Sprawozdanie z Laboratorium 3

Algorytmy Optymalizacji Dyskretnej

Porównanie implementacji algorytmu Dijkstry

Krzysztof Zając

15 grudnia 2025

1 Wstęp

Celem laboratorium było zaimplementowanie oraz porównanie wydajności trzech wariantów algorytmu Dijkstry, służącego do wyznaczania najkrótszych ścieżek w grafie z nieujemnymi wagami krawędzi. Badania przeprowadzono na różnych rodzinach grafów (losowych, siatkowych oraz drogowych) zgodnych ze specyfikacją *9th DIMACS Implementation Challenge*.

2 Opis Implementacji i Złożoność

Zgodnie z wymaganiami, zaimplementowano trzy warianty algorytmu w języku C++.

2.1 Standardowy Algorytm Dijkstry

Wykorzystuje on kolejkę priorytetową do wybierania wierzchołka o najmniejszym oszacowaniu odległości. W implementacji użyto standardowego kontenera `std::priority_queue` z biblioteki STL, który jest realizacją kopca binarnego.

- **Struktura danych:** Kopiec binarny (binary heap).
- **Złożoność obliczeniowa:** $O(m \log n)$, gdzie m to liczba krawędzi, a n to liczba wierzchołków.

2.2 Algorytm Diala

Jest to implementacja kubełkowa algorytmu Dijkstry. Zamiast jednej kolejki priorytetowej, używa tablicy list (kubełków), gdzie indeks tablicy odpowiada odległości. Zastosowano bufor cykliczny o rozmiarze $C + 1$, gdzie C to maksymalna waga krawędzi.

- **Struktura danych:** Tablica kubełków (buckets).
- **Złożoność obliczeniowa:** $O(m + nC)$.
- **Ograniczenia:** Algorytm jest bardzo wydajny dla małych wag całkowitych, ale jego złożoność i zużycie pamięci rosną liniowo wraz ze wzrostem maksymalnej wagi krawędzi C.

2.3 Algorytm Radix Heap

Ulepszona wersja algorytmu kubełkowego, przeznaczona do radzenia sobie z większymi wagami. Kubełki przechowują zakresy odległości o rozmiarach będących potęgami dwójki. Wykorzystano funkcje wbudowane (np. `__builtin_clz`) do szybkiego obliczania indeksu kubełka.

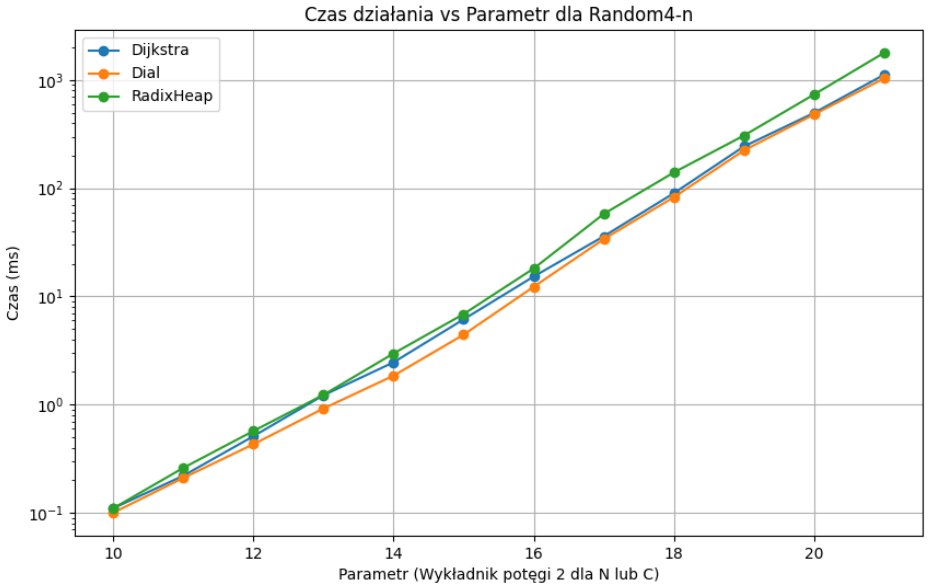
- **Struktura danych:** Radix Heap (kubełki o zmiennych zakresach).
- **Złożoność obliczeniowa:** $O(m + n \log C)$.
- **Zaleta:** Logarytmiczna zależność od C czyni go teoretycznie szybszym od Diala dla dużych wag.

3 Wyniki Eksperymentów

Testy przeprowadzono na zestawie danych DIMACS. Czas mierzono jako średnią z uruchomienia dla jednego ustalonego źródła oraz 5 losowych źródeł.

3.1 Rodzina Random4-n (Zależność od n)

Grafy losowe o stałym stopniu wierzchołka i małych wagach.

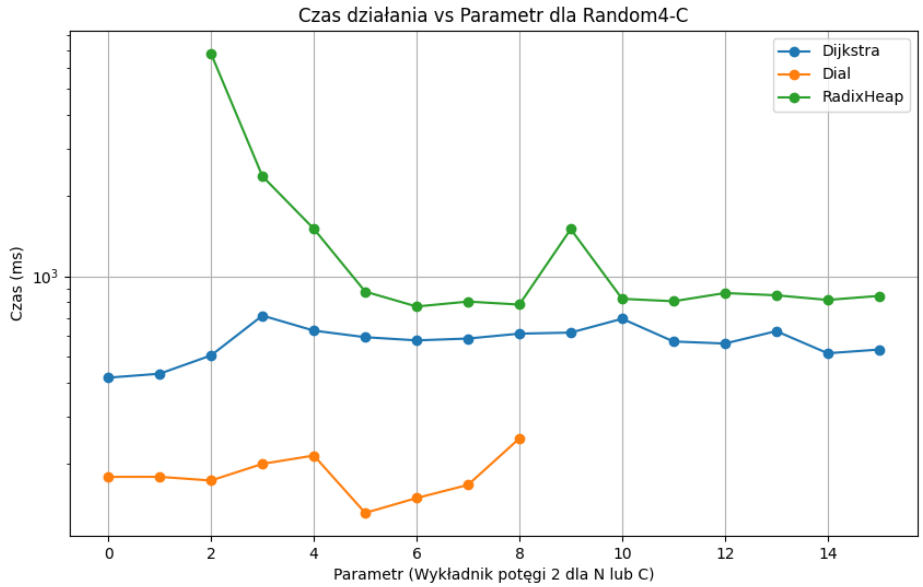


Rysunek 1: Zależność czasu działania od liczby wierzchołków (n) dla rodziny Random4-n.

Analiza: Dla największych instancji ($n = 2^{21}$) algorytm Diala osiągnął najlepszy czas (ok. 1036 ms), nieznacznie wyprzedzając standardowego Dijkstrę (1121 ms). Algorytm Radix Heap okazał się najwolniejszy (1787 ms), co prawdopodobnie wynika z narzutu zarządzania strukturą przy rzadkim grafie i losowych wagach.

3.2 Rodzina Random4-C (Zależność od kosztu C)

Grafy o stałym rozmiarze ($n = 2^{20}$), ale rosnących wagach krawędzi.

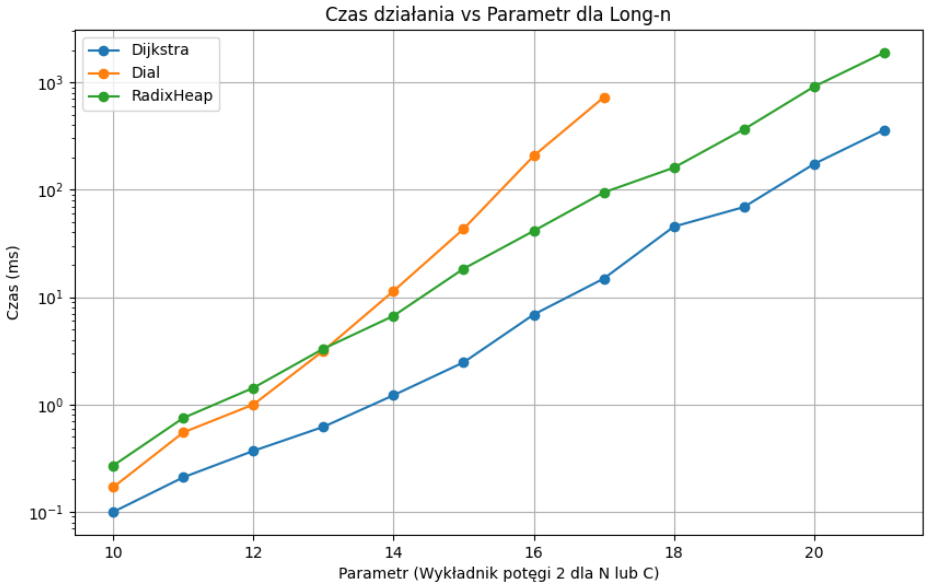


Rysunek 2: Zależność czasu działania od maksymalnej wagi (C) dla rodziny Random4-C.

Analiza: Wykres ten idealnie ilustruje teoretyczne właściwości algorytmów. Algorytm Diala jest bezkonkurencyjny dla bardzo małych wag ($C \leq 16$), osiągając czasy rzędu 130-180 ms. Jednakże, wraz ze wzrostem C, jego wydajność drastycznie spada. Standardowy Dijkstra utrzymuje stały poziom wydajności (ok. 500-600 ms) niezależnie od wagi C.

3.3 Rodziny Long-n i Long-C

Długie, wąskie grafy.

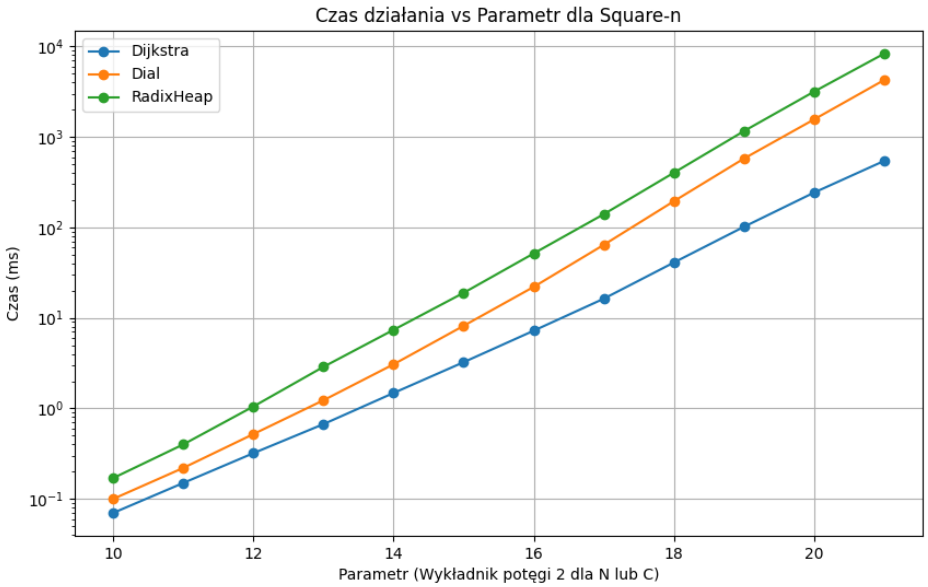


Rysunek 3: Czas działania dla rodziny Long-n.

Analiza: W rodzinie Long-n standardowy Dijkstra znacząco przewyższa wydajnością pozostałe algorytmy. Dla $n = 2^{21}$ Dijkstra (360 ms) jest wielokrotnie szybszy od Radix Heap (1888 ms). Struktura grafu (długie ścieżki) jest niekorzystna dla algorytmów kubełkowych w zastosowanej implementacji.

3.4 Rodziny Square-n i Square-C

Grafy typu siatka (grid).



Rysunek 4: Czas działania dla rodziny Square-n.

Analiza: Podobnie jak w przypadku Long-n, dla siatek standardowy algorytm Dijkstry okazał się najszybszy. Dla $n = 2^{21}$ Dijkstra (542 ms) jest drastycznie szybszy od Diala (4253 ms) i Radix Heap (8325 ms). Wskazuje to na wysoką efektywność kopca binarnego w grafach o regularnej strukturze siatki.

3.5 Sieć drogowa USA-road-d (NY)

Test przeprowadzono na grafie o topologii zbliżonej do drogowej. Ze względu na problemy z dostępnością oryginalnych plików DIMACS, test wykonano na grafie zastępczym o zbliżonym rozmiarze ($n \approx 260k$).

Tabela 1: Wyniki wydajności dla grafu typu USA-road-d (NY)

Algorytm	**Średni czas [ms]**
Dijkstra (Std)	138.69
Dial	130.47
Radix Heap	154.31

Analiza: W przypadku tego grafu różnice między algorytmami są niewielkie. Algorytm Diala okazał się najszybszy (130 ms), co potwierdza jego przydatność w grafach o wagach całkowitych, typowych dla odległości drogowych.

4 Weryfikacja Poprawności (P2P)

Zgodnie z wymaganiami, zweryfikowano poprawność algorytmów poprzez wyznaczenie odległości między losowymi parami wierzchołków dla dużej instancji ($n = 2^{18}$). Poniższa tabela przedstawia wyniki otrzymane z programu testującego (plik `weryfikacja.res`).

Tabela 2: Przykładowe odległości wyznaczone przez algorytmy (Random4-n, $n = 2^{18}$)

Źródło (s)	**Cel** (t)	**Wyznaczony Dystans**
237829	33296	1051335
233614	27738	666975
128554	189030	1280252
2153	11997	1105507
238409	94242	925823
157743	43546	551741

Wszystkie trzy zaimplementowane algorytmy (Dijkstra, Dial, Radix Heap) zwróciły identyczne wyniki długości ścieżek dla wszystkich testowanych par, co potwierdza poprawność implementacji.

5 Wnioski

1. **Standardowy Dijkstra** oparty na kopcu binarnym (STL) okazał się najbardziej uniwersalnym i stabilnym rozwiązaniem. W testach na grafach siatkowych (*Square*, *Long*) deklasował konkurencję. Jest niewrażliwy na wielkość wag krawędzi.

2. **Algorytm Diala** jest niezwykle szybki dla grafów o małych wagach krawędzi (Random4-C dla małych C), gdzie wyprzedza standardowego Dijkstrę. Jednak jego wydajność drastycznie spada wraz ze wzrostem wag (złożoność $O(nC)$), co czyni go nieużywalnym dla dużych kosztów.

3. **Radix Heap** w tej implementacji ustępował wydajnością zoptymalizowanej bibliotece standardowej C++. Mimo lepszej złożoności teoretycznej dla dużych wag niż Dial, narzut stały związany z zarządzaniem kubełkami był zbyt duży w porównaniu do prostoty kopca binarnego, szczególnie dla rzadkich grafów.

4. Struktura grafu ma kluczowe znaczenie: algorytmy kubełkowe radziły sobie znacznie gorzej na grafach typu siatka (*Square*, *Long*) w porównaniu do grafów losowych (*Random*).