**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

Wierzchołki numerowane są od 0 po kolei liczbami naturalnymi.

Wyjście

Na wyjściu 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.

Ocena złożoności:

Aby poprawnie wykonać pomiary złożoności obliczeniowej, uprościłam problem i przyjęłam stałą zależność liczby krawędzi od liczby wierzchołków. Dla grafów rzadkich E = ( |V| \* (|V|-1) \* 0.5) \* 0.25, dla gęstych E = ( | V | \* (|V|-1) \* 0.5) \* 0.75, a dla grafów o gęstości pośredniej E = ( |V| \* (|V|-1) \* 0.5) \* 0.5. Stąd wynika zmiana szacowanych złożoności algorytmów:

Rozwiązanie problemu postawionego w zadaniu dla:

* Grafów Eulera: O(V)\*O(E) + O(V2) = O(V) \* O(V2) + O(V2) = O(V3)
* Grafów z dwoma nieparzystymi wierzchołkami: O(V3) + O(V2) = O(V3)
* Grafów z liczbą nieparzystych wierzchołków większą niż 2: c\*O(V2), gdzie c – ilość wierzchołków nieparzystych.

Pomiary czasów wykonania:

Wszystkie pomiary zostały wykonane dla 50 różnych, losowo wygenerowanych grafów o tej samej liczbie wierzchołków i krawędzi, podanej jako parametry wywołania. Wyniki podane w tabelach są średnimi arytmetycznymi wszystkich pomiarów dla określonych parametrów.

Czasy zawarte w tabelach podane są w mikrosekundach.

Pomiary i porównanie złożoności z teoretyczną złożonością asymptotyczną (pesymistyczną):

Dla grafów rzadkich:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Eulerian graph | | | 2 odd vertices | | | more odd vertices | | |
|  | O(V^3) | | | O(V^3) | | | c\*O(V^2) | | |
| rzadkie | V | T(V) | q(V) | V | T(V) | q(V) | V | T(V) | q(V) |
| 1237 | 100 | 1000000 | 1,120635 | 100 | 1000000 | 1,306868 | 100 | 80000 | 1,049046 |
| 2793 | 150 | 3375000 | 1,034458 | 150 | 3375000 | 1,035566 | 150 | 180000 | 0,9984 |
| 4975 | 200 | 8000000 | 1,006521 | 200 | 8000000 | 0,982131 | 200 | 320000 | 0,973256 |
| 7781 | 250 | 15625000 | 1 | 250 | 15625000 | 1 | 250 | 500000 | 1 |
| 11212 | 300 | 27000000 | 1,008242 | 300 | 27000000 | 1,021747 | 300 | 720000 | 1,053048 |
| 15268 | 350 | 42875000 | 1,00467 | 350 | 42875000 | 1,034629 | 350 | 980000 | 1,127977 |
| 19950 | 400 | 64000000 | 1,07699 | 400 | 64000000 | 1,029599 | 400 | 1280000 | 1,195941 |

Dla grafów o gęstości „pośredniej”:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Eulerian graph | | | 2 odd vertices | | | more odd vertices | | |
|  | O(V^3) | | | O(V^3) | | | c\*O(V^2) | | |
| pośrednie | V | T(V) | q(V) | V | T(V) | q(V) | V | T(V) | q(V) |
| 2475 | 100 | 1000000 | 0,988535 | 100 | 1000000 | 0,94131 | 100 | 1000000 | 1,735809 |
| 5587 | 150 | 3375000 | 0,966332 | 150 | 3375000 | 0,901858 | 150 | 3375000 | 1,309372 |
| 9950 | 200 | 8000000 | 1,053031 | 200 | 8000000 | 0,989286 | 200 | 8000000 | 1,106237 |
| 15562 | 250 | 15625000 | 1 | 250 | 15625000 | 1 | 250 | 15625000 | 1 |
| 22425 | 300 | 27000000 | 1,036144 | 300 | 27000000 | 1,003896 | 300 | 27000000 | 0,989586 |
| 30537 | 350 | 42875000 | 0,983457 | 350 | 42875000 | 0,977865 | 350 | 42875000 | 0,961402 |
| 39900 | 400 | 64000000 | 1,090197 | 400 | 64000000 | 0,972914 | 400 | 64000000 | 0,940947 |

Dla grafów gęstych:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Eulerian graph | | | 2 odd vertices | | | more odd vertices | | |
|  | O(V^3) | | | O(V^3) | | | c\*O(V^2) | | |
| geste | V | T(V) | q(V) | V | T(V) | q(V) | V | T(V) | q(V) |
| 3712 | 100 | 1000000 | 0,86868 | 100 | 1000000 | 0,886381 | 100 | 1000000 | 1,377406 |
| 8381 | 150 | 3375000 | 0,924495 | 150 | 3375000 | 0,942704 | 150 | 3375000 | 1,093606 |
| 14925 | 200 | 8000000 | 0,968185 | 200 | 8000000 | 0,99988 | 200 | 8000000 | 1,001823 |
| 23343 | 250 | 15625000 | 1 | 250 | 15625000 | 1 | 250 | 15625000 | 1 |
| 33637 | 300 | 27000000 | 0,972028 | 300 | 27000000 | 0,960059 | 300 | 27000000 | 0,977681 |
| 45806 | 350 | 42875000 | 0,97066 | 350 | 42875000 | 0,99173 | 350 | 42875000 | 0,924742 |
| 59850 | 400 | 64000000 | 0,998248 | 400 | 64000000 | 0,964648 | 400 | 64000000 | 0,959138 |

Wnioski:

Analizując wyniki q(V) można zauważyć, że dla grafów eulerowskich złożoność została dosyć dobrze oszacowana. Lekko odbiega od normy w przypadku grafów gęstych dla małej liczby wierzchołków - występuje minimalne niedoszacowanie.

W przypadków grafów z dwoma nieparzystymi wierzchołkami oszacowanie jest trochę gorsze, ale wciąż nie najgorsze. Widzimy, że dla małej ilości wierzchołków w grafie oszacowanie nie sprawdza się.

Gdy w grafie występuje więcej niż 2 wierzchołki nieparzyste (analizowana ilość - 8), dla gęstości pośredniej i grafów gęstych występuje przeszacowanie, zaś dla grafów rzadkich - niedoszacowanie.

Wszelkie przeszacowania mogą wynikać z przyjętej pesymistycznej złożoności czasowej. Rzeczywiste wyniki mogą być lepsze niż oszacowane.

Natomiast niedoszacowania mogą wynikać z dużej ilości wywołań funkcji, dodawania elementów do *std::vector* i innych operacji, które nie są wprost związane z logiką algorytmu.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 8 nieparzystych wierzchołków | | | | | | |
| V | rzadkie | | pośrednie | | gęste | |
| 200 | 4975 | 163010 | 9950 | 381031 | 14925 | 735082 |
| 250 | 7781 | 261702 | 15562 | 672732 | 23343 | 1433095 |
| 300 | 11212 | 396842 | 22425 | 1150375 | 33637 | 2421118 |
|  |  |  |  |  |  |  |
| 10 nieparzystych wierzchołków | | | | | | |
| V | rzadkie | | pośrednie | | gęste | |
| 200 | 4975 | 5798941 | 9950 | 5528800 | 14925 | 6129935 |
| 250 | 7781 | 5936505 | 15562 | 5945874 | 23343 | 7096090 |
| 300 | 11212 | 5789614 | 22425 | 6773513 | 33637 | 8255550 |

Zrobiłam też kilka pomiarów dla większej ilości nieparzystych wierzchołków – 10.

Porównując te dwie tabele, widzimy, że czas działania algorytmu wraz ze wzrostem ilości nieparzystych wierzchołków, bardzo mocno rośnie.

Pierwsze kolumny w każdej z rozważanych grup grafów określają ilość krawędzi.