**Dokumentacja końcowa**

8. „Przygotowanie pod maraton”

Treść zadania:

W lasku Kampinoskim jest wiele ścieżek biegowo-rowerowych. Przygotowując się pod maraton zawodnik chce przebiec wszystkimi ścieżkami. Każdą ze ścieżek można pobiec w obu kierunkach i każda z nich ma określoną długość.

Należy wyznaczyć taką trasę biegaczowi która pokryje wszystkie ścieżki gdzie sumaryczny dystans będzie najmniejszy.

Dane wejściowe:

*dla 4 skrzyżowań i 5 ścieżek:*

*0 1 3*

*1 2 4*

*2 3 5*

*0 3 10*

*0 2 12*

*Odp: 41*

Zrozumienie problemu:

Problem postawiony w treści zadania jest problemem chińskiego listonosza. Sprowadza się on do znalezienia cyklu przechodzącego przez wszystkie krawędzie grafu co najmniej raz, w którym suma wag krawędzi jest najmniejszą możliwą sumą wag krawędzi spośród wszystkich takiego rodzaju cykli w grafie.

Założenia implementacji:

Informacje o grafie będą przechowywane w listach sąsiedztwa. Dla każdego wierzchołka w liście będą znajdować się jego sąsiedzi z wagą krawędzi między nimi.

Informacje o cyklu będą przechowywane w wektorze – zapis wierzchołków cyklu, oraz długość cyklu w zmiennej typu unsigned int.

Użyty język: C++

Opis algorytmu rozwiązującego problem postawiony w zadaniu:

Problem chińskiego listonosza można podzielić na trzy podproblemy:

1. Gdy każdy wierzchołek jest parzystego stopnia (dochodzi do niego parzysta ilość krawędzi), istnieje w grafie cykl Eulera (cykl, który przechodzi przez każdą krawędź dokładnie raz) – aby otrzymać wynik, wyszukujemy cykl Eulera przy pomocy rekurencyjnej procedury DFS i sumujemy wagi wszystkich krawędzi.
2. Dwa wierzchołki są nieparzystego stopnia – należy znaleźć najkrótszą ścieżkę między wierzchołkami nieparzystego stopnia (do tego posłuży nam algorytm Dijkstry), zdublować krawędzie, którymi prowadzi ścieżka i znaleźć cykl Eulera, a następnie zsumować wagi wszystkich krawędzi multigrafu.
3. Więcej niż dwa wierzchołki są nieparzystego stopnia:
   1. Wyszukujemy wszystkie wierzchołki nieparzystego stopnia.
   2. Za pomocą algorytmu Dijkstry znajdujemy najkrótsze ścieżki między nieparzystymi wierzchołkami.
   3. Wyszukujemy skojarzenie tych wierzchołków w pary o najmniejszej sumie wag krawędzi – brute-force wykorzystujący algorytm DFS
   4. Krawędzie wchodzące w skład wyznaczonych ścieżek skojarzenia dublujemy w grafie początkowym.
   5. Znajdujemy cykl Eulera i sumujemy wagi wszystkich krawędzi multigrafu.

Algorytm generujący grafy:

Analizowane programem grafy możemy podzielić na trzy różne rodzaje:

1. Grafy zawierające cykl Eulera
2. Graf z dwoma nieparzystymi wierzchołkami
3. Grafy z większą ilością nieparzystych wierzchołków (większą niż 2)

Dla każdego rodzaju stworzyłam osobne generatory:

W pierwszym etapie w każdym z generatorów dodaję do wektora podaną przez użytkownika liczbę wierzchołków, mieszam je, a następnie łączę je ze sobą po kolei (0-1, 1-2, … n-3 – n-2, n-2 – n-1).

Grafy eulerowskie:

1. Łączę ze sobą ostatni i pierwszy wierzchołek (n-1 – 0)
2. Sprawdzam czy użytkownik chce więcej krawędzi niż powstało przy wstępnym łączeniu wierzchołków:
   1. Jeśli tak - przechodzę do kroku wspólnego dla wszystkich generatorów.
   2. Jeśli nie – kończę działanie generatora.

Grafy z dwoma nieparzystymi wierzchołkami:

1. Sprawdzam czy użytkownik chce więcej krawędzi niż powstało przy wstępnym łączeniu wierzchołków:
   1. Jeśli tak – losuję dwa wierzchołki (jeden o nieparzystym stopniu, drugi o parzystym) i łączę je. Następnie znowu sprawdzam czy użytkownik chce więcej krawędzi w grafie niż do tej pory powstało:
      1. Jeśli tak – przechodzę do kroku wspólnego dla wszystkich generatorów.
      2. Jeśli nie – kończę działanie generatora.
   2. Jeśli nie – kończę działanie generatora.

Grafy z większą niż 2 liczbą nieparzystych wierzchołków:

Graf powstały po wstępnym łączeniu wierzchołków posiada dwa nieparzyste wierzchołki.

1. Dopóki graf nie posiada tylu nieparzystych wierzchołków, jaką zażądał użytkownik – losuję dwa wierzchołki o parzystych stopniach i łączę je ze sobą krawędzią.
2. Sprawdzam czy użytkownik chce więcej krawędzi niż powstało do tej pory:
   1. Jeśli tak - przechodzę do kroku wspólnego dla wszystkich generatorów.
   2. Jeśli nie – kończę działanie generatora.

Krok wspólny dla wszystkich generatorów:

1. Sprawdzam jaka zostaje reszta z dzielenia liczby krawędzi, która pozostała do stworzenia przez 3:
   1. Jeśli 1 – losuję jeden z wierzchołek i dodaję pętlę
   2. Jeśli 2 – losuję dwa wierzchołki i dodaję między nimi dwie krawędzie
   3. Jeśli trzy – losuję trzy wierzchołki i dodaję między nimi po jednej krawędzi.
2. Sprawdzam czy użytkownik chce więcej krawędzi niż powstało do tej pory:
   1. Jeśli tak (liczba ta jest zawsze podzielna przez 3) – losuję trzy wierzchołki i dodaję między nimi po jednej krawędzi.
   2. Jeśli nie – kończę działanie generatora.

Testowanie:

Testowanie będzie miało trzy wersje:

1. Poprawność – w kilku plikach txt będą zapisane informacje na temat grafów, a odpowiedź na postawiony problem będzie znana. Dzięki temu będzie można sprawnie sprawdzić czy algorytm działa poprawnie.
2. Poprawność – testowanie wg danych generowanych automatycznie (losowo) z ewentualną parametryzacją określaną przez użytkownika.
3. Złożoność – aby testować algorytm pod kątem złożoności, powstanie generator dużych grafów spójnych.

Kompilacja

Kompilacja wykonuje się poprzez wywołanie *make* w folderze z plikami źródłowymi.

Wywołanie:

./aal <flags> <parameters>

Argumenty wywołania:

|  |  |  |
| --- | --- | --- |
| **Flagi** | **Parametry** | **Znaczenie** |
| *-file* | *<file\_name>* | Wykonuje program z danymi z pliku o nazwie *file\_name*. Plik musi znajdować się w folderze *data* |
| *-eulerian* | *<number\_of\_vertices> <number\_of\_edges>* | Wykonuje program z wygenerowanymi danymi. Generowany jest graf, który posiada cykl eulera. |
| *-2odd* | *<number\_of\_vertices> <number\_of\_edges>* | Wykonuje program z wygenerowanymi danymi. Generowany jest graf, który posiada dwa nieparzyste wierzchołki |
| *-moreOdd* | *<number\_of\_vertices> <number\_of\_edges> <number\_of\_odd\_vertices>* | Wykonuje program z wygenerowanymi danymi. Generowany jest graf, który posiada więcej niż dwa nieparzyste wierzchołki. |

Parametry number\_of\_vertices, number\_of\_edges i number\_of\_odd\_vertices służą do sparametryzowania generatora.

Dodatkową flagą dodawaną przed innymi jest flaga *-analysis*, która sprawia, że liczony jest czas wykonania głównego algorytmu, potrzebny do analiz złożoności.

Format pliku w wywołaniu z parametrem *-file*:

v01 v02 le0

v11 v12 le1

...

vi1 vi2 lei

vi1 - indeks początkowego wierzchołka opisywanej krawędzi

vi2 - indeks końcowego wierzchołka opisywanej krawędzi

lei - długość opisywanej krawędzi

Wyjscie

Na wyjsciu otrzymujemy przebieg cyklu Eulera oraz jego długość.

Wyświetlana wcześniej struktura grafu wspomaga analizę problemu.

Dekompozycja:

Projekt podzielony jest na pakiety:

* Graph – struktura danych jaką jest graf i wszelkie metody potrzebne do przeanalizowania go.
* Generator – klasa generatora grafów, zależnego od parametrów wywołania programu.
* Funkcja main – wczytuje parametry wywołania, uruchamia generator grafów lub wczytuje graf z pliku i rozwiązuje problem.