AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Metody Obliczeniowe w Nauce i Technice:

Szybka transformata Fouriera

Laboratorium 8

Przemysław Lechowicz

1. Napisz klasę realizującą DFT. Proszę opisać kolejno realizowane na danych operacje posługując się teorią.

Zaimplementowana została klasa realizująca DFT:

```
class DFT {
public:
	std::complex<double>* input;
	int n;

	DFT(std::complex<double>* input1, int n1) {
		this->input = input1;
		this->n = n1;
	}

	std::vector<std::complex<double> > computeDft() {
		std::vector<std::complex<double> > output;
		for (size_t k = 0; k < n; k++) {
			std::complex<double> sum(0, 0);
			for (size_t t = 0; t < n; t++) {
				double angle = 2 * M_PI * t * k / n;
				sum += input[t] * exp(std::complex<double>(0, -angle));
			}
			output.push_back(sum);
		}
		return output;
	}
};
```

DFT jest funkcją, która odwzorowuje wektor n liczb zespolonych na inny wektor n liczb zespolonych. Stosując indeksowanie oparte na 0, niech x(t) oznacza t-ty element wektora wejściowego, a X(k) oznacza k-ty element wektora wyjściowego. Następnie funkcja DFT jest określona następującą formułą:

$$X(k) = \sum_{t=0}^{n-1} x(t) e^{\frac{-2\pi itk}{n}}$$

Wzór Eulera mówi nam, że e^{xi} = cos(x) + isin(x), dla dowolnej liczby rzeczywistej x. Co więcej, cosinus jest funkcją parzystą, więc cos (−x) = cos(x), a sinus jest funkcją nieparzystą, więc sin (−x) = - (sin(x)). Przez podstawienie:

e^{-2 \pi i t k / n} = e^{(-2 \pi t k / n) i} = \cos \left(-2 \pi \frac{t k}{n} \right) + i \sin \left(-2 \pi \frac{t k}{n} \right) = \cos \left(2 \pi \frac{t k}{n} \right) - i \sin \left(2 \pi \frac{t k}{n} \right)

Niech Re(x) będzie częścią rzeczywistą x, a Im(x) będzie częścią urojoną x. W związku z tym:

$$x(t)\,e^{-2\pi itk/n} = [\mathrm{Re}(x(t)) + i\,\mathrm{Im}(x(t))]\left[\cos\left(2\pi\frac{tk}{n}\right) - i\sin\left(2\pi\frac{tk}{n}\right)\right]$$

A następnie:

$$= \left[\mathrm{Re}(x(t))\cos\left(2\pi\frac{tk}{n}\right) + \mathrm{Im}(x(t))\sin\left(2\pi\frac{tk}{n}\right)\right] + i\left[-\mathrm{Re}(x(t))\sin\left(2\pi\frac{tk}{n}\right) + \mathrm{Im}(x(t))\cos\left(2\pi\frac{tk}{n}\right)\right]$$

2. Korzystając z implementacji stworzonej w zadaniu 1 napisz klasę realizującą FFT (korzystając z algorytmu Cooleya-Tukeya). Implementacje poprzyj stosowym materiałem teoretycznym.

Zaimplementowano algorytm FFT:

```
void separate(std::complex<double>* a, int n) {
    std::complex<double>* b = new std::complex<double>[n / 2];
    for (int i = 0; i < n / 2; i++)
        b[i] = a[i * 2 + 1];
    for (int i = 0; i < n / 2; i++)
        a[i] = a[i * 2];
    for (int i = 0; i < n / 2; i++)
        a[i + n / 2] = b[i];
    delete[] b;
}


void fft2(std::complex<double>* X, int N) {
    if (N < 2) {}
    else {
        separate(X, N);
        fft2(X, N / 2);
        fft2(X + N / 2, N / 2);
        for (int k = 0; k < N / 2; k++) {
            std::complex<double> e = X[k];
            std::complex<double> o = X[k + N / 2];
            std::complex<double> w = exp(std::complex<double>(0, -2. * M_PI * k / N));
            X[k] = e + w * o;
            X[k + N / 2] = e - w * o;
        }
    }
}
```

W tym algorytmie w pierwszej kolejności oddzielamy parzyste i nieparzyste elementy odpowiednio do dolnej i górnej połowy tablicy. Zakładamy, że wielkość tablicy jest potęgą 2.

Algorytm jest następującą sumą po elementach parzystych i nieparzystych:

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N}(2m+1)k}$$

$$\begin{aligned}
X_{k+\frac{N}{2}} &= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}m(k+\frac{N}{2})} + e^{-\frac{2\pi i}{N}(k+\frac{N}{2})} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}m(k+\frac{N}{2})} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk} e^{-2\pi mi} + e^{-\frac{2\pi i}{N}k} e^{-\pi i} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk} e^{-2\pi mi} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-\frac{2\pi i}{N/2}mk} - e^{-\frac{2\pi i}{N}k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-\frac{2\pi i}{N/2}mk} \\
&= E_k - e^{-\frac{2\pi i}{N}k} O_k
\end{aligned}$$

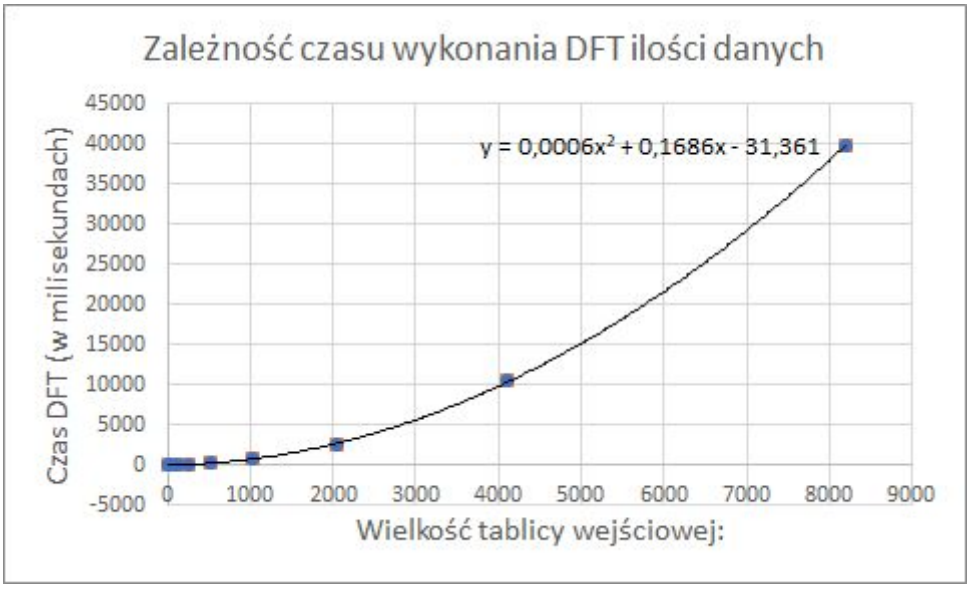
3. Dokonaj pomiarów czasu wykonywania obu transformat dla danych o różnym rozmiarze.

DFT ma złożoność O(n^2), podczas gdy FFT ma złożoność O(nlog(n)).
Został napisany kod porównujący szybkości obu algorytmów:

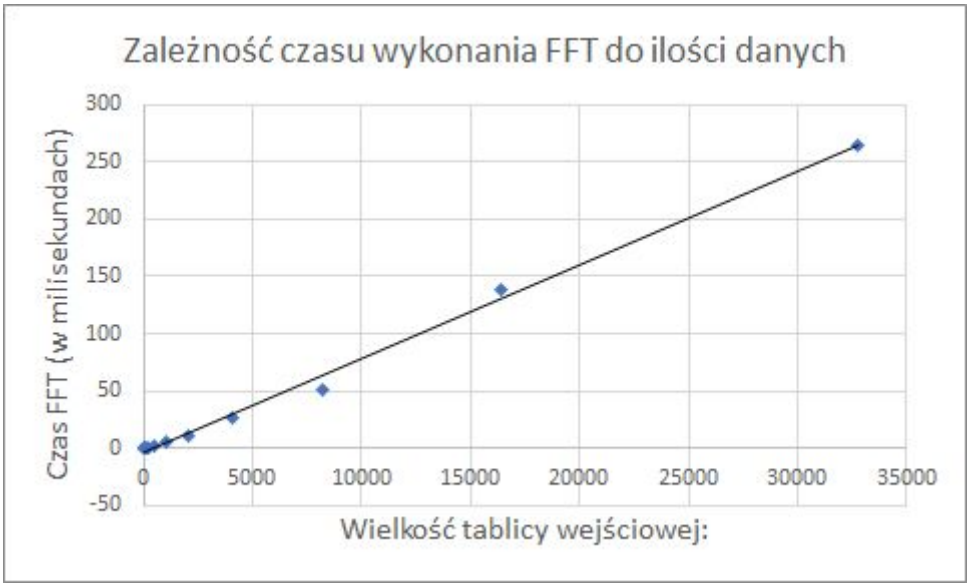
```
srand(time(NULL));
const int size = 2048;
std::complex<double> table[size];
for (int i = 0; i < size; i++) {
	table[i] = std::complex<double>(rand()%100,rand() % 100);
}
DFT dft(table, size);

uint64_t time1 = timeSinceEpochMillisec();
dft.computeDft();
uint64_t time2 = timeSinceEpochMillisec();
fft2(table, size);
uint64_t time3 = timeSinceEpochMillisec();
std::cout << time2-time1<<" "<<time3-time2<<std::endl;
```

Przeprowadziłem testy dla różnych wielkości danych:

Wielkość tablicy wejściowej:	Czas DFT (w milisekundach)	Czas FFT (w milisekundach)
8	0	0
16	0	0
32	1	0
64	6	1
128	11	1
256	51	1
512	188	2
1024	677	6
2048	2546	11
4096	10425	27
8192	39755	51



Na powyższym wykresie widzimy, że zależność czasu od ilości danych dla DFT przypomina kształtem parabolę. Zgadzałoby się to z teoretyczną złożonością czasową dla tego algorytmu. Dla FFT przeprowadziłem więcej testów i otrzymałem następujący wykres:



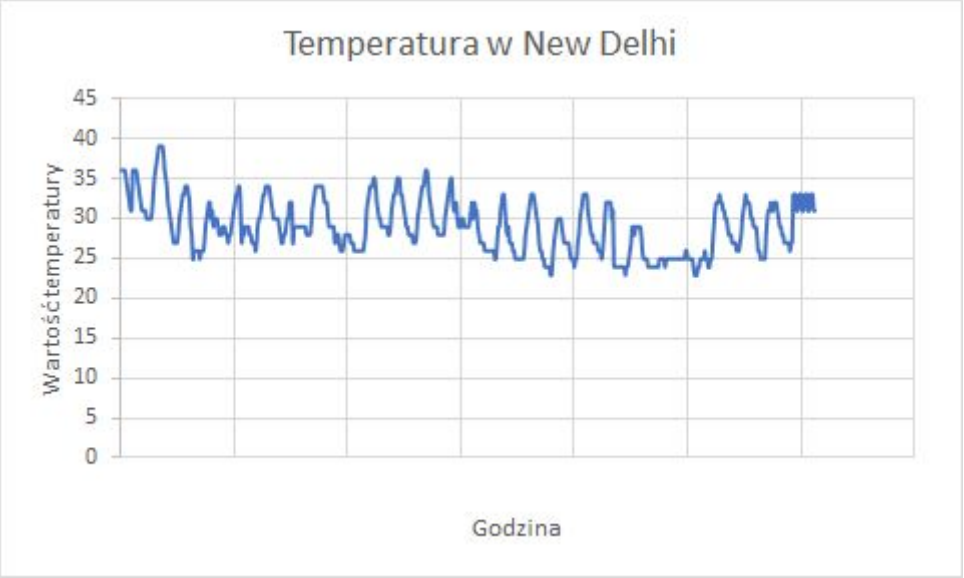
Ten wykres natomiast przypomina funkcję liniową, ale może też przedstawiać funkcję $C * nlog(n)$, gdzie C jest małą(<0.1) stałą.

DFT wykorzystuje w swoim algorytmie dwie zagnieżdżone pętle od 0 do n, które sprawiają, że cały algorytm ma złożoność $O(n^2)$.

FFT ma złożoność $O(nlog(n))$, ponieważ nie korzysta z dwóch pętli, lecz jednej, dodatkowo wywołując się rekurencyjnie. Takich wywołań rekurencyjnych jest dokładnie $2 * log(n)$, z czego każde takie wywołanie ma złożoność liniową, co sprawia, że algorytm ma złożoność $O(nlog(n))$.

4. Przetestuj implementację z zadania 2. do wykonania analizy szeregu czasowego

Wykorzystane zostały dane dotyczące temperatury pobrane ze strony: https://www.kaggle.com/mahirkukreja/delhi-weather-data/data.

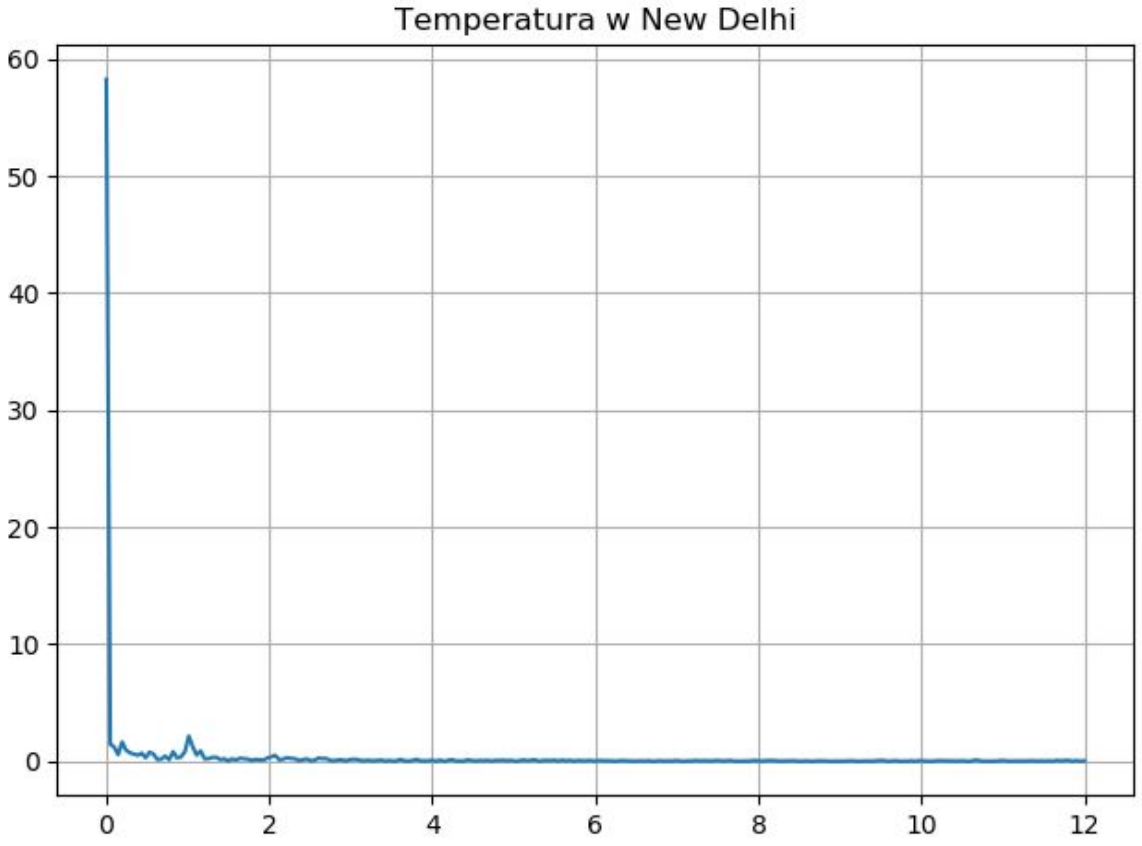


Temperatura raportowana była co godzinę. Na wykresie widzimy wartości z 25 dni.

Zostały one załadowane do programu za pomocą funkcji:

```
std::complex<double>* wczytajPlik(std::string nazwaPliku )
{
	std::complex<double> table[15570];
	std::ifstream plik;
	plik.open(nazwaPliku.c_str());

	int i = 0;
	while (true)
	{
		int a;
		plik >> a;
		if (!plik.fail()) {
			table[i] = a;
		}
		else
			break;
		i++;
	}
	return table;
}
```

Otrzymany wykres ma postać:



Na wykresie widzimy, że występuje pewna sezonowość danych. Na wykresie maksimum występuje dla częstotliwości jeden, co odpowiada sezonowości około 24 godzin, co znajduje potwierdzenie na wykresie temperatury od godziny. Oznacza to, że dane mają wzorce powtarzające się co 24 godziny