Projekt 2

Grafy – podstawowe działania

PAMSI

Projektowanie algorytmów i metody sztucznej inteligencji

Prowadzący: **mgr. Inż. Magdalena Skoczeń**
Termin zajęć: **Środa 18^{55}**

Piotr Niedziółka, 249023

29.04.2020

1. Wprowadzenie

Celem projektu była implementacja grafów w dwóch formach:

- w formie listy sąsiedztwa
- w formie macierzy sąsiedztwa

Następnie zaimplementowanie algorytmów Kruskala i Prima oraz przeprowadzenie analizy efektywności tych algorytmów.
Badania zostały wykonane dla 5 różnych wierzchołków oraz następujących gęstości grafu: *25%*,*50%*,*75%*, i dla grafu pełnego *100%*, dla 5 różnych liczb wierzchołków: *50, 100, 200, 500, 1000*.
Wzór na liczbę krawędzi:

$$L_{kraw} = \frac{W \cdot (W-1) \cdot Gestosc}{2}$$

2. Omówienie algorytmów

- Algorytm **Kruskala**

Algorytm grafowy wyznaczający minimalne drzewo rozpinające dla grafu nieskierowanego ważonego i spójnego. Znajduje drzewo, poprzez posortowanie wagami krawędzi i wybieraniem najmniejszej, jeżeli dany wierzchołek nie jest jeszcze połączony, czyli nie jest w drzewie.

Program napisałem zarówno na liście sąsiedztwa, jak i na macierzy sąsiedztwa. Potrzebna do tego była kolejka w formie kopca, lista sąsiedztwa, drzewo i struktura zawierająca wszystkie wierzchołki. Struktura sprawdza czy wierzchołki należą już do drzewa – grafu, jeżeli istnieje kilka struktur to następuje ich połączenie, a potem szukanie kolejnych wartości do dodania do struktury.

Pseudokod:

```
Algorytm Kruskal(G)
  Wejście: Graf ważony G z n wierzchołkami i m krawędziami
  Wyjście: Minimalne drzewo rozpinające T dla grafu G

for każdy wierzchołek v w G
    C(v) ← {k}
    Q ← {E}            // E - lista krawędzi
    T ← Ø
while T.size() < n-1 do
    (u,v) ← Q.removeMin()
     if C(v) ≠ C(u)
       T.Add(v,u)
       Merge(C(v),C(u)
return T
```

Złożoność obliczeniowa:

$$\boldsymbol{O(|E| \cdot log\ |V|)}$$

Gdzie: **E** = ilość krawędzi, **V** = ilość wierzchołków

- Algorytm **Prima**

Algorytm grafowy wyznaczający minimalne drzewo rozpinające dla grafu nieskierowanego ważonego i spójnego. Algorytm wylicza podzbiory krawędzi, i bada dla którego podzbioru graf będzie miał najmniejszą możliwą sumę kosztów, czyli MDR.

Program z tym algorytmem napisałem na liście sąsiedztwa, gdyż w macierzy wyrzucało mi błąd – *this->…* , prawdopodobnie spowodowane gubieniem się wskaźników lub wychodzeniem poza zakres.

Program ten bazuje na kolejce w formie stosu, drzewie, liście sąsiedztwa, oraz na tablicy, która przechowuje czy wierzchołek znajduje się w grafie *wierzchołek odwiedzony*.

Algorytm Prima zaczyna się w punkcie startowym, który jest generowany losowo, i wierzchołek ten istnieje już w drzewie. Następnie przechodzimy po wszystkich krawędziach od wierzchołka jednocześnie sprawdzając, które połączenie jest najkrótsze i wybieramy je dodając kolejny wierzchołek, który jest końcem krawędzi.

Pseudokod:

```
Algorytm PrimJarnik(G)
  Wejście: Graf ważony G z n wierzchołkami i m krawędziami
  Wyjście: Minimalne drzewo rozpinające T dla grafu G

v ← losowy wierzchołek z G
D(v) ← 0                // bieżące MST
for każdy wierzchołek u ≠ v
    D(u) ← +∞
    Q ← {E}             // E – lista krawędzi
    T ← Ø
while Q.isEmpty()=false do
    (u,e) ← Q.removeMin()
     T.Add(u,e)
     for każdy wierzchołek z ⊂Q sąsiedni do u
       if w(u,z) < D(z)
         D(z) ← w(u,z)
         Wstaw (u,z) do Q z kluczem D(z)
return T
```

Złożoność obliczeniowa:

$$\boldsymbol{O}(|\boldsymbol{E}| \cdot \boldsymbol{log}\ |\boldsymbol{V}|)$$

Dla macierzy sąsiedztwa wynosi ona

$$\boldsymbol{O}(|\boldsymbol{V}|^{\boldsymbol{2}})$$

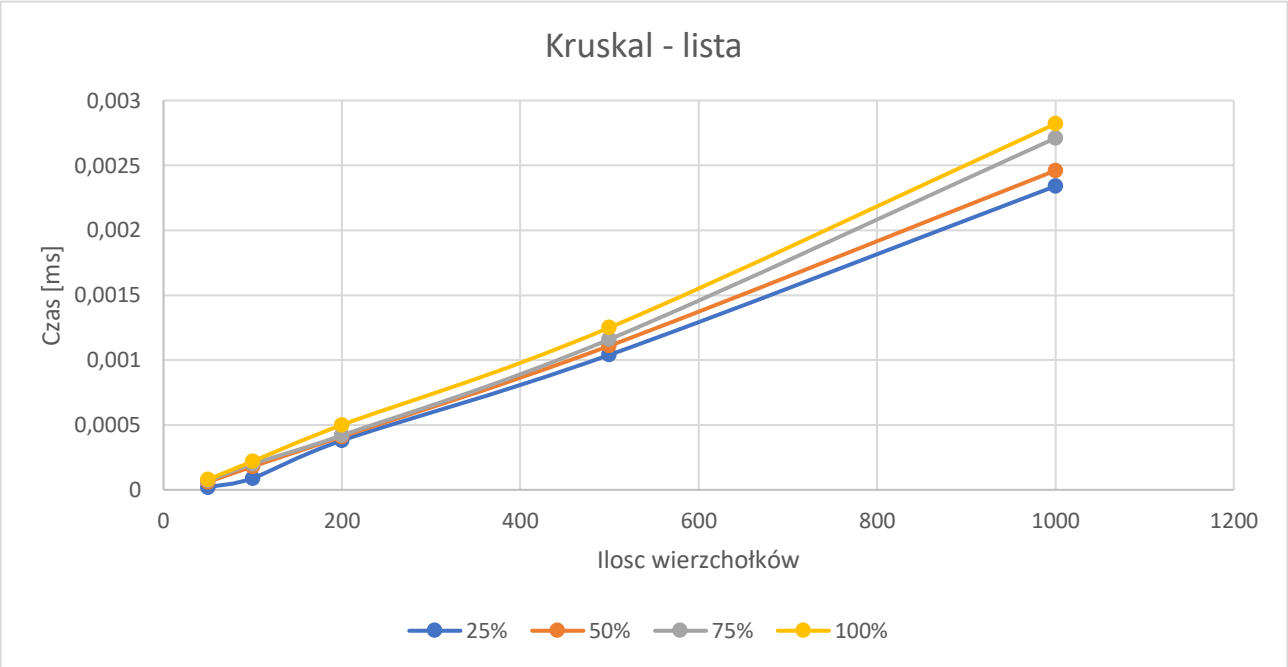
Gdzie: **E** = ilość krawędzi, **V** = ilość wierzchołków

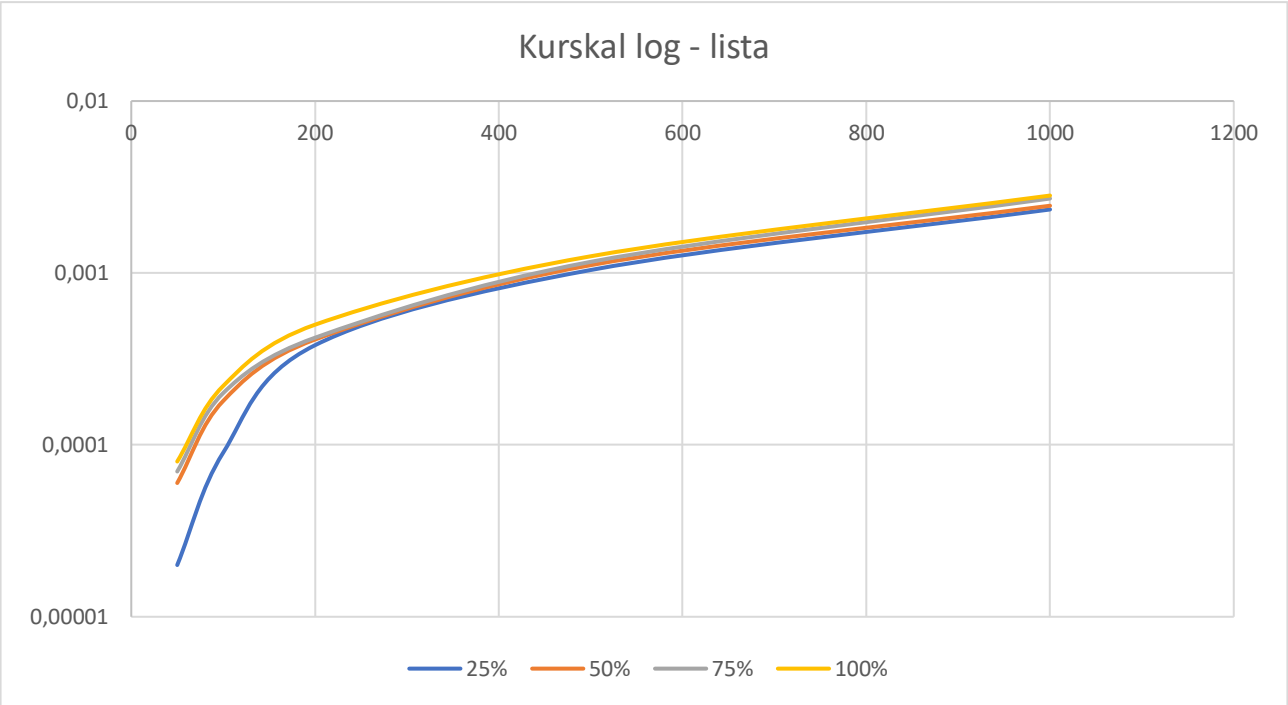
3. Przebieg badanych algorytmów

a) Algorytm Kruskala – lista sąsiedztwa

G\Il_wierz	50	100	200	500	1000
25%	0,00002	0,00009	0,00038	0,00104	0,00234
50%	0,00006	0,00018	0,00041	0,00111	0,00246
75%	0,00007	0,0002	0,00042	0,00116	0,00271
100%	0,00008	0,00022	0,0005	0,00125	0,00282

Tab. 1: Wyniki algorytmu dla listy sąsiedztwa

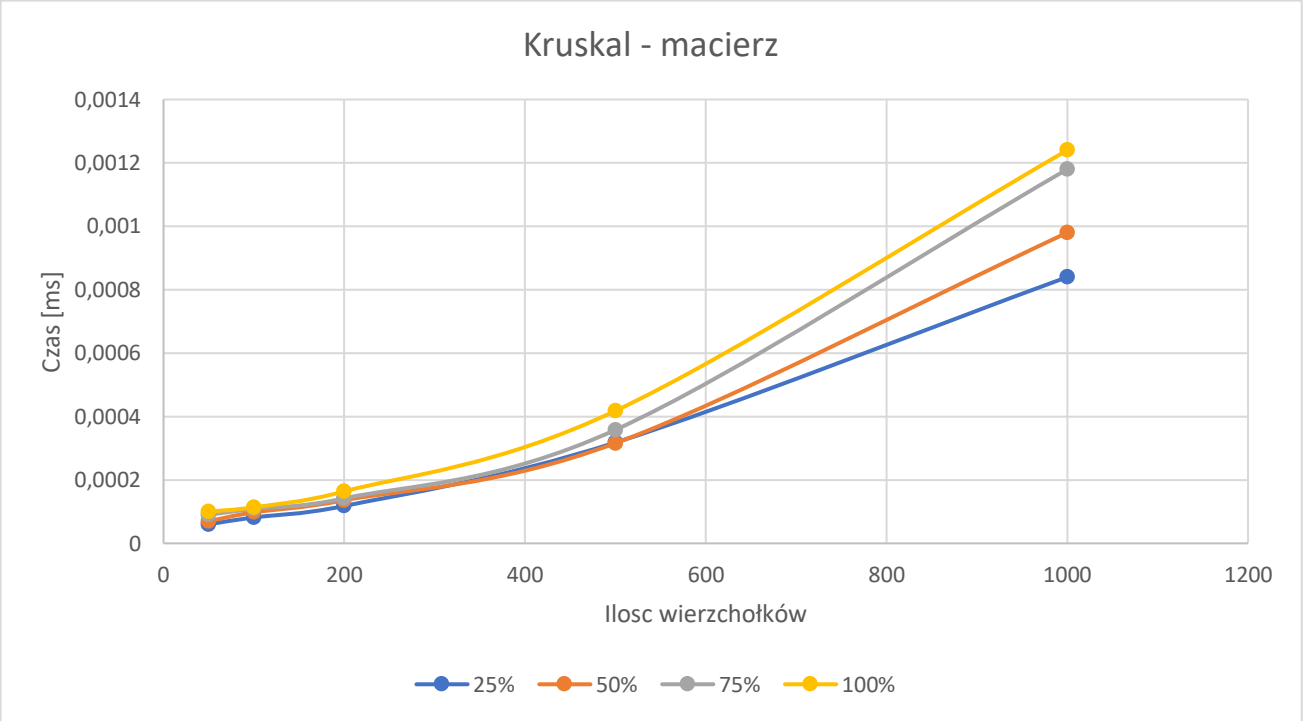


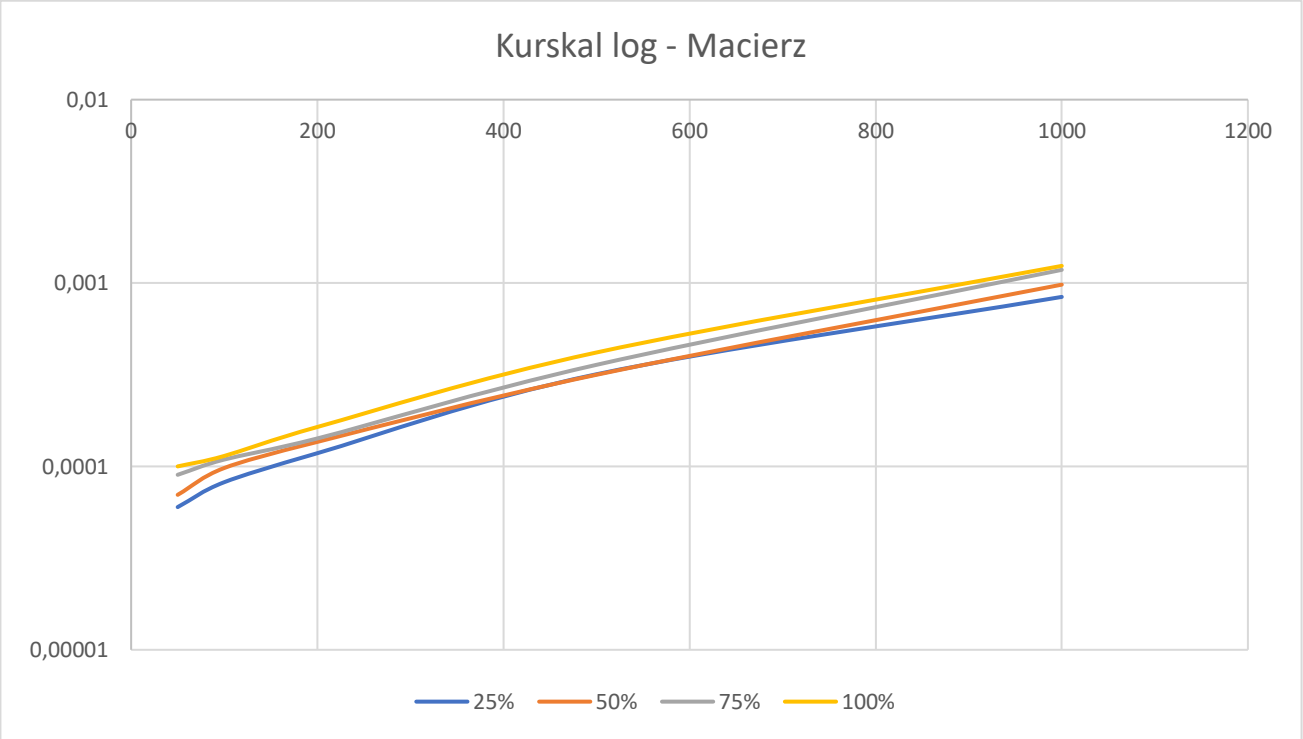


b) Algorytm Kruskala – macierz sąsiedztwa

G\Il_wierz	50	100	200	500	1000
25%	0,00006	0,000098	0,000108	0,000318	0,00084
50%	0,00007	0,000103	0,000136	0,000316	0,00098
75%	0,00009	0,000109	0,000142	0,000358	0,00118
100%	0,0001	0,000114	0,000164	0,000418	0,00124

Tab. 2: Wyniki algorytmu dla macierzy sąsiedztwa

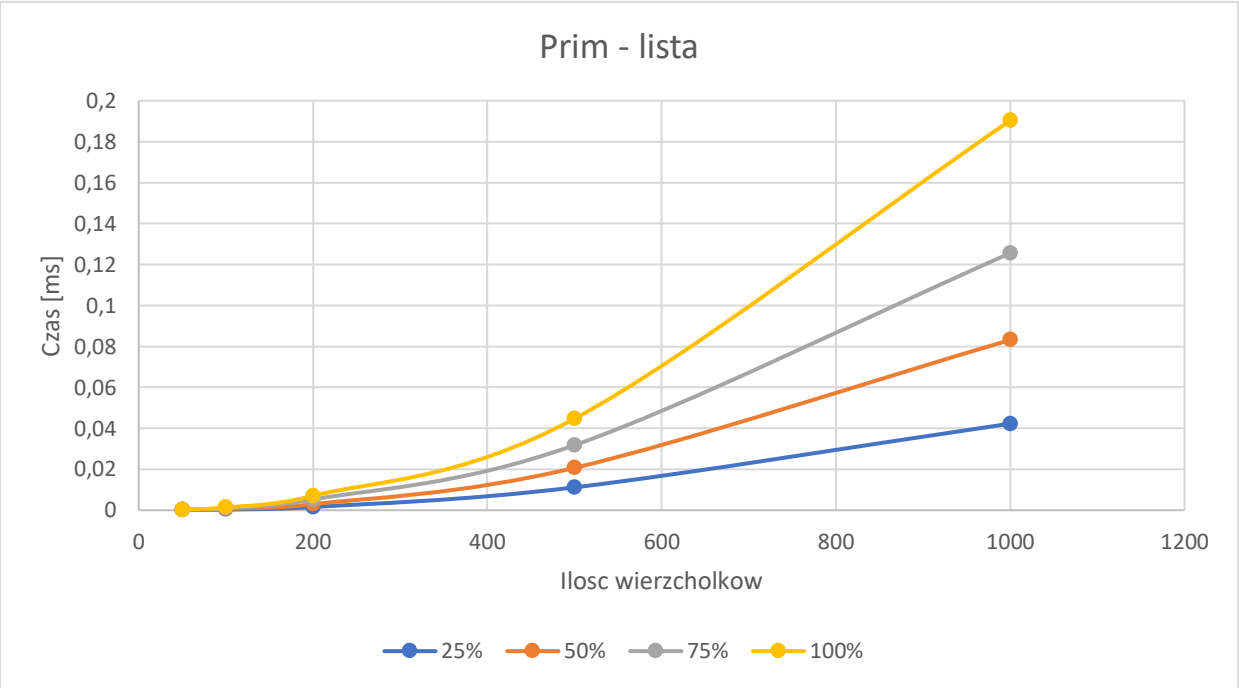


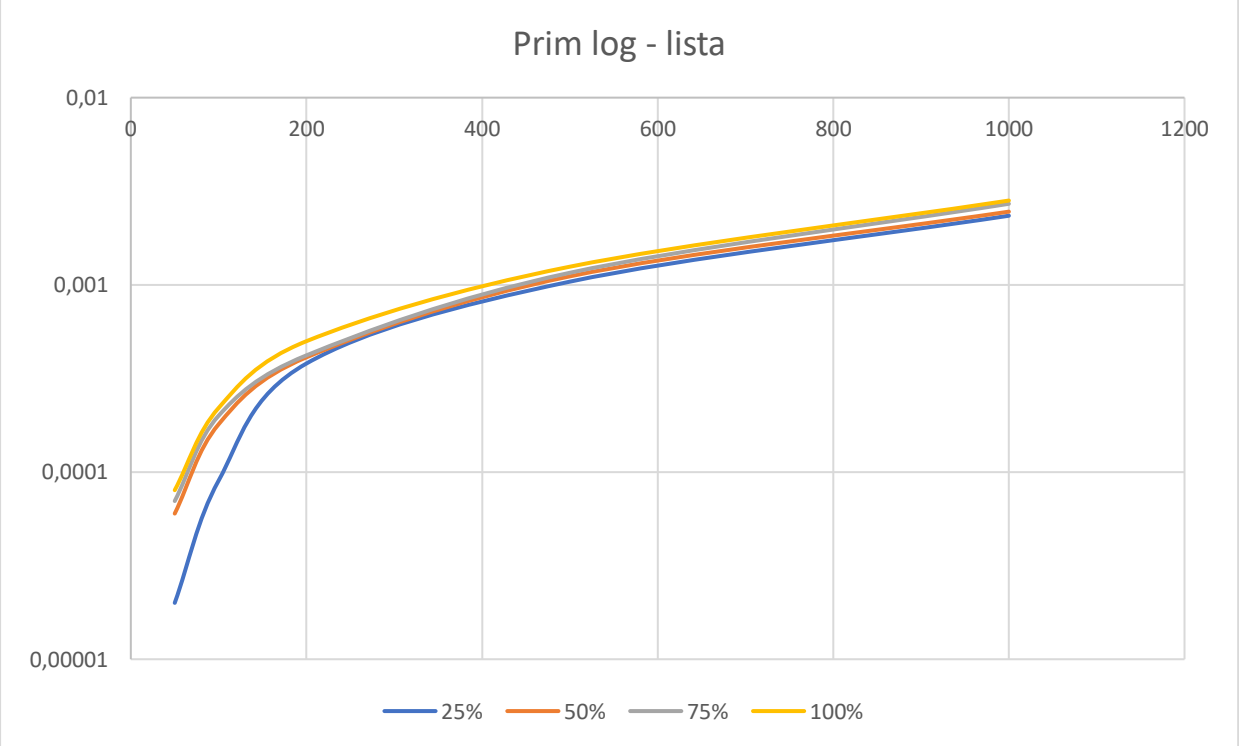


c) Algorytm Prima – lista sąsiedztwa

G\Il_wierz	50	100	200	500	1000
25%	0,000098	0,000352	0,001558	0,01116	0,04227
50%	0,000186	0,00071	0,002938	0,02069	0,08318
75%	0,000202	0,000958	0,00525	0,03176	0,12556
100%	0,00032	0,001396	0,00705	0,04468	0,19036

Tab. 3: Wyniki algorytmu dla listy sąsiedztwa





4. Podsumowanie

W programie pojawił się problem w zaimplementowaniem algorytmu Prima jako macierz sąsiedztwa. Jednak porównując te dwa algorytmy, szybszy powinien być ten na liście, ponieważ na macierzy osiąga on złożoność kwadratową, jednak nie udało mi się go napisać.

Po zaimplementowaniu algorytmu Kruskala na liście można zauważyć, że jest on szybszy pod wszelkimi względami od algorytmu Prima, zarówno dla małych, jak i dużych grafów. Czas wykonywania się tego algorytmu jest krótszy o ponad połowę, jest to pewnie spowodowane implementacją, a także założeniem algorytmu. Pierwszy – Kruskala – szuka najmniejszej krawędzi w grafie i dodaje do drzewa, drugi zaś przeszukuje wszystkie krawędzie prowadzące z wierzchołka, dlatego czasy są dłuższe.

Po utworzeniu wykresów można zauważyć, że wykres algorytmu Kruskala na macierzy nie wygląda na logarytmiczny, może to być spowodowane implementacją. Jednak jego czasy są zbliżone do listowej implementacji, a nawet w przypadku dużych grafów szybsze. Punkt, gdzie ilość wierzchołków wynosi 500 jest szczególnym, gdyż występuje nagły spadek szybkości algorytmu.

Wyniki algorytmów zaimplementowanych na liście na wykresie logarytmicznym przypominają wykres logarytmiczny, tak jak wynosi ich złożoność.

5. Bibliografia

[1] http://www.cplusplus.com/doc/tutorial/dynamic/
[2] https://www.softwaretestinghelp.com/graph-implementation-cpp/
[3] https://pl.wikipedia.org/wiki/Algorytm_Kruskala
[4] https://pl.wikipedia.org/wiki/Algorytm_Prima
[5] https://eduinf.waw.pl/inf/alg/001_search/0141.php
[6] http://lukasz.jelen.staff.iiar.pwr.edu.pl/styled-2/page-2/index.php