1. Cel i zakres programu

`przetnij-graf` to narzędzie konsolowe służące do **podziału nieskierowanego, prostego grafu** na zadaną liczbę podgrafów, z zachowaniem równowagi wielkości w granicach określonego marginesu procentowego. Umożliwia to:

- wyodrębnienie spójnych fragmentów grafu,
- analizę każdego podgrafu niezależnie,
- eksport wyników w formacie tekstowym (CSR-like) lub binarnym,
- opcjonalny podgląd w terminalu.

2. Interfejs użytkownika

2.1. Wywołanie

`przetnij-graf <input file> <N> <M> [-o <output file>] [-t] [-b]`

- `<input file>` – plik z grafem w prostym formacie tekstowym:

  ```
  V E
  src  dest
  …
  src_E dest_E
  ```

- `<N>` – liczba podgrafów (N 1),
- `<M>` – margines nierównowagi, interpretowany jako **maksymalny udział procentowy** wierzchołków jednego podgrafu (np. 20.0 oznacza, że żadna część nie może mieć $>$ 20% wszystkich wierzchołków).
- `-o <output file>` – ścieżka pliku wyjściowego (domyślnie `plik.out`),
- `-t` – dodatkowo drukuje reprezentację CSR-like w terminalu,
- `-b` – zamiast formatu tekstowego generuje plik binarny.

2.2. Format wyjściowy

2.2.1. Tekstowy (CSR-like)

1. `rowPos[0..V]` – tablica długości `V+1`, gdzie `rowPos[i]` wskazuje początek listy sąsiadów wierzchołka `i` w tablicy `rowNodes`.
2. `rowNodes[0..2E-1]` – spłaszczona lista wszystkich sąsiadów (bo graf nieskierowany: każda krawędź zapisana dwukrotnie).
3. `component[0..V-1]` – numer podgrafu (`0…splitCount-1`) dla każdego wierzchołka.
4. `compCounts[0..splitCount-1]` – liczba wierzchołków w każdej części.
5. `-1` – separator kończący dump.

2.2.2. Binarne

- Little-endianowy zapis kolejno:
  1. `int32` – `numVertices`
  2. `int32` – `numEdges`
  3. Dla każdej z `numEdges` krawędzi:
     - `int32 src`,
     - `int32 dest`.
- **Uwaga:** nie zapisujemy `splitCount`.

3. Struktury danych i funkcje publiczne

3.1. Struktury

```
typedef struct Edge {
    int src, dest;
} Edge;

typedef struct Graph {
    int    numVertices;  // liczba wierzchołków
    int    numEdges;     // liczba krawędzi
    Edge  *edges;        // tablica krawędzi
    int    splitCount;   // liczba podgrafów (ustawiana w split.c)
} Graph;
```

3.2. Kluczowe funkcje publiczne

Funkcja	Sygnatura	Opis
`createGraph`	`Graph *createGraph(int V, int E)`	Alokuje `Grap`
`freeGraph`	`void freeGraph(Graph *g)`	Zwalnia `edge`
`copyGraph`	`Graph *copyGraph(const Graph *orig)`	Głęboka kopi
`graphFromTextFile`	`Graph *graphFromTextFile(const char *fn)`	Wczytuje pro
`graphToTextFile`	`void graphToTextFile(const Graph *g, const char *fn)`	Zapis CSR-li
`graphToBinaryFile`	`void graphToBinaryFile(const Graph *g, const char *fn)`	Zapis binarny
`graphToString`	`char *graphToString(const Graph *g)`	Generuje CS
`graphFromString`	`Graph *graphFromString(const char *str)`	Odtwarza `Gr`
`splitGraph`	`Graph *splitGraph(Graph *orig, int number, float marginPct)`	Główna funk odrzuca (`NUL`

4. Przepływ głównego algorytmu (`splitGraph`)

1. **Przygotowanie**

- `number++` → pracujemy na `N+1` grupach;
- `balancedGraph = copyGraph(orig);`
- `totalV = orig->numVertices`.

2. **Obliczenie `desiredSizes[0..number-1]`**

```
base = totalV / number;
rem  = totalV % number;
for(c=0; c<number; ++c)
  desiredSizes[c] = base + (c < rem ? 1 : 0);
```

3. **Wielośródłowy BFS**
   - Wylicz stopnie (`degree[u]`), wybierz `number` wierzchołków-ziaren z najwyższym stopniem.
   - Kolejka elementów `(comp, u)`; oznacz `component[u] = comp`, `visited[u] = true`.
   - Rozszerzaj BFS aż każda grupa osiągnie `desiredSizes[comp]`.
4. **Dopełnienie**
   - Dla każdej krawędzi `(u,v)`: jeżeli `u` ma `component`, `v` nie – przypisz `v` do tej samej grupy (i odwrotnie).
5. **Filtrowanie krawędzi**
   - Przepisz tylko te krawędzie, których oba końce są w tym samym `component`.
6. **Weryfikacja marginesu**

```
for(i=0; i<number; ++i){
  perc = compCounts[i]*100.0f/totalV;
  if(perc > marginPct) { invalid = true; break; }
}
if(invalid) { freeGraph(balancedGraph); return NULL; }
```

7. **Zakończenie**
   - `balancedGraph->splitCount = number-1;`
   - Zwróć `balancedGraph`.

5. Przykłady użycia

5.1. Podział na 3 części, margines 20%, zapis tekstowy

Wejście (`graph.txt`):

```
8 8
0 1
0 2
1 3
```

```
2 3
3 4
4 5
5 6
6 7
```

`./przetnij-graf graph.txt 3 20.0 -o out.txt`

Wygeneruje `out.txt` z CSR-like dump i komunikat:

`Graph written to text file: out.txt`

Opcjonalnie:

`./przetnij-graf graph.txt 3 20.0 -o out.txt -t`

– wyświetli dump w terminalu.

5.2. Podział na 4 części, margines 15%, zapis binarny

`./przetnij-graf graph.txt 4 15.0 -o out.bin -b`

Wygeneruje `out.bin` (little-endian) i:

`Graph written to binary file: out.bin`

6. Obsługa błędów i kody wyjścia

Kod wyjścia	Komunikat na `stderr`	Przyczyna
1	`Usage: przetnij-graf <input> <N> <M> [-o <out>] [-t] [-b]`	Zbyt mało argumentów
2	`Failed to load graph from file: <input file>`	Błąd otwarcia / parsow
3	`Failed to split graph into <N> parts with margin <M>%`	Algorytm nie spełnił wa
(inny 0)	–	Błąd zapisu pliku wyjśc

7. Środowisko i zależności

- **Kompilator**: GCC 15.1 ([gcc.gnu.org][1])
- **Make**: GNU Make (4.0)
- **Biblioteki C**: standardowa biblioteka C (glibc 2.30 lub ekwiwalent)
- **Python** (opcjonalnie, do `visualize_graph.py`): 3.8, z modułami `networkx` i `matplotlib`

MAKEFILE:

```
CC = gcc
CFLAGS = -Wall -O2
all: przetnij-graf
przetnij-graf: main.o graph.o split.o
```

```
    $(CC) $(CFLAGS) -o $@ $^
%.o: %.c %.h
    $(CC) $(CFLAGS) -c $<
clean:
    rm -f *.o przetnij-graf
```

8. Wytyczne stylu kodu

- **Formatowanie**: 4-znakowy tabulator lub 2 spacje; limit długości linii 80 znaków.
- **Nazewnictwo**:
  - Struktury i typy: `PascalCase` (`Graph`, `Edge`).
  - Funkcje: `camelCase` (`splitGraph`, `graphToTextFile`).
  - Zmienne lokalne: `snake_case` (`desired_sizes`, `row_pos`).
- **Komentarze**:
  - Każda funkcja publiczna – krótki blok w nagłówku pliku `.h`.
  - W kodzie – przy nietrywialnych fragmentach algorytmów (BFS, korekcja marginesu).
- **Zarządzanie pamięcią**: zawsze sprawdzać wynik alokacji (`malloc`/`calloc`) i zwalniać w każdej ścieżce zakończenia.

9. Licencja i autorzy

- **Licencja**: Do ustalenia.
- **Autorzy**:
  - Karol Juszczak (karol.juszczak.stud@pw.edu.pl)
  - Yaroslav Shevchuk (yaroslav.shevchuk.stud@pw.edu.pl)