Artur Woliński, 258952 30.05.2022r

Projektowanie algorytmów i metody sztucznej inteligencji
Projekt 2.

Wprowadzenie

Zadaniem projektowym było zaimplementowanie algorytmu Dijkstry i wykonie szeregu testów do sprawdzenia efektywności dla różnych rodzajów implementacji grafów.

Algorytm testowany były dla grafów o różnym stopniu wypełnienia oraz ilości wierzchołków:

Ilość wierzchołków:

- 100
- 200
- 500
- 1000
- 2000

Gęstości grafów:

- 25%
- 50%
- 75%
- 100%

Opis algorytmu Dijkistry

Algorytm ten pozwala znaleźć koszty dojścia od wierzchołka startowego v do każdego innego wierzchołka w grafie (o ile istnieje odpowiednia ścieżka). Dodatkowo wyznacza on poszczególne ścieżki. Zasada pracy jest następująca:

Tworzymy dwa zbiory wierzchołków Q i S. Początkowo zbiór Q zawiera wszystkie wierzchołki grafu, a zbiór S jest pusty. Dla wszystkich wierzchołków u grafu za wyjątkiem startowego v ustawiamy koszt dojścia d (u) na nieskończoność. Koszt dojścia d (v) zerujemy. Dodatkowo ustawiamy poprzednik p (u) każdego wierzchołka u grafu na niezdefiniowany. Poprzedniki będą wyznaczały w kierunku odwrotnym najkrótsze ścieżki od wierzchołków u do wierzchołka startowego v. Teraz w pętli dopóki zbiór Q zawiera wierzchołki, wykonujemy następujące czynności:

1. Wybieramy ze zbioru Q wierzchołek u o najmniejszym koszcie dojścia d (u).
2. Wybrany wierzchołek u usuwamy ze zbioru Q i dodajemy do zbioru S.
3. Dla każdego sąsiada w wierzchołka u, który jest wciąż w zbiorze Q, sprawdzamy, czy

d (w) > d (u) + waga krawędzi u–w.

Jeśli tak, to wyznaczamy nowy koszt dojścia do wierzchołka w jako:

d (w) ← d (u) + waga krawędzi u–w.

Następnie wierzchołek u czynimy poprzednikiem w:

p (w) ← u.

Złożoność obliczeniowa

Złożoność obliczeniowa algorytmu Dijkstry zależy od liczby V wierzchołków i E krawędzi grafu. O rzędzie złożoności decyduje implementacja kolejki priorytetowej:

wykorzystując „naiwną" implementację poprzez zwykłą tablicę, otrzymujemy algorytm o złożoności O(V^{2})}O(V^{2})

w implementacji kolejki poprzez kopiec, złożoność wynosi O(E\log V)O(E\log V)

po zastąpieniu zwykłego kopca kopcem Fibonacciego złożoność zmienia się na { O(E+V\log V)O(E+V\log V)

Pierwszy wariant jest optymalny dla grafów gęstych (czyli jeśli E=\Theta (V^{2})E=\Theta (V^2)), drugi jest szybszy dla grafów rzadkich (E=\Theta (V)), (E=\Theta (V)),trzeci jest bardzo rzadko używany ze względu na duży stopień skomplikowania i niewielki w porównaniu z nim zysk czasowy.

Pomiar efektywności

ilość wierzchołków	Gęstość			
	25%	50%	75%	100%
100	8,43	9,13	18,04	10,97
200	29,60	31,71	35,55	39,75
500	183,06	245,90	190,50	272,22
1000	840,33	1717,61	1452,20	1448,95
2000	5372,10	5999,84	7651,29	8665,74

Tabela 1. Macierz sąsiedztwa

ilość wierzchołków	Gęstość			
	25%	50%	75%	100%
100	5,84	7,26	13,17	11,66
200	26,04	26,75	59,02	57,24
500	197,88	386,39	977,91	2164,56
1000	1830,41	2912,82	6498,24	9699,97
2000	8221,09	20768,60	28598,10	38101,30

Tabela 2. Lista sąsiedztwa

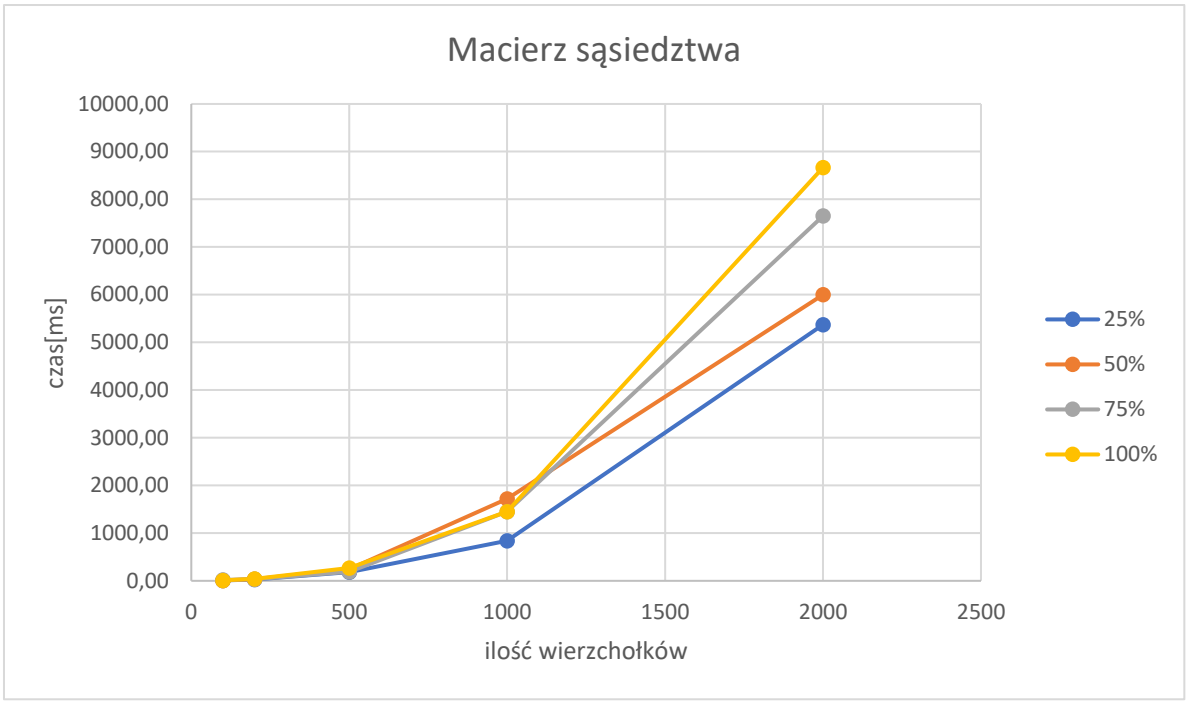

Tabela 3. macierz sąsiedztwa

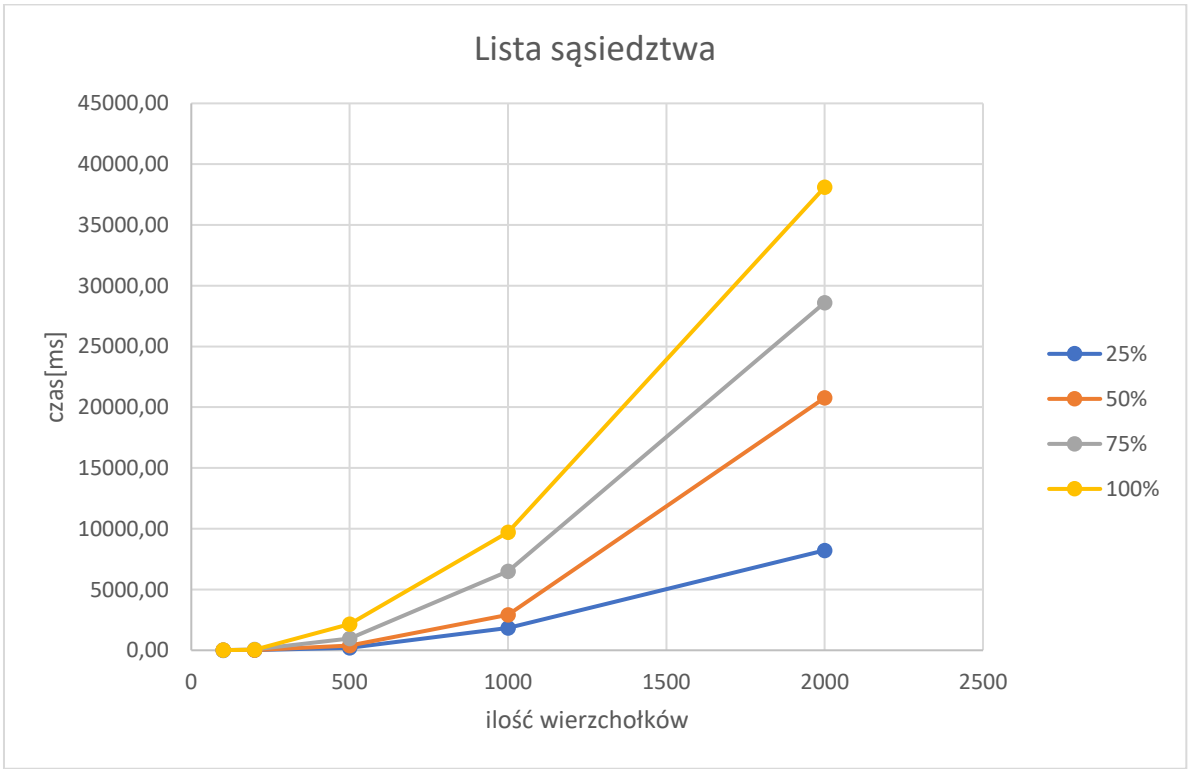

Tabela 4. Lista sąsiedztwa

Porównanie czasów przeszukiwania

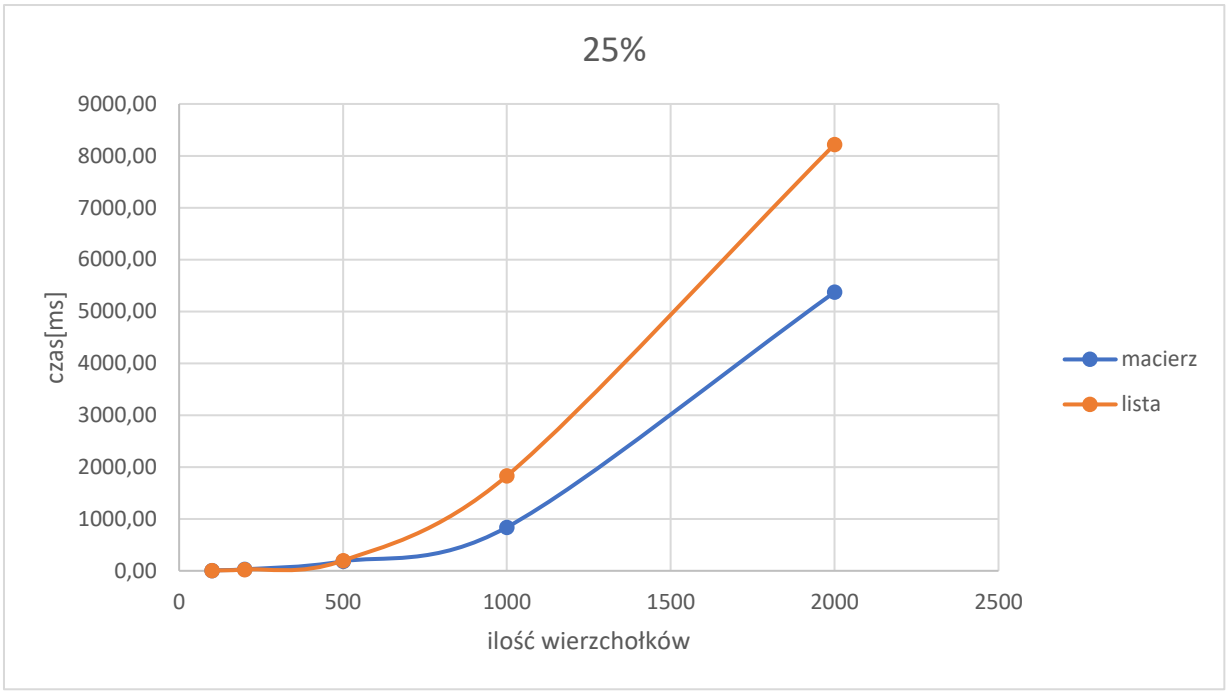

Tabela 5. 25% wypełnienia

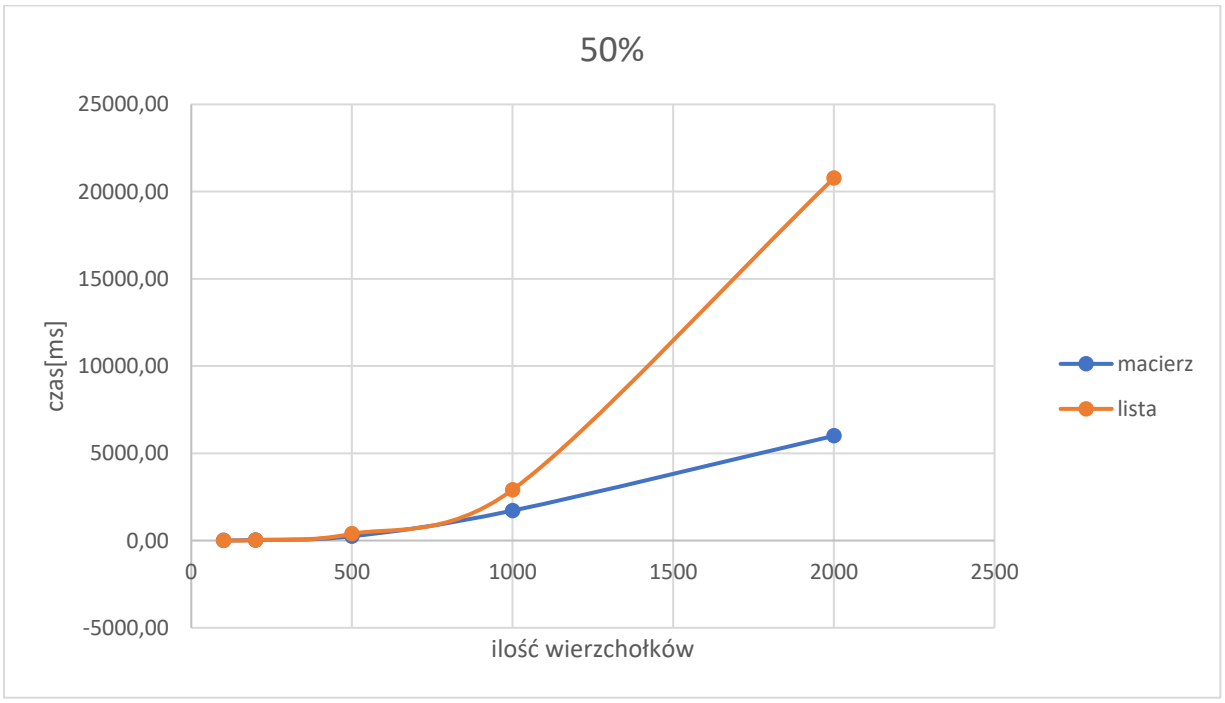

Tabela 6. 50% wypełnienia

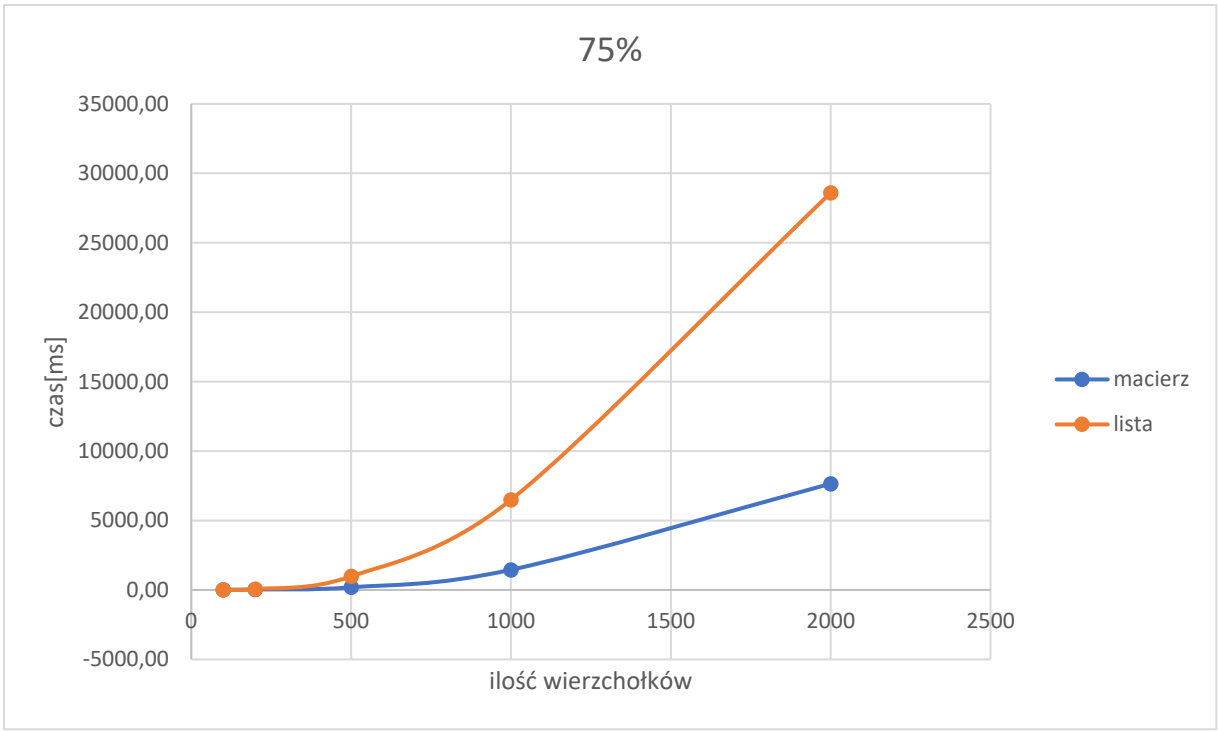

Tabela 7. 75% wypełnienia

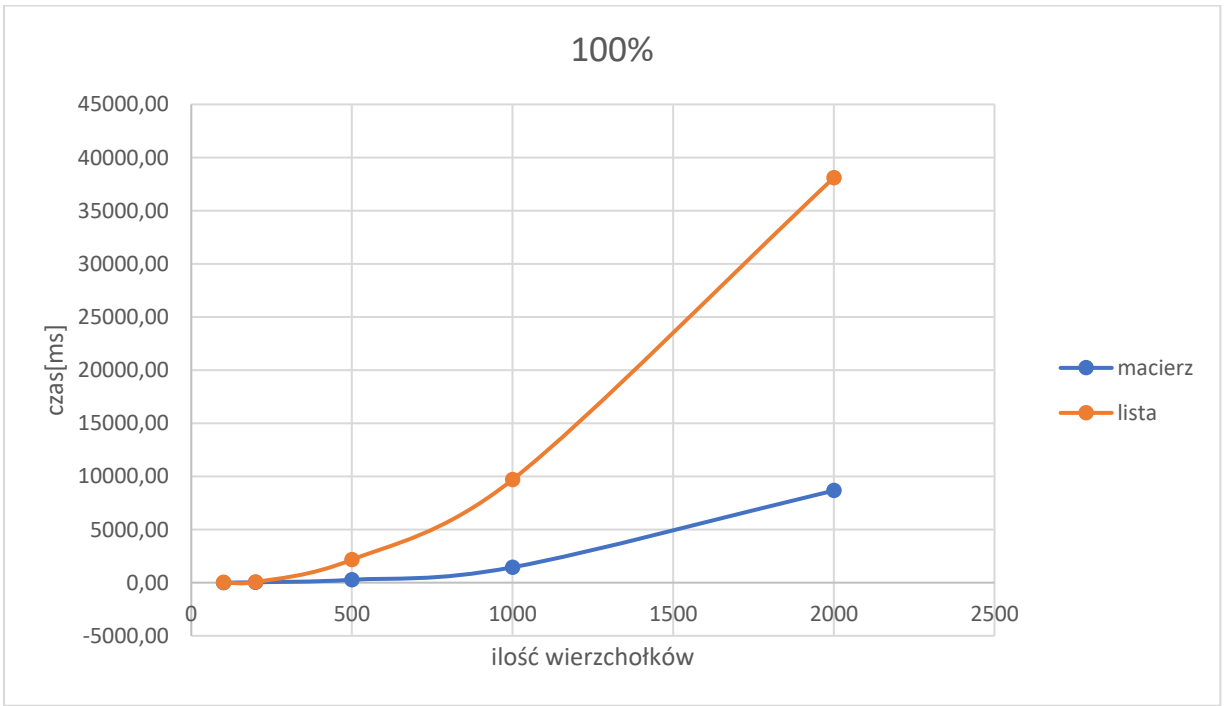

Tabela 8. 100% wypełnienia

Wnioski

- Algorytm Dijkistry działa bardziej efektywnie dla macierzy sąsiedztwa
- Dla macierzy sąsiedztwa efektywność wzrasta wraz z ilością wierzchołków

- Dla macierzy sąsiedztwa różnice w czasie działania są niewielkie, natomiast dla listy sąsiedztwa są dosyć znaczne, widać to najwyraźniej przy dużych ilościach wierzchołków.

Bibliografia

- https://pl.wikipedia.org/wiki/Algorytm_Dijkstry
- https://eduinf.waw.pl/inf/alg/001_search/0138.php
- https://www.algorytm.edu.pl/olimpiada-informatyczna/algorytm-dijkstry