# Sprawozdanie – Otoczka wypukła, algorytm Jarvisa

## Cel ćwiczenia

* Poznanie wybranych problemów geometrycznych.
* Zrozumienie problemu otoczki wypukłej.

## Podstawowe zagadnienia

### Algorytmy geometryczne – rozwiązywane problemy

Algorytmy geometryczne zajmują się rozwiązywaniem problemów związanych z obiektami geometrycznymi, takimi jak punkty, odcinki, wielokąty czy okręgi. Przykładowe zagadnienia to: wyznaczanie otoczki wypukłej, przecięcia odcinków, triangulacja, czy znajdowanie najbliższej pary punktów.

### Otoczka wypukła

Otoczka wypukła zbioru punktów to najmniejszy zbiór wypukły, który zawiera wszystkie punkty tego zbioru. Formalnie, dla danego zbioru punktów ( P ), otoczka wypukła ( S ) to taki zbiór wypukły, że każdy punkt z ( P ) należy do wnętrza lub brzegu ( S ).

### Pojęcia wykorzystywane w rozwiązaniu

* **Punkt** – para współrzędnych (x, y).
* **Otoczka wypukła** – minimalny wielokąt wypukły zawierający wszystkie punkty.
* **Kąt biegunowy** – kąt pomiędzy dodatnią półosią OX a wektorem do danego punktu.
* **Orientacja** – określenie, czy trzy punkty są współliniowe, tworzą skręt w lewo czy w prawo.

## Wyznaczanie otoczki wypukłej zbioru P

Celem jest znalezienie najmniejszego zbioru wypukłego ( S ), takiego że wszystkie punkty ( P ) znajdują się w jego wnętrzu lub na brzegu.

## Algorytm Jarvisa (Gift Wrapping)

**Dane:** zbiór punktów ( P )  
**Wynik:** podzbiór ( S ) – punkty otoczki wypukłej

### Główne kroki:

1. Wyznacz punkt ( P\_0 ) o najmniejszej współrzędnej y (jeśli kilka, to o najmniejszej x) oraz ( Q\_0 ) o największej współrzędnej y.
2. Ustaw ( r = P\_0 ).
3. Powtarzaj, dopóki ( r Q\_0 ):
   * Dodaj ( r ) do ( S ).
   * Wyznacz punkt ( t ) ze zbioru ( P ), który ma najmniejszy kąt biegunowy względem ( r ) (dla dodatniej półosi OX).
   * Ustaw ( r = t ).
4. Ustaw ( r = Q\_0 ).
5. Powtarzaj, dopóki ( r P\_0 ):
   * Dodaj ( r ) do ( S ).
   * Wyznacz punkt ( t ) ze zbioru ( P ), który ma najmniejszy kąt biegunowy względem ( r ) (dla ujemnej półosi OX).
   * Ustaw ( r = t ).
6. Zwróć ( S ).

*Uwaga: Można połączyć pętlę od ( P\_0 ) do ( Q\_0 ) i od ( Q\_0 ) do ( P\_0 ) w jedną.*

## Przykład

Dane punkty:

0 0; 0 2; -2 2; 4 2; 3 3; 2 5; 5 5; 1 6; 0 8;

Wizualizacja:

Przykład otoczki wypukłej

Wynik:  
( S = { 0 0; 4 2; 5 5; 0 8; -2 2; } )

## Zadanie

* Napisanie w języku C++ programu implementującego algorytm Jarvisa dla problemu wyznaczenia otoczki wypukłej.
* Użytkownik powinien mieć możliwość podania liczby punktów oraz ich współrzędnych (z klawiatury).
* Przetestowanie programu (poprawność działania, odporność na błędy, itp.).
* Przygotowanie sprawozdania.

## Przykład działania programu

**Dane wejściowe:**

9  
0 0  
0 2  
-2 2  
4 2  
3 3  
2 5  
5 5  
1 6  
0 8

**Wynik:**

0 0;  
4 2;  
5 5;  
0 8;  
-2 2;

## Opis kluczowych fragmentów kodu

### Klasa Point

Reprezentuje punkt na płaszczyźnie:

class Point {  
public:  
 int x, y;  
 explicit Point(int x = 0, int y = 0);  
 bool operator==(const Point& other) const;  
 bool operator<(const Point& other) const;  
};

Dzięki przeciążeniu operatorów możliwe jest łatwe porównywanie i sortowanie punktów oraz eliminacja duplikatów.

### Funkcja orientation

Określa orientację trzech punktów (czy skręcają w lewo, prawo, czy są współliniowe):

int JarvisMarch::orientation(const Point& p, const Point& q, const Point& r) {  
 int val = (q.y - p.y) \* (r.x - q.x) -  
 (q.x - p.x) \* (r.y - q.y);  
 if (val == 0) return 0;  
 return (val > 0) ? 1 : 2;  
}

Jest to kluczowe dla wyboru kolejnego punktu otoczki.

### Algorytm Jarvisa (convexHull)

Główna funkcja wyznaczająca otoczkę wypukłą:

std::vector<Point> JarvisMarch::convexHull(const std::vector<Point>& points) {  
 int n = points.size();  
 if (n < 3) return {};  
  
 std::vector<Point> hull;  
 int l = 0;  
 for (int i = 1; i < n; i++)  
 if (points[i].x < points[l].x ||  
 (points[i].x == points[l].x && points[i].y < points[l].y))  
 l = i;  
  
 int p = l, q;  
 do {  
 hull.push\_back(points[p]);  
 q = (p + 1) % n;  
 for (int i = 0; i < n; i++) {  
 if (orientation(points[p], points[i], points[q]) == 2)  
 q = i;  
 }  
 p = q;  
 } while (p != l);  
  
 return hull;  
}

Wybiera kolejno punkty otoczki, zawsze szukając tego o najmniejszym kącie względem poprzedniego.

### Obsługa wejścia użytkownika (Menu::inputPoints)

Zapewnia poprawność i unikalność wprowadzanych punktów:

std::vector<Point> Menu::inputPoints() {  
 int n;  
 std::cout << "Enter number of points: ";  
 while (!(std::cin >> n) || n < 3) {  
 std::cout << "Invalid input. Enter an integer >= 3: ";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 }  
 std::vector<Point> points;  
 std::set<Point> unique\_points;  
 for (int i = 0; i < n; ) {  
 int x, y;  
 std::cout << "Enter point " << (i + 1) << ": ";  
 if (std::cin >> x >> y) {  
 Point pt(x, y);  
 if (unique\_points.insert(pt).second) {  
 points.push\_back(pt);  
 ++i;  
 } else {  
 std::cout << "Duplicate point ignored. ";  
 }  
 } else {  
 std::cout << "Invalid input. Enter integers for x and y.\n";  
 std::cin.clear();  
 std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');  
 }  
 }  
 return points;  
}

Dzięki temu program jest odporny na błędy użytkownika i duplikaty.

## Złożoność czasowa algorytmu Jarvisa

Algorytm Jarvisa (Gift Wrapping) ma złożoność czasową:

* **O(n \* h)**

gdzie: - **n** – liczba wszystkich punktów wejściowych, - **h** – liczba punktów należących do otoczki wypukłej (rozmiar wyniku).

Wynika to z faktu, że dla każdego punktu otoczki (h iteracji) przeglądane są wszystkie punkty wejściowe (n porównań), aby znaleźć kolejny punkt otoczki o najmniejszym kącie biegunowym względem bieżącego.

W najgorszym przypadku, gdy wszystkie punkty leżą na otoczce (np. punkty na okręgu), złożoność wynosi **O(n^2)**. W praktyce, gdy h jest dużo mniejsze od n, algorytm może być szybszy od innych metod o złożoności O(n log n).