pierwszy z wymienionych przypadków, wystarczy policzyć cztery wyznaczniki z równania prostej przechodzącej przez dwa punkty – odpowiednio det(A, B, C), det(A, B, D), det(C, D, A) oraz det(C, D, B). Ponieważ punkty mają leżeć po przeciwnych stronach prostej, odpowiednie wyznaczniki muszą mieć przeciwne znaki. Wystarczy więc sprawdzić, czy iloczyny det(A, B, C) · det(A, B, D) oraz det(C, D, A) · det(C, D, B) są ujemne.

W drugim z wymienionych przypadków co najmniej jeden z wyznaczników rozpatrywanych w pierwszym przypadku będzie równy zero. Należy dodatkowo sprawdzić, czy punkt, dla którego wyznacznik jest równy zero, należy do drugiego odcinka.

Oto specyfikacja problemu sprawdzenia, czy dwa odcinki się przecinają, oraz zapis algorytmu w postaci funkcji w pseudokodzie:

Specyfikacja

Dane: xA, yA, xB, yB, xC, yC, xD, yD – liczby rzeczywiste, współrzędne punktów A, B, C i D tworzących odcinki AB i CD.

Wynik: wartość logiczna **prawda**, gdy odcinki AB i CD się przecinają, lub **fałsz** – w przeciwnym przypadku.

```
funkcja PrzecOdcinki(xA,yA,xB,yB,xC,yC,xD,yD)
   w1 ← det(xA,yA,xB,yB,xC,yC)
   w2 ← det(xA,yA,xB,yB,xD,yD)
   w3 ← det(xC,yC,xD,yD,xA,yA)
   w4 ← det(xC,yC,xD,yD,xB,yB)
   jeśli w1 * w2 < 0 oraz w3 * w4 < 0 to
      zwróć prawda i zakończ
   jeśli PktWOdcinku(xA,yA,xB,yB,xC,yC) to
      zwróć prawda i zakończ
   jeśli PktWOdcinku(xA,yA,xB,yB,xD,yD) to
      zwróć prawda i zakończ
   jeśli PktWOdcinku(xC,yC,xD,yD,xA,yA) to
      zwróć prawda i zakończ
   jeśli PktWOdcinku(xC,yC,xD,yD,xB,yB) to
      zwróć prawda i zakończ
   zwróć fałsz i zakończ
```

W zapisie funkcji `PrzecOdcinki` zastosowaliśmy dwie funkcje omówione w tym temacie: funkcję `det`, obliczającą wyznacznik, oraz funkcję `PktWOdcinku`, sprawdzającą przynależność punktu do odcinka. Zwróć uwagę, że funkcję `PktWOdcinku` na potrzeby tego problemu można uprościć, wykorzystując obliczone wyznaczniki.

Kod źródłowy funkcji `PrzecOdcinki` jest następujący:

Kod źródłowy funkcji sprawdzającej, czy dwa odcinki się przecinają

```
bool PrzecOdcinki(punkt A, punkt B, punkt C, punkt D)
{
    float w1, w2, w3, w4;
    w1=det(A.x,A.y,B.x,B.y,C.x,C.y);
    w2=det(A.x,A.y,B.x,B.y,D.x,D.y);
    w3=det(C.x,C.y,D.x,D.y,A.x,A.y);
    w4=det(C.x,C.y,D.x,D.y,B.x,B.y);
    if (w1*w2<0 && w3*w4<0) return true;
    if (PktWOdcinku(A,B,C)) return true;
    if (PktWOdcinku(A,B,D)) return true;
    if (PktWOdcinku(C,D,A)) return true;
    if (PktWOdcinku(C,D,B)) return true;
    return false;
}
```

Rysunek 9.10 przedstawia przykład wykonania programu sprawdzającego, czy dwa odcinki się przecinają.

```
Podaj wspolrzedne koncow odcinka AB.
Wspolrzedne punktu A.
x = 0
y = 0
Wspolrzedne punktu B.
x = 2
y = 2
Podaj wspolrzedne koncow odcinka CD.
Wspolrzedne punktu C.
x = 0
y = 2
Wspolrzedne punktu D.
x = 2
y = 0
Odcinki sie przecinaja.
```

Rys. 9.10. Przykład wykonania programu sprawdzającego, czy dwa odcinki się przecinają

Ćwiczenie 4

Napisz program sprawdzający, czy dwa odcinki się przecinają, zgodnie ze specyfikacją podaną na s. 157.

Zapamiętaj

Żeby zbadać, czy odcinki *AB* i *CD* się przecinają, wystarczy sprawdzić, czy punkty *A* i *B* leżą po przeciwnych stronach prostej przechodzącej przez punkty *C* i *D* oraz czy punkty *C* i *D* leżą po przeciwnych stronach prostej przechodzącej przez punkty *A* i *B*. Dodatkowo należy rozpatrzeć przypadki, gdy koniec jednego odcinka należy do drugiego odcinka.

9.3. Przynależność punktu do wielokąta wypukłego

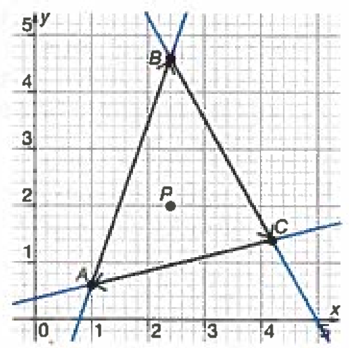
Najpierw sprawdzimy, czy punkt należy do trójkąta. Następnie zbadamy, czy punkt należy do dowolnego **wielokąta wypukłego**.

Wielokąt wypukły to wielokąt, w którym każdy z kątów ma miarę mniejszą niż 180°.

Wektor, s. 155

Przynależność punktu do trójkąta

Boki trójkąta możemy potraktować jako **wektory**. Koniec jednego wektora będzie początkiem następnego. Jeśli punkt P należy do trójkąta i nie leży na żadnym z jego boków, to musi znajdować się po tej samej stronie każdego z wektorów. Ilustruje to rysunek 9.11: punkt P znajduje się po prawej stronie wektorów $\overrightarrow{AB}$, $\overrightarrow{BC}$ i $\overrightarrow{CA}$. Wówczas znaki wyznaczników z równania prostej przechodzącej przez dwa punkty, odpowiednio $\det(A, B, P)$, $\det(B, C, P)$, $\det(C, A, P)$, są takie same.



Rys. 9.11. Ilustracja przynależności punktu *P* do trójkąta *ABC*, gdy punkt *P* nie leży na żadnym z boków trójkąta

Jeśli punkt będzie leżał na którymś z boków trójkąta (może być też jego wierzchołkiem), wtedy co najmniej jeden z wyznaczników będzie równy zero.

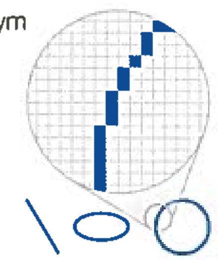
Dobra rada

Jeśli każdy odcinek o końcach należących do danego wielokąta zawiera się w wielokącie, to wielokąt ten jest wypukły. Zwróć uwagę, że każdy trójkąt jest wielokątem wypukłym.

A to ciekawe

Jak na ekranie komputera rysowane są odcinki, okręgi, elipsy?

W grafice rastrowej obraz jest siatką pikseli. Aby dany kształt był jak najwierniejszym odwzorowaniem oczekiwanego, trzeba wybrać te piksele, które muszą zostać pokolorowane. W 1962 r. amerykański informatyk Jack Bresenham opracował algorytm, który umożliwia rysowanie odcinka na urządzeniach rastrowych. Wykorzystuje on operacje tylko na liczbach całkowitych, co jest ważne przy implementacjach algorytmu (nie wymaga to czasochłonnych operacji na liczbach rzeczywistych). Algorytm Bresenhama przystosowano później do rysowania innych kształtów, np. okręgu czy elipsy.



Oto specyfikacja problemu przynależności punktu do trójkąta:

Specyfikacja

Dane: xA, yA, xB, yB, xC, yC – liczby rzeczywiste, współrzędne niewspółliniowych punktów A, B, C tworzących trójkąt ABC,
xP, yP – liczby rzeczywiste, współrzędne punktu P.

Wynik: wartość logiczna `prawda`, jeżeli punkt P należy do trójkąta ABC, lub `fałsz` – w przeciwnym przypadku.

Funkcję `PktWTrojkacie`, realizującą opisany algorytm badania przynależności punktu do trójkąta, można zapisać w pseudokodzie następująco:

```
funkcja PktWTrojkacie(xA,yA,xB,yB,xC,yC,xP,yP)
    w1 ← det(xA,yA,xB,yB,xP,yP)
    w2 ← det(xB,yB,xC,yC,xP,yP)
    jeśli w1 * w2 < 0 to zwróć fałsz i zakończ
    w2 ← det(xC,yC,xA,yA,xP,yP)
    jeśli w1 * w2 < 0 to zwróć fałsz i zakończ
    zwróć prawda i zakończ
```

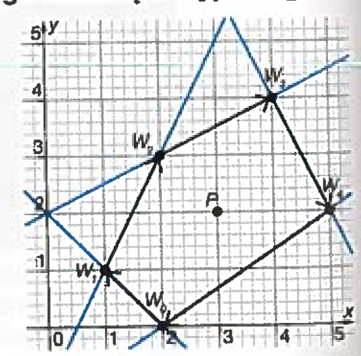
Obliczamy wyznaczniki w1 i w2, określające położenie punktu P względem prostych wyznaczonych przez boki AB i BC. Jeśli iloczyn wyznaczników jest ujemny, to punkt P nie należy do trójkąta. W przeciwnym przypadku obliczamy nową wartość zmiennej w2 dla trzeciego wyznacznika (dla prostej przechodzącej przez punkty C oraz A) i ponownie badamy znak iloczynu wyznaczników w1 i w2.

Ćwiczenie 5

Napisz program sprawdzający, czy punkt P należy do trójkąta ABC, zgodnie ze specyfikacją podaną powyżej.

Przynależność punktu do dowolnego wielokąta wypukłego

Omówiony algorytm badania przynależności punktu do trójkąta można zastosować dla dowolnego wielokąta wypukłego. Załóżmy, że punkty $W_i(x_i, y_i)$, gdzie $i = 0, ..., n-1$, są wierzchołkami wielokąta wypukłego o n bokach. Odcinki W_iW_{i+1} dla $i = 0, ..., n-2$ oraz $W_{n-1}W_0$ tworzą kolejne boki wielokąta. Należy sprawdzić, czy punkt P leży po tej samej stronie wszystkich prostych wyznaczonych przez wektory $\overline{W_iW_{i+1}}$ i wektor $\overline{W_{n-1}W_0}$. Ilustruje to rysunek 9.12.



Rys. 9.12. Ilustracja przynależności punktu P do wielokąta wypukłego, gdy punkt P nie leży na żadnym z jego boków

Specyfikacja problemu badania przynależności punktu do wielokąta wypukłego może być następująca:

Specyfikacja

Dane: n – liczba wierzchołków wielokąta wypukłego, n ⩾ 3,
`W[0..n-1]` – tablica przechowująca współrzędne punktów będących kolejnymi wierzchołkami wielokąta wypukłego, każdy element tablicy pamięta informację o jednym punkcie, żadne trzy punkty nie są współliniowe, a współrzędne punktu to dwie liczby rzeczywiste (x_i, y_i), $i = 0, ..., n-1$,
xP, yP – liczby rzeczywiste, współrzędne punktu P.

Wynik: wartość logiczna `prawda`, gdy punkt P należy do wielokąta wypukłego o wierzchołkach z tablicy `W`, lub `fałsz` – w przeciwnym przypadku.

Oto funkcja `PktWWielWyp`, realizująca opisany algorytm badania przynależności punktu do wielokąta wypukłego, zapisana w pseudokodzie:

```
funkcja PktWWielWyp(n,W,xP,yP)
   w1 ← det(W[0].x,W[0].y,W[1].x,W[1].y,xP,yP)
   dla i ← 1, ..., n - 1 wykonuj
      w2 ← det(W[i].x,W[i].y,W[(i+1) mod n].x,
               W[(i+1) mod n].y,xP,yP)
      jeśli w1 * w2 < 0 to zwróć fałsz i zakończ
   zwróć prawda i zakończ
```

Obliczamy najpierw wyznacznik określający położenie punktu P względem prostej wyznaczonej przez dwa pierwsze wierzchołki z tablicy `W` (zmienna w1). Następnie w pętli obliczamy wyznaczniki określające położenie punktu P względem prostych wyznaczonych przez końce kolejnych boków (zmienna w2). Zwróć uwagę, że indeks drugiego wierzchołka dzielimy z resztą przez n. Dzięki temu dla i = n − 1 otrzymamy wartość `(i + 1) mod n = 0`, czyli indeks końcowego wierzchołka ostatniego boku (czyli `W[0]`). Jeśli iloczyn w1*w2 dla którejś z wartości w2 będzie ujemny, to punkt P nie należy do wielokąta.

Dane opisujące wielokąt wygodnie jest przygotować w pliku tekstowym. Pierwszy wiersz może zawierać jedną liczbę całkowitą n określającą liczbę wierzchołków, następne n wierszy – po dwie liczby rzeczywiste będące współrzędnymi kolejnych wierzchołków wielokąta. Na przykład opis wielokąta wypukłego z rysunku 9.12 będzie taki jak na rysunku 9.13.

```
5
2 0
1 1
2 3
4 4
5 2
```

Rys. 9.13. Opis pięciokąta z rysunku 9.12 w pliku tekstowym

Oto kod źródłowy funkcji `CzytajWiel`, wczytującej opis wielokąta z pliku tekstowego:

Kod źródłowy funkcji wczytującej z pliku tekstowego informację o liczbie wierzchołków wielokąta wypukłego oraz współrzędne tych wierzchołków

```
1. void CzytajWiel(vector<punkt> &W)
2. {
3.     int i, n;
4.     ifstream we("wielwyp.txt");
5.     we>>n; W.resize(n);
6.     for (i=0;i<n;i++) we>>W[i].x>>W[i].y;
7.     we.close();
8. }
```

Parametr `W` funkcji `CzytajWiel` jest przekazywany przez referencję. Jest on typu `vector`. Każdy jego element jest strukturą typu punkt, która pamięta współrzędne jednego wierzchołka. W linii 4 otwierany jest plik *wielwyp.txt* z opisem wielokąta, w linii 5 wczytywana jest liczba wierzchołków i ustalany jest rozmiar parametru `W`. Pętla w linii 6 wczytuje współrzędne kolejnych punktów.

Kod źródłowy funkcji `PktWWielWyp`, sprawdzającej przynależność punktu do wielokąta wypukłego, jest następujący:

Kod źródłowy funkcji sprawdzającej, czy punkt należy do wielokąta wypukłego

```
bool PktWWielWyp(vector<punkt> W, punkt P)
{
    int n=W.size();
    float w1, w2;
    w1=det(W[0].x,W[0].y,W[1].x,W[1].y,P.x,P.y);
    for (int i=1;i<n;i++)
    {
        w2=det(W[i].x,W[i].y,W[(i+1)%n].x,W[(i+1)%n].y,P.x,P.y);
        if (w1*w2<0) return false;
    }
    return true;
}
```

Ćwiczenie 6

Napisz program sprawdzający, czy punkt P należy do wielokąta, zgodnie ze specyfikacją podaną na s. 161. Opis wielokąta wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *wielwyp.txt*).

Zapamiętaj

Aby określić przynależność punktu P do wielokąta wypukłego, wystarczy sprawdzić położenie punktu P względem prostych wyznaczonych przez wektory tworzące boki wielokąta. Koniec jednego wektora jest początkiem następnego.

Podsumowanie

- Położenie punktów względem prostej można określić, korzystając z równania ogólnego prostej $Ax + By + C = 0$. Współrzędne punktu podstawiamy do lewej strony równania. Jeśli punkt należy do prostej, to wartość wyrażenia jest równa 0. Dla punktów znajdujących się po tej samej stronie prostej znaki otrzymanych wyników są takie same.
- Żeby zbadać, czy punkt P należy do odcinka AB, trzeba sprawdzić, czy należy on do prostej przechodzącej przez punkty A i B oraz czy współrzędne punktu P zawierają się pomiędzy współrzędnymi punktów A i B.
- Równanie prostej przechodzącej przez dwa punkty można zapisać z wykorzystaniem wyznacznika macierzy. Dla punktu P leżącego po prawej stronie prostej zawierającej wektor $\overline{AB}$ wartość $\det(A, B, P)$ jest ujemna, a dla punktu leżącego po lewej stronie prostej – dodatnia. Dla punktów leżących na prostej wyznacznik ma wartość 0.
- Żeby zbadać przecinanie się odcinków AB i CD, wystarczy sprawdzić, czy punkty A i B leżą po przeciwnych stronach prostej przechodzącej przez punkty C i D oraz czy punkty C i D leżą po przeciwnych stronach prostej przechodzącej przez punkty A i B. Dodatkowo należy rozpatrzeć przypadki, gdy koniec jednego odcinka należy do drugiego odcinka.
- Badając przynależność punktu do wielokąta wypukłego, sprawdzamy położenie tego punktu względem prostych wyznaczonych przez wektory tworzące boki wielokąta. Koniec jednego wektora jest początkiem następnego. Pierwszy wierzchołek wielokąta jest początkiem wektora wyznaczającego pierwszy bok i równocześnie końcem wektora tworzącego ostatni bok.
- Przynależność punktu do trójkąta badamy tak samo jak przynależność do dowolnego wielokąta wypukłego. Każdy trójkąt jest wielokątem wypukłym.

Zadania

1 Napisz program obliczający pole trójkąta ABC, którego wierzchołkami są niewspółliniowe punkty $A(x_A, y_A)$, $B(x_B, y_B)$ i $C(x_C, y_C)$.

Wskazówka: Możesz skorzystać z poniższego wzoru na pole trójkąta ABC.

$$P_{ABC} = \frac{1}{2} \cdot \left|(x_B - x_A) \cdot (y_C - y_A) - (x_C - x_A) \cdot (y_B - y_A)\right|$$

2 Punktem kratowym nazywamy punkt o współrzędnych całkowitych. Wczytaj z klawiatury współrzędne dwóch różnych punktów kratowych A i B o wartościach każdej ze współrzędnych z zakresu od 1 do 9. Oblicz liczbę punktów kratowych znajdujących się po każdej ze stron prostej przechodzącej przez punkty A i B w obrębie kwadratu, którego wierzchołki znajdują się w punktach o współrzędnych (0, 0), (0, 10), (10, 10) i (10, 0). Na przykład dla punktów $A(1, 1)$ i $B(2, 2)$ po obu stronach prostej przechodzącej przez te punkty znajduje się po 55 punktów kratowych spełniających warunki zadania.

* **3** Napisz program sprawdzający położenie punktu P względem prostej zawierającej wektor $\overrightarrow{AB}$, gdzie punkt A leży poniżej punktu B. Dane punktów A, B i P wczytaj z klawiatury. Program powinien wypisać jeden z komunikatów: punkt leży na prostej poniżej punktu A, punkt należy do odcinka AB, punkt leży na prostej powyżej punktu B, punkt nie należy do prostej przechodzącej przez punkty A i B.

** **4** Punktem kratowym nazywamy punkt o współrzędnych całkowitych. Napisz program, który obliczy liczbę punktów kratowych należących do trójkąta ABC, którego wierzchołki mają współrzędne całkowite. Załóż, że boki trójkąta nie zawierają punktów kratowych poza wierzchołkami trójkąta.

** **5** Napisz program obliczający pole wielokąta wypukłego. Dane wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *wielokat_1.txt*). W pliku w pierwszym wierszu znajduje się jedna liczba całkowita dodatnia $n \geqslant 3$, określająca liczbę wierzchołków wielokąta. W następnych n wierszach pliku znajdują się po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne kolejnych wierzchołków w kartezjańskim układzie współrzędnych.

*** **6** Napisz program, który sprawdzi przynależność punktu P do wielokąta wklęsłego. Dane wielokąta wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *wielokat_2.txt*). W pliku w pierwszym wierszu znajduje się jedna liczba całkowita dodatnia $n \geqslant 3$, określająca liczbę wierzchołków wielokąta. W następnych n wierszach pliku znajdują się po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne kolejnych wierzchołków w kartezjańskim układzie współrzędnych. Współrzędne punktu P wczytaj z klawiatury.

*** **7** Napisz program znajdujący tzw. wypukłą otoczkę zbioru punktów. Wypukła otoczka to podzbiór zbioru punktów tworzący wielokąt wypukły zawierający wszystkie punkty zbioru. Dane do programu wczytaj z pliku tekstowego, który otrzymasz od nauczyciela (np. *punkty_1.txt*). W pliku w pierwszym wierszu znajduje się jedna liczba całkowita dodatnia $n \geqslant 3$, określająca liczbę punktów, w następnych n wierszach pliku znajdują się po dwie liczby rzeczywiste oddzielone spacją, określające współrzędne kolejnych punktów w kartezjańskim układzie współrzędnych. Wynik, czyli współrzędne punktów tworzących wypukłą otoczkę, zapisz w pliku tekstowym (np. *punkty_1_wynik.txt*) – w każdym wierszu pliku współrzędne jednego punktu oddzielone spacją.

10. Fraktale

W przyrodzie można zaobserwować wiele obiektów, których fragmenty są podobne do całości. Podobieństwo znajdziemy np. w liściu paproci, kalafiorze czy drzewach. Dzięki rozwojowi technik komputerowych rysowanie przybliżeń takich obiektów jest ułatwione. W tym temacie zajmiemy się algorytmami, które to umożliwiają, i narysujemy przybliżenia wybranych figur.

Cele lekcji

- Dowiesz się, czym jest fraktal.
- Poznasz zbiór Cantora.
- Narysujesz krzywą Kocha oraz płatek Kocha.
- Dowiesz się, czym jest drzewo binarne, i utworzysz przykład takiej figury.
- Narysujesz dywan Sierpińskiego.
- Zastosujesz rekurencję oraz język JavaScript z biblioteką p5.js.
- Poznasz metodę IFS, służącą do generowania fraktali.

Fraktal to figura, którą cechuje m.in. samopodobieństwo, czyli fragmenty tej figury są podobne do całości. Pierwsze znane opisy takich figur pochodzą z XIX w., choć wówczas nie używano jeszcze pojęcia „fraktal".

Fraktal

10.1. Zbiór Cantora

Zbiór Cantora jest podzbiorem odcinka jednostkowego (o długości 1), który można opisać w następujący sposób:

Zbiór Cantora

- zbiór Cantora stopnia 0 to odcinek jednostkowy,
- zbiór Cantora stopnia 1 otrzymujemy z odcinka jednostkowego: dzielimy go na trzy równe części i usuwamy część środkową,
- zbiór Cantora wyższego stopnia tworzymy ze zbioru Cantora stopnia o jeden mniejszego: dzielimy każdy jego odcinek na trzy równe części i usuwamy części środkowe.

Proces dzielenia odcinków tworzących zbiór Cantora i usuwania z nich części środkowych jest nieskończony, więc będziemy generować jedynie zbiory Cantora określonego stopnia. Rysunek 10.1 ilustruje zbiory Cantora stopni od 0 do 3.

Rys. 10.1. Zbiory Cantora stopni od 0 do 3 (w danej linii przedstawiony jest zbiór jednego stopnia)

Warto wiedzieć

Zbiór Cantora odkrył w 1875 r. irlandzki matematyk Henry J.S. Smith, opisał go natomiast niemiecki matematyk Georg F.L.P. Cantor w 1883 r. Prace Cantora są kluczowe dla podstaw współczesnej matematyki.