Dokumentacja programu do dzielenia grafów

Bartosz Szumerowski, Damian Mazur

1 Cel projektu

Program ten ma za zadanie podzielić graf podany przez użytkownika w pliku w formie tekstowej na części ustalone przez użytkownika o podanym, w procencie, maksymalnym marginesie różnicy ilości wierzchołków w grafach powstałych w wyniku podzielenia grafu. Użytkownik ma możliwość ustalenia ilości, na które graf ma być podzielony, jaki jest maksymalny margines różnicy między dwoma grafami po ich podziale na dwie części, oraz w jakim formacie pliku mają zostać zwrócone grafy.

2 Zgodność i Kompatybilność

Program do dzielenia grafów napisany w języku C działa na komputerach z systemem operacyjnym GNU/Linux. Program kompilowany jest za pomocą kompilatora **GCC (GNU C Compiler)**. Przed uruchomieniem programu należy zainstalować podany kompilator. Można to zrobić w powłoce Debiana i większości dystrybucji opartych o niego, przy użyciu polecenia "***sudo apt install gcc***".

3 Kompilacja

Aby skompilować program wystarczy wpisać komende "***make***" w katalogu zawierającym wszystkie pliki należące do programu. To skompiluje cały kod zawarty w plikach do pliku wykonywalnego "a.out".

4 Uruchamianie

Aby uruchomić program trzeba wpisać polecenie "***/.a.out***" wraz z przynajmniej jednym argumentem. Program uruchamia się za podaniem do trzech parametrów , gdzie:

- **Pierwszy argument** - nazwa pliku zawierający graf zapisany w formacie tekstowym. Format tego argumentu to napis.

- **Drugi argument** - ile razy ma zostać wykonananane dzielenie grafu. W przypadku podania wartości 0, graf wejściowy zostanie podany na wyjście bez zmian. Formatem tego argumentu jest liczba naturalna wraz z zerem. (opcjonalne, domyślnie 1)
- **Trzeci argument** - o jaki maksymalny margines wyrażony w procentach mają różnić się ilości wierzchołków w podzielonych. Formatem tego argumentu jest liczba nieujemna (nie musi być całkowita) zapisana w formie dziesiętnej. (opcjonalne, domyślnie 10)

Przykład ***"/.a.out graf.txt 5 25"***. Program z takimi argumentami dzieli graf podany w pliku o nazwie "graf.txt" 5 razy z marginesem 25% i grafy mają zostać zapisane w formie tekstowej do pliku graf.out.

Istnieje specjalne wywołanie programu w celu wypisania pomocy - ***"/.a.out --help"*** lub ***"/.a.out -h"***. Wypisze on listę i opis dostępnych argumentów, flag i metod dzielenia, a następnie zwróci wartość 0. Podanie po wywołaniu pomocy jakichkolwiek innych argumentów będzie traktowane jako błąd.

5 Dostępne flagi

- **-o plik.out** - argument zawierający ścieżkę do pliku wyjściowego, do którego ma zostać zapisany graf wyjściowy. Domyślnie ustawiona jest flaga -o z plikiem wyjściowym "graf.out", chyba że pojawi się flaga -t.
- **-t** - flaga określająca, że graf wyjściowy ma być wypisany w terminalu.
- **-b** - flaga zmieniająca format wypisywania grafu z tekstowego na binarny.
- **-m** - flaga określa metodę (algorytm) dzielenia grafu - dostępne metody zostały opisane w następnej sekcji.
- **-s** - flaga określa nasienie (seed) funkcji losującej dla algorytmu Kargera. Domyślnie nasienie jest pobierane z systemowego zegara.
- **-i** - flaga określa ilość iteracji dla algorytmu Kargera. Domyślnie wartość wynosi co najmniej 50 i jest obliczana ze wzoru

$$50 * \left(\frac{n}{10}\right)^2$$

gdzie n jest liczbą wierzchołków. Wartość argumentu podana przez użytkownika nie może być mniejsza od 1.

6 Dostępne metody

- **1 - Algorytm Kargera** - metoda ta wykorzystuje algorytm Kargera, który jest algorytmem probabilistycznym, co oznacza że nie ma stuprocentowej szansy na powodzenie. Prawdopodobieństwo na poprawny wynik

przy pojedyńczym wykonaniu jest znikome i spada wraz z większymi grafami, dlatego domyślna ilość iteracji jest spora i rośnie wraz z ilością wierzchołków w grafie. Użytkownik może zdefiniować własną ilość iteracji jeżeli zależy mu na lepszym wyniku lub przeciwnie, na większej wydajności.

- **2 lub 3 - Prosty algorytm** - jest to prosty algorytm brute force sprawdzający każdą możliwą kombinację wierzchołków. Dzięki temu zagwarantowany jest najlepsze możliwe rozwiązanie, lecz kosztem bardzo słabej wydajności. Złożoność tej metody to

$$O(2^n)$$

gdzie n to ilość wierzchołków w grafie. Metoda ta ma dwa warianty. Przy wariancie 2 będzie ona także sprawdzała czy grafy po podzieleniu są spójne i nie dokona podziału jeżeli takowe nie są (podobnie jak algorytm Kargera). Przy wariancie 3 nie będzie dokonywała takiego sprawdzenia przez co wynikiem podziału mogą być więcej niż 2 grafy. Po podziale grafy te będą traktowane jako oddzielne spójne grafy, a nie jako jeden niespójny (na którym podział nie byłby możliwy, gdyż sam w sobie byłby podzielony).

7 Przykłady uruchomienia

- **"./a.out graf.in 5 20"** - wykonuje 5 podzieleń na grafie/grafach w pliku graf.in z marginesem różnicy 20% i zapisuje wyjście do pliku graf.out w trybie tekstowym
- **"./a.out graf.in -t"** - wykonuje 1 podzielenie w grafie/grafach w pliku graf.in z marginesem różnicy 10% i wypisuje wyjście na standardowe wyjście w trybie tekstowym
- **"./a.out graf123.in 10 -o graf123.out -b"** - wykonuje 10 podzieleń w grafie/grafach w pliku graf123.in z marginesem różnicy 10% i zapiuje wyście do pliku graf123.out w trybie binarnym

8 Działanie programu

Program do działania wymaga przynajmniej jednego argumentu, pierwszego. Drugi i trzeci argument są opcjonalne. Domyślna wartość dla drugiego argumentu to 1, a dla trzeciego to 10. Jeżeli użytkownik nie poda pierwszego argumentu to program nie zadziała. Na końcu program stworzy pliki, których ilość jest zależna od drugiego argumentu, w formacie tekstowym. Ostatecznie, gdy program nie napotka na żadne błędy, zwróci wartość 0.

9 Możliwe komunikaty błędów

W trakcie działania programu, może on napotkać na różne błędy. Te błędy zostaną skomunikowane odpowiednimi wiadomościami, które omówienie znajduje się poniżej. Po napotkaniu chociaż jednego błędu program natychmiast kończy działanie i zwraca wartość -1.

- **"Nie określono pliku wejściowego"** - Nie podano pierwszego argumentu z nazwą pliku wejściowego.
- **"Podano dodatkowe argumenty po prośbie o pomoc"** - Po użyciu "–help" lub "-h" jako pierwszego argumentu podano dodatkowe argumenty.
- **"Podana liczba podziałów jest ujemna"** - Podana liczba podziałów (drugi argument) jest liczbą ujemną, a nią nie powinien być.
- **"Podana różnica procentowa jest ujemna"** - Podany maksymalny margines (trzeci argument) jest liczbą ujemną, a nią nie powinien być.
- **"Podano za dużo argumentów"** - Liczba podanych argumentów nie będących flagami jest za duża (więcej niż 3).
- **"Podano flagę -o bez podania nazwy pliku wyjściowego"** - Po fladze "-o" nie podano argumentu z nazwą pliku wyjściowego.
- **"Próba użycia -o dwukrotnie"** - Podano flagę "-o" dwukrotnie.
- **"Próba użycia -t dwukrotnie"** - Podano flagę "-t" dwukrotnie.
- **"Próba użycia -b dwukrotnie"** - Podano flagę "-b" dwukrotnie.
- **"Próba użycia -m dwukrotnie"** - Podano flagę "-m" dwukrotnie.
- **"Próba użycia -s dwukrotnie"** - Podano flagę "-s" dwukrotnie.
- **"Próba użycia -i dwukrotnie"** - Podano flagę "-i" dwukrotnie.
- **"Próba użycia -o i -t jednocześnie"** - Argumenty zawierają zarówno flagę "-o", jak i "-t" - flagi te się wykluczają.
- **"Podano flagę -m bez podania numeru metody"** - Podano flagę "-m" lecz nie podano następnego argumentu określającego numer metody.
- **"Podano flagę -s bez podania nasienia"** - Podano flagę "-s" lecz nie podano następnego argumentu określającego nasienie.
- **"Podano flagę -i bez podania ilości iteracji"** - Podano flagę "-i" lecz nie podano następnego argumentu określającego liczbę iteracji.
- **"Podano niepoprawny identyfikator metody przy fladze -m"** - Podany numer metody jest niepoprawny - nie jest 1, 2 lub 3.

- **"Podano niepoprawną ilość iteracji przy fladze -i"** - Podana liczba iteracji jest mniejsza od 1.
- **"Zdefiniowano nasienie przy jednoczesnym określeniu prostej metody cięcia"** - Podano flagę "-s" przy jednoczesnym określeniu metody cięcia ("-m") na 2 lub 3.
- **"Zdefiniowano ilość iteracji przy jednoczesnym określeniu prostej metody cięcia"** - Podano flagę "-i" przy jednoczesnym określeniu metody cięcia ("-m") na 2 lub 3.
- **"Próba użycia nieznanej flagi"** - Podano argument, który zaczyna się myślnikiem i nie jest znaną flagą. Błąd ten może się pojawić przy próbie podana pliku, którego nazwa zaczyna się myślnikiem. W takim przypadku jedynym wyjściem jest zmiana nazwy pliku.
- **"Błąd otwierania pliku wejściowego"** - W trakcie otwierania pliku wejściowego wystąpił błąd. Może to wynikać m.in. z tego, że nie istnieje lub użytkownik nie ma uprawnień do jego odczytu.
- **"Błąd otwierania pliku wyjściowego"** - W trakcie otwierania pliku wyjściowego wystąpił błąd. Może to wynikać m.in. z tego, że użytkownik nie ma uprawnień do zapisu w danym miejscu lub istnieje folder o nazwie pliku wyjściowego.
- **"Błąd formatu pliku wejściowego"** - W trakcie czytania pliku wejściowego wystąpił błąd, który najpewniej jest efektem błędnego formatu pliku wejściowego.
- **"Pusta sekcja połączeń"** - W trakcie przetwarzania grafu wczytanego z pliku, napotkano sekcję połączeń, która jest pusta - różnica między dwoma kolejnymi elementami w szóstej lub daszej linii, albo między ostatnim, a następnym elementem dwóch z tych linii, wynosi 0 lub 1.
- **"Błędna konfiguracja sekcji połączeń"** - Podobne do pustej sekcji połączeń, ale różnica jest ujemna.
- **"Próba połączenia wierzchołka z samym sobą"** - Wczytany z pliku graf zawiera połączenie wierzchołka z samym sobą.
- **"Próba ponownego połączenia wierzchołków"** - Wczytany z pliku graf zawiera dwa wierzchołki, które są ze sobą połączone więcej niż jeden raz.
- **"Błąd programu"** - Jeżeli wystąpi to jest to najprawdopodobniej problem z samym programem, a nie z wejściem użytkownika, czy plikiem wejściowym.

10 Rodzaj i struktura grafu

Grafy, przyjmowane na wejście i przekazywane na wyjście, są grafami nieskierowanymi bez wag. Graf nie musi być spójny, a więc można traktować go jako kilka grafów. Wierzchołki grafów mają ustalone pozycje. Pozycje te są określone przez X i Y będące liczbami naturalnymi lub zerem

11 Format plików tekstowych

Format ten jest używany zarówno przez pliki wejściowe, jak i pliki wyjściowe gdy wyjście jest ustawione na format tekstowy.

- **Pierwsza linia** - Występuje tylko w plikach wyjściowych, zawiera informację o liczbie udanych podziałów grafu.
- **Druga linia** - Określa ilość kolumn, tzn. X wierzchołka musi być mniejsze od tej wartości.
- **Trzecia linia** - Tablica określająca pozycje X wierzchołków; jeżeli pozycja X danego wierzchołka została określona prze n-ty element tej tablicy to identyfikatorem tego elementu jest n.
- **Czwarta linia** - Każda wartość to początek fragmentu tablicy w trzeciej linii, który określa pozycje X wierzchołków dla danego wiersza. Fragment kończy się wraz z początkiem kolejnego fragmentu lub końcem tablicy w trzeciej linii; ilość elementów w czwartej tablicy określa liczbę wierszy, tzn. Y wierzchołka musi być mniejsze od tej wartości.
- **Piąta linia** - Tablica zawierająca identyfikatory wierzchołków, wykorzystywana do określenia połączeń między wierzchołkami.
- **Kolejne linie** - Każda wartość to początek fragmentu tablicy w piątej linii, który kończy się wraz z początkiem następnego fragmentu (z tej lub następnej linii) lub końcem ostatniej linii. Linia ta może występować wielokrotnie, co oznacza, że graf podzielony jest na kilka grafów.

12 Format plików binarnych

Jako że format plików tekstowych można sprowadzić do kilku tablic liczb, gdzie kolejne liczby są oddzielone średnikami, a kolejne tablice znakami nowej linii, format plików binarnych ma identyczną strukturę, ale różni się sbosobem w jaki zapisano te tablice. Liczby zapisane są na czterech bajtach z kolejnością little endian i są one oznaczone (w przypadku wystąpienia liczb ujemnych, wczytywanie zakończy się błędem, wyjątkiem jest liczba -1 która oznacza separator tablic). Elementy w tablicy występują bez żadnych znaków je dzielących, gdyż takie nie są potrzebne - liczba ma zawsze długość cztery bajty. W formacie tym liczba -1 (0xFFFFFFFF) ma specjalne znaczenie i oddziela od siebie kolejne tablice.

Plik może, ale nie musi się kończyć tą liczbą. Jedyną różnicą w samym znaczeniu tablic w stosunku do formatu tekstowego jest to, że w binarnych plikach wyjściowych nie występuje pierwsza tablica, a więc nie ma podanej informacji o liczbie udanych podziałów grafu.

13 Moduły projektu

- **main.c** - Odpowiada za kolejnych funkcji z różnych modułów, mających na celu wczytanie pliku, podzielenie i zapisanie podzielonego grafu do pliku.
- **arguments.c/arguments.h** - Odpowiada za przetworzenie argumentów i flag z linii polecenia.
- **array.c/array.h** - Implementacja tablicy dynamicznej.
- **cut.c/cut.h** - Odpowiada za cięcie grafów. Nie zawiera implementacji metod szukających możliwych cięć a funkcję zarządzającą szukaniem cięć, a następnie ich wykonywaniem. Zawiera też funkcje wspólne dla wszystkich metod szukania cięć.
- **errors.c/errors.h** - Zawiera jedną funkcję - error - która to odpowiada za ładne wypisanie błędu i zakończenie działania procesu z kodem -1.
- **file.c/file.h** - Odpowiada za wczytanie pliku w formacie tekstowym i zapisanie do formatu tekstowego lub binarnego. Moduł ten nie odpowiada za tworzenie struktury grafu, a linie pliku są przechowywane w tablicach int'ów, którymi są w kontekście obu używanych formatów.
- **graph.c/graph.h** - Odpowiada za tworzenie struktury grafu/grafów na podstawie wczytanego pliku (tablic int'ów), oraz do jej ponownego zapisaniu po modyfikacjach. Przy tworzeniu grafu brana jest jedynie pod uwagę piąta linia i kolejne linie, czyli to jak dane wierzchołki są ze sobą połączone. Poprzednie linie nie są brane pod uwagę, gdyż poprzednie linie definiują ilość i pozycje wierzchołków, które pozostają niezmienne.
- **karger_cut.c/karger_cut.h** - Implementacja algorytmu Kargera do szukania możliwego cięcia grafu (metoda 1).
- **list.c/list.h** - Implementacja listy dwukierunkowej.
- **simple_cut.c/simple_cut.h** - Zawiera prosty algorytm szukający cięcia grafu (metoda 2 lub 3).

W folderze src/old znajdują się też dwa nieużywane modułu projektu będące pozostałością po nieudanej implementacji algorytmu spektralnego.

14 Działanie programu

Program rozpoczyna się od przetworzenia argumentów i flag wejściowych w module arguments.c. Przetworzone argumenty trafiają następnie do struktury je przechowującej:

```
typedef struct
{
 const char* inputFile; // NULL means it wasn't specified
 const char* outputFile; // NULL means stdout

 int divisions;
 double maxDiff; // in percentages

 bool useBinaryMode;

 int method;

 bool definedSeed; // works only with method = 0
 int seed;

 int iterations; // if < 1 - auto
} Arguments;
```

Program następnie próbuje wczytać plik o ścieżce ze zmiennej inputFile. Plik przechowywane jest w strukturze zawierającej jego kolejne linie:

```
typedef struct
{
    int maxNodes;

    Array* xCoords; // of int
    Array* xCoordsStart; // of int

    Array* conns; // of int
    Array* connStarts; // of Array* of int
} File;
```

Użyty w strukturze typ "Array" jest tablicą dynamiczną zdefiniowaną w array.h:

```
typedef struct
{
    void* arr;
    int len;
    int _elementSize;
    int _fullSize;
    bool _ofPointers;
} Array;
```

Nazwy elementów tej struktury powinny być dość zrozumiałe, nie licząc "ofPointers". Pole te określane jest przy tworzeniu tablicy i w przy wartości true, przy dodawaniu wartości zamiast wskaźników na wartości, przyjmuje bezpośrednio wartości. Tak samo przy pobieraniu elementu z tablicy, zamiast wkaźnika na ten element, zwróci ten element. Ustawienie tego może znacznie ułatwić operacje na tablicy, gdy elementami tablicy są wskaźniki. Co do samej struktury, pola zaczynające się podkreślnikiem powinny być traktowane jako prywatne pola tablicy i nie powinny być modyfikowane z zewnątrz. W kolejnym kroku linia piąte i ostatnie linie, tzn. "conns" i "connStarts" są konwertowane na tablicę grafów (o długości takiej samej jak długość tablicy "connStarts"), gdzie graf i jego wierzchołki mają następujące struktury:

```
typedef struct
{
    int id, pos;
    Array* conns; // of Node*
    bool mark;

    bool simpleCut_secondPart;
} Node;

typedef struct
{
    Array* nodes; // of Node*
} Graph;
```

Następnie grafy są dzielone (w zależności od wybranej metody) algorytmem Kargera lub prostym algorytmem brute force. Algorytmy te są wykorzystywane do znalezienia możliwego cięcia. Znalezione cięcie jest następnie zwracane w formie struktury:

```
typedef struct
{
    // list node of list of graphs
    ListNode* graph;

    // array of booleans defining side of each node
    bool* sideArray;

    // number of required cuts
    int cuts;
} Cut;
```

Po znalezieniu możliwych cięć wybierane jest cięcie wymagające najmniejszej liczby usuniętych wierzchołków (z najmniejszą wartością "cuts"). W podanej powyżej strukturze można też zobaczyć typ ListNode, który to jest elementem listy dwukierunkowej. Lista dwukierunkowa i jej elementy mają następujące pola:

```
typedef struct
{
    void* val;
    void* prev;
    void* next;
} ListNode;

typedef struct
{
    ListNode* first;
    ListNode* last;
    int size;
} List;
```

Po podzieleniu grafy zapisywane są z powrotem do struktury "File", modyfikując "conns" i "connStarts", a plik jest następnie zapisywany do pliku wyjściowego "outputFile" w zdefiniowanym przez użytkownika ("useBinaryMode") trybie.

15 Repozytorium git

`https://github.com/SBQD-nng/Dzielenie-Grafu`