Politechnika Wrocławska

Wydział Informatyki i Telekomunikacji

Badania operacyjne i optymalizacja dyskretna

Wtorek, 13:15-15:00

Metoda Ścieżki Krytycznej - Najkrótsza ścieżka

Autorzy:

Aleksandra Rozmus, 252954

Paweł Gołębiowski, 243710

Prowadzący:

dr inż. Mariusz Makuchowski

29 października 2023

Politechnika Wrocławska

Spis treści

1 Wstęp

Grafy stanowią podstawowy i uniwersalny obiekt matematyczny stosowany w różnych dziedzinach nauki i techniki, od informatyki po nauki społeczne. Jednym z kluczowych zagadnień dotyczących grafów jest problem znajdowania najkrótszej ścieżki między dwoma wierzchołkami. Istnieje wiele algorytmów służących do rozwiązywania tego problemu, z których każdy ma swoje unikatowe cechy, zalety i ograniczenia [1].

W tym sprawozdaniu skupimy się na dwóch klasycznych algorytmach: algorytmie Dijkstry oraz algorytmie Bellmana-Forda. Algorytm Dijkstry, nazwany tak na cześć swojego twórcy, Edsgera Dijkstry, jest jednym z najbardziej znanych algorytmów do znajdowania najkrótszych ścieżek w grafach ważonych bez krawędzi o ujemnych wagach. Z kolei algorytm Bellmana-Forda, nazwany na cześć jego twórców, Richarda Bellmana i Lestera Forda, jest bardziej ogólnym rozwiązaniem, które potrafi radzić sobie z grafami zawierającymi krawędzie o ujemnych wagach, a także wykrywać cykle o ujemnej wadze [2].

Celem tego sprawozdania jest eksploracja i porównanie tych dwóch algorytmów pod kątem ich wydajności, zakresu zastosowań oraz specyficznych cech. Przeanalizujemy ich działanie na różnych typach grafów, badając jak zachowują się w różnych warunkach oraz jakie mają ograniczenia.

2 Dane wejściowe

Jako dane wejściowe wykorzystane przy sprawdzeniu poprawności działania stworzonych algorytmów wykorzystano dane dostępne w pliku *data25.txt* na stronie mariusz.makuchowski.staff.iiar.pwr.wroc.pl.

```
25
 0  0  0  1  1  9  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 6  5  0  7  0  2  7  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 4  6  7  0  1  5  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  4  0  4  0  1  0  6  0  6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1  3  3  0  0  0  0  0  4  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 4  7  5  0  0  3  0  0  5  2  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 2  0  9  3  4  2  3  0  9  0  8  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  6  7  2  0  0  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 9  0  5  0  0  5  4  6  0  0  0  9  0  8  2  0  0  0  0  0  0  0  0  0  0
 8  3  2  2  0  6  5  7  6  0  0  0  6  0  6  0  0  0  0  0  0  0  0  0  0
 0  3  0  0  6  7  0  0  8  2  6  0  0  0  0  6  0  0  0  0  0  0  0  0  0
 0  0  3  0  6  0  0  0  0  0  7  2  0  3  1  2  0  0  0  0  0  0  0  0  0
 0  6  1  0  0  0  5  0  6  0  4  5  6  0  2  0  8  0  2  0  0  0  0  0  0
 0  0  6  0  8  0  0  0  0  0  6  9  0  0  0  0  0  1  5  0  0  0  0  0  0
 0  0  1  1  0  4  4  0  1  5  8  1  3  0  9  0  0  0  0  7  0  0  0  0  0
 0  0  4  0  0  0  0  9  8  5  8  0  8  0  6  2  0  6  0  6  0  0  0  0  0
 5  8  0  0  3  0  0  0  0  0  8  0  1  5  2  0  9  0  0  1  7  0  4  0  0
 4  3  4  4  6  3  9  0  5  0  1  5  3  9  1  0  0  0  0  1  3  0  0  0  0
 0  3  9  0  7  0  0  6  4  2  0  7  5  2  0  0  2  6  0  0  4  2  7  9  5
 7  0  0  0  0  0  6  0  1  0  0  5  0  2  0  1  0  8  0  8  0  0  7  2  7
 0  0  0  0  2  4  0  0  6  5  7  9  0  5  0  0  0  0  0  4  8  0  0  0  7
 7  0  0  0  0  0  0  0  4  6  0  4  0  0  0  0  7  7  0  6  5  0  0  0  4
 2  0  3  0  0  0  9  3  1  0  0  7  7  0  0  6  8  8  0  0  0  7  2  0  0
 0  9  0  1  8  0  0  8  6  0  0  0  8  7  0  0  0  7  0  0  5  9  8  6  0
```

Listing 1: Dane wejściowe

gdzie:

- pierwszy wiersz - liczba wierzchołków grafu,
- kolejne wiersze - macierz połączeń między wierzchołkami.

3 Zaimplementowane algorytmy

Podczas laboratorium zaimplementowano dwa algorytmy. Dla obu algorytmów dane były wczytywane z pliku i zapisywane do pliku wyjściowego.

3.1 Algorytm Dijkstry

Algorytm Dijkstry został stworzony przez Edsgera Dijkstry w 1956 roku i od tego czasu stanowi jeden z podstawowych algorytmów stosowanych do znajdowania najkrótszej ścieżki w grafach ważonych. Kluczowym założeniem algorytmu Dijkstry jest to, że nie może on operować na grafach z krawędziami

o ujemnych wagach. Działanie algorytmu opiera się na lokalnym wyborze najkrótszej krawędzi, rozpoczynając od wybranego wierzchołka startowego. W każdym kroku, spośród wierzchołków jeszcze nieprzetworzonych, wybierany jest ten o aktualnie najkrótszym oszacowaniu odległości od źródła. Algorytm kontynuuje działanie, aż wszystkie wierzchołki zostaną przetworzone lub osiągnięte.

```
def dijkstra(n, matrix, start_vertex):
    distances = [float('infinity')] * n
    distances[start_vertex] = 0
    priority_queue = [(0, start_vertex)]
    predecessors = [-1] * n

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        if current_distance > distances[current_vertex]:
            continue

        for i in range(n):
            if matrix[current_vertex][i] and current_distance + matrix[current_vertex][i] <
            ↪  distances[i]:
                distances[i] = current_distance + matrix[current_vertex][i]
                predecessors[i] = current_vertex
                heapq.heappush(priority_queue, (distances[i], i))

    paths = {}
    for i in range(n):
        if distances[i] != float('infinity'):
            path = []
            current = i
            while current != -1:
                path.append(current)
                current = predecessors[current]
            path.reverse()
            paths[i] = path
        else:
            paths[i] = []

    return distances, paths
```

Listing 2: Algorytm Dijkstry

3.2 Algorytm Bellmana-Forda

Algorytm Bellmana-Forda, nazwany na cześć Richarda Bellmana i Lestera Forda, jest nieco starszy i oferuje bardziej ogólne podejście do problemu najkrótszej ścieżki. Jego główną zaletą jest zdolność do obsługi grafów z krawędziami o ujemnych wagach oraz wykrywanie cykli o ujemnej wadze. Algorytm

ten działa poprzez iteracyjne relaksowanie krawędzi, co oznacza aktualizację odległości do wierzchołków docelowych, jeśli znaleziono krótszą ścieżkę przez dany wierzchołek pośredniczący. Proces ten jest powtarzany dla każdej krawędzi w grafie n-1 razy, gdzie n to liczba wierzchołków. Ostatecznym krokiem jest sprawdzenie, czy istnieją krawędzie, które można jeszcze zrelaksować, co wskazywałoby na obecność cyklu o ujemnej wadze.

```python
def bellman_ford(n, matrix, start_vertex):
    distances = [float('infinity')] * n
    distances[start_vertex] = 0
    predecessors = [-1] * n
    edges = []

    for i in range(n):
        for j in range(n):
            if matrix[i][j]:
                edges.append((i, j, matrix[i][j]))

    for _ in range(n-1):
        for edge in edges:
            u, v, w = edge
            if distances[u] != float('infinity') and distances[u] + w < distances[v]:
                distances[v] = distances[u] + w
                predecessors[v] = u

    for edge in edges:
        u, v, w = edge
        if distances[u] != float('infinity') and distances[u] + w < distances[v]:
            print("Graf zawiera cykl o ujemnej wadze")
            return None, None

    paths = {}
    for i in range(n):
        if distances[i] != float('infinity'):
            path = []
            current = i
            while current != -1:
                path.append(current)
                current = predecessors[current]
            path.reverse()
            paths[i] = path
        else:
            paths[i] = []

    return distances, paths
```

Listing 3: Algorytm Bellmana-Forda

Chociaż oba algorytmy służą do rozwiązywania tego samego problemu, różnią się podejściem, zakre-

sem zastosowań oraz efektywnością w różnych scenariuszach. Algorytm Dijkstry jest zazwyczaj szybszy dla grafów bez krawędzi o ujemnych wagach, podczas gdy algorytm Bellmana-Forda jest bardziej uniwersalny, ale zwykle mniej wydajny dla dużych grafów. [3] [4] [5]

4 Dane wyjściowe

Poniżej przedstawiono uzyskane dane wyjściowe dla pliku *data25.txt* dla obu algorytmów. Dane te pokrywają się z oczekiwanymi.

```
Dijkstra: ([0, 5, 5, 1, 1, 2, 10, 7, 6, 6, 8, 8, 10, 12, 8, 12, 12, 9, 13, 10, 14, 12, 13,
↪  16, 15], {0: [0], 1: [0, 4, 1], 2: [0, 4, 5, 2], 3: [0, 3], 4: [0, 4], 5:
[0, 4, 5], 6: [0, 4, 5, 9, 6], 7: [0, 3, 7], 8: [0, 4, 5, 8], 9: [0, 4, 5, 9], 10: [0, 4, 5,
↪  8, 10], 11: [0, 3, 7, 11], 12: [0, 4, 5, 9, 14, 17, 12], 13: [0, 4, 5, 9, 14, 17, 19,
↪  13], 14: [0, 4, 5, 9, 14], 15: [0, 4, 5, 9, 14, 17, 12, 15], 16: [0, 4, 5, 9, 14, 17,
↪  19, 16], 17: [0, 4, 5, 9, 14, 17], 18: [0, 4, 5, 9, 14, 18], 19:
[0, 4, 5, 9, 14, 17, 19], 20: [0, 4, 5, 9, 14, 17, 19, 20], 21: [0, 4, 5, 9, 14, 17, 19,
↪  21], 22: [0, 4, 5, 9, 14, 17, 22], 23: [0, 4, 5, 9, 14, 17, 19, 20, 23], 24: [0, 4, 5,
↪  9, 14, 17, 19, 24]})
Bellman-Ford: ([0, 5, 5, 1, 1, 2, 10, 7, 6, 6, 8, 8, 10, 12, 8, 12, 12, 9, 13, 10, 14, 12,
↪  13, 16, 15], {0: [0], 1: [0, 4, 1], 2: [0, 4, 5, 2], 3: [0, 3], 4: [0, 4], 5: [0, 4, 5],
↪  6: [0, 3, 7, 6], 7: [0, 3, 7], 8: [0, 4, 5, 8], 9: [0, 4, 5, 9], 10: [0, 4, 5, 8, 10],
↪  11: [0, 3, 7, 11], 12: [0, 4, 5, 9, 14, 17, 12], 13: [0, 4, 5,
9, 14, 17, 19, 13], 14: [0, 4, 5, 9, 14], 15: [0, 4, 5, 9, 14, 17, 12, 15], 16: [0, 4, 5, 9,
↪  14, 17, 19, 16], 17: [0, 4, 5, 9, 14, 17], 18: [0, 4, 5, 9, 14, 18], 19: [0, 4, 5, 9,
↪  14, 17, 19], 20: [0, 4, 5, 9, 14, 17, 19, 20], 21: [0, 4, 5, 9, 14, 17, 19, 21], 22: [0,
↪  4, 5, 9, 14, 17, 22], 23: [0, 4, 5, 9, 14, 17, 19, 20, 23], 24: [0, 4, 5, 9, 14, 17, 19,
↪  24]})
```

Listing 4: Dane wyjściowe

Czas obliczeń dla poszczególnych algorytmów:

- algorytm Dijkstry: 0.0003842000000000012s,
- algorytm Bellmana-Forda: 0.001581100000000002s.

5 Badania

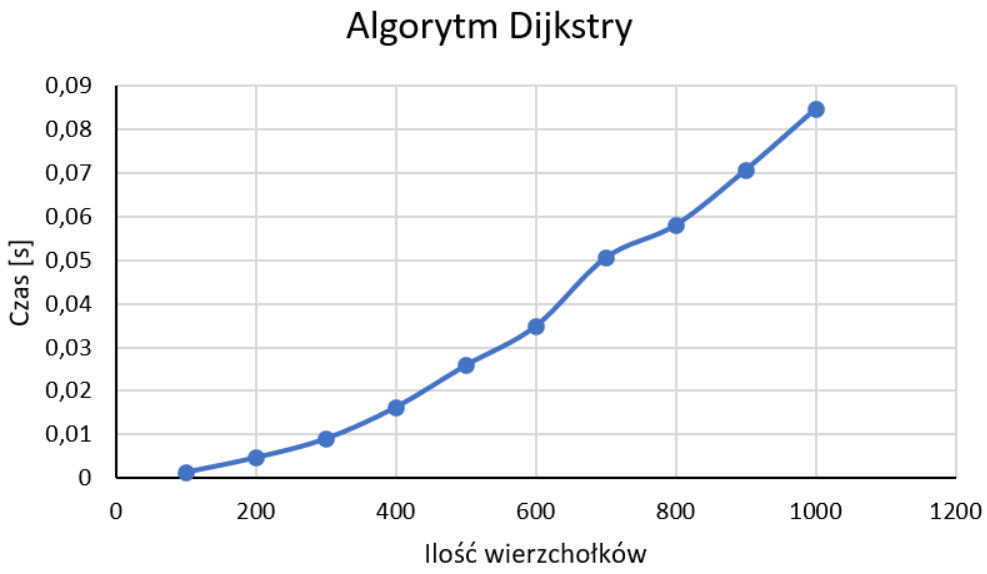
Nasze badania koncentrowały się głównie na pomiarze czasu wykonania algorytmów Dijkstry i Bellmana-Forda. W tym celu:

- Wybraliśmy różnorodne grafy o różnych rozmiarach i gęstościach. To pozwoliło nam na analizę zachowania algorytmów w szerokim zakresie warunków, od grafów rzadkich po grafy gęste.

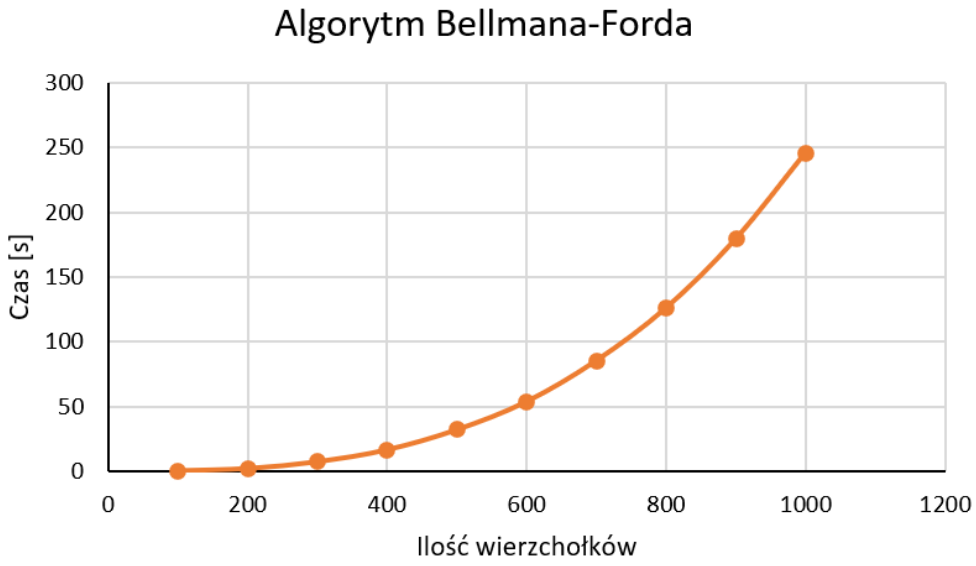
- Algorytmy zostały zaimplementowane z uwzględnieniem najlepszych praktyk programistycznych, aby zapewnić ich optymalne działanie.
- Dla każdego wybranego grafu oba algorytmy były uruchamiane wielokrotnie, a wynikowy czas wykonania był rejestrowany. Wszystkie testy były przeprowadzane w kontrolowanym środowisku, aby minimalizować wpływ czynników zewnętrznych na wyniki.
- Po zebraniu danych przeprowadziliśmy analizę, aby zidentyfikować tendencje, wzorce i wyjątki w zachowaniu algorytmów w zależności od charakterystyki grafu.

Tabela 1: Czasy obliczeń dla wybranej ilości wierzchołków

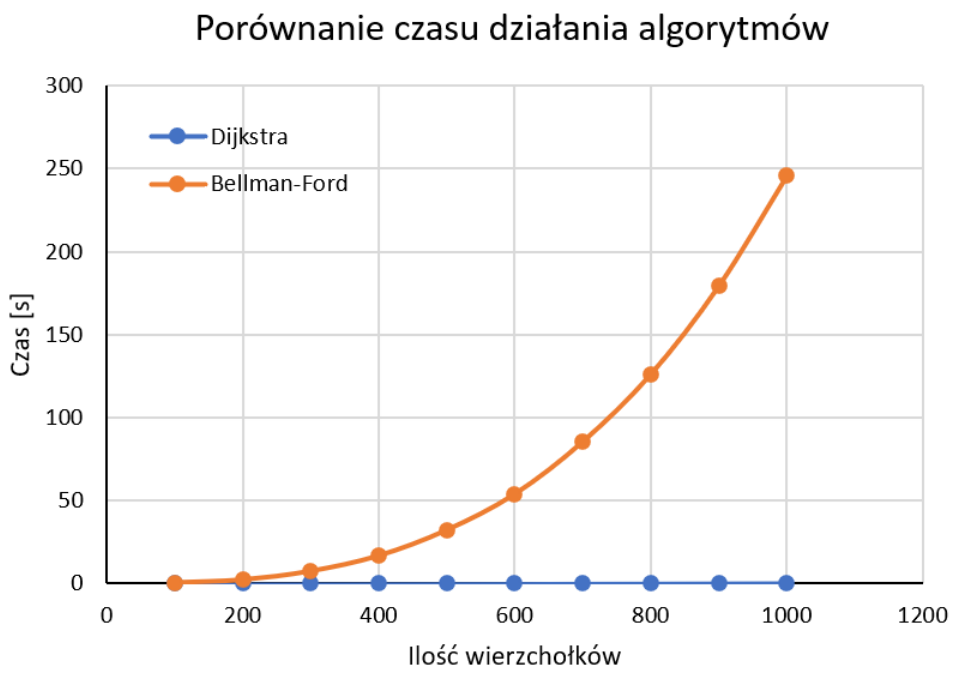
	Dijkstra	Bellman-Ford
Ilość krawędzi	Czas [s]	
100	0,001243	0,254338
200	0,004601	1,964812
300	0,008926	7,169732
400	0,016136	16,45437
500	0,02574	31,95524
600	0,034733	53,67267
700	0,050433	85,31411
800	0,05795	126,0185
900	0,070608	179,3666
1000	0,084636	245,8798



Rysunek 5.1: Czas wykonywania algorytmu Dijkstry



Rysunek 5.2: Czas wykonywania algorytmu Bellmana-Forda



Rysunek 5.3: Porównanie czasu wykonywania algorytmów

6 Wnioski

- Obie metody, zarówno algorytm Dijkstry, jak i Bellmana-Forda, są efektywnymi narzędziami do rozwiązywania problemu najkrótszej ścieżki w grafach. Jednakże algorytm Dijkstry jest bardziej odpowiedni dla grafów, które nie mają krawędzi o ujemnych wagach, podczas gdy algorytm Bellmana-Forda może być stosowany do grafów z krawędziami o dowolnej wadze.
- W testach wydajnościowych algorytm Dijkstry zazwyczaj okazywał się być szybszym rozwiązaniem dla większości grafów, zwłaszcza tych bez krawędzi o ujemnych wagach. Algorytm Bellmana-Forda, chociaż bardziej uniwersalny, może być znacznie mniej efektywny w przypadku dużych grafów, ze względu na jego złożoność czasową.
- Jednym z kluczowych atutów algorytmu Bellmana-Forda jest jego zdolność do wykrywania cykli o ujemnej wadze w grafie. Jest to cecha nieosiągalna dla algorytmu Dijkstry.
- Mimo, że algorytm Dijkstry jest zazwyczaj szybszy, jego implementacja, zwłaszcza z użyciem struktur danych takich jak kopce, może być bardziej skomplikowana niż bardziej intuicyjna i prosta implementacja algorytmu Bellmana-Forda.
- Wybór odpowiedniego algorytmu zależy od specyficznych wymagań problemu. Jeśli pewne jest,

że graf nie zawiera krawędzi o ujemnych wagach, algorytm Dijkstry jest zazwyczaj preferowany ze względu na jego wydajność. Jeśli jednak graf może zawierać krawędzie o ujemnych wagach lub istnieje potrzeba wykrycia cykli o ujemnej wadze, algorytm Bellmana-Forda staje się niezbędny.

- Złożoność czasowa algorytmu Bellmana-Forda dla grafu liczącego n wierzchołków i e krawędzi w przypadku pesymistycznym jest równa O(en). Biorąc pod uwagę, że w przypadku braku krawędzi wielokrotnych liczba krawędzi jest zawsze mniejsza od n^2, można powiedzieć, że złożoność czasowa algorytmu to O(n^3).
- Złożoność czasowa algorytmu Dijkstry dla przypadku kiedy dane są przechowywane w postaci kopca wynosi O(logn).

7 Bibliografia

[1] Leiserson C. E. Rivest R. L. Stein C. Cormen, T. H. *Introduction to algorithms (3rd ed.)*. MIT press, 2009.

[2] Zbigniew Czech, Sebastian Deorowicz, and Piotr Fabian. *Algorytmy i struktury danych: wybrane zagadnienia*. Wydawnictwo Politechniki Śląskiej, 2010.

[3] E. W. Dijkstra. *A note on two problems in connexion with graphs*. Numerische mathematik, 1(1), 269-271, 1959.

[4] R. Bellman. *On a routing problem*. Quarterly of applied mathematics, 16(1), 87-90, 1958.

[5] L. R. Ford. *Network flow theory*. Rand Corporation, 1956.