Nazwa kursu: Projektowanie Algorytmów i Metody Sztucznej Inteligencji

Tytuł: Projekt 2 (grafy)

Data oddania: 08.05.2019 r.

Termin zajęć: Środa, 11-13

Prowadzący: dr inż. Łukasz Jeleń

Dane studenta: Patryk Szczygieł 241578

1. Cel ćwiczenia

   Celem projektu było zaimplementowanie grafu przechowującego elementy określonego typu i dwóch algorytmów (Kruskala oraz Prima) służących do wyznaczania minimalnego drzewa rozpinającego. Implementacje grafu trzeba było zrobić na dwa sposoby: za pomocą listy sąsiedztwa oraz macierzy sąsiedztwa. Należało porównać algorytmy, przeprowadzając analizę efektywności dla 100 losowych instancji w zależności od typu reprezentacji grafu, ilości wierzchołków oraz gęstości grafu.

2. Algorytm Kruskala

   Jest to jeden z algorytmów służących do wyznaczania minimalnego drzewa rozpinającego, czyli takiego, które łączy wszystkie wierzchołki danego grafu spójnego, mające najmniejszą możliwą sumę wag krawędzi. Na początku, algorytm dzieli wszystkie wierzchołki na osobne klastry, a następnie bierze krawędź o najmniejszej wadze. Jeśli krawędź ta łączy wierzchołki z różnych klastrów to jest ona dołączana do rozwiązania, natomiast wierzchołki są łączone w jeden klaster. Czynność ta jest powtarzana (v-1) razy, gdzie v to liczba wierzchołków grafu.
   Jeśli chodzi o złożoność algorytmu to przy implementacji jego kolejki za pomocą kopca otrzymujemy $\mathrm{E} * \log V$ dla wstawiania kolejnych elementów do kolejki. Po dodaniu usuwania z czasem $\log V$ E razy i wzięcia całkowitego czasu pętli while mamy $(\mathrm{E} + \mathrm{V}) * \log V$ co po uproszczeniu da $O(E \log V)$.

3. Algorytm Prima

   Algorytm Prima jest drugim narzędziem do wyznaczania minimalnego drzewa rozpinającego. Działa on jednak na innej zasadzie niż Algorytm Kruskala. Na początku, wybierany jest punkt startowy, czyli dowolny wierzchołek w danym grafie. Dla każdego wierzchołka określamy, że jego koszt wynosi nieskończoność, natomiast wybranemu wierzchołkowi przypisujemy wartość 0. Potem określane są wierzchołki, które są osiągalne dla wierzchołka startowego oraz wybierany jest ten o najmniejszym koszcie. Wybranemu sąsiadowi jest przypisywana wartość oraz jest on dodawany wraz z krawędzią do drzewa. Następnie algorytm znów wybiera wierzchołek o najmniejszym koszcie, który jest w zasięgu drzewa. Jeśli wartość sprawdzanego wierzchołka jest

mniejsza niż dotychczas przypisana jest ona nadpisywana. Czynność ta jest powtarzana aż drzewo nie obejmie wszystkich wierzchołków grafu.
Złożoność algorytmu jest zależna od implementacji kolejki. Najmniejsza będzie przy implementacji kolejki priorytetowej przy pomocy kopca gdzie usuwanie elementu zajmie $\log V$. Odbędzie się to E razy więc mamy $\mathrm{E} * \log V$. Ewentualne nadpisywanie to $\log V$, a więc razem mamy $(\mathrm{E} + \mathrm{V}) * \log V$. Po uproszczeniu złożoność wyniesie $O(E \log V)$.

| | liczba wierzchołków | | | | | |
|---|---|---|---|---|---|---|
| gęstość | | 10 | 50 | 100 | 500 | 750 |
| | 25% | 0,0000022s | 0,0000027s | 0,0000094s | 0,0001836s | 0,000441s |
| | 50% | 0,0000019s | 0,0000046s | 0,0000154s | 0,0002865s | 0,0007432s |
| | 75% | 0,0000022s | 0,0000073s | 0,000021s | 0,0004117s | 0,0011849s |
| | 100% | 0,0000033s | 0,0000523s | 0,0002149s | 0,0058705s | 0,0127033s |

*Tab. 1. Wyniki pomiarów dla algorytmu Kruskala (reprezentacja macierzowa)*

| liczba wierzchołków | | | | | | |
|---|---|---|---|---|---|---|
| gęstość | | 10 | 50 | 100 | 500 | 750 |
| | 25% | 0,0000045s | 0,0000599s | 0,0001923s | 0,0047004s | 0,0136959s |
| | 50% | 0,0000058s | 0,0000563s | 0,0002012s | 0,0073262s | 0,0201609s |
| | 75% | 0,0000057s | 0,000066s | 0,0002216s | 0,0110347s | 0,0263363s |
| | 100% | 0,0000041s | 0,0000636s | 0,0002547s | 0,0137352s | 0,0325123s |

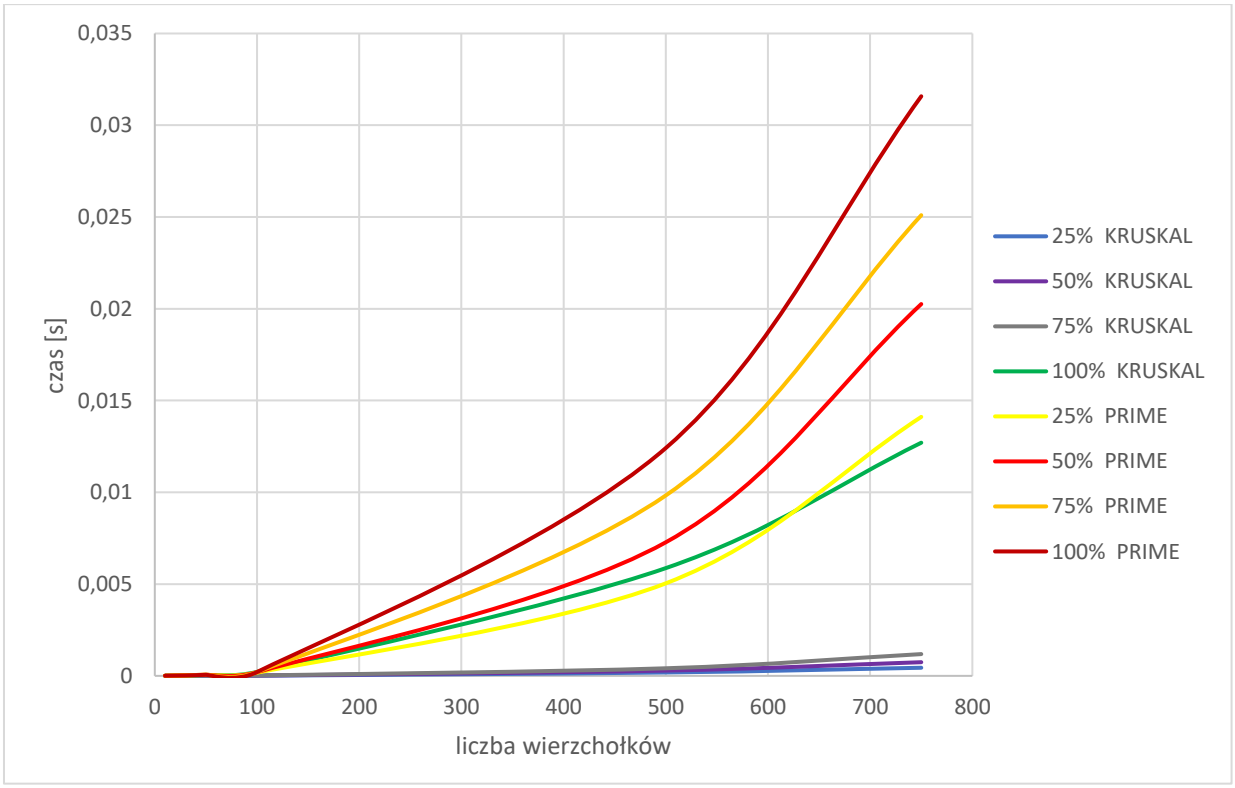
*Tab. 2. Wyniki pomiarów dla algorytmu Kruskala (reprezentacja listowa)*

| | | liczba wierzchołków | | | | |
|---|---|---|---|---|---|---|
| gęstość | | 10 | 50 | 100 | 500 | 750 |
| | 25% | 0,0000041s | 0,0000518s | 0,0001741s | 0,00504s | 0,0141103s |
| | 50% | 0,0000052s | 0,0000612s | 0,0001997s | 0,0072773s | 0,0202574s |
| | 75% | 0,0000058s | 0,0000636s | 0,000206s | 0,0098288s | 0,0251001s |
| | 100% | 0,000005s | 0,0000553s | 0,0002079s | 0,0124123s | 0,031579s |

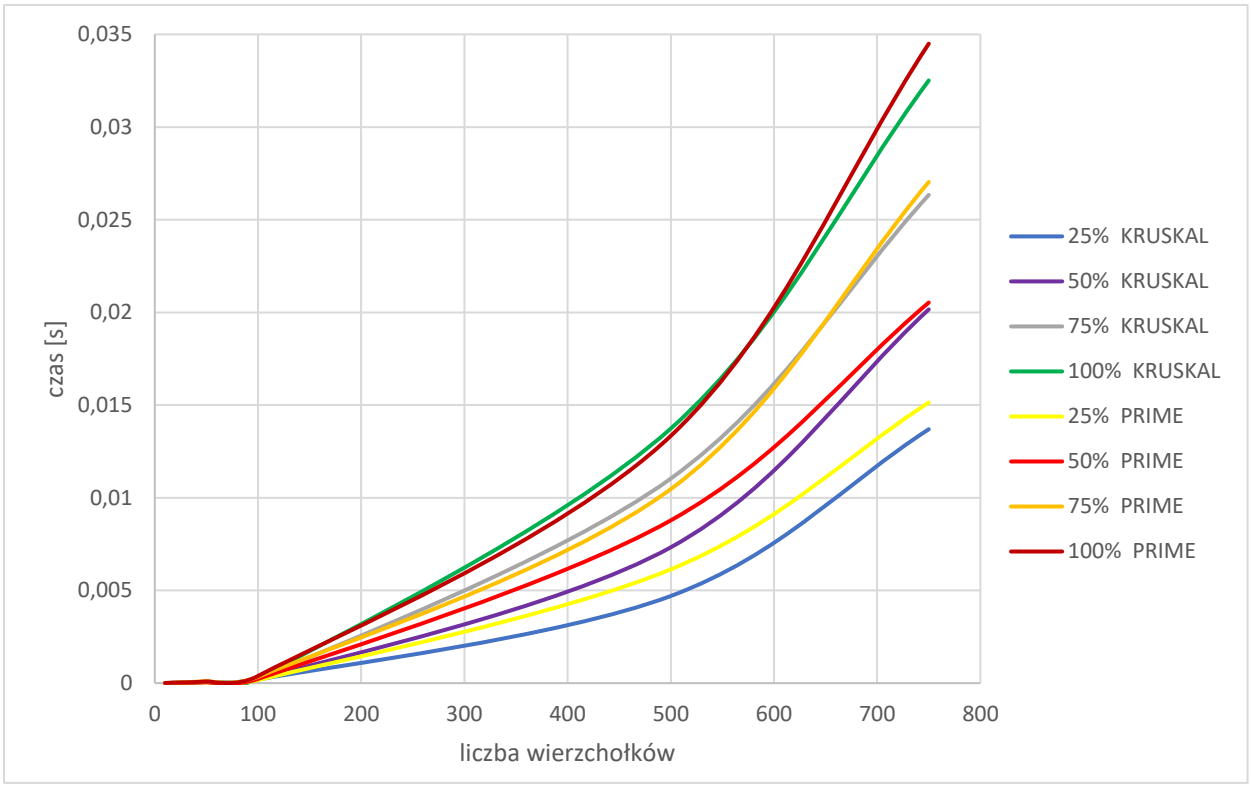
*Tab. 3. Wyniki pomiarów dla algorytmu Prima (reprezentacja macierzowa)*

| | | liczba wierzchołków | | | | |
|---|---|---|---|---|---|---|
| gęstość | | 10 | 50 | 100 | 500 | 750 |
| | 25% | 0,0000041s | 0,0000451s | 0,0001663s | 0,0061378s | 0,0151379s |
| | 50% | 0,0000041s | 0,0000504s | 0,0002288s | 0,0087793s | 0,020532s |
| | 75% | 0,0000052s | 0,0001019s | 0,0003381s | 0,0104664s | 0,0270361s |
| | 100% | 0,0000077s | 0,0000942s | 0,0003837s | 0,013337s | 0,0344957s |

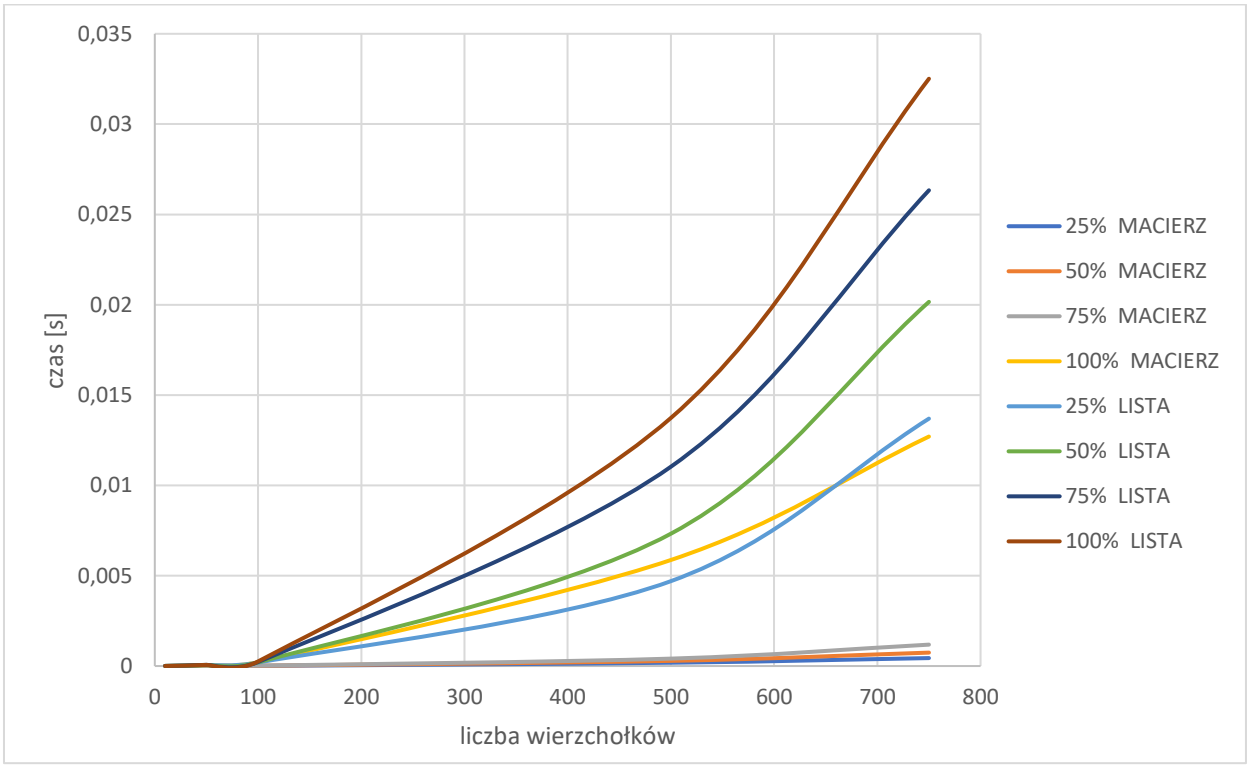
*Tab. 4. Wyniki pomiarów dla algorytmu Prima (reprezentacja listowa)*



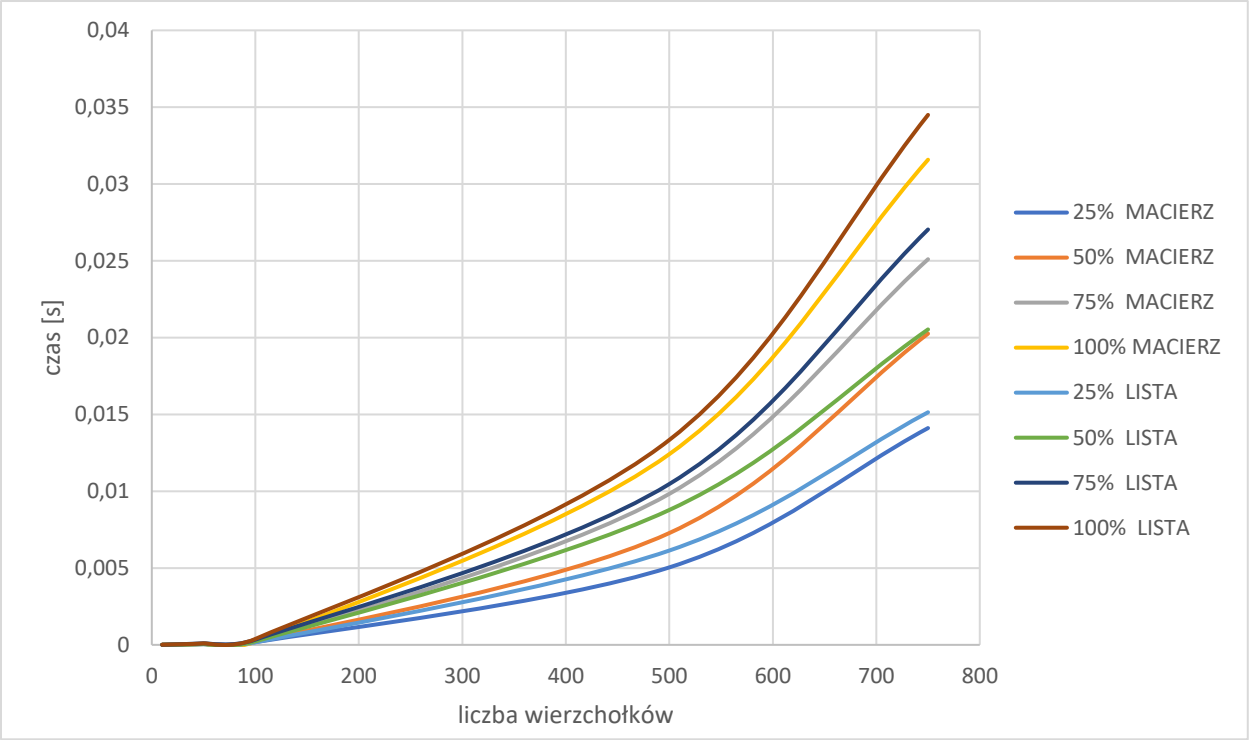

*Rys. 1. Porównanie algorytmu Kruskala i Prima dla reprezentacji macierzowej*




*Rys. 2. Porównanie algorytmu Kruskala i Prima dla reprezentacji listowej*




*Rys. 3. Porównanie reprezentacji macierzowej i listowej dla algorytmu Kruskala*




*Rys. 4. Porównanie reprezentacji macierzowej i listowej dla algorytmu Prima*

4. Specyfikacja komputera, na którym prowadzono pomiary

   Procesor: Intel Core i5-7300HQ (2.5 GHz, 3.5 GHz Turbo, 6 MB Cache)
   Ram: 8GB

5. Wnioski

   Oba algorytmy są stosunkowo szybkie i osiągają bardzo podobne czasy w analizie efektywności. Według przewidywań oba algorytmy działają szybciej dla mniejszej gęstości grafu. Przy reprezentacji listowej dla mniejszych gęstości algorytm Kruskala jest minimalnie szybszy, natomiast przy gęstościach zbliżonych do grafu pełnego algorytm Prima okazuje się być nieznacznie lepszy. Przy reprezentacji macierzowej algorytm Kruskala jest szybszy dla każdej gęstości.
   Jeśli chodzi o porównanie reprezentacji grafu dla tego samego algorytmu, stosowanie reprezentacji macierzowej może być nieco lepsze. Reprezentacja listowa jednak ma taką przewagę, iż nie trzeba w niej deklarować ilości elementów.

6. Literatura


1) http://www.algorytm.org/algorytmy-grafowe/algorytm-prima.html
   (korzystano 22.04 - 04.05.2019)
2) http://www.algorytm.org/algorytmy-grafowe/algorytm-kruskala.html
   (korzystano 22.04 - 04.05.2019)
3) https://pl.wikipedia.org/wiki/Reprezentacja_grafu#Reprezentacja_przez_list%C4%99_s%C4%85siedztwa
   (korzystano 22.04 - 04.05.2019)
4) https://pl.wikipedia.org/wiki/Algorytm_Kruskala
   (korzystano 22.04 - 04.05.2019)
5) https://pl.wikipedia.org/wiki/Algorytm_Prima
   (korzystano 22.04 - 04.05.2019)
6) http://algorytmika.wikidot.com/mst
   (korzystano 22.04 - 04.05.2019)
7) http://algorytmy.ency.pl/artykul/algorytm_kruskala
   (korzystano 22.04 - 04.05.2019)
8) http://algorytmy.ency.pl/artykul/minimalne_drzewo_rozpinajace
   (korzystano 22.04 - 04.05.2019)
9) https://mattomatti.com/pl/a0146
   (korzystano 22.04 - 04.05.2019)