29.04.2020r.

Jacek Multan

indeks: 248964

Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Projekt 2 – Grafy

Dr Łukasz Jeleń

czwartek 9:15 – 11:00

Wstęp

Celem projektu było zbadanie złożoności obliczeniowej algorytmu Dijkstry dla reprezentacji grafu za pomocą listy sąsiedztwa i macierzy sąsiedztwa. Schemat ten służy do wyszukiwania najkrótszej ścieżki w grafie ważonym. Jest to przykład algorytmu zachłannego, czyli takiego, gdzie w każdym kroku dokonuje się wyboru najlepszego w danej chwili rozwiązania.

Algorytm

Każdemu wierzchołkowi przypisujemy indeks w tablicy, następnie wypełniamy ją maksymalnymi dostępnymi liczbami, poza pierwszym elementem zawierającym 0. Podobnie można uczynić, by przechowywać poprzedników danego wierzchołka. Złożoność obliczeniowa tych czynności wynosi O(V). Później dodajemy do kolejki priorytetowej wierzchołki. Dopóki nie będzie ona pusta, wykonujemy pętlę zdejmującą minimum oraz, w przypadku mojej implementacji, sprawdzającą czy od momentu dodania, droga do wierzchołka była aktualizowana. Jeśli tak nie było, sprawdzamy czy opłaca się zmienić drogę, na prowadzącą jakąkolwiek z przyległych krawędzi i dodajemy wierzchołek, do którego ona wiedzie, do kolejki. Operacja zwrócenia minimum lub dodania nowego wierzchołka dokonuje się w czasie O(log V), gdy jest ona zaimplementowana za pomocą kopca, natomiast pętla wykona się tyle razy ile jest krawędzi w grafie. Cała złożoność obliczeniowa wynosi więc O(max(V, E log V)) czyli O(E log V).

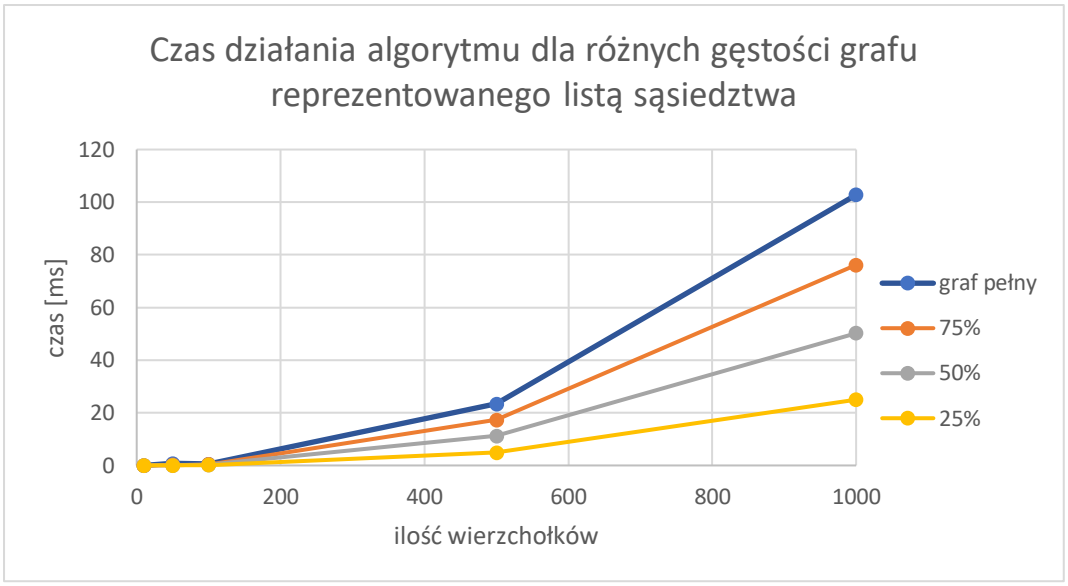
Przebieg eksperymentu

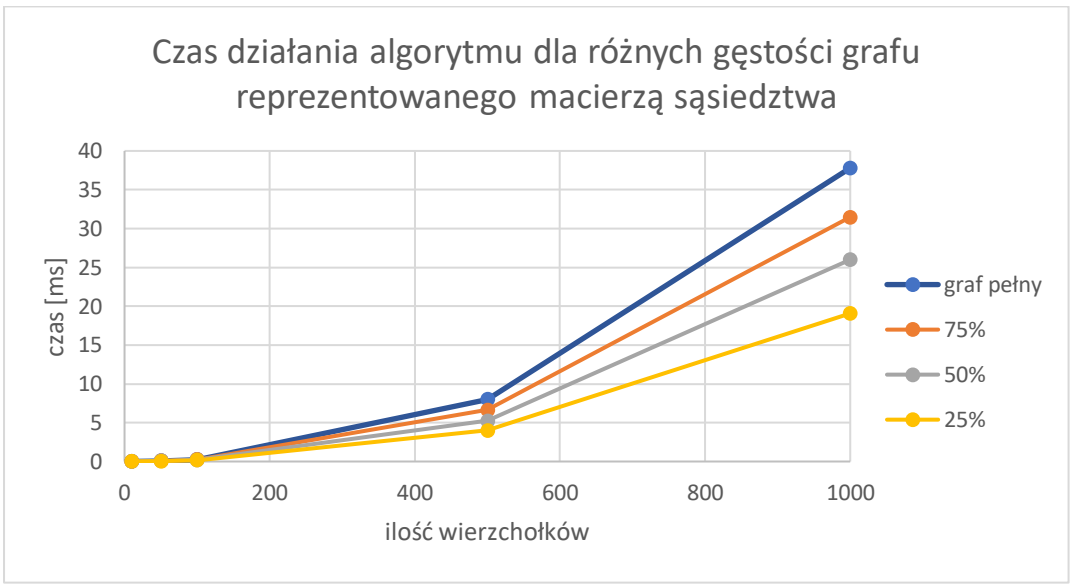
Podczas doświadczenia program losował po 100 grafów o gęstości 25%, 50%, 75% oraz 100% dla 10, 50, 100, 500 oraz 1000 wierzchołków. Następnie zapisywał je do pliku, wczytywał je do macierzy oraz listy sąsiedztwa i dla każdej reprezentacji liczył czas działania algorytmu Dijkstry. Program udostępniał także możliwość generowania pliku wyjściowego zawierającego drogę do danego wierzchołka oraz wszystkich poprzedników w kolejności, w celu sprawdzenia poprawności działania.

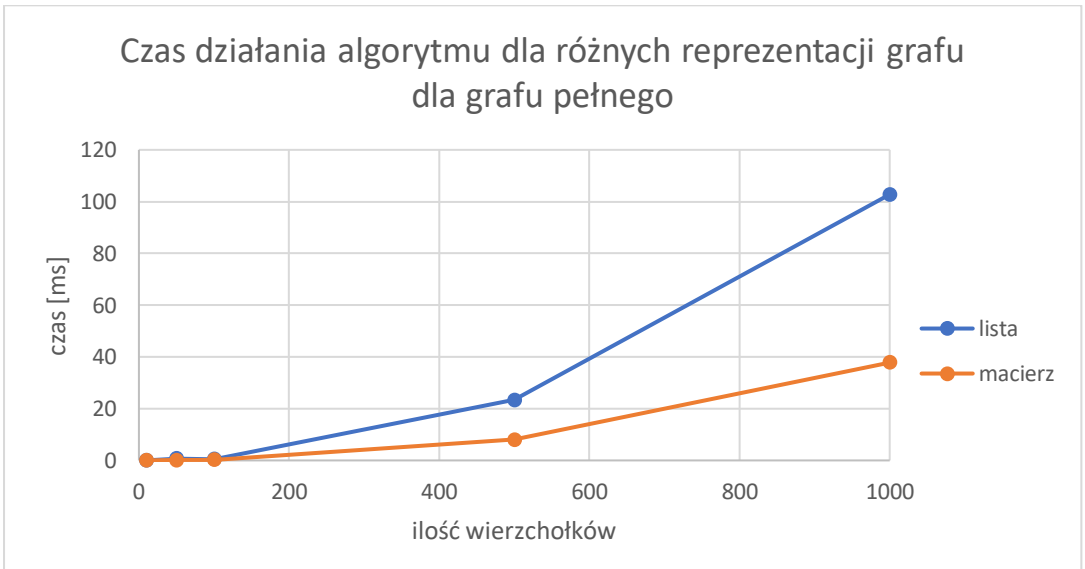
Wyniki

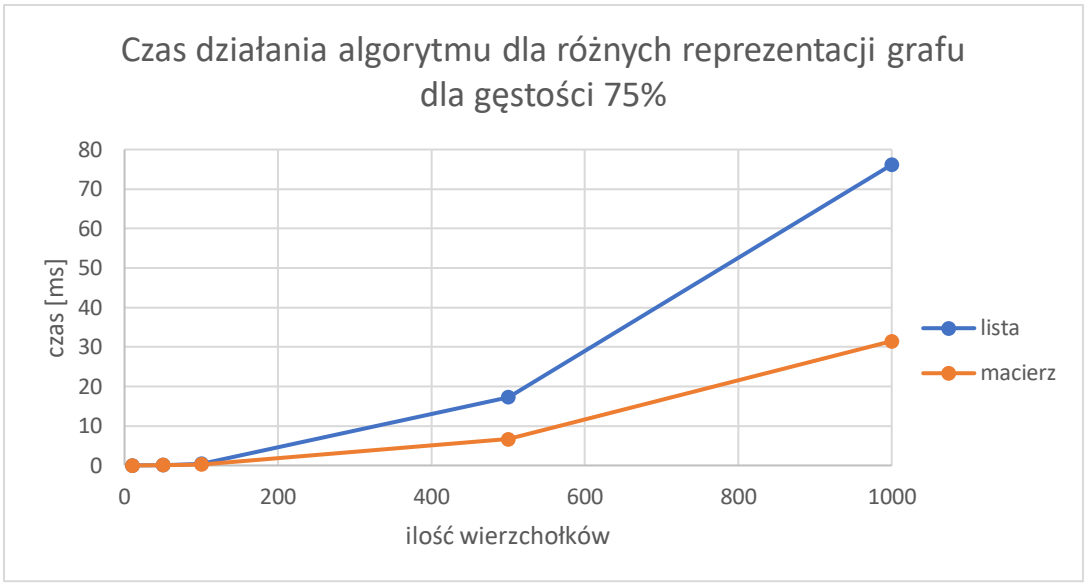
LISTA	czas [ms]				
wierzchołków gęstość	10	50	100	500	1000
pełny	0,0088	0,776	0,453	23,32	102,76
75%	0,0082	0,095	0,385	17,27	76,16
50%	0,0071	0,072	0,250	11,20	50,19
25%	0,0046	0,045	0,151	4,84	24,97

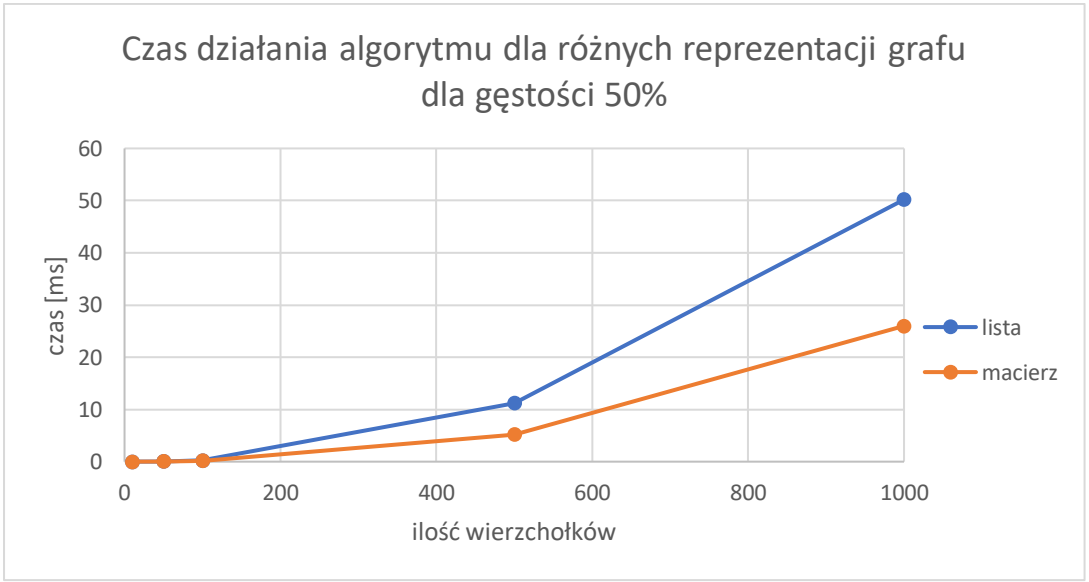
MACIERZ	czas [ms]				
wierzchołków gęstość	10	50	100	500	1000
pełny	0,0072	0,086	0,264	7,98	37,80
75%	0,0069	0,077	0,239	6,66	31,47
50%	0,0062	0,066	0,200	5,22	25,99
25%	0,0042	0,053	0,159	4,00	19,07

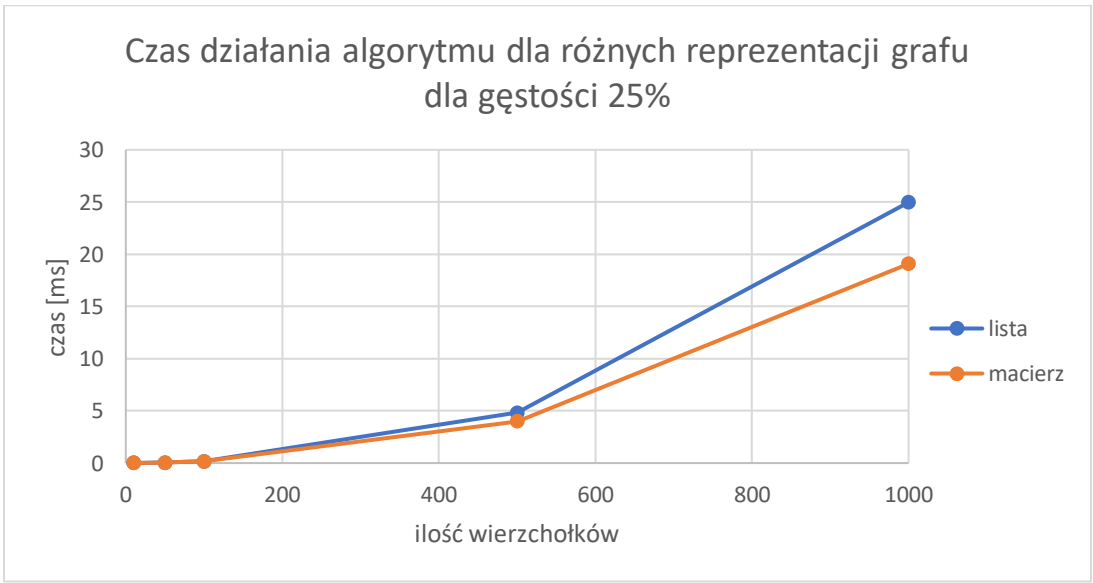

Czas działania algorytmu dla różnych gęstości grafu reprezentowanego listą sąsiedztwa
120
100
80
60
40
20
0
czas [ms]
0
200
400
600
800
1000
ilość wierzchołków
graf pełny
75%
50%
25%


Czas działania algorytmu dla różnych gęstości grafu reprezentowanego macierzą sąsiedztwa
40
35
30
25
20
15
10
5
0
czas [ms]
0
200
400
600
800
1000
ilość wierzchołków
graf pełny
75%
50%
25%


Czas działania algorytmu dla różnych reprezentacji grafu dla grafu pełnego
120
100
80
60
40
20
0
czas [ms]
0
200
400
600
800
1000
ilość wierzchołków
lista
macierz


Czas działania algorytmu dla różnych reprezentacji grafu dla gęstości 75%
czas [ms]
80
70
60
50
40
30
20
10
0
0
200
400
600
800
1000
ilość wierzchołków
lista
macierz


Czas działania algorytmu dla różnych reprezentacji grafu dla gęstości 50%
czas [ms]
60
50
40
30
20
10
0
0
200
400
600
800
1000
ilość wierzchołków
lista
macierz


Czas działania algorytmu dla różnych reprezentacji grafu dla gęstości 25%
czas [ms]
30
25
20
15
10
5
0
0
200
400
600
800
1000
ilość wierzchołków
lista
macierz

Wnioski:

Algorytm działa szybciej dla mniejszych gęstości grafu. Wynika to z faktu, że mają one mniej krawędzi. Jest to szczególnie widoczne w wypadku listy sąsiedztwa, gdyż powoduje to zmniejszoną ilość iteracji. W wypadku macierzy nawet jeśli graf nie jest pełny, to zostaje to potwierdzone dopiero po sprawdzeniu, że odpowiedni element w tablicy wskazuje na NULL. Program działał wolniej dla implementacji na liście sąsiedztwa. Uwidaczniało się to wraz ze wzrostem gęstości grafu i ilości wierzchołków, a tym samym przyrostem liczby krawędzi. Najprawdopodobniej przyczyną tego jest dostęp do każdego elementu w tablicy w czasie stałym, natomiast lista wymaga iteracji, więc czas jest liniowy. Macierz jednak też ma też swoją wadę, jaką jest wolniejsze dodawanie i usuwanie wierzchołków.

Źródła

- **https://stackoverflow.com**
- **https://en.cppreference.com/**
- **https://eduinf.waw.pl/inf/alg/001_search/0113.php**
- **https://pl.wikipedia.org/wiki/Algorytm_Dijkstry**
- **https://eduinf.waw.pl/inf/alg/001_search/0138.php**
- **https://www.youtube.com/watch?v=gdmfOwyQlcI**
- **http://cpp0x.pl/forum/temat/?id=18251**