Politechnika Wrocławska

Badania operacyjne i optymalizacja dyskretna

Sprawozdanie z wykonania zadania

Autorzy: Sebastian Żółkiewicz *259337* Piotr Kulczycki *259257*

Kod przedmiotu: W04ISA-SM0401G, Grupa: 2

Termin zajęć: czwartek 11:15 - 13:00

Politechnika
Wrocławska

Spis treści

1 Numer ćwiczenia

Wykonano zadanie MaxFlow - algorytm Ford-Fulkerson oraz Autorska metoda odcięć w 2 odsłonach.

2 Termin oddania + okres spóźnienia

Termin oddania zadania 21.11.2024.

3 Problem MaxFlow

Problem maksymalnego przepływu (MaxFlow) polega na znalezieniu maksymalnej ilości przepływu, który może przejść od źródła do ujścia w sieci przepływowej. Sieć ta jest reprezentowana jako graf skierowany, gdzie wierzchołki to węzły, a krawędzie mają przypisane pojemności (maksymalny przepływ, jaki może przez nie przejść).

4 Dane wejściowe

Dane dostarczone przez prowadzącego reprezentują graf skierowany z dodatnimi wagami na krawędziach.

- Pierwsza liczba N określa liczbę wierzchołków grafu, w tym przypadku $N = 5$.
- Kolejne N wierszy zawiera po N liczb, które przedstawiają wagę krawędzi od węzła A (określonego przez numer wiersza) do węzła B (określonego przez numer kolumny).
- Waga wynosząca 0 oznacza brak połączenia między wierzchołkami A i B.

Przykładowa macierz wag wygląda następująco:

$$\begin{matrix} 5 & 0 & 0 & 0 & 8 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 3 & 0 & 6 & 0 \\ 8 & 0 & 0 & 0 & 8 \\ 0 & 9 & 0 & 6 & 0 \end{matrix}$$

5 Algorytm Forda-Fulkersona

Algorytm Forda-Fulkersona jest jednym z najczęściej używanych algorytmów rozwiązujących ten problem. Opiera się on na iteracyjnym znajdowaniu ścieżek powiększających przepływ i aktualizowaniu stanu przepływów w sieci, aż nie będzie można znaleźć żadnej ścieżki powiększającej przepływ. W przypadku, gdyby pojemności krawędzi były liczbami całkowitymi, algorytm ten zawsze kończy się po skończonej liczbie kroków.

Działanie algorytmu (pseudokod)

Algorithm 1: Algorytm Forda-Fulkersona dla problemu MaxFlow

1: flow $\leftarrow 0$ {Początkowy przepływ wynosi 0}
2: **while** istnieje ścieżka powiększająca przepływ w grafie **do**
3: Znajdź ścieżkę powiększającą przepływ w grafie (BFS)
4: $c_{\min} \leftarrow$ minimalna pojemność na tej ścieżce
5: Zwiększ przepływ o $c_{\min}$ wzdłuż tej ścieżki
6: **for** każdą krawędź (u, v) na ścieżce **do**
7: Zaktualizuj przepływ:

$$\text{flow}(u, v) \leftarrow \text{flow}(u, v) + c_{\min}$$

8: Zaktualizuj pojemność krawędzi:

$$c(u, v) \leftarrow c(u, v) - c_{\min}$$

9: Zaktualizuj pojemność odwrotnej krawędzi:

$$c(v, u) \leftarrow c(v, u) + c_{\min}$$

10: **end for**
11: **end while**
12: Zwróć przepływ flow jako wynik

Złożoność czasowa

Złożoność czasowa algorytmu Forda-Fulkersona zależy od wybranego sposobu znajdowania ścieżek powiększających przepływ. Jeśli do tego celu używamy przeszukiwania wszerz (BFS), wówczas algorytm znajdzie ścieżki powiększające w czasie $O(E)$, gdzie E to liczba krawędzi w grafie.

Złożoność czasowa całego algorytmu zależy od liczby iteracji, w których znajdowana jest nowa ścieżka powiększająca przepływ. W najgorszym przypadku algorytm może wykonywać tyle iteracji, ile wynosi maksymalny przepływ (gdy przepływ zwiększa się o 1 jednostkę w każdej iteracji). Zatem złożoność czasowa algorytmu Forda-Fulkersona w najgorszym przypadku wynosi:

$$O(\text{max\_flow} \cdot E)$$

gdzie max_flow to maksymalny przepływ w sieci.

6 Greed1MinCut

Algorytm `Greed1MinCut` wykonuje iteracyjne wycinanie wierzchołków w celu znalezienia minimalnego przecięcia. W każdej iteracji algorytm wybiera wierzchołek, który ma połączenie z już usuniętymi wierzchołkami, i oblicza wartość przecięcia. Proces powtarza się, dopóki nie zostaną wycięte wszystkie wierzchołki.

Za każdym razem wybierany jest wierzchołek którego skutek (wartość przecięcia po przeniesieniu) będzie jak najmniejsza, bez uwzględnienia obecnej wartości przecięcia.

7 Greed2MinCut

Algorytm `Greed2MinCut` jest podobny do `Greed1MinCut`, ale różni się sposobem wyboru wierzchołka który jako następny zostanie odcięty.

Za każdym razem wybierany jest wierzchołek którego łącza przepustowość prowadząca do odciętych już wierzchołków jest największa.

Działanie algorytmu (pseudokod)

Algorithm 2: Algorytm typu GreedMinCut

1: nodes_to_cut $\leftarrow$ $Nodes/[source, sink]$ {Wszystkie wierzchołki z wykluczeniem pierwszego i ostatniego }
2: nodes_cuted $\leftarrow$ $[sink]$ {ostatniou wierchołek traktujemy jako już odcięty }
3: minCut $\leftarrow$ $CalcCutValue()$ {Początkowa wartość przecięcia }
4: **while** nodes_to_cut jest niepusty **do**
5: nodes_candidate $\leftarrow$ $filter$(nodes_to_cut, nodes_cuted) {Spośród wszystkich nieodciętych wierzchołków, wybierz te które mają połączenie z dowolnym już wyciętym wierzchołkiem}
6: **if** nodes_candidate.$isEmpty() = true$ **then**
7: Przerwij pętle while
8: **else**
9: Wybierz kandydata i przenieś go do wierzchołków odciętych
10: **end if**
11: minCut $\leftarrow$ min(**CalcCutValue**(), minCut) {Zapisz najmniejszą wartość minCut}
12: **if CalcCutValue**() == $None$ **then**
13: Return 0 {sink i source nie są połączone}
14: **end if**
15: **end while**
16: Zwróć ostatnią przyjętą wartość **minCut**

Złożoność czasowa

Złożoność czasowa obu algorytmów zależy od liczby wierzchołków i krawędzi w grafie. W obu przypadkach głównym elementem jest iteracyjne przeszukiwanie wierzchołków do wycięcia, gdzie dla każdego wierzchołka obliczany jest koszt przecięcia. Zakładając, że algorytmy przetwarzają wszystkie wierzchołki, a każda iteracja wymaga obliczenia przepływów dla wszystkich krawędzi, złożoność czasowa dla obu algorytmów wynosi:

$$O(V^2 \cdot E)$$

gdzie V to liczba wierzchołków, a E to liczba krawędzi w grafie. Złożoność może się różnić w zależności od struktury grafu oraz tego, jak efektywnie wykonywane są obliczenia przepływów.

W praktyce algorytm Greed2MinCut nie musi dla każdego kandydata wierzchołka liczyć wartości przecięcia. Dzięki dobranemu sposobowi wyboru wierzchołka do odcięcia, obliczenia wyboru wierzchołka sprowadziły się do sumy tablicy tworzonej do sprawdzenia czy dany wierzchołek może być wybrany. Te metody sprawiają że algorytm mimo tego że częściej daje rozwiązania sub-optymalne (nie optymalne), działa znacząco szybciej niż jego odpowiednik Greed1MinCut

8 Badania

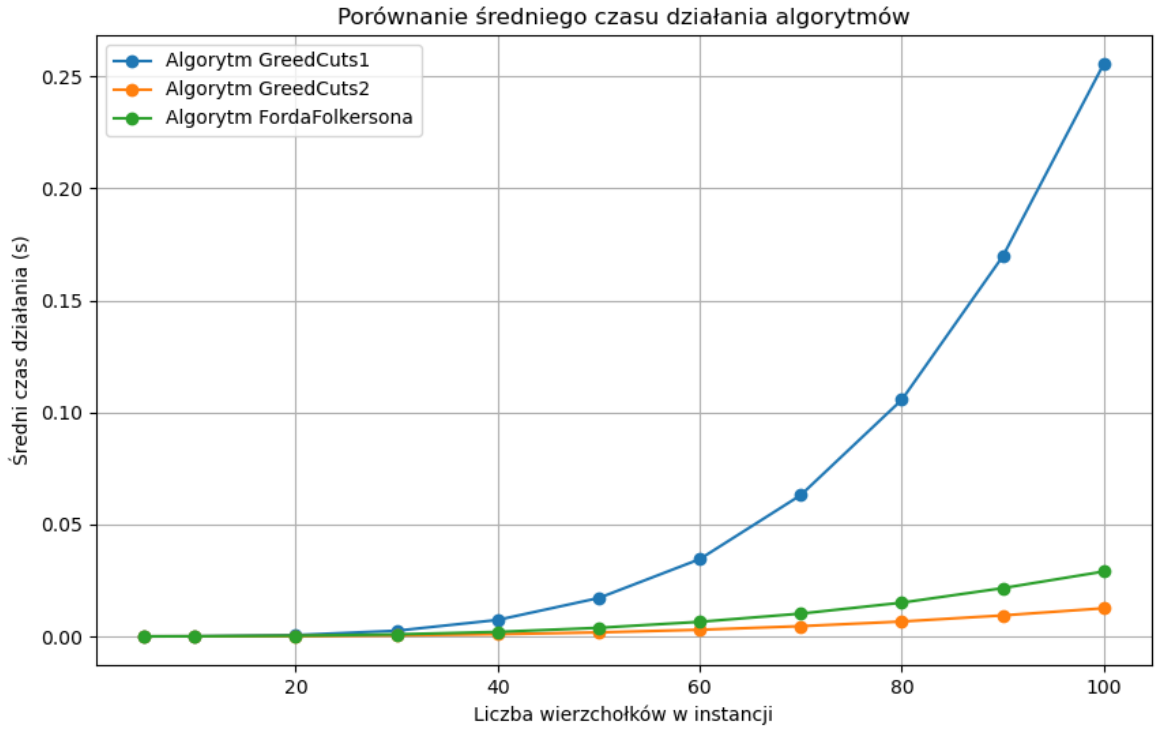
W ramach przeprowadzonych badań dokonano porównania trzech algorytmów znajdowania maksymalnego przepływu. Celem tych badań było porówanie ich złożoności czasowej.

8.1 Metodyka badań

- **Dane wejściowe:** Przeprowadzono testy na różnej wielkości grafach skierowanych z nieujemnymi wagami. Dane testowe były generowane za pomocą funkcji znajdującej się w jednym z plików źródłowych.
- **Kryteria porównawcze:** Czas działania algorytmów, zmierzony wielokrotnie.
- **Środowisko testowe:** Pomiary przeprowadzono na komputerze stacjonarnym CPU $AMD\ Ryzen^{TM}\ 5\ 3600 \times 12$.

8.2 Wyniki

Na poniższym wykresie 1 przedstawiono średni czas trwania algorytmu w zależności od wielkości instancji.



Rysunek 1: Wykres średniego czasu działania algorytmów w zależności od wielkości instancji.

Średnie czasy wykonania pokrywają się z oczekiwaniami, choć nie spodziewaliśmy się otrzymać prawie dwa razy lepszych wyników czasu działania względem algorytmu referencyjnego.

8.3 Pomyłkowość

Podczas badań mierzono ile razy algorytmy **Greed1MinCut** i **Greed2MinCut**, podadzą wynik inny niż algorytm referencyjny. Co ciekawe dla łącznej ilości przeprowadzonych obliczeń (11000 instancji), algorytm **Greed1MinCut** nie pomylił się ani razu, gdzie średnia ilość pomyłek dla algorytmu **Greed2MinCut** wymościła 5 +/- 2.

9 Wnioski

Z przeprowadzonych badań wynika, że w raz ze zmieniającą się specyfikacją problemu oraz zmianą funkcji celu (np. na taką która wlicza czas obliczeń algorytmu), istnieje szerokie spektrum możliwości poprawy i przyśpieszenia algo

- Algorytm ten, mimo swojej prostoty, może wykazywać słabą efektywność w przypadku grafów o dużych rozmiarach, ze względu na liniową zależność z maksymalnym przepływem i liczbą krawędzi. Złożoność czasowa w najgorszym przypadku wynosi $O(\text{max\_flow} \cdot E)$, co może prowadzić do dużych czasów obliczeniowych przy wysokich wartościach przepływu.
- Algorytm **Greed1MinCut** mimo wielu prób nie zwrócił ani jednej błędnej wartości dla generowanych problemów.
- Algorytm **Greed2MinCut** mimo rzadko pojawiających się błędów działa szybciej niż algorytm referencyjny.
- Dla aplikacji wymagających jak najszybszego rozwiązania problemu MaxFlow, algorytmy Greedy mogą być preferowane, szczególnie jeśli dokładność nie jest krytyczna, a priorytetem jest czas obliczeń. Dla mniejszych grafów lub gdy wymagane jest absolutne rozwiązanie optymalne, algorytm Forda-Fulkersona może nadal stanowić dobry wybór.

10 Źródła

- Materiały z wykładu i zajęć.
- Materiały dostarczone przez prowadzącego: http://mariusz.makuchowski.staff.iiar.pwr.wroc.pl
- Strona z algorytmem referencyjnm GeeksForGeeks