# Wstęp Teoretyczny

**W teorii grafów sieć przepływowa** (znana również jako sieć transportowa) jest grafem skierowanym, w którym każda krawędź ma pojemność, a każda krawędź otrzymuje przepływ. Wielkość przepływu na krawędzi nie może przekraczać pojemności krawędzi. Często w badaniach operacyjnych graf skierowany jest nazywany siecią, wierzchołki nazywane są węzłami, a krawędzie – łukami. Przepływ musi spełniać ograniczenie, że wielkość przepływu do węzła jest równa wielkości przepływu z niego, chyba że jest źródłem, które ma tylko przepływ wychodzący lub ujście, które ma tylko przepływ przychodzący.

**Algorytm Forda-Fulkersona** to algorytm, który rozwiązuje problem maksymalnego przepływu. To znaczy, biorąc pod uwagę sieć z wierzchołkami i krawędziami pomiędzy tymi wierzchołkami, które mają określone wagi, ile „przepływu" może jednocześnie przetwarzać sieć? Przepływ może oznaczać wszystko, ale zazwyczaj oznacza dane przez sieć komputerową.

Algorytm Forda-Fulkersona zakłada, że danymi wejściowymi będzie graf G wraz z wierzchołkiem źródłowym s i wierzchołkiem ujścia t. Graf to dowolna reprezentacja grafu ważonego, w którym wierzchołki są połączone krawędziami o określonych wagach. Aby rozróżniać początek i koniec sieci przepływu, musi istnieć również wierzchołek źródłowy i wierzchołek ujścia.

# Opis skryptu

Ten skrypt składa się z dwóch głównych klas. Pierwsza to Edge w którym inicjalizują się dane krawędzi, tzn. ich źródło, ujście, oraz pojemność. Druga to Graph. W niej inicjalizujemy dwa słowniki: edges który zawiera w sobie poszczególne obiekty klasy Edge, adjacents zawiera liste wszystkich sąsiadujących krawędzi.

Funkcja **add_edge()**

Przyjmuje 3 parametry **source**(źródło), **sink**(ujście), oraz **capacity**(pojemność).

add_edge() najpierw sprawdza czy source równa się sink. Jak nie to tworzy dwa obiekty klasy Edge dla odpowiednich dwóch wierzchołków i dodaje ich do

słownika edge. Dalej musimy te dane dodać również do słownika adjacents. Jeśli tego source i sink niema w adjacents to dodajemy.

Funkcja **find_path()**

To jest normalny algorytm BFS, który wyszukuje możliwą ścieżke od źródła do ujścia i zwraca ją. Najpierw funkcja sprawdza czy źródło jest równe ujście. Dalej iteruje po wszystkich krawędzi obliczając wartość rezydualną dla każdej. Potem sprawdza czy ta wartość jest większa od zera(czy jest możliwość na zwiększenie dalszego przepływu), jak nie to przechodzimy na następną krawędź a poprzednią dodajemy do ścieżki. Potem zwracamy wynik działania.

Funkcja **ford_fulkerson()**

Korzystając z BFS, możemy dowiedzieć się, czy istnieje ścieżka od źródła do ujścia. BFS buduje również tablicę path. Korzystając z tablicy path, przechodzimy przez znalezioną ścieżkę i znajdujemy możliwy przepływ przez tę ścieżkę, znajdując minimalną pojemność rezydualną wzdłuż ścieżki. Później dodamy przepływ znalezionej ścieżki do ogólnego przepływu. Pod koniec sumujemy cały przepływ i go zwracamy

# Sposób uruchamiania skryptu

Dany skrypt nie zawiera żadnej obcej biblioteki więc nie ma potrzeby nic instalować. Dla sprawdzenia poprawności danych zwracanych przez **ford_fulkerson(obowiązkowo wskazać nazwę wierzchołków źródła i ujścia)** zaimplementowałem strukturę unittest. Ona od razu tworzy dwa grafy, oraz sprawdza ich maksymalny przepływ.

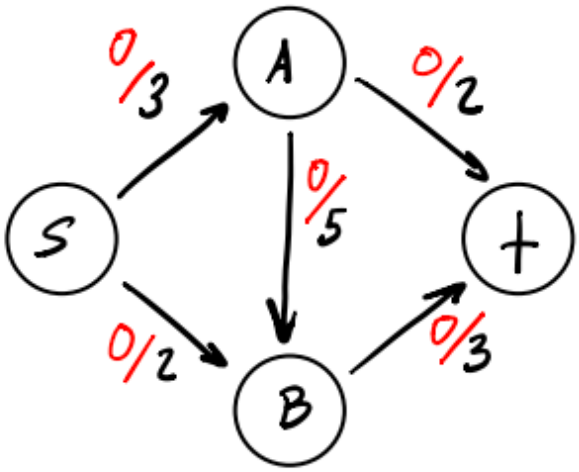
## Uruchamianie w konsoli:

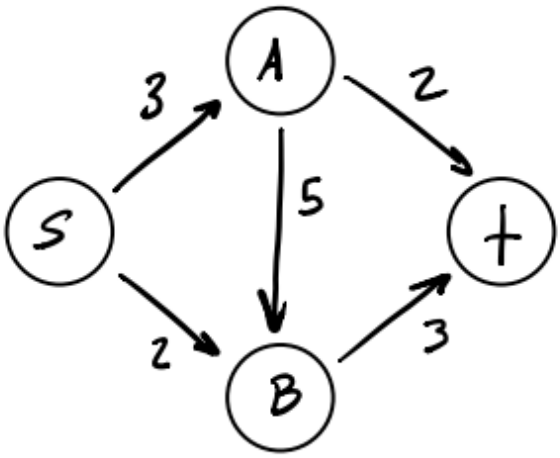
**Windows:** python Fulkerson.py

**Linux:** python3 Fulkerson.py

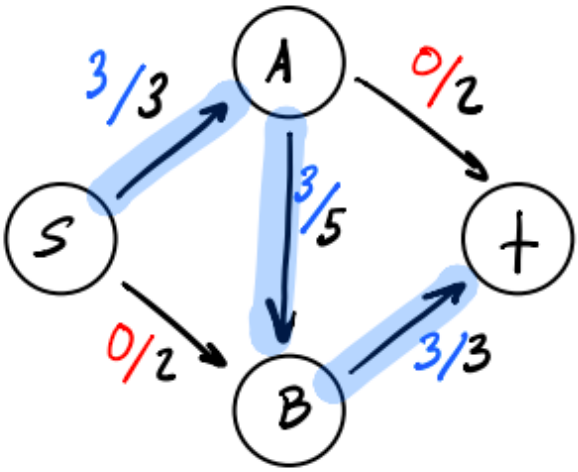
## Vizualizacja algorytmu

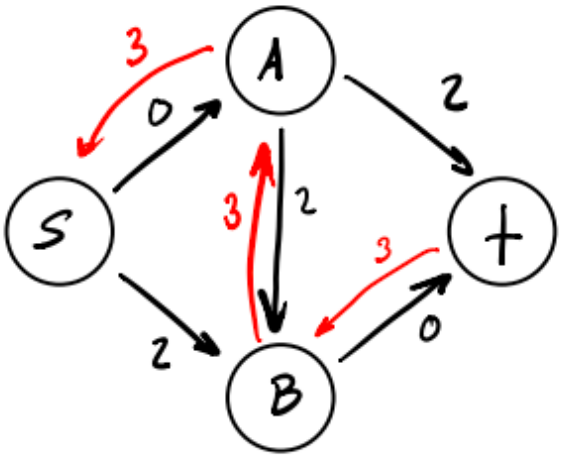
1.Graf początkowy



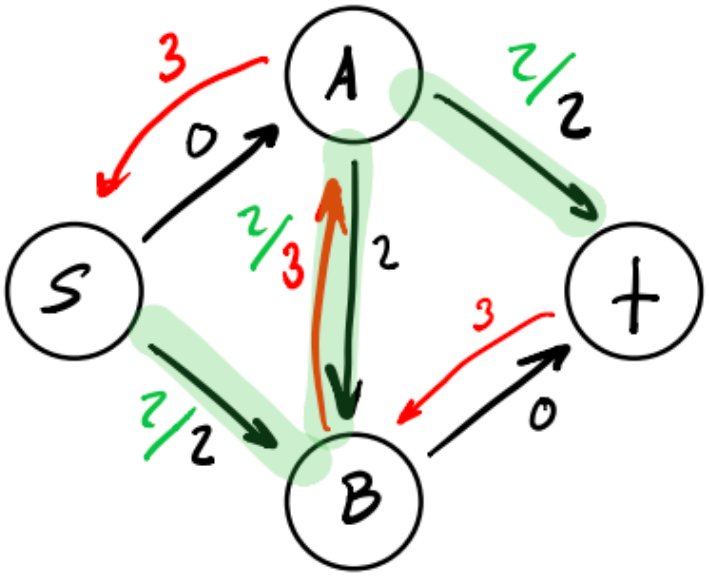




Przeprowadzamy raz algorytm i osiągamy przepływ 3. Aktualizujemy graf rezydualny

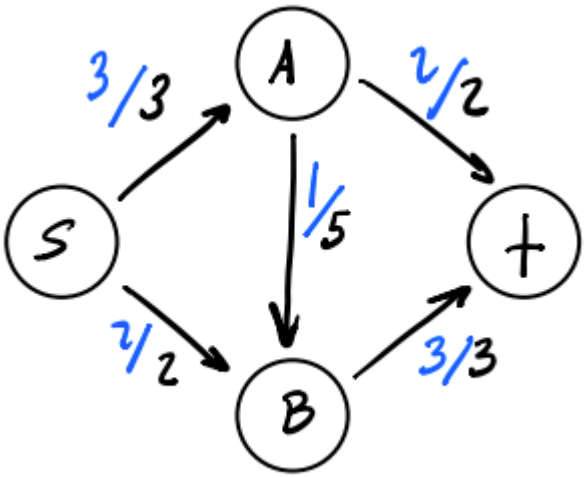


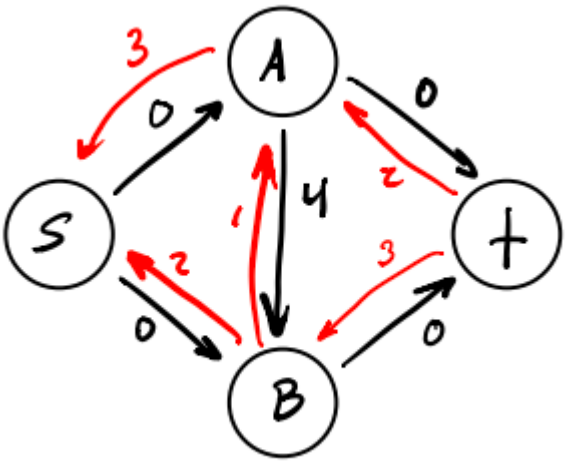




Przeszukujemy zaktualizowany graf rezydualny w poszukiwaniu nowej ścieżki s-t. Nie ma już dostępnych krawędzi do przodu, ale możemy użyć krawędzi tylnej, aby zwiększyć przepływ. Możemy zmniejszyć przepływ wzdłuż krawędzi A-B o 2, co pozwoli nam wykorzystać obie krawędzie prowadzące do t!




Rozszerzamy bieżący przepływ o nasze wyniki powyżej i graf rezydualny.




Makszmalny przepływ równy 5.