Obliczenia inżynierskie w chmurze

Sprawozdanie z projektu – program wyznaczający obwód punktów na płaszczyźnie.

Antoni Struski 299075

04.01.2025

Spis treści

1. Opis działania kodu

Ze względu na kiepską znajomość pythona kod napisano w c++, a następnie przekonwertowano na pythona za pomocą konwertera internetowego. Kod w języku python kożysta z dodatkowej biblioteki „numpy”, której instalacja zdefiniowana jest w pliku Dockerfile. W kodzie można wyróżnić sześć części:

1.1. Zdefiniowanie klasy wsp przechowującej współrzędne x i y punktów.

```
class wsp{
public:

     wsp() { x = 0; y = 0; }
     wsp(double xx, double yy) { x = xx; y = yy; }

     void set_x(double xx) { x = xx; }
     void set_y(double yy) { y = yy; }
     double  get_x() { return x; }
     double  get_y() { return y; }

private:
     double x, y;
};
```

1.2. Wczytanie ilości „rozmiar” oraz współrzędnych punktów z pliku dane.txt i umieszczenie ich w tablicy „w” przechowującej klasę wsp. Na początku umieszczono także licznik mierzący ile czasu potrzebował program na znalezienie obwodu.

```
time_t begin, end;
time(&begin);

ifstream dane("dane.txt");
     if(!dane.is_open()) {cerr << "Error opening the file!";return 0;}
```

```
	int rozmiar;
	dane>>rozmiar;

	if(rozmiar<3) { cerr << "Error za mało punktów"; return 0; }

	wsp* w;
	w = (wsp*)malloc(rozmiar * sizeof(*w));
	double x, y;
	for(int i = 0; i < rozmiar; ++i) { dane >> x >> y; w[i].set_x(x);
	w[i].set_y(y); }//wczytanie z pliku
dane.close();
```

1.3. Sortowanie rosnąco tablicy „w" względem współrzędnej y

```
for(int i = 0; i < rozmiar - 1; i++) {
	for(int j = 0; j < rozmiar - i - 1; j++) {
		if(w[j].get_y() > w[j + 1].get_y()) {swap(w[j], w[j + 1]);}
	}
}
```

1.4. Szukanie obwodu podzielonego na dwa fragmenty – górny i dolny od skrajnych punktów po prawej i lewej oraz zapisanie ich w wektorach zapisane w wektorach obw_G i obw_D.

```
vector<wsp> obw_D;
int bottom = 0;
obw_D.push_back(w[0]);
for(int i = 1; i < rozmiar; i++) {
	if(w[i].get_x() <= obw_D.back().get_x()) { obw_D.push_back(w[i]); }
	if(w[i].get_x() > obw_D.front().get_x()) { obw_D.insert(obw_D.begin(),
w[i]); bottom++; }
}
//obwód górny ułożony współzegarowo
vector<wsp> obw_G;
int top = 0;
obw_G.push_back(w[rozmiar - 1]);
for(int i = rozmiar - 2; i > 0; i--) {
	if(w[i].get_x() >= obw_G.back().get_x()) { obw_G.push_back(w[i]); }
	if(w[i].get_x() < obw_G.front().get_x()) { obw_G.insert(obw_G.begin(),
w[i]); top++; }
}
```

1.5. Usuwanie z obwodu punktów tworzących kąty wklęsłe poprzez sprawdzenie, czy dany punkt leży pod czy nad linią łączącą jego dwóch sąsiadów.

```
//sprawdzanie wklęsłości w górnym obwodzie
do {
	spr1 = 0;
	i = 0;
	while(i <= obw_G.size() - 3) {
		a = (obw_G[i+2].get_y() - obw_G[i].get_y()) / (obw_G[i+2].get_x()
- obw_G[i].get_x());
		b = obw_G[i].get_y() - obw_G[i].get_x() * a;
		if(obw_G[i + 1].get_y() < a * obw_G[i + 1].get_x() + b) {
			obw_G.erase(obw_G.begin() + i + 1); spr1 = 1; }
		else i++;
	}
} while(spr1==1);

//sprawdzanie wklęsłości w dolnym obwodzie
do {
	spr1 = 0;
	i = 0;
	while(i <= obw_D.size() - 3) {
		a = (obw_D[i + 2].get_y() - obw_D[i].get_y()) / (obw_D[i +
2].get_x() - obw_D[i].get_x());
		b = obw_D[i].get_y() - obw_D[i].get_x() * a;
		if(obw_D[i + 1].get_y() > a * obw_D[i + 1].get_x() + b) {
			obw_D.erase(obw_D.begin() + i + 1); spr1 = 1;
		}
		else i++;
	}
} while(spr1 == 1);
```

1.6. Wyświetlanie wyników, zapis wyników do pliku wynik.txt, wyświetlenie czasy pracy programu, wyczyszczenie tablic.

```
	cout << endl; for(int i = 0; i < obw_G.size(); i++) { cout <<
obw_G[i].get_x() << "  " << obw_G[i].get_y() << endl; }
	cout << endl; for(int i = 0; i < obw_D.size(); i++) { cout <<
obw_D[i].get_x() << "  " << obw_D[i].get_y() << endl; }

	//zapisywanie wyników do plików
	ofstream wynik;
	wynik.open ("wynik.txt");
	wynik << "x\ty"<<endl;
	for(int i = 0; i < obw_G.size(); i++) { wynik << obw_G[i].get_x() << "\t"
<< obw_G[i].get_y() << endl; }
	for(int i = 1; i < obw_D.size(); i++) { wynik << obw_D[i].get_x() << "\t"
<< obw_D[i].get_y() << endl; }
	wynik.close();
```

```
    time(&end);
    time_t elapsed = end - begin;
    cout << endl << "czas: " << elapsed;

    free(w);
    return(0);
```

2. Obsługa chmury obliczeniowej

Na platformie Microsoft Azure utworzono maszynę wirtualną z systemem operacyjnym Ubuntu. Połączono się z nią poprzez okno poleceń oraz program filezilla, za pomocą którego przesłano plik dockera, dane.txt oraz kod programu w pythonie.

Zawartość pliku Dockerfile:

```
FROM python:3.9

RUN apt-get update
RUN apt-get install vim -y
RUN mkdir -p /home/struskawka

RUN pip install numpy

COPY dane.txt /home/struskawka/dane.txt
COPY obw.py /home/struskawka/obw.py

WORKDIR /home/struskawka
# Run "
CMD ["python","/home/struskawka/obw.py"]
# CMD ["/bin/bash"]
```

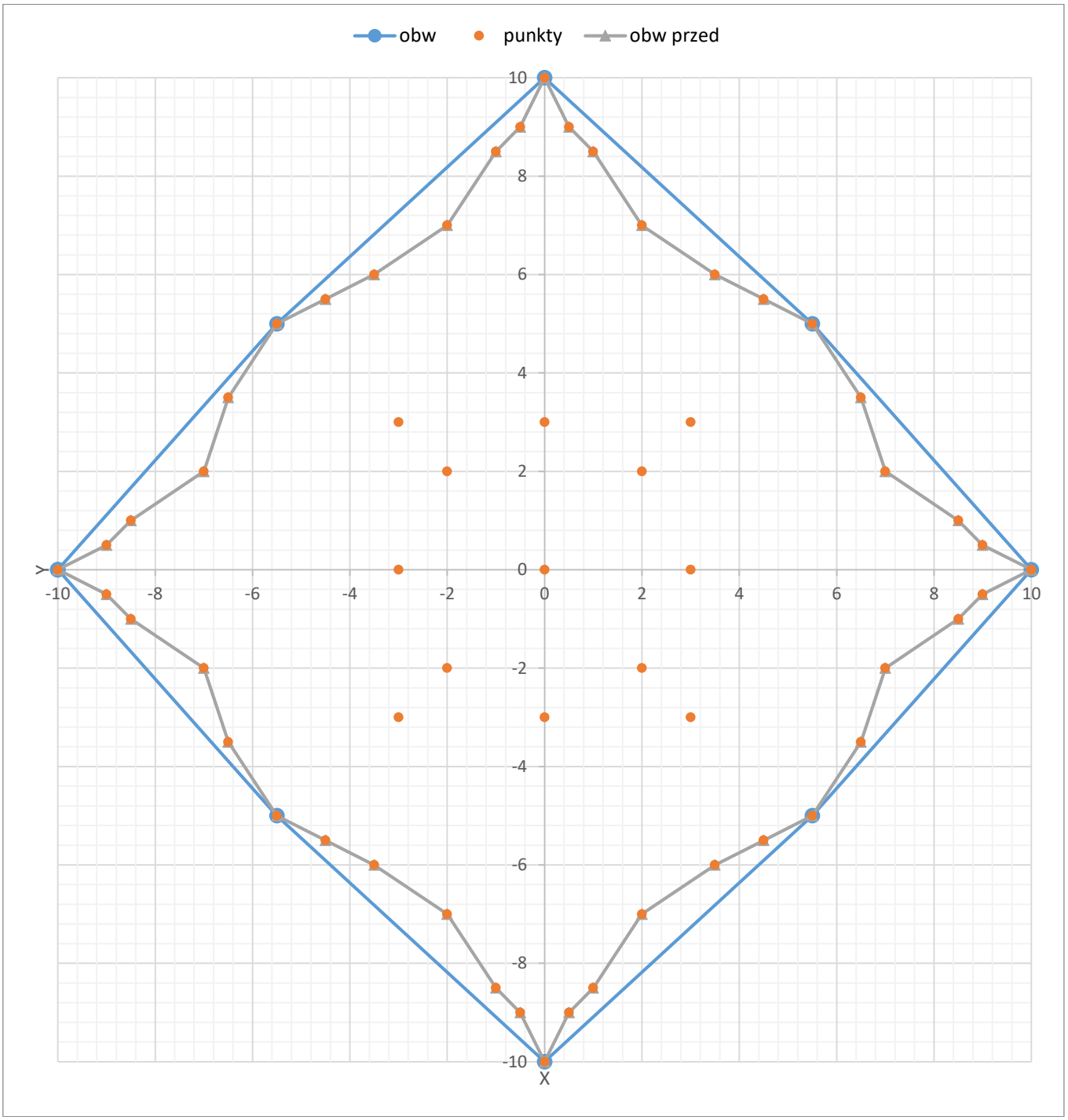
Zainstalowano na niej dockera i na podstawie dockerfile utworzono obraz z danymi, kodem programu i zainstalowaną biblioteką numpy. Utworzono kontener, w którym wywołano program. Po otrzymaniu wyników wszystko zamknięto i skasowano.

Użyte komendy:

```
ssh user@IP_vm_address
sudo apt-get update
sudo apt install docker.io
sudo docker build -t python39_obwod .
sudo docker images
sudo docker run --name python39_obwod_kontener -it python39_obwod
sudo docker ps -a
sudo docker rm [id]
```

3. Prezentacja wyników

Przypadek testowy z 57 punktami dobranymi specjalnie w taki sposób, żeby na obwodzie pojawiały się kąty wklęsłe. Na wykresie szarą linią zaznaczono obwód przed ich usunięciem (wynik między częściami kodu 1.4 i 1.5), a niebieską ostateczny wynik (po części 1.5). Czas trwania obliczeń 0,004 084 s.



Przypadek z wygenerowanym 1000 punktów o losowych współrzędnych pomiędzy 100 i -100. Czas trwania obliczeń 0,025 310 s.