Specyfikacja Implementacyjna

Dominik Wawrzyniuk

16.11.2020

1 Opis modułów

1.1 Moduł główny

Zawiera pliki "main.cpp", "IO.cpp", "IO.h", "input.cpp" oraz "input.h"

1.1.1 main.cpp

Zawiera całą logikę programu w funkcji main. Nie zawiera innych funkcji.

```
/* pseudokod: main.cpp */

int main(char* sciezka) {
        input dane = czytajDane(sciezka);
        graf graf = zinterpretujDane(dane);
        wynik wynik = obliczWynik(graf);
        zapiszWynik(wynik, input);
}
```

Najpierw dane są wczytane z pliku i zapisane w strukturze zdefiniowanej w plikach input.cpp oraz input.h. Następnie te dane są zamieniane na dane, które algorytm przyjmie, i nastepnie jemu przekazane. Na koniec działania program zapisze wynik. Struktury danych graf i wynik są zdefiniowane w module Algorytm.

1.1.2 input.cpp i input.h

Plik input.h definiuje strukturę input, która jest pojemnikiem na dane wczytane z pliku. Definiuje on również klasy pomocnicze które będą przechowywane w input oraz dwie struktury przechowywane w klasach. Podczas wykonywania programu tylko jedna struktura input będzie używana jednocześnie.

```
/* plik: input.h */

struct budynek {
        int ID;
        string nazwa;
        int wymog;
};
class Budynki {
        private:
                budynek* budynki;
                int ile;
                int index = 0;
        public:
                Budynki(int);
                ~Budynki();
                void add(int, string, int);
                budynek get(int);
};
struct handel {
        int ID_fabryki;
        int ID_apteki;
        int limit;
        float koszt;
};
class Handle {
        private:
                handel** handle;
                int rzedy;
                int kolumny;
                int index_i = 0;
                int index_j = 0;
        public:
                Handle(int, int);
                ~Handle();
                void add(int, int, int, float);
                handel get(int, int);
                handel get(int);
};
struct input {
        Budynki* fabryki;
        Budynki* apteki;
        Handle* handle;
};
```

Struktura input będzie zawierać wszystkie informacje które są potrzebne do interpretacji danych dla algorytmu, czyli wskaźniki do obu rodzajów budynków i do handli. Struktura budynek przechowuje dane dotyczące budynku, a klasa Budynki jest pojemnikiem na wskaźnik do budynek z opcją dodawania nowych elementów. Funkcja ta będzie wstawiać nowe wartości sortując je względem identyfikatora. Opcja ta zgłosi również błąd jeżeli nastąpi próba ponownego dodania tego samego indeksu. Struktura handel i klasa Handle są analogią klasy Budynki dla handli wymienionych w pliku, ale zaimplementowana jako macierz.

1.1.3 IO.cpp, IO.h

Zawiera funkcje służące odczytowi danych z pliku oraz zapisujących wynik do pliku, wraz ze wszelkimi funkcjami sprawdzającymi błędy we wprowadzonych danych.

```
/* plik: IO.h */

input czytajDane(char*);
void zapiszWynik(int*);
```

Funkcja czytajDane przeczyta dane z pliku podanego jako argument, sprawdzając w trakcie wykonywania błędy zadeklarowane w oddzielnych funkcjach pliku IO.cpp.

```
/* pseudokod: IO.cpp */

input przeczytaj(char* sciezka) {
        input ret;
        ifstream file;
        file.open(sciezka);
        sprawdzCzyPierwszaLinijkaJestPoprawna(file);
        int ile_fabryk = policzIleFabryk(file);
        sprawdzCzyDrugaLinijkaJestPoprawna(file);
        int ile_aptek = policzIleAptek(file);
        sprawdzCzyTrzeciaLinijkaJestPoprawna(file);
        file.open(sciezka);
        Budynki fabryki = Budynki(ile_fabryk);
        wczytajDaneDoFabryk(&fabryki, file);
        ret.fabryki = &fabryki;
        Budynki apteki = Budynki(ile_aptek);
        wczytajDaneDoAptek(&apteki, file);
        ret.apteki = &apteki;
        Handle handle = Handle(ile_fabryk, ile_aptek);
        wczytajDaneDoHandli(&handle, file);
        ret.handle = &handle;
        file.close();
        return ret;
}
```

```
input czytajDane(char* sciezka) {
        sprawdzCzyPlikIstnieje(sciezka);
        sprawdzCzyPlikJestTekstowy(sciezka);
        return przeczytaj(sciezka);
}

void zapiszWynik(wynik wynik, input in) {
        ofstream file;
        file.open("../Plik_wyjsciowy/Wynik.txt");
        for (int i = 0; i < in.handle->ile; i++) {
                for (int j = 0; j < in.handle->ile; j++) {
                        drukuj(wynik.zakupy[i][j],
                         wynik.ceny[i][j],
                         in.fabryki->get(i),
                         in.apteki->get(j));
                }
        }
        drukuj(wynik.suma);
        file.close();
}
```

Funkcje zaczynające nazwę od sprawdz są funkcjami które mogą rzucić potencjalny błąd, ale błędy mogą też być znalezione w trakcie wczytywania danych do poszczególnych klas. Po wczytaniu danych do wymaganej struktury program przekaże je z powrotem do pliku main.cpp. Program nie sprawdza czy identyfikatory w handlach pokrywają się z podanymi przy fabrykach i aptekach, ponieważ ten błąd zostanie wykryty na etapie konwertowania danych do postaci przyjmowanej przez algorytm. Zapisz wynik otrzyma zdefiniowaną w module Algorytm strukturę wynik, którą wykorzysta do zapisania wyniku w pliku wyjściowym.

1.2 Algorytm

Zawiera pliki "alg.cpp", "alg.h", "conv.cpp", "conv.h", "graf.h", "wynik.h".

1.2.1 graf.h

Plik graf.h zawiera strukturę reprezentującą postać grafu, zapisaną w macierzy. Będzie przechowywać limity dostaw budynków i koszt za sztukę leku.

```
/* plik: graf.h */

struct graf {
        int** limity;
        int** koszty;
        int wezly;
};
```

Postać ta pozwoli na ropatrzenie problemu jako problem max flow min cost, do którego dostosowany został algorytm.

1.2.2 wynik.h

Plik wynik.h zawiera strukturę w której algorytm zapisze i przekaże dalej wynik przez niego obliczony. Zawiera informację dotyczącą ile handli zostało dokonanych z kim, ile dany handel kosztował, i jaka była łączna cena zakupów.

```
/* plik: wynik.h */

struct wynik {
        int** zakupy;
        float** ceny;
        float suma;
};
```

1.2.3 conv.cpp i conv.h

Pliki conv zawierają funkcje służące konwersji danych wczytanych z pliku do struktury graf, która jest przyjmowana przez algorytm.

```
/* plik: conv.h */

graf zinterpretujDane(input);
```

Funkcja graf zamieni podane jej dane na postać grafową.

```
/* pseudokod: conv.cpp */

graf zinterpretujDane(input in) {
        int wezly = in.fabryki->ile + in.apteki->ile + 2;
        graf graf;
        graf.wezly = wezly;
        graf.limity = new int* [wezly];
        graf.koszty = new float* [wezly];
        zapiszKran(&graf, in.fabryki);
        zapiszWartosci(&graf, in.handle);
        zapiszZlew(&graf, in.apteki);
        return graf;
}
```

Funkcje zapisz zapisują w odpowiednim wierszu macierzy grafowej wartości limitu i kosztu danej krawędzi.zapiszKran i zapiszZlew korzystają z limitów w fabrykach i aptekach, a zapisz wartości z handli. Po ukończeniu tej czynności funkcja zwraca powstały graf.

1.2.4 alg.cpp i alg.h

Plik alg.cpp oraz alg.h zawierają algorytm który rozwiąże problem przekształcony w problem max flow min cost. Algorytm do tego zastosowany będzie oparty na sukcesywnym znajdywaniu najkrótszej ścieżki za pomocą algorytmu Bellmana Forda. Algorytm ten jest najlepszy do wykorzystania tutaj, ponieważ oryginalny problem łatwo jest przekształcić w postać grafu, a rozwiązanie problemu metodami standardowymi dla programowania całkowitego byłoby wolniejsze w tym konkretnym rodzaju problemu, gdzie wszystkie ograniczenia oprócz kosztów byłyby 1, -1 albo 0.

```
/* plik: alg.h */

wynik obliczWynik(graf);
```

Funkcja obliczWynik przyjmuje graf, a zwraca strukturę wynik zawierającą wszystkie części poszukiwanego rozwiązania.

```
/* pseudokod: alg.cpp */

bool algorytmBellmanaForda(graf graf, int* flow);

void aktualizujWynik(graf graf, int* flow, wynik* wynik);

wynik obliczWynik(graf graf) {
        wynik wynik;
        int* flow;
        while(algorytmBellmanaForda(graf, flow)) {
                aktualizujWynik(graf, flow, &wynik);
        }
        return wynik;
}
```

Algorytm będzie wywoływać metodę algorytmBellmanaForda, która będzie znajdywać najtańszą ścieżkę w grafie, i informować czy taką znalazła jako informację zwrotną. Po każdym wywołaniu będzie aktualizować wynik używając nowych przepływów znalezionych przez algorytm Bellmana Forda. Na koniec zwróci wynik.

2 Testowanie

Program zostanie przetestowany bez użycia środowisk testowych, za pomocą wbudowanego w język C++ makra assert. Program nie zawiera wielu metod do testowania dlatego nie użyto środowiska testowego. Testy znajdą się obok folderu OLKA, w folderze o nazwie Testy. Testowane będą pliki IO.cpp, conv.cpp i alg.cpp, a ich testy będą zapisane w pliku o szablonie [nazwa_pliku]_test.cpp, w folderze o nazwie pliku.

2.1 IO_test.cpp

Testy dla pliku IO.cpp:

```
/* plik: IO_test.cpp */

void test_BrakPliku();
void test_PlikNieTekstowy();
void test_PlikNieIstnieje();
void test_PlikZNiepoprawnymZapisemDanych();
void test_PlikZBrakiemNiektorychDanych();
void test_PlikZLiczbamiNienaturalnymi();
void test_PlikZKosztamiDluzszymiNizFloat();
void test_PlikZeSprzecznymiIdentyfikatorami();
void test_PlikZeZbytDlugaNazwaFirmy();
void test_PlikZeZbytDLugaLinijka();
void test_PoprawnyPlik();
void test_ZapisBezPodanychParametrow();
void test_ZapisPoprawny();
```

2.2 conv_test.cpp

Testy dla pliku conv.cpp:

```
/* plik: conv_test.cpp */

void test_BrakWejscia();
void test_NiepelneWejscie();
void test_PoprawneWejscie();
```

2.3 alg_test.cpp

Testy dla pliku alg.cpp:

```
/* plik: conv_test.cpp */

void test_BrakGrafu();
void test_NiepelnyGraf();
void test_PoprawnyGraf();
```

3 Uwagi końcowe

Cały program będzie wykonany w języku C++.