Projektowanie algorytmów i metody sztucznej inteligencji, bud C-3,s.103

Temat projektu: Grafy - problem najkrótszej ścieżki

Projekt wykonał:

Jakub Kędzierski 248915

Data: 30.04.2020r

Termin zajęć: czwartek 9:15-11:00

Prowadzący: Dr inż. Łukasz Jeleń

---

1. Wstęp

W projekcie zbadano efektywność algorytmu znalezienia najkrótszej ścieżki między dwoma wierzchołkami w grafie w zależności od jego reprezentacji. Do rozwiązania tego problemu zastosowano algorytm Dijkstry. Graf zaimplementowano na dwa sposoby przy użyciu:

- listy sąsiedztwa
- macierzy sąsiedztwa

Dla każdej reprezentacji grafu zaimplementowano standardowe metody dostępu. Dodatkowo program ma możliwość wczytania grafu z pliku tekstowego oraz zapisu działania algorytmu do pliku. Kod programu napisano w języku C++, a testy przeprowadzono na systemie Linux (dystrybucja Ubuntu). Efektywność algorytmu badano dla:

- 2 reprezentacji grafu
- 5 różnych liczb wierzchołków: 10, 50, 100, 500, 1000
- 4 gęstości grafu: 25%, 50%, 75%, 100% (graf pełny)

Dla każdej gęstości grafu innej niż 100%, krawędzie tworzone były losowo, aby uniknąć niespójnych wyników ponieważ systematyczne ułożenie krawędzi mogłoby mieć wpływ na szybkość algorytmu. Dla każdego przypadku przeprowadzono 100 prób z których wyliczono średnią czasu przebiegu algorytmu. Po zakończeniu testów zobrazowano wyniki w postaci tabel i wykresów oraz porównano otrzymane rezultaty z założeniami teoretycznymi.

2. Opis algorytmu

Algorytm Dijkstry służy do znajdowania najkrótszej ścieżki pomiędzy wybranymi wierzchołkami w grafie o nieujemnych wagach krawędzi. Algorytm podczas swojej pracy wyznacza nie tylko najkrótszą odległość między dwoma konkretnymi wierzchołkami, ale najkrótszą ścieżkę od zadanego wierzchołka startowego do wszystkich pozostałych wierzchołków.

Krótki, uproszczony opis algorytmu:

- stworzenie zbioru zawierającego wszystkie wierzchołki w grafie

- ustawienie początkowych odległości od wierzchołka startowego (0 – wierzchołek startowy, oo- wierzchołki pozostałe)

- w pętli dopóki nasz zbiór zawierający wierzchołki nie jest pusty: wybieramy ze zbioru wierzchołek o najmniejszym koszcie dojścia, dla wybranego wierzchołka dla wszystkich wierzchołków sąsiadujących dokonujemy relaksacji krawędzi na określonych warunkach i przypisujemy nowy koszt ścieżki.

Wpływ na szybkość algorytmu ma reprezentacja grafu – powodem jest częste odwoływanie się do wierzchołków sąsiadujących. Operacja ta powinna być wykonywana w krótszym czasie dla grafów niepełnych reprezentowanych na liście sąsiedztwa. Wynika to ze sposobu przedstawienia grafu:

- lista sąsiedztwa: każdy wierzchołek grafu posiada własną listę krawędzi (listę wierzchołków sąsiadujących),

- macierz sąsiedztwa: krawędzie grafu są przedstawione w dwuwymiarowej tablicy, tam gdzie krawędź nie istnieje tablica przechowuje znacznik NULL.

Podczas odwoływania się do metody pobierającej krawędzie danego wierzchołka na liście sąsiedztwa otrzymujemy gotową listę, w przeciwieństwie do macierzy, gdzie musimy iterować po „pustych polach", czyli krawędziach które nie istnieją. W sytuacji gdy graf jest pełny, sposób reprezentacji grafu nie powinien odgrywać dużej roli w szybkości przebiegu algorytmu. Istotną rolę w szybkości funkcjonowania algorytmu odgrywa sposób zaimplementowania kolejki priorytetowej przechowującej odległości od wierzchołku. W projekcie kolejkę zaimplementowano w postaci kopca, co skróciło czas pobierania z kolejki najmniejszej wartości klucza.

Złożoność obliczeniowa algorytmu dla grafu reprezentowanego na liście sąsiedztwa to O((v+e) log v), czyli w najgorszym przypadku **O(e·logv),** gdzie e to liczba krawędzi, a v liczba wierzchołków.

Oczekiwana złożoność obliczeniowa dla grafu reprezentowanego na macierzy sąsiedztwa to O(v^2 + e log v) , co w najgorszym przypadku sprowadza się również do **O(e·logv).**

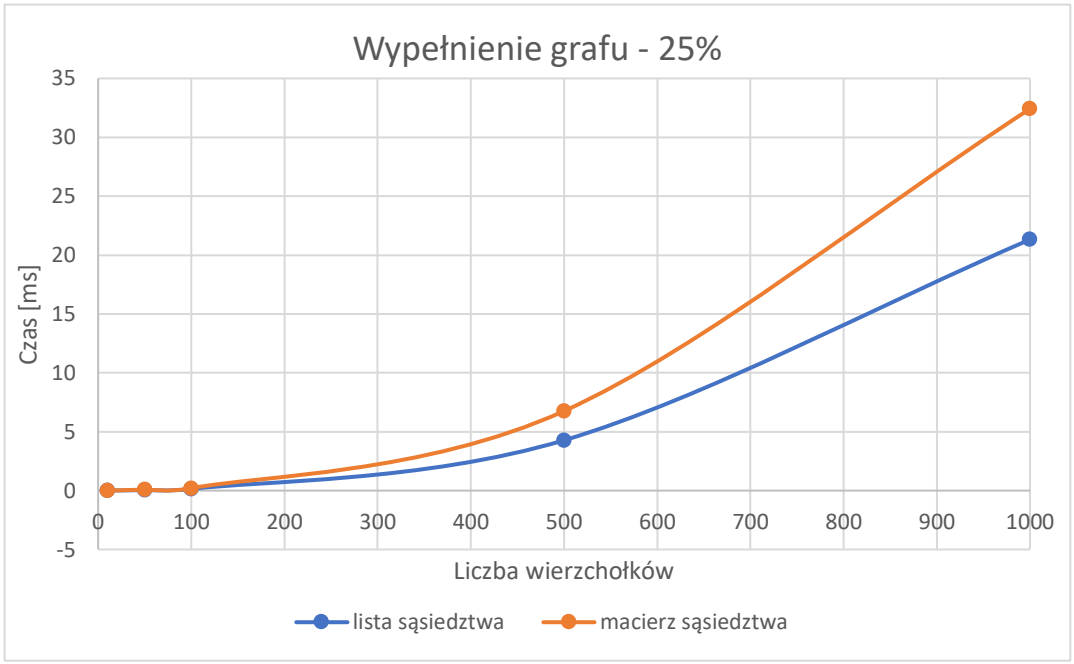
3. Wyniki eksperymentów

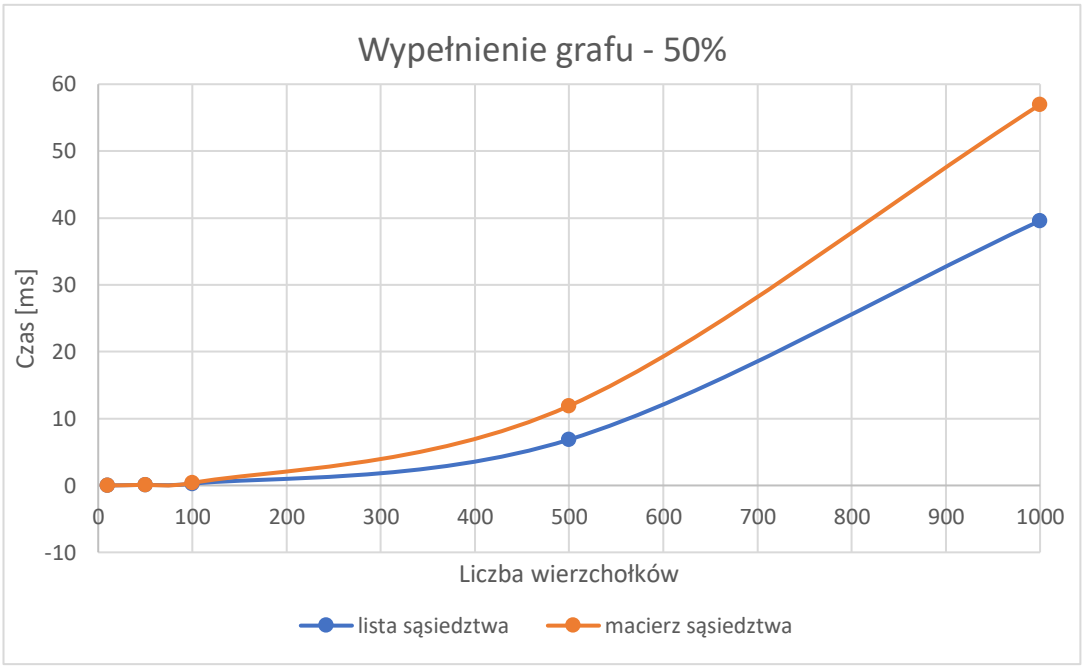
Wyniki podane w tabelach i przedstawione na wykresie to wyniki czasu przebiegu algorytmu uśrednione ze 100 prób dla każdego przypadku. Czas podany jest w ms.

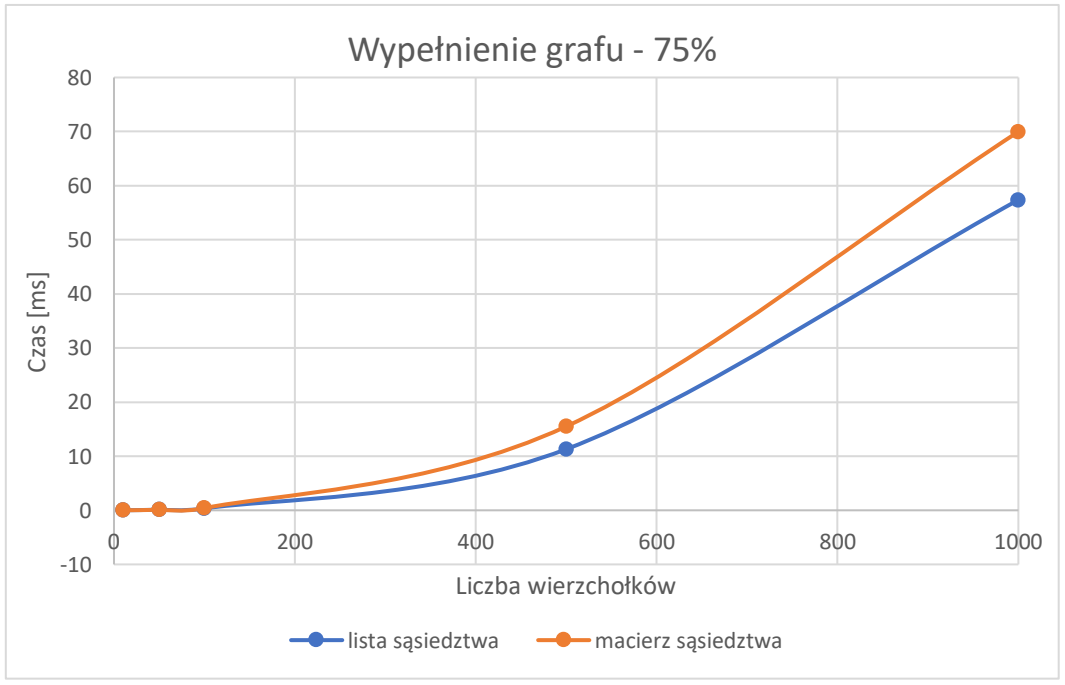
Lista sąsiedztwa					
gęstość grafu (%) ↓	**liczba wierzchołków**				
	10	**50**	**100**	**500**	**1000**
0,25	0,0017	0,045	0,14	4,3	21,3
0,5	0,0027	0,093	0,25	6,9	39,6
0,75	0,0037	0,113	0,35	11,3	57,4
1	0,0043	0,120	0,42	15,7	76,0

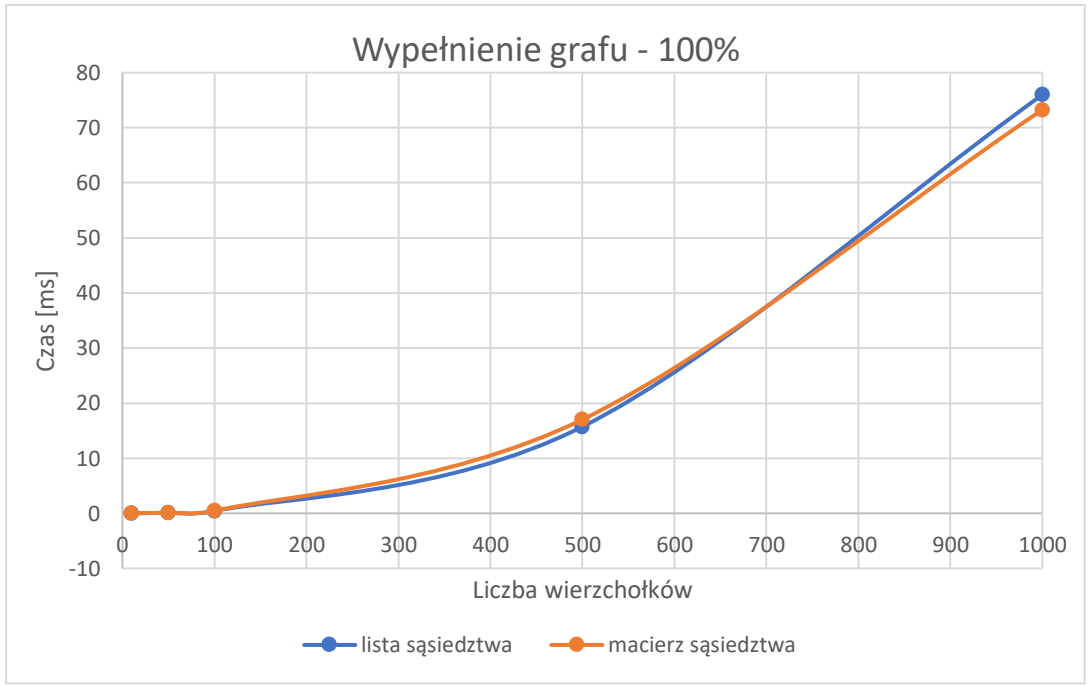
Macierz sąsiedztwa					
gęstość grafu (%) ↓	**liczba wierzchołków**				
	10	**50**	**100**	**500**	**1000**
0,25	0,0024	0,083	0,20	6,8	32,4
0,5	0,0029	0,098	0,37	11,9	57,0
0,75	0,0040	0,099	0,40	15,5	70,0
1	0,0060	0,124	0,46	17,0	73,3

Wykresy dla każdego wypełnienia grafu



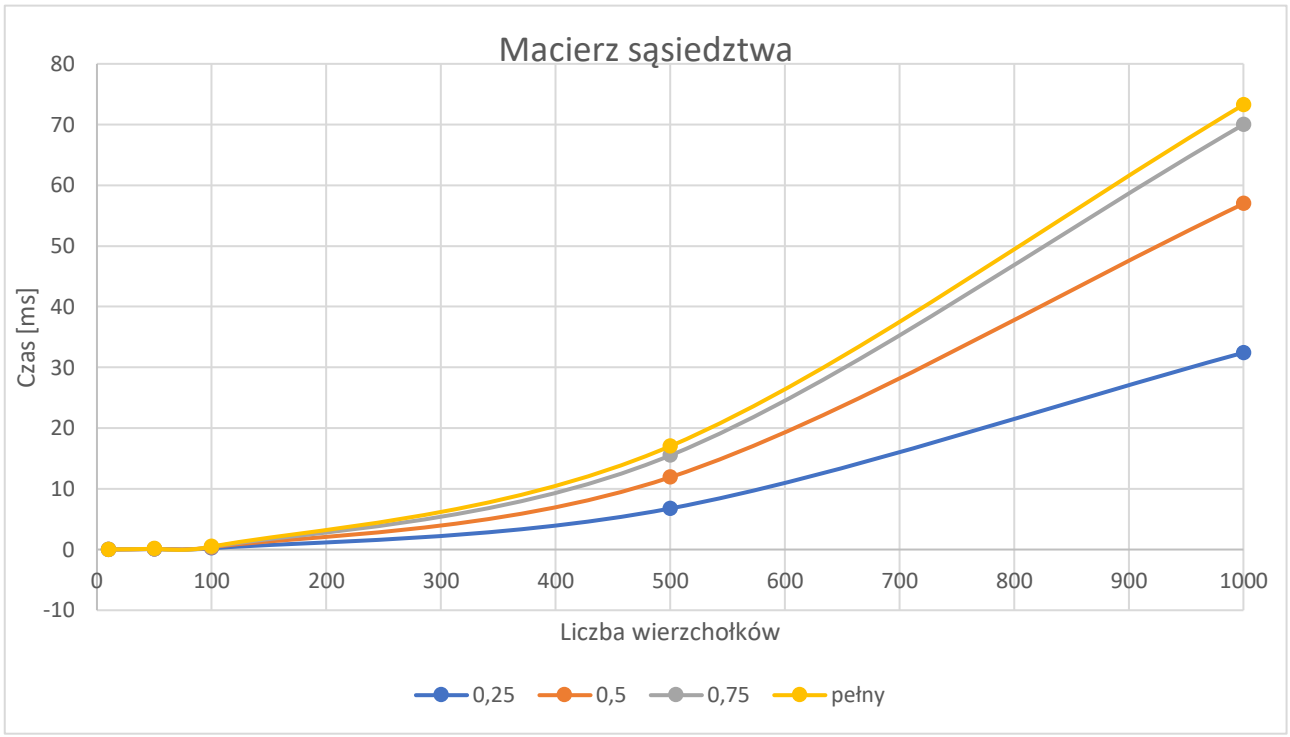


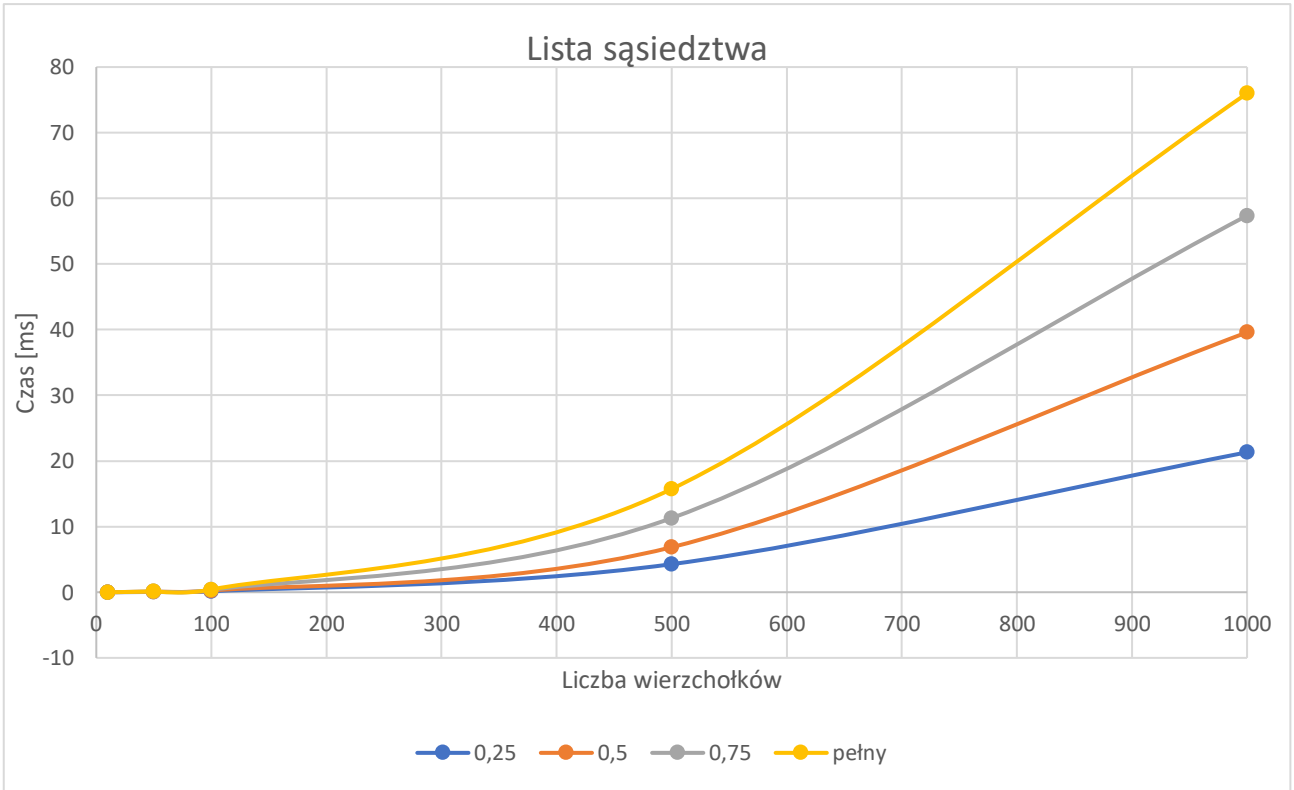




Wykresy dla każdej reprezentacji grafu

Legendy opisują wypełnienie grafu w %





4. Wnioski

Wnioski jakie można otrzymać po analizie otrzymanych wyników są zgodne z wcześniejszymi założeniami teoretycznymi. Krzywe zobrazowane na wykresach pokrywają się z oczekiwanymi złożonościami obliczeniowymi (O(e·logv)). Długość działania algorytmu jest zależna od liczby krawędzi, wierzchołków oraz reprezentacji grafu. Czas wykonania algorytmu zwiększał się wraz ze wzrostem liczby krawędzi oraz liczby wierzchołków. Dla grafów pełnych był najdłuższy i zgodnie z przewidywaniami podobny dla różnych reprezentacji grafów. Dla mniejszych gęstości był krótszy dla grafu reprezentowanego na liście sąsiedztwa ponieważ ta implementacja zapewnia krótszy czas działania metody udostępniającej wszystkie krawędzie danego wierzchołka (*„incidentEdges(int)”*). Przykładowo: przy 25% gęstości grafu, czasy dla macierzy sąsiedztwa są średnio o 50-60% większe od czasów listy sąsiedztwa.

Zaimplementowanie kolejki priorytetowej z której korzysta algorytm Dijkstry za pomocą kopca poprawiło szybkość działania algorytmu. Podczas pracy z program, często pojawiał się problem z wyciekami pamięci co można byłoby wyeliminować poprzez wykorzystanie biblioteki STL (np. klasa vector, wskaźniki inteligentne). Dostrzeżono znaczne różnice w szybkości algorytmu przy uruchamianiu programu przez IDE oraz z poziomu terminala. Możliwe jest otrzymanie odmiennych czasów dla innych konfiguracji sprzętowych.

5. Bibliografia

Do analizy algorytmów korzystałem z:

- Wolna encyklopedia Link: https://www.wikipedia.org/ [data dostępu: kwiecień 2020]
- - Portal o tematyce informatycznej: https://stackoverflow.com/ [data dostępu: kwiecień 2020]
- Portal o tematyce informatycznej:
  Link: https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-using-priority_queue-stl/ [data dostępu: kwiecień 2020]
- -serwis edukacyjny „Algorytmy Struktury Danych”
  Link : https://eduinf.waw.pl/inf/alg/001_search/0113.php#P4 [data dostępu: kwiecień 2020]