

Lab 2: Maksymalne przepływy

W ramach laboratorium należy zaimplementować algorytm Forda-Fulkersona, a następnie użyć go jako składnika algorytmu wyznaczania spójności krawędziowej grafów.

Zadanie 1 [🔗](#)

Dany jest graf skierowany $G = (V, E)$, funkcja $c: E \rightarrow \mathbb{N}$ dająca wagi krawędziom, oraz wyróżnione wierzchołki s i t . Należy znaleźć maksymalny przepływ w grafie G pomiędzy s i t , tzn. funkcję $f: E \rightarrow \mathbb{N}$ spełniającą warunki definicji przepływu, zapewniającą największą przepustowość.

Do rozwiązania zadania należy wykorzystać algorytm Forda-Fulkersona, porównując dwie strategie znajdowania ścieżek powiększających:

- przy użyciu przeszukiwania metodą DFS
- przy użyciu przeszukiwania metodą BFS (algorytm Edmondsa-Karpa)

Zadanie 2 [dodatkowe – temu tematowi poświęcimy laboratorium 3]

Dany jest graf nieskierowany $G = (V, E)$. *Spójnością krawędziową* grafu G nazywamy minimalną liczbę krawędzi, po których usunięciu graf traci spójność. Przykładowo:

- spójność krawędziowa drzewa = 1
- spójność krawędziowa cyklu = 2
- spójność krawędziowa n -kliki = $n-1$

Opracuj i zaimplementuj algorytm obliczający spójność krawędziową zadanego grafu G , wykorzystując algorytm Forda-Fulkersona oraz następujący fakt:

(Tw. Menger) Minimalna ilość krawędzi które należy usunąć by zadane wierzchołki s, t znalazły się w różnych komponentach spójnych jest równa ilości krawędziowo rozłącznych ścieżek pomiędzy s i t

Wskazówka: jak można zinterpretować ilość krawędziowo rozłącznych ścieżek jako problem maksymalnego przepływu?

Proponowana kolejność prac

- wczytaj graf korzystając z funkcji `loadDirectedWeightedGraph` z biblioteki `dimacs` (funkcja działa tak samo jak `loadWeightedGraph` z poprzedniego laboratorium, ale zwraca graf skierowany, czyli listę trójek (u, v, w) oznaczających krawędź z wierzchołka u do v o wadze w).

- zaimplementuj konwersję grafu na strukturę, która pozwoli na wygodną implementację algorytmu Forda-Fulkersona - w szczególności potrzebny będzie wydajny dostęp do sąsiednich wierzchołków (listy/zbiory sąsiedztwa) oraz miejsce na informację o aktualnym przepływie przez każdą krawędź. Przydatna będzie możliwość znajdowania krawędzi "w przeciwną stronę".
- zaimplementuj przeszukiwanie DFS/BFS na sieci residualnej (pamiętaj, że sieć residualna może mieć krawędzie, których nie ma w oryginalnym grafie!)
- zaimplementuj algorytm uaktualniania przepływu przy użyciu zadanej ścieżki powiększającej
- zaimplementuj liczenie spójności krawędziowej

Pomocne pliki

W ramach laboratorium należy wykorzystać:

- [dimacs.py](#) – wczytywanie grafów (teraz również funkcja do wczytywania grafów skierowanych)
- [graphs-lab2.zip](#) – grafy testowe
 - zadanie 1 - folder `flow`
 - zadanie 2 - folder `connectivity`

W przykładowych grafach szukamy przepływów między pierwszym i ostatnim wierzchołkiem!