

Sprawozdanie z Mini-Projekt 3

Mateusz Czarnecki

Spis treści

1. Cel projektu
2. Rozwiązanie problemu
3. Szczegóły implementacyjne
4. Sposób wywołania programu
5. Wnioski i spostrzeżenia

1. Cel projektu

Celem Mini-Projektu-3 realizowanego przeze mnie na zajęciach Podstaw Informatyki było oprogramowanie i opisanie w sprawozdaniu programu wyszukującego najkrótszej drogi we wczytywanym z pliku labiryncie, z punktu **Start** do punktu **Stop**. Znalezienie rozwiązania do powyższego problemu miało na celu poszerzenie mojej wiedzy na temat ważonych grafów skierowanych oraz wykorzystania ich w algorytmicznych problemach.

Program powinien w dowolnie wybrany sposób pobierać dane definiujące labirynt z pliku tekstowego, a także posłużyć się algorytmem rekurencyjnym **DFS**, służącym do znajdowania najkrótszej drogi w grafie. Dozwolone było sformułowanie grafu na kilka sposobów: jako **macierz sąsiedztwa**, **listy sąsiedztwa** lub **macierz incydencji**.

2. Rozwiązanie problemu

W celu rozwiązania problemu, na zajęciach z Podstaw Informatyki dowiedziałem się, czym jest **skierowany graf ważony**, algorytm **DFS**, a także zapoznałem się ze szczegółami dotyczącymi implementacji labiryntu w programie. W ramach projektu, napisałem program spełniający postawione w celach założenia. Zdecydowałem się na graf skierowany w postaci **macierzy sąsiedztwa**, która w istocie stanowi tablicę liczb całkowitych, do której w odpowiedni sposób przypisywane są wartości wag krawędzi labiryntu. Labirynt definiowany jest za pomocą pliku zawierającego krawędzie, po których można się poruszać. Każda krawędź zdefiniowana jest za pomocą pary punktów: wierzchołka, z którego następuje połączenie, oraz wierzchołka, do którego następuje połączenie (Kolejność punktów ma znaczenie). Taki sposób definicji labiryntu gwarantuje jednoznaczność połączeń: zapisujemy wszystkie możliwe drogi dojścia do celu. Wagi generowane są losowo w wartościach od 0 do 10, z zaokrągleniem do jednego miejsca po przecinku przez stworzony przeze mnie generator.

3. Szczegóły implementacyjne

Program napisany został w języku C. Automat korzysta z bibliotek standardowych: **stdio.h**, **stdlib.h**, **time.h**. Zawiera się w jednym pliku: **main.c**.

Składa się z następujących funkcji:

int wczytaj_krawedzie() – Funkcja wczytuje krawędzie z pliku podanego w pierwszym argumencie wywołania, dynamicznie modyfikuje zmienną przechowującą ilość krawędzi. Funkcja zwraca 0 w przypadku poprawnego wczytania krawędzi z pliku i 1 w przypadku wystąpienia błędu.

void wczytaj_wagi() – Funkcja generuje dla każdej wczytanej krawędzi losowe wagi z przedziału (0,10), zaokrąglone do jednego miejsca po przecinku.

void zeruj_macierz() – Funkcja iteruje po macierzy i ją zeruje.

void wczytaj_macierz() – Funkcja zapisuje w utworzonej tabeli macierzy wartości wag dla poszczególnych krawędzi - definiuje połączenia w macierzy sąsiedztwa.

void zeruj_wektor() – Funkcja iteruje po podanym wektorze i go zeruje.

void DFS() – Funkcja reprezentuje rekurencyjny algorytm służący do znajdowania najkrótszej drogi w labiryncie. Zaznacza bieżący wierzchołek jako odwiedzony, po czym przechodzi do kolejnych sąsiadów wierzchołka (między którymi w macierzy sąsiedztwa określone jest połączenie) i wykonuje dla nich tą samą operację. Odwiedzone wierzchołki, a także sumę wag krawędzi zapisuje odpowiednio w tablicach **drogi[]** oraz **suma[]**, posługuje się również zmiennymi pomocniczymi takimi jak: **start**, **koniec**, **liczba_drog**, **liczba_ruchow**. W przypadku napotkania wierzchołka końcowego, algorytm wywołuje się rekurencyjnie dla startowego wierzchołka i szuka kolejnych dostępnych wierzchołków. Po znalezieniu wszystkich dostępnych wierzchołków, algorytm kończy swoje działanie.

void wyswietl_najkrotsza_droge() – Funkcja wskazuje, która ze znalezionych przez algorytm dróg jest najkrótsza oraz drukuje ją na wyjście **stdout**.

int main() – Funkcja główna programu określają parametry wywołania, wywołuje wszystkie poprzednie funkcje, deklaruje zmienne oraz tablice, wskazuje błędy, a także zwalnia pamięć po zaalokowanych dynamicznie tablicach.

4. Sposób wywołania programu

Aby wywołać program w sposób poprawny, należy podać argument określający **nazwę pliku**, w którym znajdują się wypisane krawędzie definiujące labirynt. W przypadku niepodania pierwszego argumentu, program poinformuje nas o błędzie oraz poda wzorec prawidłowego wywołania. Kolejne argumenty są opcjonalne. Drugi argument odpowiada za **długość ściany labiryntu**, np. wartość **3** odpowiada labiryntowi **3x3**, **4** – **4x4** itd. W przypadku niepodania drugiego argumentu, program przyjmuje wielkość labiryntu 3x3. Trzeci argument to liczba określająca **numer wierzchołka startowego**. W przypadku braku argumentu, start wyniesie **1**. Ostatni, czwarty argument **odpowiada za numer wierzchołka końcowego**. W przypadku braku, program ustawi wierzchołek końcowy jako liczbę o dwa mniejszą od ilości pól w labiryncie.

Gotowe przykłady wywołania:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

`./a.out 4x4.txt 4 1 14`

0	1	2
3	4	5
6	7	8

`./a.out 3x3.txt 3 1 7`

0	1
2	3

`./a.out 2x2.txt 2 1 2`

kolor pomarańczowy – ściany

kolor zielony – start

kolor czerwony – koniec

W przypadku, gdy dane w pliku nie zgadzają się z wprowadzoną długością ściany labiryntu, program nie będzie działać poprawnie.

5. Wnioski i spostrzeżenia

Podsumowując wykonaną pracę, stworzony przeze mnie program spełnia założenia wymienione w celach projektu. Ze względu na rekurencyjną naturę algorytmu, a także brak doświadczenia w tworzeniu podobnych projektów, implementacja DFS oraz stworzenie sposobu przechowywania labiryntu okazały się dla mnie wymagającym zadaniem. Dzięki projektowi zdobyłem wiedzę o algorytmach opartych na skierowanych grafach, a także doświadczenie w pracy nad algorytmami rekurencyjnymi.