Algorytm Bellmana-Forda

Dawid Kosiński

1 Treść zadania

Napisać program, do znajdowania najkrótszych ścieżek między zadanym wierzchołkiem grafu a wszystkimi pozostałymi wierzchołkami tego grafu. Program wykorzystuje algorytm Bellmana-Forda.

Program uruchamiany jest z linii poleceń z wykorzystaniem następują-cych przełączników:

-g plik wejściowy z grafem
-w plik wejściowy z wierzchołkami startowymi
-o plik wyjściowy z wynikami

2 Analiza zadania

Zagadnienie przedstawia problem znajdywania najkrótszej trasy z wierz-chołka początkowego do reszty wierzchołków w grafie. Graf może posiadać ujemne wagi, dlatego program dotyczy zastosowania algorytmu Bellmana-Forda.

2.1 Struktury danych

W programie wykorzystano tak zwaną listę sąsiedztwa do przechowywa-nia grafu. Jest to tak naprawdę wektor wierzchołków, a każdy wierzchołek posiada swoją listę wychodzących krawędzi i kosztów ich pokonania. Do-datkowo w programie wykorzystano kontener unordered map, który jest po-trzebny do ”kodowania”indeksów wierzchołków i odkodowywania. Potrzeba takiego kontenera wynika z możliwości podania dowolnych indeksów wierz-chołków przez użytkownika, a program musi w jakiś sposób móc je łatwo rozpoznawać.

2.2 Algorytmy

Program najpierw pobiera graf z pliku tekstowego, a następnie wczytuje startowe wierzchołki. Potem wykonywane jest znajdywanie optymalnych tras. Algorytm Bellmana-Forda wykonuje się w czasie $O(|V| * |E|)$. Złożoność pa-mięciowa wynosi $O(|V|)$

3 Specyfikacja zewnętrzna

Program jest uruchamiany z linii poleceń.

Należy przekazać do programu nazwy plików: wejściowego z grafem, wej-ściowego z wierzchołkami startowymi i wyjściowego po odpowiednich prze-łącznikach (odpowiednio: -g dla pliku z grafem, -w dla pliku z wierzchołkami startowymi, -o dla pliku wyjściowego), np.

program -g Pierwszy.txt -w Drugi.txt -o Wyjsciowy.txt
program -w Drugi.txt -o Wyjsciowy.txt -g Pierwszy.txt

Pliki są plikami tekstowymi. Przełączniki mogą być podane w dowolnej ko-lejności. Uruchomienie programu bez żadnego parametru lub z parametrem

program
program -g

powoduje wyświetlenie krótkiej pomocy. Uruchomienie programu z niepra-widłowymi parametrami powoduje wyświetlenie komunikatu

Podano bledna liczbe parametrow! Prosze podac 3 parametry wraz z ich przelacznikami.

i wyświetlenie pomocy.

Podanie nieprawidłowej nazwy pliku powoduje wyświetlenie odpowied-niego komunikatu:

Program nie jest w stanie otworzyc pliku wejsciowego o nazwie: " << _fileName << ". Prosze sie upewnic, ze na pewno istnieje taki plik w folderze projektu.

4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs (komunikację z użytkownikiem) od logiki apli-kacji (znajdywania tras).

4.1 Ogólna struktura programu

W funkcji głównej wywołana jest funkcja Load_parameters, która odpo-wiada za załadowanie parametrów i sprawdzenie czy są poprawne. Gdy pro-gram nie został wywołany prawidłowo, zostaje wypisany stosowny komuni-kat i program się kończy. Następnie wywoływane są funkcje Load graph_ i Load variables . Funkcje te ładują graf z pierwszego pliku oraz wierzchołki startowe z drugiego pliku. Następnie wywoływana jest funkcja Setup File , która czyści plik wyjściowy (ze względu na algorytm zapisywania wyników

do pliku jest to koniecznie). Następnie sprawdzane są poprawności plików tekstowych podanych przez użytkownika za pomocą trzech wywołań funk-cji FileCorrect , które zwracają odpowiednie wartości typu **bool**. Kluczowym fragmentem programu jest wykonanie algorytmu Bellmana Forda za pomocą funkcji Bellman Ford, która zwraca **false**, jeśli zostanie napotkany ujemny cykl w grafie i następnie kończy program. W przypadku, gdy ujemnych cykli nie ma, to wynik jest zapisywany do pliku i następuje przejście do kolejnego wierzchołka startowego.

4.2 Szczegółowy opis typów i funkcji

Szczegółowy opis typów i funkcji zawarty jest w załączniku.

5 Testowanie

Program został przetestowany na różnego rodzaju plikach. Pliki niepo-prawne z grafem (puste) powodują zgłoszenie braku szukanych wierzchołków w pliku wyjściowym. Pliki z grafem w którym występują ujemne cykle po-wodują zapisanie stosownej informacji do pliku wyjściowego. W przypadku, gdy plik z wierzchołkami startowymi jest pusty, to wtedy plik wyjściowy też jest pusty.

6 Wnioski

Program wykorzystujący algorytm Bellmana-Forda okazał się bardzo pra-cochłonnym zadaniem, ale dzięki temu miałem okazję nauczyć się wielu rozwiązań, o których wcześniej nie miałem pojęcia. Przede wszystkim na-uczyłem się przeszukiwać Internet, aby zdobyć informacje o potrzebnych mi narzędziach czy funkcjach. Projekt pomógł mi również znacząco podnieść umiejętności korzystania z rozwiązań pokazanych na wykładach tj. wektory, listy, struktury, obsługa plików czy dzielenie projektu na pliki nagłówkowe i źródłowe. Zrozumienie działania algorytmu Bellmana-Forda przyszło mi bardzo szybko, lecz miałem problem z przeniesieniem tego na język C++. Największe trudności miałem we właściwym opakowaniu i zrozumieniu jak działać na danych z plików tekstowych. Dodatkowo czymś nowym i na po-czątku niezrozumiałym było dla mnie tworzenie dokumentacji w Doxygen. Projekt nauczył mnie również współpracy z platformą GitHub, którą mimo początkowych kłopotów traktuję jako bardzo pożyteczne narzędzie do prze-chowywania swoich prac. Bardzo pomógł mi fakt, że wykłady są nagrywane

i udostępniane na platformie, dzięki czemu mogłem szukać tam rozwiązań i pomysłów potrzebnych do rozwiązania projektu.

Dodatek
Szczegółowy opis typów i funkcji

Algorytm Bellmana-Forda

Wygenerowano przez Doxygen 1.9.3

Rozdział 1

Indeks klas

1.1 Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Rozdział 2

Indeks plików

2.1 Lista plików

Tutaj znajduje się lista wszystkich udokumentowanych plików z ich krótkimi opisami:

Rozdział 3

Dokumentacja klas

3.1 Dokumentacja struktury Edge

Struktura opakowująca wprowadzane krawędzie do programu.

`#include <model.h>`

Atrybuty publiczne

- int **begin**
- int **end**
- double **cost**

3.1.1 Opis szczegółowy

Struktura opakowująca wprowadzane krawędzie do programu.

Parametry

begin	Zmienna przechowująca indeks początku krawędzi
end	Zmienna przechowująca indeks końca krawędzi.
cost	Zmienna przechowująca koszt przejścia krawędzi.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- Algorytm Bellmana-Forda/Algorytm Bellmana-Forda/model.h

3.2 Dokumentacja struktury Graph

Struktura opakowująca wprowadzony graf do programu.

`#include <model.h>`

Atrybuty publiczne

- vector< Node > **nodes**
- unordered_map< int, int > **umap_nodes**
- unordered_map< int, int > **umap_nodes_reversed**

3.2.1 Opis szczegółowy

Struktura opakowująca wprowadzony graf do programu.

Parametry

nodes	Wektor wierzchołkow znajdujących się w grafie.
umap_nodes	Mapa zawierająca indeksy wierzchołkow z przypisanymi do nich wartościami od 0, do n, gdzie n to liczba wierzchołków w grafie.
umap_nodes_reversed	Mapa zawierająca indeksy wierzchołków, które są przypisane do wartości od 0, do n, gdzie n to liczba wierzchołków w grafie.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- Algorytm Bellmana-Forda/Algorytm Bellmana-Forda/model.h

3.3 Dokumentacja struktury Node

Struktura opakowująca wprowadzane wierzchołki do programu.

`#include <model.h>`

Atrybuty publiczne

- int **index**
- list< Edge > **edges**

3.3.1 Opis szczegółowy

Struktura opakowująca wprowadzane wierzchołki do programu.

Parametry

index	Zmienna przechowująca indeks wierzchołka
edges	Lista przechowująca krawędzie, którymi można przejść z tego wierzchołka do innych.

Dokumentacja dla tej struktury została wygenerowana z pliku:

- Algorytm Bellmana-Forda/Algorytm Bellmana-Forda/model.h

3.4 Dokumentacja struktury Params

Struktura zawierająca parametry przekazywane do programu.

`#include <widok.h>`

Atrybuty publiczne

- string **graphFile**
- string **pathsFile**
- string **outputFile**

3.4.1 Opis szczegółowy

Struktura zawierająca parametry przekazywane do programu.

Parametry

graphFile	Plik z grafem
pathsFile	Plik z wierzchołkami, od których program będzie szukał najlepszych tras.
outputFile	Plik z wynikami programu (trasy i koszty).

Dokumentacja dla tej struktury została wygenerowana z pliku:

- Algorytm Bellmana-Forda/Algorytm Bellmana-Forda/widok.h

Rozdział 4

Dokumentacja plików

4.1 Dokumentacja pliku Algorytm Bellmana-Forda/Algorytm Bellmana-Forda/model.h

```
#include <vector>
#include <list>
#include <string>
#include <utility>
#include <iterator>
#include <algorithm>
#include <iostream>
#include <unordered_map>
#include <sstream>
#include <fstream>
```

Komponenty

- struct Edge

  Struktura opakowująca wprowadzane krawędzie do programu.
- struct Node

  Struktura opakowująca wprowadzane wierzchołki do programu.
- struct Graph

  Struktura opakowująca wprowadzony graf do programu.

Funkcje

- bool Find (const Graph &graph, const int &index, int &_where)

  Funkcja szukająca wierzchołka w grafie.
- void addEdge_Directed (Graph &graph, const int begin, const int end, const double cost)

  Funkcja dodająca skierowaną krawędź do grafu.
- void Save (Graph &_graph, const string &_fileName, const int index, const vector< double > &_distances, const vector< string > &_paths)

  Funkcja zapisująca wyniki algorytmu Bellmana-Forda dla wprowadzonych danych. Zapis jest dokonywany w pliku tekstowym podanym przez użytkownika.
- bool Bellman_Ford (Graph &graph, const int index, const string &fileName)

  Najistotniejsza funkcja programu wykonująca algorytm Bellmana-Forda.

4.1.1 Dokumentacja funkcji

4.1.1.1 addEdge_Directed()

```
void addEdge_Directed (
            Graph & graph,
            const int begin,
            const int end,
            const double cost )
```

Funkcja dodająca skierowaną krawędź do grafu.

Przebieg funkcji:

1. Stwórz tymczasową krawędź i zainicjuj jej parametry wprowadzonymi parametrami do funkcji.
2. Sprawdź czy istnieje wierzchołek początkowy i końcowy w grafie. Jeśli, któryś nie istnieje, to go stwórz.
3. Wepchnij na koniec listy krawędzi wierzchołka początkowego ową krawędź.

Parametry

in,out	*graph*	Graf do którego są dodawane krawędzie.
in	*begin*	Indeks wierzchołka z którego wychodzi krawędź.
in	*end*	Indeks wierzchołka w którym się kończy krawędź.
in	*cost*	Koszt przejścia z wierzchołka o indeksie begin do wierzchołka o indeksie end.

4.1.1.2 Bellman_Ford()

```
bool Bellman_Ford (
            Graph & graph,
            const int index,
            const string & fileName )
```

Najistotniejsza funkcja programu wykonująca algorytm Bellmana-Forda.

Przebieg działania algorytmu:

1. Zainicjuj wektor dystansów do każdego wierzchołka wartościami maksymalnymi, a komórkę odpowiadającą wierzchołkowi startowemu zainicjuj zerem.
2. Zainicjuj wektor tras indeksem startowego wierzchołka.
3. Wykonaj krok 4 N razy, gdzie N to liczba wierzchołków w grafie. Dlaczego nie N-1 razy? Bo podczas tej pętli algorytm jest w stanie sprawdzić czy nie występuje ujemny cykl. Jeśli w ostatniej iteracji wektor distances się zmieni, to znaczy, że wystąpił ujemny cykl.

4. Wykonaj krok 5 i 6 dla każdej krawędzi w sprawdzanym wierzchołku.
5. Sprawdź, czy aktualny optymalny dystans do początku tej krawędzi nie wynosi MAX_INT (zamiast nieskończoności). Jeśli dystans do początku tej krawędzi wynosi MAX_INT to przejdź do kolejnej krawędzi w tym wierzchołku.
6. Jeśli aktualny optymalny dystans do początku tej krawędzi po dodaniu kosztu krawędzi jest mniejszy od dystansu do końca tej krawędzi, to zaaktualizuj optymalny trasę i zaaktualizuj wygląd trasy w wektorze paths.
7. Jeśli została wykonana ostatnia iteracja głównej pętli, i nie dokonano zmian, to przejdź do zapisu danych w pliku. W przypadku, gdy dokonano zmian, to zwróć fałsz, symbolizujący pojawienie się ujemnego cyklu.

4.1.1.3 Find()

```
bool Find (
            const Graph & graph,
            const int & index,
            int & _where )
```

Funkcja szukająca wierzchołka w grafie.

Funkcja iteruje przez wszystkie wierzchołki i sprawdza czy istnieje jakiś o indeksach początku i końca krawędzi. Jeśli zostanie znaleziony, to funkcja się kończy i przekazywany parametr _where zawiera informacje o tym, gdzie jest ten wierzchołek. Parametr _where jest tylko istotny dla poszukiwania początkowego wierzchołka, gdyż końcowy jest tylko tworzony, jeśli go nie ma, aby istniał w grafie. Jeśli nie zostanie znaleziony, zostaje przekazana o tym stosowna informacja i w kolejnych wykonywach instrukcjach programu brakujący wierzchołek zostaje dodany.

Parametry

in	*graph*	Przeszukiwany graf.
in	*index*	Indeks szukanego wierzchołka.
in,out	*_where*	Zmienna przechowująca indeks znalezionego wierzchołka.

Zwraca

false nie znaleziono wierzchołka.

true znaleziono wierzchołka.

4.1.1.4 Save()

```
void Save (
            Graph & _graph,
            const string & _fileName,
            const int index,
            const vector< double > & _distances,
            const vector< string > & _paths )
```

Funkcja zapisująca wyniki algorytmu Bellmana-Forda dla wprowadzonych danych. Zapis jest dokonywany w pliku tekstowym podanym przez użytkownika.

Format zapisu: Wierzcholek startowy: index np. index->2->3 : cost np. index->4->5 : cost

Parametry

in	*_graph*	Graf z którego są zapisywane wyniki.
in	*_fileName*	Nazwa pliku wyjściowego.
in	*index*	Indeks wierzchołka, dla którego są zapisywane wyniki.
in	*_distances*	Koszty optymalnych tras ułożone w kolejności odpowiadającej ich destynacjom (indeks wektora distances odpowiada indeksowi w wektorze wierzchołków).
in	*_paths*	Wypisane optymalne trasy ułożone w kolejności odpowiadającej ich destynacjom (indeks wektora distances odpowiada indeksowi w wektorze wierzchołków).

4.2 model.h

Idź do dokumentacji tego pliku.

```
1
4 #pragma once
5 #include <vector>
6 #include <list>
7 #include <string>
8 #include <utility>
9 #include <iterator>
10 #include <algorithm>
11 #include <iostream>
12 #include <unordered_map>
13 #include <sstream>
14 #include <fstream>
15 using namespace std;
23 struct Edge
24 {
25     int begin;
26     int end;
27     double cost;
28 };
35 struct Node
36 {
37     int index;
38     list<Edge> edges;
39 };
47 struct Graph
48 {
49     vector <Node> nodes;
50     unordered_map<int, int> umap_nodes;
51     unordered_map<int, int> umap_nodes_reversed;
52 };
66 bool Find(const Graph& graph, const int& index, int & _where);
79 void addEdge_Directed(Graph& graph, const int begin, const int end, const double cost);
93 void Save(Graph& _graph, const string& _fileName, const int index, const vector <double>& _distances,
       const vector <string>& _paths);
94
106 bool Bellman_Ford(Graph& graph, const int index, const string & fileName);
```

4.3 Dokumentacja pliku Algorytm Bellmana-Forda/Algorytm Bellmana-Forda/widok.h

```
#include <iostream>
#include "model.h"
```

Komponenty

- struct Params

  Struktura zawierająca parametry przekazywane do programu.

Funkcje

- bool Load_parameters (Params &params, const int &Liczba_parametrów, char ∗parametry[ ])

  Funkcja, która obsługuje podawane parametry do programu wraz z ich przełącznikami.
- void Load_graph (const string &name, Graph &_graph)

  Funkcja ładująca graf z pliku tekstowego.
- void Load_variables (const string &name, vector< int > &variables)

  Funkcja wczytująca wierzchołki, dla których program będzie szukać najkrótszych tras do innych.
- void Setup_File (const string &fileName)

  Funkcja przygotowująca plik wyjściowy do pracy.
- void DisplayGraph (const Graph &graph)

  Funkcja wyświetlająca graf w konsoli (Funkcja debuggowa).
- void PrintNegativeCycle (const string &_fileName)

  Funkcja wpisująca do pliku tekstowego informację o ujemnym cyklu w grafie.
- void PrintNotFound (const string &_fileName, const int &_var)

  Funkcja wpisująca do pliku tekstowego informację o braku podanego wierzcholka.
- bool CheckVar (const int &_var, const Graph &graph)

  Funkcja sprawdzająca, czy podany wierzchołek znajduje się w grafie.
- bool FileCorrect (const string &_fileName)

  Funkcja sprawdzająca czy podany plik wejściowy jest dostępny dla programu.

4.3.1 Dokumentacja funkcji

4.3.1.1 CheckVar()

```
bool CheckVar (
            const int & _var,
            const Graph & graph )
```

Funkcja sprawdzająca, czy podany wierzchołek znajduje się w grafie.

Parametry

in	*_graph*	Przeszukiwany graf.
in	*_var*	Indeks szukanego wierzchołka.

4.3.1.2 DisplayGraph()

```
void DisplayGraph (
            const Graph & graph )
```

Funkcja wyświetlająca graf w konsoli (Funkcja debuggowa).

Parametry

in,out	*graph*	Wyświetlany graf.

4.3.1.3 FileCorrect()

```
bool FileCorrect (
            const string & _fileName )
```

Funkcja sprawdzająca czy podany plik wejściowy jest dostępny dla programu.

Parametry

in	*_fileName*	nazwa sprawdzanego pliku.

Zwraca

true program jest w stanie otworzyć plik tekstowy.

false program nie jest w stanie otworzyć pliku tekstowego.

4.3.1.4 Load_graph()

```
void Load_graph (
            const string & name,
            Graph & _graph )
```

Funkcja ładująca graf z pliku tekstowego.

Przebieg funkcji:

1. Wczytuj dane do zmiennej temp i zależnie od wartości zmiennej sequence wykonaj odpowiednie kroki:
   (a) Gdy sequence % 5 == 0 to przekonwertuj temp na typ int i zapisz w zmiennej beginning.
   (b) Gdy sequence % 5 == 1 to sprawdź czy krawędź jest skierowana, czy dwukierunkowa i ustaw odpowiednią wartość zmiennej directed.
   (c) Gdy sequence % 5 == 2 to przekonwertuj temp na typ int i zapisz w zmiennej end.
   (d) Gdy sequence % 5 == 4 to przekonwertuj temp na typ double i zapisz w zmiennej cost. Następnie zależnie od wartości zmiennej directed wykonaj funkcję addEdge_Directed() raz, lub podwójnie.
2. Zinkrementuj sequence i wczytuj dalej krawędzie.

\params[in] name Nazwa pliku tekstowego zawierającego graf.

Parametry

in,out	*_graph*	Graf, który jest ładowany z pliku tekstowego.

Zwraca

false jest niepoprawna liczba parametrow

true jest poprawna liczba parametrow

4.3.1.5 Load_parameters()

```
bool Load_parameters (
            Params & params,
            const int & Liczba_parametrów,
            char * parametry[] )
```

Funkcja, która obsługuje podawane parametry do programu wraz z ich przełącznikami.

Przebieg funkcji:

- Zainicjowanie wektora parametrów temp, w którym będą zawierać się rzutowane parametry na typ string.
- Wpisanie parametrów do wektora temp.
- Główna pętla funkcji, która iteruje przez wektor temp i gdy napotka przełącznik, to wpisuje go do wektora params w następującej kolejności:
  1. Plik wejściowy z grafem
  2. Plik wejściowy z wierzchołkami startowymi
  3. Plik wyjściowy z wynikami

  Opis działania:

1. Iteruj przez pomocniczy wektor, aż do napotkania przełącznika.
2. Element następujący przełącznik jest wartością parametru odpowiadającego dla przełącznika.
3. Zapisz parametr do wektora parametrów.

Parametry

in,out	*params*	Wektor parametrów.
in	*Liczba_parametrów*	Liczba parametrów wprowadzonych do programu.
in	*parametry*	Wprowadzone parametry do programu wraz z ich odpowiadającymi przełącznikami.

4.3.1.6 Load_variables()

```
void Load_variables (
            const string & name,
            vector< int > & variables )
```

Funkcja wczytująca wierzchołki, dla których program będzie szukać najkrótszych tras do innych.

Parametry

in	*name*	Nazwa pliku tekstowego, z którego są wczytywane zmienne.
in,out	*variables*	Wektor zmiennych zawierający wierzchołki, dla których program będzie szukać najkrótszych tras do innych wierzchołków.

4.3.1.7 PrintNegativeCycle()

```
void PrintNegativeCycle (
            const string & _fileName )
```

Funkcja wpisująca do pliku tekstowego informację o ujemnym cyklu w grafie.

Parametry

in	*_fileName*	Nazwa wyjściowego pliku tekstowego.

4.3.1.8 PrintNotFound()

```
void PrintNotFound (
            const string & _fileName,
            const int & _var )
```

Funkcja wpisująca do pliku tekstowego informację o braku podanego wierzcholka.

Parametry

in,out	*_fileName*	Nazwa wyjściowego pliku tekstowego.
in	*_var*	Indeks wierzcholka.

4.3.1.9 Setup_File()

```
void Setup_File (
            const string & fileName )
```

Funkcja przygotowująca plik wyjściowy do pracy.

Jej zadaniem jest wyczyszczenie aktualnej zawartości pliku. Potrzebne jest to, gdyż implementacja zapisywania danych do pliku dopisuje dane, a nie nadpisuje starych.

Parametry

in	*fileName*	Nazwa pliku wyjściowego.

4.4 widok.h

Idź do dokumentacji tego pliku.

```
1
4 #pragma once
5 #include <iostream>
6 #include "model.h"
7 using namespace std;
13 struct Params
14 {
15     string graphFile;
16     string pathsFile;
17     string outputFile;
18 };
38 bool Load_parameters(Params& params, const int& Liczba_parametrów, char* parametry[]);
54 void Load_graph(const string& name, Graph& _graph);
59 void Load_variables(const string& name, vector <int>& variables);
66 void Setup_File(const string& fileName);
71 void DisplayGraph(const Graph& graph);
75 void PrintNegativeCycle(const string& _fileName);
80 void PrintNotFound(const string& _fileName, const int& _var);
85 bool CheckVar(const int& _var, const Graph& graph);
91 bool FileCorrect(const string& _fileName);
```

Indeks