

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynierskich
Katedra Informatyki

DOKUMENTACJA PROJEKTOWA ZAAWANSOWANE PROGRAMOWANIE

...Algorytm listy dwukierunkowej z zastosowaniem GitHub...

Autor:
Imie Nazwisko

Prowadzący:
mgr inż. Dawid Kotlarski

Nowy Sącz 2024

Spis treści

1. Ogólne określenie wymagań	4
1.1. Przykład	4
1.2. Instalacja	5
2. Analiza problemu	6
2.1. Wczytywanie danych z pliku CSV	6
2.2. Przechowywanie danych w strukturze drzewiastej	6
2.3. Operacje analityczne	7
2.4. Testowanie jednostkowe	7
3. Projektowanie	9
3.1. Narzędzia	9
3.2. Ustawienia kompilatora	9
3.3. Używanie narzędzi	9
3.3.1. Git	9
3.3.2. Google Test	10
3.3.3. Opis narzędzia: Google Test	10
4. Implementacja	12
4.1. Struktura projektu	12
4.2. Funkcja wczytywania danych z pliku CSV	12
4.3. Klasa Drzewo	13
4.4. Testowanie jednostkowe	15
5. Wnioski	16
5.1. Wnioski dotyczące wczytywania danych	16
5.2. Wnioski dotyczące przechowywania danych	16
5.3. Wnioski dotyczące analizy danych	16
5.4. Wnioski dotyczące testowania jednostkowego	16
5.5. Podsumowanie	16
Literatura	18

Spis rysunków	18
Spis tabel	19
Spis listingów	20

1. Ogólne określenie wymagań

Projekt ma na celu stworzenie systemu do analizy i przechowywania danych energetycznych. System będzie składał się z kilku modułów, które umożliwią wczytywanie danych z plików CSV, przechowywanie ich w strukturach danych, oraz wykonywanie operacji analitycznych na tych danych.

Główne wymagania systemu obejmują:

- Wczytywanie danych z plików CSV i przechowywanie ich w wektorze obiektów klasy Punkt.
- Przechowywanie danych w strukturze drzewiastej, umożliwiającej szybki dostęp do danych na podstawie dat.
- Implementacja funkcji analitycznych, takich jak obliczanie sumy autokonsumpcji i eksportu energii dla określonych przedziałów czasowych.
- Logowanie błędów podczas wczytywania danych z plików CSV.
- Testowanie jednostkowe za pomocą biblioteki Google Test.

1.1. Przykład

Poniżej znajduje się przykładowy kod programu, który wczytuje dane z pliku CSV, przechowuje je w strukturze drzewiastej, oraz oblicza sumę autokonsumpcji dla określonego dnia i ćwiartki¹:

```
1 #include <iostream>
2 #include <vector>
3 #include "Punkt.h"
4 #include "Drzewo.h"
5 #include "CSVReader.h"
6
7 int main() {
8     std::vector<Punkt> dane = wczytajDane("dane.csv");
9
10    Drzewo drzewo;
11    for (const auto& punkt : dane) {
12        drzewo.dodajPunkt(punkt);
13    }
14 }
```

¹

```
15     std::cout << "Suma autokonsumpcji: " << drzewo.  
        sumaAutokonsumpcji(2021, 10, 3, 2) << std::endl;  
16  
17     return 0;  
18 }
```

Listing 1. ObliczanieAutokonsumpcji

Powyższy kod wczytuje dane z pliku `dane.csv` za pomocą funkcji `wczytajDane`. Następnie dodaje te dane do struktury drzewa za pomocą metody `dodajPunkt`. Na końcu oblicza sumę autokonsumpcji dla określonej daty i ćwiartki, a wynik wyświetla na standardowym wyjściu.

1.2. Instalacja

Aby zainstalować i uruchomić projekt, należy wykonać następujące kroki:

- Sklonuj repozytorium projektu: `git clone <URL_repozytorium>`
- Przejdź do katalogu projektu: `cd <katalog_projektu>`
- Skompiluj projekt za pomocą kompilatora g++: `g++ -o program Program.cpp CSVReader.cpp Drzewo.cpp Punkt.cpp -lgtest -lgtest_main -pthread`
- Uruchom skompilowany program: `./program`

Upewnij się, że masz zainstalowane wszystkie niezbędne biblioteki, takie jak Google Test do testowania jednostkowego. Możesz je zainstalować za pomocą menadżera pakietów, np. `apt`:

```
sudo apt-get install libgtest-dev  
sudo apt-get install cmake  
cd /usr/src/gtest  
sudo cmake .  
sudo make  
sudo cp *.a /usr/lib
```

2. Analiza problemu

2.1. Wczytywanie danych z pliku CSV

Jednym z głównych wyzwań projektu jest poprawne wczytywanie danych z pliku CSV. Plik CSV może zawierać błędy, takie jak niepoprawne formatowanie danych. W projekcie zastosowano mechanizm logowania błędów do pliku `log_error.txt`, który rejestruje wszelkie problemy napotkane podczas wczytywania danych².

```

1 std::vector<Punkt> wczytajDane(const std::string& nazwaPliku) {
2     std::ifstream plik(nazwaPliku);
3     std::string linia;
4     std::vector<Punkt> dane;
5
6     std::getline(plik, linia); // Pomijamy nagłówek
7
8     while (std::getline(plik, linia)) {
9         std::stringstream ss(linia);
10        std::string data;
11        double autokonsumpcja, eksport, import, pobor, produkcja;
12
13        if (ss >> data >> autokonsumpcja >> eksport >> import >>
14            pobor >> produkcja) {
15            dane.push_back(Punkt(data, autokonsumpcja, eksport,
16                                import, pobor, produkcja));
17        } else {
18            std::ofstream logError("log_error.txt", std::ios::app);
19            logError << "Błąd wczytywania linii: " << linia <<
20            std::endl;
21        }
22    }
23    return dane;
24 }
```

Listing 2. wczytajPunkty

2.2. Przechowywanie danych w strukturze drzewiastej

Kolejnym wyzwaniem jest przechowywanie danych w sposób umożliwiający szybki dostęp do nich na podstawie daty. W projekcie wykorzystano strukturę drzewiastą, która przechowuje dane w zagnieżdżonych mapach³.

²wczytajPunkty

³

```

1 void Drzewo::dodajPunkt(const Punkt& punkt) {
2     int rok, miesiac, dzien, cwartka;
3     rozbijDate(punkt.getData(), rok, miesiac, dzien, cwartka);
4
5     // Doda dane do struktury drzewa
6     dane[rok][miesiac][dzien][cwartka].push_back(punkt);
7 }

```

Listing 3. dodajPunktDoDrzewa

2.3. Operacje analityczne

Aby umożliwić analizę danych, zaimplementowano funkcje obliczające sumy auto-konsumpcji oraz eksportu energii dla określonych przedziałów czasowych⁴.

```

1 double Drzewo::sumaAutokonsumpcji(int rok, int miesiac, int dzien,
2     int cwartka) {
3
4     // Sprawdzenie dost ȳpno ȳci danych
5     if (dane.find(rok) != dane.end() &&
6         dane[rok].find(miesiac) != dane[rok].end() &&
7         dane[rok][miesiac].find(dzien) != dane[rok][miesiac].end()
8         &&
9         dane[rok][miesiac][dzien].find(cwartka) != dane[rok][
10        miesiac][dzien].end()) {
11         for (const auto& punkt : dane[rok][miesiac][dzien][cwartka
12        ]) {
13             suma += punkt.getAutokonsumpcja();
14         }
15     }
16
17     return suma;
18 }

```

Listing 4. obliczSumeAutokonsumpcji

2.4. Testowanie jednostkowe

Aby zapewnić poprawność działania programu, zastosowano testy jednostkowe za pomocą biblioteki Google Test⁵.

⁴

⁵

```
1 #include <gtest/gtest.h>
2 #include "Drzewo.h"
3 #include "Punkt.h"
4
5 TEST(DrzewoTest, TestSumaAutokonsumpcji) {
6     Drzewo drzewo;
7     Punkt punkt("2021-10-03 10:15", 150.0, 100.0, 50.0, 80.0,
8     200.0);
9     drzewo.dodajPunkt(punkt);
10    ASSERT_EQ(drzewo.sumaAutokonsumpcji(2021, 10, 3, 2), 150.0);
11 }
```

Listing 5. testObliczSumeAutokonsumpcji

3. Projektowanie

3.1. Narzędzia

W projekcie będziemy korzystać z następujących narzędzi:

- Język programowania: C++
- Kompilator: g++
- System kontroli wersji: Git
- Biblioteki dodatkowe: Google Test (do testowania jednostkowego)

3.2. Ustawienia kompilatora

Aby skompilować projekt, używamy kompilatora g++ z następującymi ustawieniami:

- `-std=c++11` - Ustawienie standardu C++
- `-Wall` - Włączanie wszystkich ostrzeżeń
- `-lgtest` - Linkowanie z biblioteką Google Test
- `-lgtest_main` - Linkowanie z główną biblioteką Google Test
- `-pthread` - Używanie wątków POSIX

Przykładowe polecenie do kompilacji projektu:

```
g++ -std=c++11 -Wall -o program Program.cpp CSVReader.cpp Drzewo.cpp Punkt.cpp -l
```

3.3. Używanie narzędzi

3.3.1. Git

System kontroli wersji Git będzie używany do zarządzania kodem źródłowym projektu. Podstawowe komendy Git:

- `git clone <URL_repozytorium>` - Sklonowanie repozytorium
- `git add <plik>` - Dodanie pliku do śledzenia
- `git commit -m «wiadomość»` - Zatwierdzenie zmian z wiadomością

- `git push` - Wysłanie zmian do zdalnego repozytorium
- `git pull` - Pobranie zmian ze zdalnego repozytorium

3.3.2. Google Test

Google Test jest biblioteką do testowania jednostkowego. Testy są definiowane w plikach testowych, a następnie kompilowane i uruchamiane razem z projektem. Przykładowy test jednostkowy⁶:

```
1 #include <gtest/gtest.h>
2 #include "Drzewo.h"
3 #include "Punkt.h"
4
5 TEST(DrzewoTest, TestSumaAutokonsumpcji) {
6     Drzewo drzewo;
7     Punkt punkt("2021-10-03 10:15", 150.0, 100.0, 50.0, 80.0,
8     200.0);
9     drzewo.dodajPunkt(punkt);
10    ASSERT_EQ(drzewo.sumaAutokonsumpcji(2021, 10, 3, 2), 150.0);
11 }
```

Listing 6. testSumaAutokonsumpcji

3.3.3. Opis narzędzia: Google Test

Google Test to framework do testowania jednostkowego w C++. Umożliwia definiowanie testów, które sprawdzają poprawność działania kodu. Poniżej znajduje się przykładowy scenariusz użycia Google Test:

- Najpierw instalujemy Google Test za pomocą menedżera pakietów:

```
sudo apt-get install libgtest-dev
sudo apt-get install cmake
cd /usr/src/gtest
sudo cmake .
sudo make
sudo cp *.a /usr/lib
```

- Następnie tworzymy plik testowy, w którym definiujemy testy, np. `test_Drzewo.cpp`.

⁶

- Kompilujemy projekt razem z testami:

```
g++ -std=c++11 -Wall -o test_program test_Drzewo.cpp Drzewo.cpp Punkt.cpp
```

- Uruchamiamy skompilowany program testowy, aby wykonać testy:

```
./test_program
```

4. Implementacja

4.1. Struktura projektu

Projekt składa się z następujących plików:

- `CSVReader.cpp` - Implementacja funkcji do wczytywania danych z plików CSV.
- `CSVReader.h` - Nagłówek dla pliku `CSVReader.cpp`.
- `Drzewo.cpp` - Implementacja klasy `Drzewo`, która przechowuje i analizuje dane.
- `Drzewo.h` - Nagłówek dla pliku `Drzewo.cpp`.
- `Punkt.cpp` - Implementacja klasy `Punkt`, która przechowuje pojedynczy rekord danych.
- `Punkt.h` - Nagłówek dla pliku `Punkt.cpp`.
- `Program.cpp` - Główny plik programu, który wczytuje dane, przechowuje je w strukturze drzewiastej i wykonuje operacje analityczne.
- `test_Drzewo.cpp` - Testy jednostkowe dla klasy `Drzewo`.
- `test_Punkt.cpp` - Testy jednostkowe dla klasy `Punkt`.

4.2. Funkcja wczytywania danych z pliku CSV

Implementacja funkcji `wczytajDane` w pliku `CSVReader.cpp`⁷:

```
1 #include "CSVReader.h"
2 #include "Punkt.h"
3 #include <fstream>
4 #include <sstream>
5
6 std::vector<Punkt> wczytajDane(const std::string& nazwaPliku) {
7     std::ifstream plik(nazwaPliku);
8     std::string linia;
9     std::vector<Punkt> dane;
10
11     std::getline(plik, linia); // Pomijamy nagłówek
12
13     while (std::getline(plik, linia)) {
14         std::stringstream ss(linia);
```

7

```

15     std::string data;
16     double autokonsumpcja, eksport, import, pobor, produkcja;
17
18     if (ss >> data >> autokonsumpcja >> eksport >> import >>
19     pobor >> produkcja) {
20         dane.push_back(Punkt(data, autokonsumpcja, eksport,
21         import, pobor, produkcja));
22     } else {
23         std::ofstream logError("log_error.txt", std::ios::app);
24         logError << "B Ć d wczytywania linii: " << linia <<
25         std::endl;
26     }
27 }

```

Listing 7. wczytajDaneZPliku

4.3. Klasa Drzewo

Implementacja klasy Drzewo w pliku Drzewo.cpp, która przechowuje dane w strukturze drzewiastej i umożliwia wykonywanie operacji analitycznych⁸:

```

1  #include "Drzewo.h"
2  #include "Punkt.h"
3  #include <sstream>
4  #include <iostream>
5
6  // Funkcja rozbijaj ca dat Ć na rok, miesi c, dzie i wartk ě
7  void Drzewo::rozbijDate(const std::string& data, int& rok, int&
8  miesiac, int& dzien, int& chwila) {
9      std::stringstream ss(data);
10     std::string rok_str, miesiac_str, dzien_str, chwila_str,
11     minuta_str;
12     char separator;
13
14     ss >> rok_str >> separator >> miesiac_str >> separator >>
15     dzien_str >> chwila_str >> separator >> minuta_str;
16
17     rok = std::stoi(rok_str);
18     miesiac = std::stoi(miesiac_str);
19     dzien = std::stoi(dzien_str);
20     chwila = std::stoi(chwila_str);
21     minuta = std::stoi(minuta_str);
22 }

```

8

```
18     int godzina = std::stoi(godzina_str);
19     int minuta = std::stoi(minuta_str);
20
21     if (godzina < 6) {
22         cwartka = 0;
23     } else if (godzina < 12) {
24         cwartka = 1;
25     } else if (godzina < 18) {
26         cwartka = 2;
27     } else {
28         cwartka = 3;
29     }
30 }
31
32 // Metoda dodaj ca nowy punkt do drzewa
33 void Drzewo::dodajPunkt(const Punkt& punkt) {
34     int rok, miesiac, dzien, cwartka;
35     rozbijDate(punkt.getData(), rok, miesiac, dzien, cwartka);
36
37     dane[rok][miesiac][dzien][cwartka].push_back(punkt);
38 }
39
40 // Funkcja obliczaj ca sum ̑ autokonsumpcji
41 double Drzewo::sumaAutokonsumpcji(int rok, int miesiac, int dzien,
42     int cwartka) {
43     double suma = 0.0;
44
45     if (dane.find(rok) != dane.end() &&
46         dane[rok].find(miesiac) != dane[rok].end() &&
47         dane[rok][miesiac].find(dzien) != dane[rok][miesiac].end()
48         &&
49         dane[rok][miesiac][dzien].find(cwartka) != dane[rok][
50     miesiac][dzien].end()) {
51         for (const auto& punkt : dane[rok][miesiac][dzien][cwartka
52     ]) {
53         suma += punkt.getAutokonsumpcja();
54     }
55     }
56
57     return suma;
58 }
```

Listing 8. rozbicieDaty

4.4. Testowanie jednostkowe

Testy jednostkowe dla klasy `Drzewo` w pliku `test_Drzewo.cpp`⁹:

```
1 #include <gtest/gtest.h>
2 #include "Drzewo.h"
3 #include "Punkt.h"
4
5 TEST(DrzewoTest, TestSumaAutokonsumpcji) {
6     Drzewo drzewo;
7     Punkt punkt("2021-10-03 10:15", 150.0, 100.0, 50.0, 80.0,
8     200.0);
9     drzewo.dodajPunkt(punkt);
10    ASSERT_EQ(drzewo.sumaAutokonsumpcji(2021, 10, 3, 2), 150.0);
11 }
```

Listing 9. `testObliczSumeAutokonsumpcji`

⁹

5. Wnioski

Projekt ten umożliwił stworzenie systemu do analizy i przechowywania danych energetycznych, spełniającego określone wymagania. Poniżej znajdują się główne wnioski z realizacji projektu:

5.1. Wnioski dotyczące wczytywania danych

Funkcja `wczytajDane` skutecznie przetwarza dane z plików CSV i przechowuje je w strukturze wektora obiektów klasy `Punkt`. Mechanizm logowania błędów okazał się przydatny w identyfikacji niepoprawnych linii w plikach CSV, co pozwala na ich późniejszą korektę.

5.2. Wnioski dotyczące przechowywania danych

Struktura drzewiasta zastosowana w klasie `Drzewo` pozwala na efektywne przechowywanie i szybki dostęp do danych na podstawie daty. Zastosowanie zagnieżdżonych map umożliwia łatwe grupowanie danych według roku, miesiąca, dnia i ćwiartki.

5.3. Wnioski dotyczące analizy danych

Implementacja funkcji analitycznych, takich jak `sumaAutokonsumpcji` i `sumaEksportu`, umożliwia szybkie i dokładne obliczenia na dużych zbiorach danych. Struktura drzewiasta pozwala na szybkie wyszukiwanie i agregowanie danych, co jest kluczowe dla wydajności systemu.

5.4. Wnioski dotyczące testowania jednostkowego

Zastosowanie biblioteki Google Test do testowania jednostkowego pozwoliło na wcześnie wykrywanie błędów i zapewnienie wysokiej jakości kodu. Testy jednostkowe okazały się nieocenione w procesie weryfikacji poprawności działania funkcji i metod klas.

5.5. Podsumowanie

Projekt zakończył się sukcesem, spełniając wszystkie założone cele i wymagania. System jest w stanie skutecznie wczytywać, przechowywać i analizować dane energetyczne, zapewniając jednocześnie wysoką wydajność i dokładność. W przyszłości

można rozważyć rozszerzenie funkcjonalności systemu o dodatkowe analizy i raportowanie danych.

Spis rysunków

Spis tabel

Spis listingów

1.	ObliczanieAutokonsumpcji	4
2.	wczytajPunkty	6
3.	dodajPunktDoDrzewa	7
4.	obliczSumeAutokonsumpcji	7
5.	testObliczSumeAutokonsumpcji	8
6.	testSumaAutokonsumpcji	10
7.	wczytajDaneZPliku	12
8.	rozbicieDaty	13
9.	testObliczSumeAutokonsumpcji	15