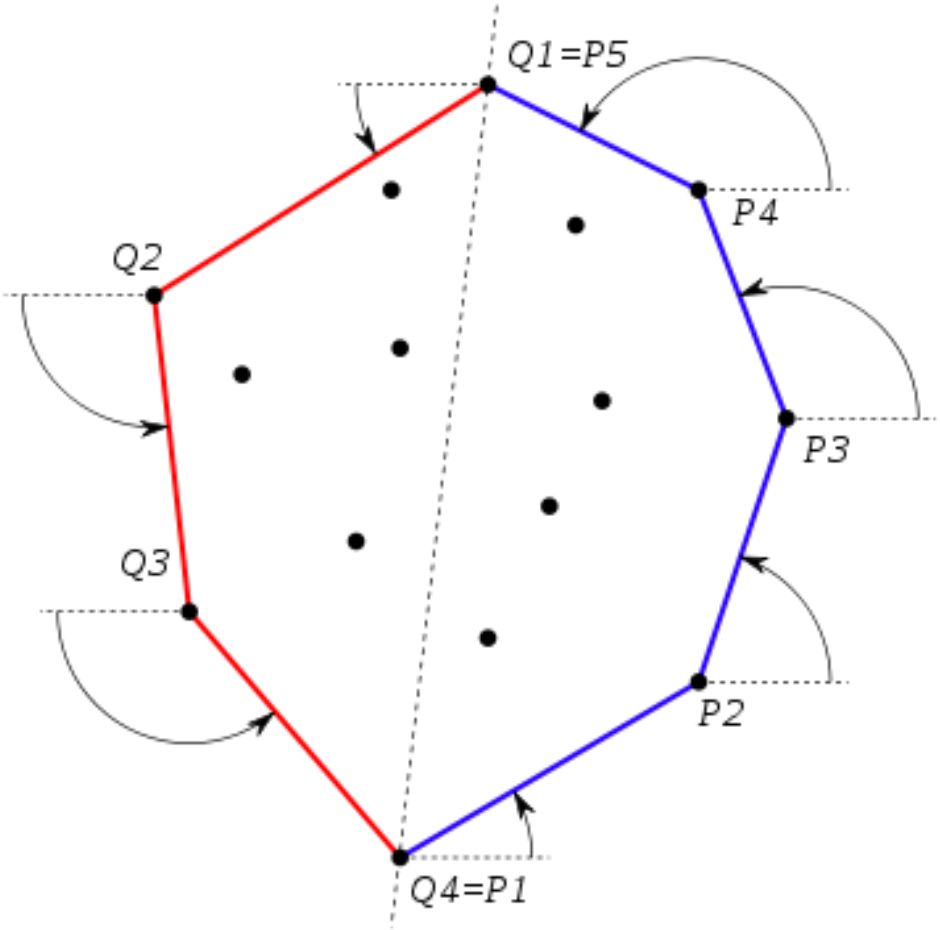
Algorytm Jarvisa

Sprawozdanie z laboratorium 13 – Łukasz Konieczny, LK4

1. Wstęp teoretyczny

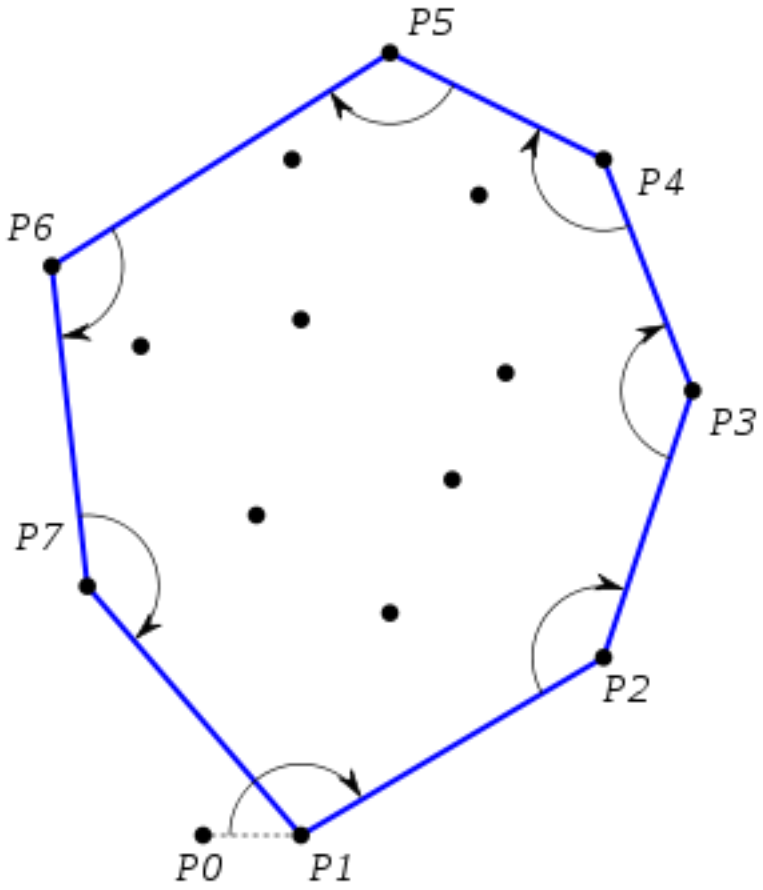
Algorytm Jarvisa należy do algorytmów geometrycznych, służących do rozwiązywania problemów związanych z geometrią analityczną. Jego celem jest wyznaczenie punktów, które połączone ze sobą stanowią wielokąt okalający wszystkie pozostałe punkty w zbiorze (jest „otoczką wypukłą” zbioru)

- Na początku wyznaczamy punkty zbioru znajdujące się możliwie najniżej, oraz najwyżej na płaszczyźnie. Punkt najniższy nazwiemy punktem „p”, a najwyższy „q”
- Wyznaczamy prawą część otoczki, poruszając się przeciwnie do ruchu wskazówek zegara. Rozpoczynając od punktu najbliższego „p” z jego prawej strony, wyznaczamy kąt który każdy inny punkt wyznacza z „p”. Punkt „s” o najmniejszym kącie pomiędzy wektorem PS a wektorem [1, 0] będzie częścią otoczki. Do zbioru wynikowego dodajemy punkt „p” i przechodzimy do wyznaczonego punktu
- Kontynuujemy porównania, dopóki nie dotrzemy do punktu „q”
    - Po dotarciu do punktu „q”, wykonujemy analogiczną procedurę, wyznaczając lewą stronę otoczki. Kontynuujemy do momentu, w którym dotrzemy do punktu „p”



Alternatywnie, możemy wyznaczyć całą otoczkę za jednym podejściem:

- Zaczynając od punktu „p”, porównujemy kąty pomiędzy wektorami PS, a poprzednim znalezionym wektorem
- Wybieramy punkt, dla którego ten kąt jest największy
- Kontynuujemy, aż do powrotu do punktu „p”



2. Opis implementacji

W implementacji skorzystaliśmy z iloczynu wektorowego do wyznaczania wzajemnego położenia punktu w zbiorze (funkcja „wyznaczOrientacje”)

```
// 0 kiedy punkty są współliniowe, 1 gdy zgodnie ze wskazówkami zegara, 2
kiedy przeciw wskazówkom
int wyznaczOrientacje(Punkt p, Punkt q, Punkt 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;
}
```

```
void jarvis(Punkt* zbior, int n) {
    // Muszą być co najmniej 3 punkty żeby wyznaczyć otoczkę
    if (n < 3) return;

    vector<Punkt> wynik;

    // Wyznacza punkt na samym dole
    int l = 0;

    for(int i = 1; i < n; i++)
        if(zbior[i].y < zbior[l].y || (zbior[i].y == zbior[l].y && zbior[i].x
< zbior[l].x))
            l = i;

    int p = l, q;
    do
    {
        wynik.push_back(zbior[p]);

        // Wyznacza punkt "q", dla którego punkty "p", "q" i "x" są ustawione
przeciwnie do wskazówek zegara dla każdego punktu "x"
        q = (p+1)%n;
        for (int i = 0; i < n; i++)
        {
           if (wyznaczOrientacje(zbior[p], zbior[i], zbior[q]) == 2)
               q = i;
        }

        p = q;

    } while (p != l);

    for (int i = 0; i < wynik.size(); i++)
        cout << "(" << wynik[i].x << ", " << wynik[i].y << ")\n";
}
```

3. Wnioski

Podczas zajęć zapoznaliśmy się z ideą algorytmów geometrycznych, oraz zaimplementowaliśmy w języku C++ algorytm Jarvisa, co pozwoliło nam lepiej zrozumieć działanie i zastosowanie tego typu algorytmów.

Bibliografia

https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/

https://pl.wikipedia.org/wiki/Algorytm_Jarvisa