Specyfikacja funkcjonalna programu dzielącego graf

Adam Domański, Oliwier Osiński

30.04.2025

Link do repozytorium:

`https://github.com/t0q1/JIMP2_graph_C/tree/main`

Cel projektu

Celem projektu jest stworzenie programu dzielącego graf w języku C. Program ma podzielić graf określoną liczbę razy, przy jak najmniejszej liczbie usuniętych krawędzi. Różnica w liczbie wierzchołków w nowo utworzonych grafach nie może się różnić o więcej niż podany przez użytkownika margines procentowy. Wszystkie parametry mają być przyjmowane z linii poleceń. Program ma wypisywać otrzymany graf w trybie tekstowym lub binarnym w zależności od preferencji użytkownika.

Argumenty wywołania programu

Do prawidłowego uruchomienia programu należy podać następujące argumenty:

- ścieżka pliku wejściowego: ścieżka do pliku, który zawiera tekstową interpretację grafu;
- liczba podzieleń grafu (N): dodatnia liczba całkowita, której domyślna wartość wynosi 1;
- margines różnicy procentowej między powstałymi grafami (M): nieujemna liczba całkowita, której domyślna wartość wynosi 10 (wartości interpretowane w procentach);
- ścieżka pliku wyjściowego: ścieżkę do pliku, w którym zostanie zapisany graf po dokonaniu podziałów

Plik wejściowy

Do prawidłowego dokonania podziału grafu potrzebne będą informacje o grafie, który ma zostać podzielony. Dane te mają być zapisane w formacie tekstowym w pliku txt.

Plik z grafem wejściowym składa się z następujących linii:

- Maksymalna liczba węzłów w wierszu.
- Indeksy węzłów w poszczególnych wierszach.
- Wskaźniki na pierwsze indeksy węzłów w liście wierszy.
- Grupy węzłów połączone przy pomocy krawędzi.
- Wskaźniki na pierwsze węzły w poszczególnych grupach.

Przykład grafu w pliku wejściowym:

```
3
0;2;1;0
0;2;3
1;3;4;4;2
0;3;5
```

Dane wyjściowe

Wyniki operacji programu mogą zostać wyświetlone lub zapisane na dwa różne sposoby.

1. W domyślnym trybie tekstowym najpierw w pierwszej linii zwracana jest wartość pomyślnie podzielonych grafów, a następnie podzielone grafy w identycznym formacie jak w pliku wejściowym.
2. W trybie binarnym, gdzie ma być zwracane tylko grafy. Tryb ten nie ma z góry określonego formatu.

Warto zaznaczyć, że w zależności od podanych flag, program może jednocześnie wyświetlić wynik w terminalu, jak i zapisać go do pliku.

Program przyjmuje następujące flagi:

- -h - flaga wyświetla instrukcje obsługi programu
- -o plik.out - flaga przyjmuje argument w postaci ścieżki do pliku, w którym ma zostać zapisany wynik operacji
- -t - flaga powoduje wyświetlenie wyniku w terminalu
- -b - flaga zmienia sposób wyświetlania wyniku z tekstowego na binarny

Specyfikacja implementacyjna programu dzielącego graf

Struktura plików

Opisywany program składa się z następujących katalogów i plików:

- Makefile - plik wykonywalny kompilujący i uruchamiający różne testowe scenariusze programu;
- bin - katalog zawierający skompilowany program o nazwie "graf";
- include - katalog zawierający pliki nagłówkowe;
- src - katalog zawierający pliki źródłowe c;
- tests - katalog zawierający testowe pliki wejściowe opisujące grafy

Argumenty wywołania

Do poprawnego uruchomienia programu niezbędne jest podanie jednego obowiązkowego argumentu. Opcjonalnie można dodać dwa dodatkowe argumenty zmieniające parametry programu.

1. **plik.in** - ścieżka do pliku, w którym zapisany jest graf przeznaczony do podziału.

   W przypadku, gdy podany plik nie istnieje, program zróci błąd o treści: Blad: Nie udalo sie otworzyc pliku wejsciowego o podanej sciezkce. Przerywam dzialanie. i zakończy działanie.

   Natomiast, gdy dane przedstawiające graf są niepoprawne, program zwróci błąd o treści: "Blad: Dane w pliku przedstawiajace graf sa niepoprawne. Przerywam dzialanie." i zakończy działanie.

2. **N** (opcjonalny) - dodatnia liczba całkowita podzieleń grafu na podgrafy, której wartość domyślna wynosi **1**.

   W przypadku, gdy nie poda się wartości liczbowej, program zwróci błąd o treści: "Blad: Liczba podzieleń grafu została niepoprawnie zdefiniowana. Przerywam działanie." i zakończy działanie.

   Natomiast, gdy podana liczba, będzie niedodatnie lub zmiennoprzecinkowa, to program zwróci błąd o treści: "Blad: Liczba podzieleń grafu musi być większa bądź równa 1. Przerywam działanie." i zakończy działanie.

3. **M** (opcjonalny) - nie ujemna liczba całkowita nie przekraczająca wartości 100, liczba przedstawia graniczną wartość procentową, pod którą musi się zmieścić różnica wierzchołków podzielonych podgrafów, jej domyślna wartość wynosi **10**.

   W przypadku, gdy nie poda się wartości liczbowej, program zwróci błąd o treści: "Blad: Liczba marginesu różnicy procentowej została niepoprawnie zdefiniowana. Przerywam działanie." i zakończy działanie.

   Natomiast, gdy podana liczba, będzie ujemna lub zmiennoprzecinkowa, to program zwróci błąd o treści: "Blad: Liczba marginesu różnicy procentowej między wierzchołkami powstałych grafów musi znajdować się w przedziale [0-100]. Przerywam działanie." i zakończy działanie.

Przykłady użycia argumentów podczas wywoływania programu znajdują się w sekcji **Uruchomienie programu**.

Flagi

Program przyjmuje następujące flagi:

- **-h** - flaga wyświetlająca informacje o argumentach programu oraz dostępne flagi wraz z ich opisami.
- **-o plik.out** - flaga przyjmująca jako argument ścieżkę do pliku, do którego ma zostać zapisany wynik końcowy programu. Domyślna wartość argumentu flagi to **"wynik.txt"**.
- **-b** - flaga zmienia tryb wyświetlania wyniku z domyślnie tekstowego na binarny.
- **-t** - flaga sprawiająca, że wynik końcowy programu zostanie wypisany w terminalu. Można łączyć z flagą -b, wtedy w terminalu zostanie wyświetlony wynik binarny.

Przykłady użycia flag podczas wywoływania programu znajdują się w sekcji **Uruchomienie programu**.

Uruchomienie programu

Do kompilacji programu należy użyć komendy **make** w terminalu lub od razu wybrać wcześniej przygotowany scenariusz testowy, wpisując **make {nazwa_testu}** o których więcej w sekcji **Testy**.

Aby uruchomić program należy w teminalu wywołać plik wykonywalny znajdujący się w katalogu bin: **/bin/graf**, a następnie podać odpowiednie argumenty, o których mowa była w sekcjach **Argumenty wywołania** oraz **Flagi**.

Przykład wywołania programu wczytującego graf z pliku /tests/graf.csrrg, wypisującego wynik w trybie tekstowym w terminalu oraz do pliku podzial.txt:
./bin/graf /tests/graf.csrrg 2 20 -o podzial.txt -t

Przykład wywołania programu wczytującego graf z pliku /tests/graf2.csrrg, wypisującego wynik w trybie binarnym w terminalu:
./bin/graf /tests/graf2.csrrg -t -b

Format wyjściowy

Program zawsze zapisuje końcowy wynik w pliku, który został podany w odpowiedniej fladze podczas wywoływania (domyślnie "wynik.txt").

Na wynik końcowy w trybie tekstowym (domyślny) składa się:

- Liczba udanych podziałów grafu w pierwszej linijce.
- Następnie graf w takim samym formacie co w pliku wejściowym.

W przypadku trybu binarnego, podawany jest jedynie sam graf. Sposób zapisania grafu w postaci binarnej przedstawia sekcja **Plik binarny**.

Plik binarny

W trybie binarnym każda liczba reprezentowana jest binarnie na 32 bitach. Znak **\n** jest reprezantowany przez ciąg 32 jedynek, natomiast znak **;** przez ciąg 31 jedynek i zera.

Przykładowo:

- Liczba 5 jest reprezentowana przez 00000000000000000000000000000101
- Znak **\n** przez 11111111111111111111111111111111
- Znak **;** przez 11111111111111111111111111111110

  Przykład: graf w formie tekstowej: 5\n0;2;4;1;3;0;1;2;3\n0;3;5;9\n0;1;4;7;5;5;8;6;8;7\n0;2;5;7
  zapisa w postaci binarnej:
  00000000000000000000000000000101 11111111111111111111111111111111
  00000000000000000000000000000000 11111111111111111111111111111110
  00000000000000000000000000000010 11111111111111111111111111111110
  00000000000000000000000000000100 11111111111111111111111111111110
  00000000000000000000000000000001 11111111111111111111111111111110
  00000000000000000000000000000011 11111111111111111111111111111110
  00000000000000000000000000000000 11111111111111111111111111111110
  00000000000000000000000000000001 11111111111111111111111111111110
  00000000000000000000000000000010 11111111111111111111111111111110

00000000000000000000000000000011 11111111111111111111111111111111
00000000000000000000000000000000 11111111111111111111111111111110
00000000000000000000000000000011 11111111111111111111111111111110
00000000000000000000000000000101 11111111111111111111111111111110
00000000000000000000000000001001 11111111111111111111111111111111
00000000000000000000000000000000 11111111111111111111111111111110
00000000000000000000000000000001 11111111111111111111111111111110
00000000000000000000000000000100 11111111111111111111111111111110
00000000000000000000000000000111 11111111111111111111111111111110
00000000000000000000000000000101 11111111111111111111111111111110
00000000000000000000000000000101 11111111111111111111111111111110
00000000000000000000000000001000 11111111111111111111111111111110
00000000000000000000000000000110 11111111111111111111111111111110
00000000000000000000000000001000 11111111111111111111111111111110
00000000000000000000000000000111 11111111111111111111111111111111
00000000000000000000000000000000 11111111111111111111111111111110
00000000000000000000000000000010 11111111111111111111111111111110
00000000000000000000000000000101 11111111111111111111111111111110
00000000000000000000000000000111

Zapis grafu w formacie binarnym nie zawiera spacji, tylko stały ciąg zer i jedynek. Powyższy przykład zawiera spacje dla łatwiejszej wizualizacji.

Pliki źródłowe

W katalogu src znajdują się następujące pliki:

- **main.c** - plik, w którym obsługiwane są argumenty i flagi wywołujące program oraz inicjalizuje podział grafu.
- **grapf.c** - plik zawierający funkcje do obsługi struktury grafu oraz funkcje odpowiedzialne za faktyczny podział grafu.
- **file_graph.c** - plik zawierający funkcje odpowiedzialne za odczyt grafu z pliku, jak i jego zapis do pliku.
- **getopt.c** - plik zawierający funkcje do poprawnego i bezbłędnego odczytu argumentów oraz flag wywoływanego programu.

Struktury

Struktura grafu:

```
typedef struct {
    int n;
    int start;
    int parent;
    Node ** adj;
} Graph;
```

gdzie:

- n - liczba wierzchołków grafu
- start oraz parent - zmienne potrzebne do podziału grafu
- adj - "lista"wierzchołków grafu

Struktura wierzchołka:

```
typedef struct n{
    int vertex;
    struct n *next;
} Node;
```

gdzie:

- vertex - etykieta wierzchołka
- next - wskaźnik do następnego wierzchołka

Funkcja podziału grafu

Użyty w naszych programie algorytm do podziału grafu jest uproszczoną wersją algorytmu Kerninghana-Lina.
W pierwszym kroku wierzchołki są reprezentowane w tablicy intów o długości ilości wierzchołków w grafie, etykieta węzła odpowiada indeksowi w tej tablicy. Do tablicy przypisywane są po równo wartości (0 i 1), wartości te będą reprezentować na koniec dwa grafy, jeden z wierzchołkami o wartościach 0 i drugi z wierzchołkami o wartościach 1. Następnie przy pomocy pętli, algorytm przechodzi kolejno wszystkie wierzchołki w tablicy i sprawdza czy przeniesienie danego wierzchołka (zmiana wartość na przeciwną) do drugiej części zmniejsza liczbę przecięć. Warto zaznaczyć, że wierzchołek jest przenoszony tylko wtedy gdy nie zaburzy równowagi dwóch części. Następnie z powstałej tablicy składającej się z 0 i 1, są tworzone dwa grafy. W przypadku gdy powstałe grafy spełniają warunek różnicy liczby wierzchołków, to pierwotny dzielony graf w tablicy grafów jest podmieniany na pierwszy nowo powstały graf, a drugi powstały graf jest dodawany na koniec tablicy grafów. W przypadku gdy nie spełniają tego warunku, indeks wskazujący na obecny graf w tablicy grafów jest zwiększany o jeden i proces podziału zaczyna się od początku dla kolejnego grafu w kolejce. Po zakończeniu wszyskich podziałów grafów, wszyskie grafy z tablicy grafów zostają scalone w jeden duży graf, który ma tyle samo wierzchołków co graf wczytywany z pliku wejściowego. Ze względu na parametry start i parent w strukturze grafu, każdy graf zna pierwotne wartości etykiet z wczytywanego grafu. Następnie funkcja zwróci liczbę udanych podziałów grafów i cały proces dobiegnie do końca.