Sprawozdanie obliczenia naukowe
Lista 4

Jakub Kowal

Algorytm Dijkstry

Opis algorytmu

Klasyczny algorytm Dijkstry dla grafu skierowanego z nieujemnymi wagami. Tablica d zawiera dystanse od s. MinHeap H przechowuje wierzchołki z kluczami $= d[v]$. Kod wykonuje: insert(s,0) w pętli extractMin -> rozluźnianie sąsiadów i wstawianie dla niespotkanych wcześniej węzłów, lub decreaseKey dla odkrytych krótszych ścieżek. Dodatkowo algorytm zwraca wcześniej, gdy wyciągnięty wierzchołek to dest (wcześniejsze zakończenie w przypadku, gdy podany jest szuakany węzeł).

Złożoność

n — liczba węzłów
m — liczba krawędzi

1. O(n) Inicjalizacje:
    - O(n) inicjalizacja wektora d
    - O(n) inicjalizacja wektora heap i pos w MinHeap
    - O(log n) wstawienie s do MinHeap (Koszt klasycznego heapify)
2. O(n) Główna pętla:
    - O(log n) extractMin (kolejny heapify, tylko że w dół, żeby dostać najmniejszy element)
    - O(m) pętla po krawędziach aktualnego węzła:
        - O(log n) insert lub decreaseKey (oba korzystają z heapify)

Złożoność końcowa:
Worst case: $O(nm \log n)$
ale jako że algorytm przechodzi przez każdy węzeł i każdą krawędź tylko raz, to otrzymujemy: $O((n+m) \log n)$

Algorytm Diala

Opis algorytmu

Implementacja algorytmu diala z $C + 1$ kubełkami. Algorytm dodaje węzeł do kubełka na podstawie kosztu dotarcia do tego węzła. Po kubełkach chodzi się cyklicznie, dlatego że jest ich C + 1 zamiast nC + 1. Główna idea algorytmu opiera się na szukaniu następnego niepustego kubełka, wyciaganiu z niego jego węzłów i dla każdego z tych węzłów sprawdzenie jego krawędzi i defacto decreaseKey na jego sąsiadach. Powtarzamy to aż nie przejrzymy wszystkich węzłów.

Złożoność obliczeniowa

n — liczba węzłów
m — liczba krawędzi
C — maksymalny koszt krawędzi

1. O(n) Inicjalizacje:
   - O(n) inicjalizacja wektora d
   - O(C+1) inicjalizacja wektora kubełki
   - O(C+1) wstawienie pierwszego źródła do unordered_set (operacja kosztu O(size))
2. O(Cn) Główna pętla:
   - O(n) Pętla chodząca po aktualnym kubełku:
     - O(m) Pętla chodząca no krawędziach aktualnego węzła:
       * O(C+1) wstawianie lub wstawianie i usuwanie węzła do którego prowadzi krawędź

Złożoność obliczeniowa:
Worst case: O($Cn * n * m * (C + 1)$)
ale tak naprawdę: O($m * (C + 1) + Cn$), bo każda krawędź będzie wstawiana lub wstawiania i usuwana tylko raz oraz w najgorszym przypadku będziemy szukać niepustych kubełków $n * C$ razy.
W praktyce jednak erase i insert są avg O(1) więc otrzymujemy: O($m + nC$)

Algorytm RadixHeap

Opis algorytmu

Algorytm radixHeap polega na przypasowaniu węzłów do odpowiednich kubełków na podstawie dystansu węzła i zakresów kubełków. W momencie gdy węzły znajdują się w pierwszym kubełku ich dystans jest zapisywany do wektora d oraz ich sąsiedzi są dodawani do odpowiednich im kubełków. Jeśli pierwszym

pustym kubełkiem nie jest ogólnie pierwszy kubełek, to reskalujemy poprzednie kubełki na zakres tego kubełka i odpowiednio przydzielamy węzły, które były w tym kubełku.

Złożoność obliczeniowa

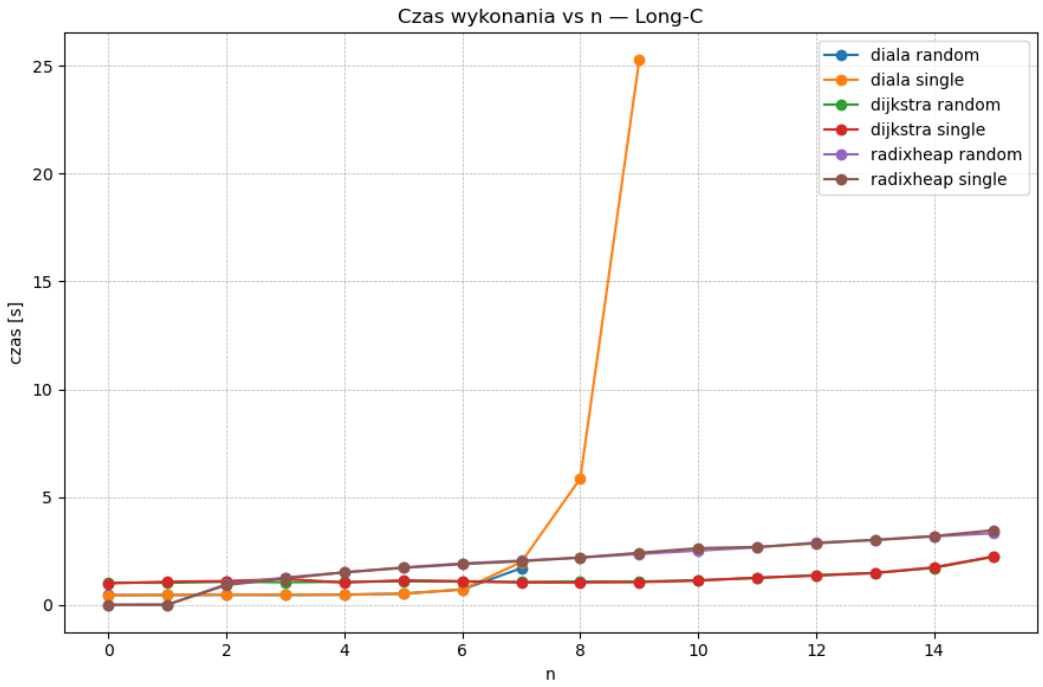
n — liczba węzłów
m — liczba krawędzi
C — maksymalny koszt krawędzi
K+1 — liczba kubełków, $K = \lceil \log_2(nC) \rceil$

1. O(n) Inicjalizacje:
    - O(K+1) inicjalizacja wektora z kubełkami
    - O(n) inicjalizacja wektora z dystanstami
    - O(K+1) znajdowanie pierwszego niepustego kubełka
2. O(n) Główna pętla
    - O(n) przejście po każdym węźle w kubełku 0
        - O(m) przejście po każdej krawędzi tego węzła
            - O(K+1) wstawienie węzła do odpowiedniego kubełka
    - albo
    - O(n) znalezienie najmniejszego węzła w kubełku
    - O(K) zmiana zakresów kubełków do naszego kubełka
    - O(nK) przeniesienie węzłów do odpowiednich kubełków
    - O(K+1) znalezienie następnego niepustego kubełka

Złożoność obliczeniowa:
Worst case: O$(n * n * m * (K + 1))$
Faktyczna złożoność: O$(nK + m)$, bo każdego węzła można szukać K razy i sprawdzamy każdą krawedź

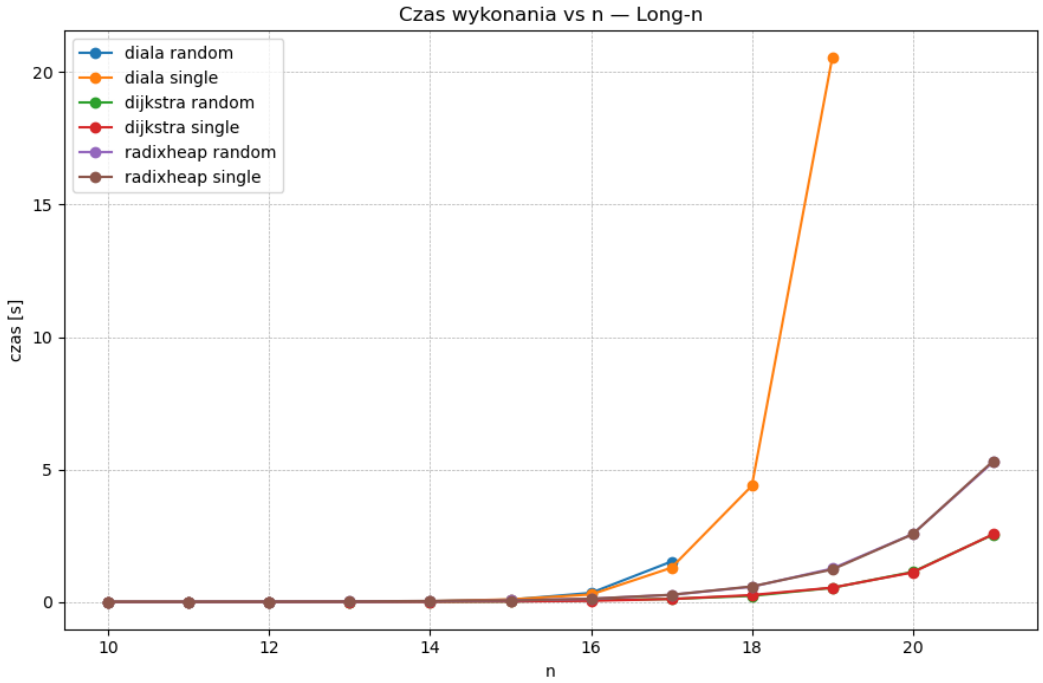
Wyniki

Long-C



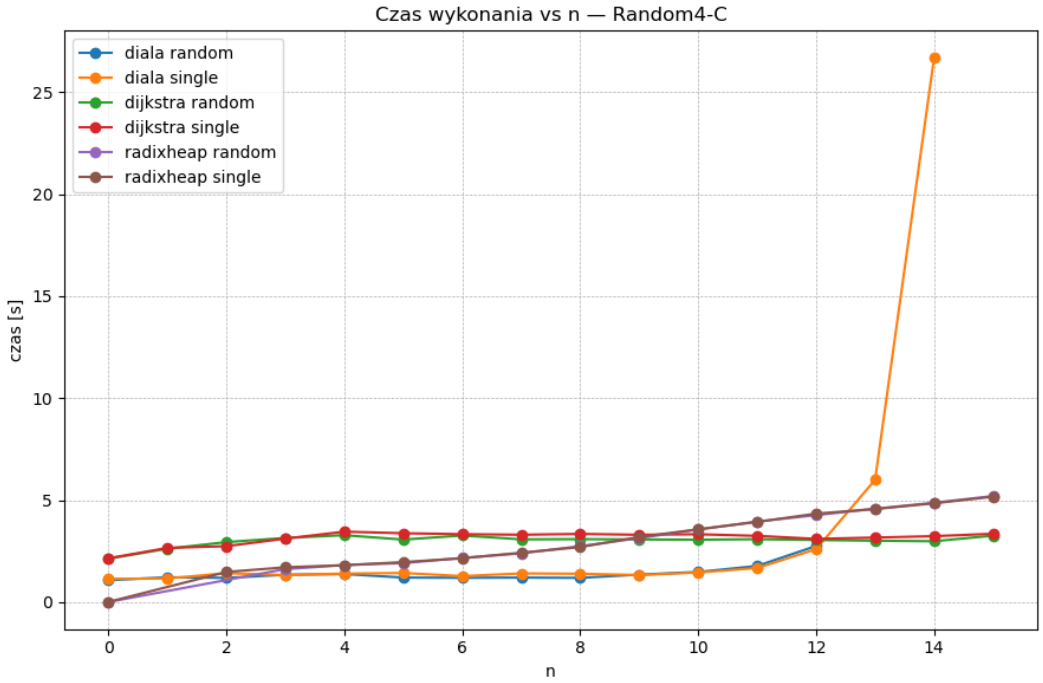
para punktów	Dijkstra	Diala	RadixHeap
1132:1048550	9977506694892	—	5927598271354
883387:436016	15007112688755	—	8797270024275
614326:981930	10261814404395	—	6085585268672
759110:303208	3341417388807	—	1927921886058
556279:740320	438969887073	—	255443251955

Long-n



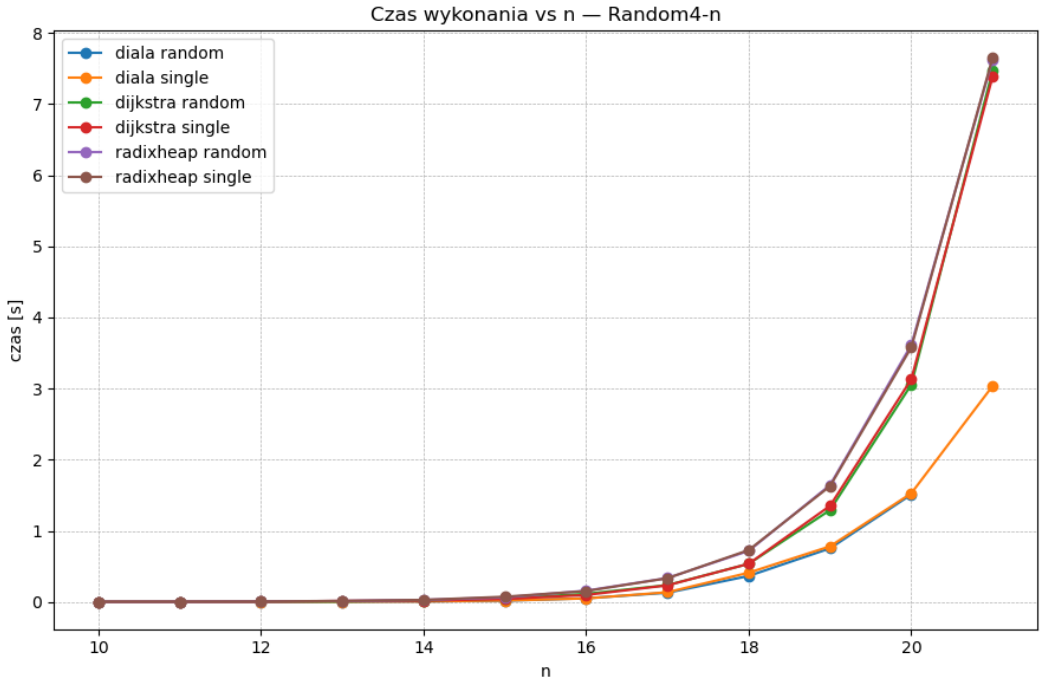
para punktów	Dijkstra	Diala	RadixHeap
2264:2097100	40153593769	—	40123865041
2058338:1598840	3649047545	—	3648557277
1196604:1668171	24699258046	—	24679412525
1338161:1995750	10449131856	—	10445292862
1044442:1462211	9477083147	—	9463161620

Random4-C



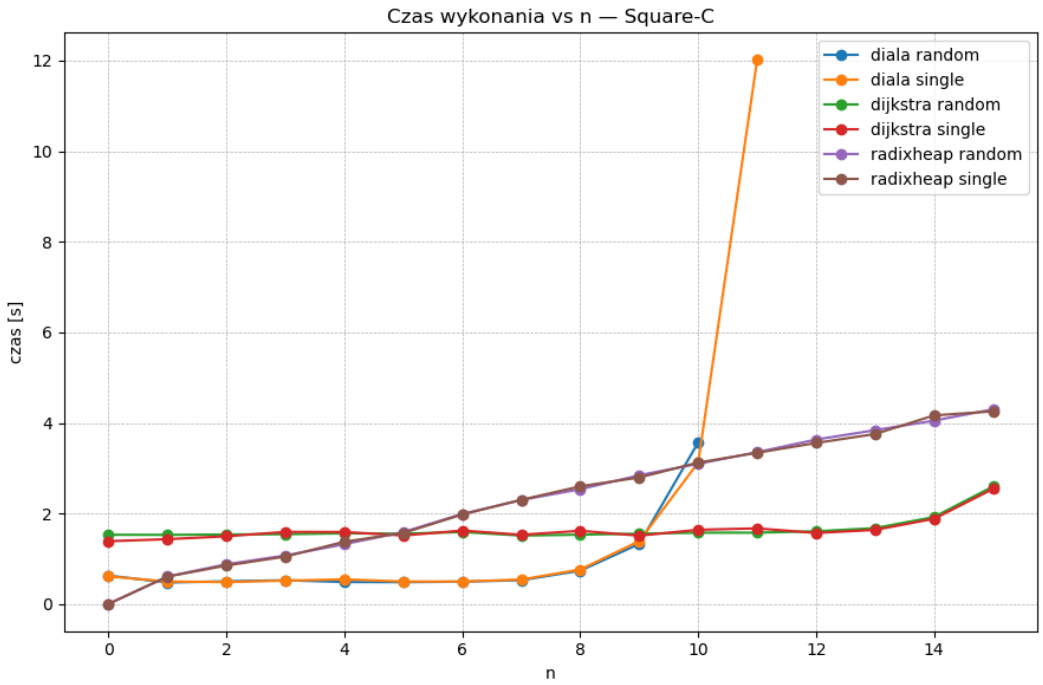
para punktów	Dijkstra	Diala	RadixHeap
1132:1048550	9309840586	—	4636957039
720434:860007	5926831647	—	3682228985
52096:634200	5468955645	—	3601865745
450409:482164	8885919162	—	4065478853
728824:140169	5806743752	—	4118393018

Random4-n



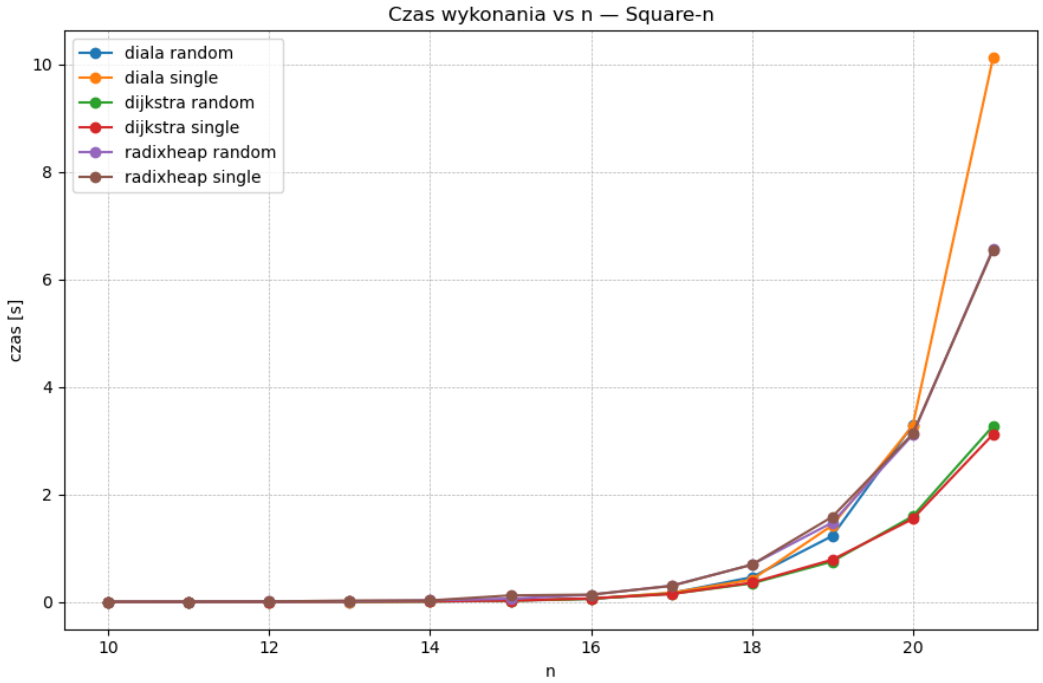
para punktów	Dijkstra	Diala	RadixHeap
2264:2097100	10809934	10809934	10809934
950468:875668	4595248	7328346	4595248
18269:2035405	9839957	6638851	9839957
923484:232792	8076717	6269065	8076717
1289340:962118	7315314	7344071	7315314

Square-C



para punktów	Dijkstra	Diala	RadixHeap
1132:1048550	736302363522	—	193475230816
354877:289865	1073533415715	—	286982391364
31826:40417	488686089524	—	121275042871
642148:513734	1190578167273	—	204373005951
559536:823607	594581624910	—	103742229848

Square-n



para punktów	Dijkstra	Diala	RadixHeap
2264:2096652	805277316	805277316	805277316
676522:600324	133155573	376660365	133155573
1417923:144930	366456815	79722686	366456815
1413391:1822333	576540241	532190650	576540241
1420366:678997	291149179	631449945	291149179

Usa-road-d

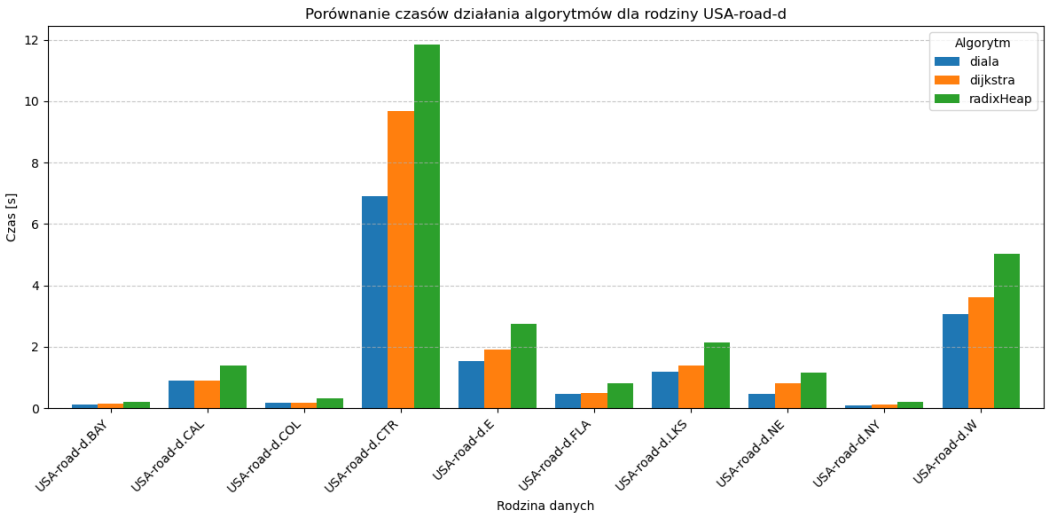


Tabela dla USA-road-d.CTR

para punktów	Dijkstra	Diala	RadixHeap
358:14067697	17810392	17810392	17810392
13644729:1993552	12970380	12970380	12970380
12380073:1701271	9237645	9237645	9237645
11149768:1323170	8760111	8760111	8760111
263249:9126376	5348057	5348057	5348057

Tabela dla USA-road-d.W.p2p

para punktów	Dijkstra	Diala	RadixHeap
6023:6261690	14207173	14207173	14207173
5670998:5145040	2572734	2572734	2572734
734605:3610144	15365457	15365457	15365457
3928410:3063857	18297859	18297859	18297859
183469:1134694	1535678	1535678	1535678

Wnioski

Jak widać na powyższych wykresach każdy z algorytmów ma swoje wady i zalety. Algorytm Diala dla niskich wag krawędzi jest bardzo szybki, ale wraz ze wzrostem wad diametralnie skacze jego czas wykonywania. Algorytm Radix Heap potrafi być szybszy od reszty dal niskich wartości wag, ale wraz z ich wzrostem jego czas wykonywania również rośnie. Warto dodać, że pomimo tego ten

wzrost nie jest porównywalny do Diala. Wzrost Radix Heap jest akceptowalny i dalej rozsądny w kontekście naszych zadań. Najbardziej stabilnym algorytmem jest Dijkstra. Pomimo, że dla niskich kosztów krawędzi nie jest najszybsza z rozpatrywanych algorytmów, to jednak najlepiej sobie radzi z rosnącym c.