

## Algorytmy i złożoność

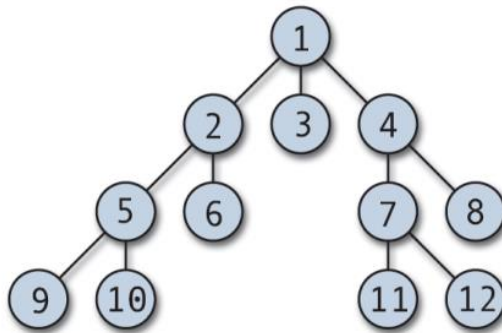
### Projekt 3

#### Nr.tematu 25 (BFS)

Dmytro Kruhlov 48720

#### Opis algorytmu

Przeszukiwanie wszerz (ang. breadth-first search, BFS) – jeden z najprostszych algorytmów przeszukiwania grafu. Przechodzenie grafu rozpoczyna się od zadanego wierzchołka  $s$  i polega na odwiedzeniu wszystkich osiągalnych z niego wierzchołków. Wynikiem działania algorytmu jest drzewo przeszukiwania wszerz o korzeniu w  $s$ , zawierające wszystkie wierzchołki osiągalne z  $s$ . Do każdego z tych wierzchołków prowadzi dokładnie jedna ścieżka z  $s$ , która jest jednocześnie najkrótszą ścieżką w grafie wejściowym. Algorytm działa prawidłowo zarówno dla grafów skierowanych jak i nieskierowanych.



#### Złożoność

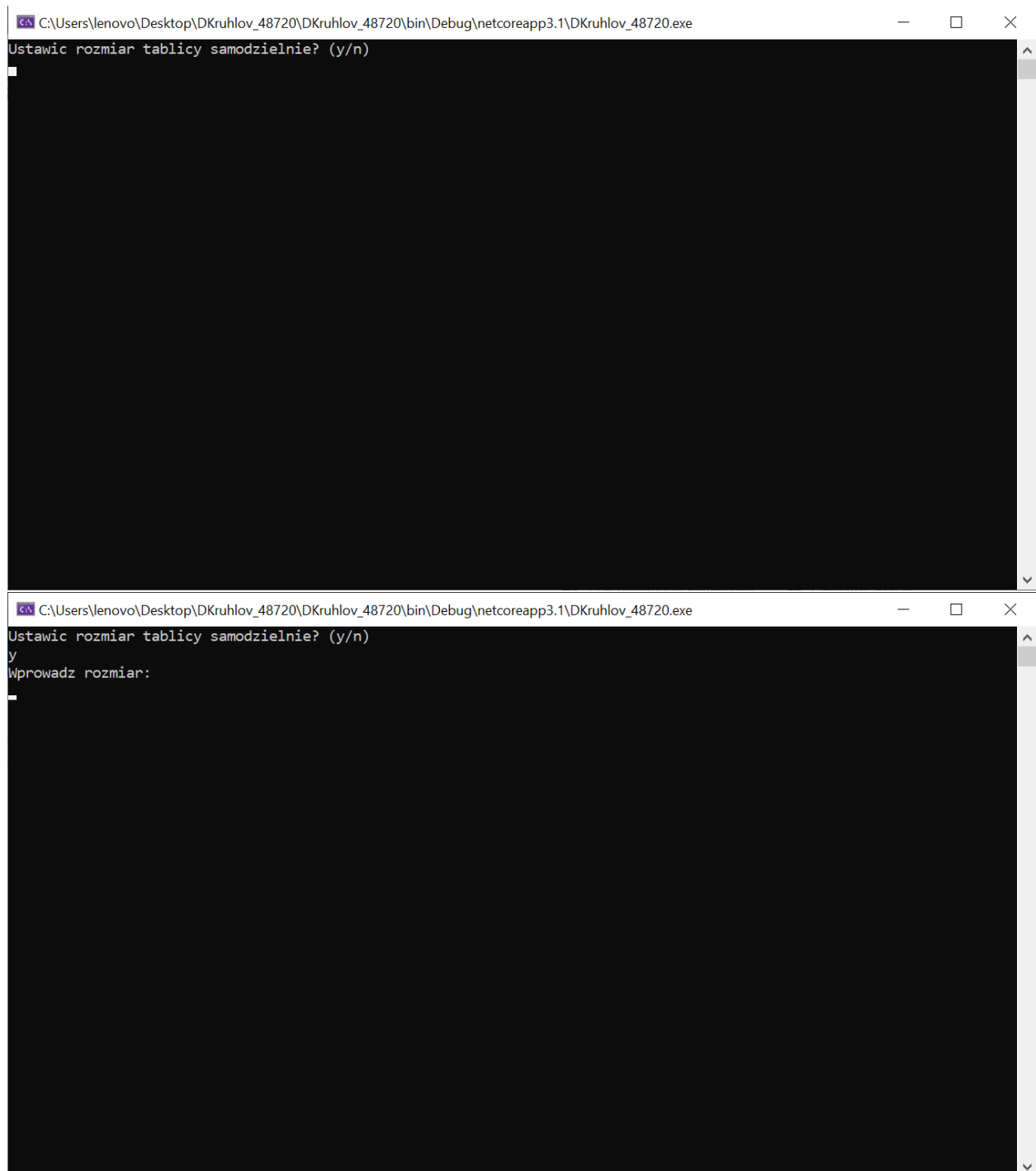
Dla każdej krawędzi i każdego wierzchołka algorytm wykonuje stałą liczbę działań, stąd złożoność czasowa to  $O(V + E)$ .

Maksymalna liczba wierzchołków przechowywanych jednocześnie w kolejce to  $V$ , czyli maksymalna ilość używanej pamięci to  $O(V)$ .

#### Instrukcja korzystania z programu

- I. Otwórz program
- II. Wybrać, czy chcesz ustawić rozmiar tablicy samodzielnie (y – tak, n – nie). Jeśli nie, sam program wybierze rozmiar tablicy.
- III. Program tworzy tablicę i wypełnia ją liczbami 0 i 1 (macierz sąsiedztwa).
- IV. Program wyświetla macierz sąsiedztwa.
- V. Program opisuje kroki, które podjął, aby przejść z węzła START do pozostałych węzłów.
- VI. Program wyświetla, ile czasu i pamięci poświęcono na proces.
- VII. Program pyta, czy chcesz zacząć od nowa (y/n).

## Przykład działania programu



```
C:\Users\lenovo\Desktop\DKruhlov_48720\DKruhlov_48720\bin\Debug\netcoreapp3.1\DKruhlov_48720.exe
Ustawic rozmiar tablicy samodzielnie? (y/n)
█

C:\Users\lenovo\Desktop\DKruhlov_48720\DKruhlov_48720\bin\Debug\netcoreapp3.1\DKruhlov_48720.exe
Ustawic rozmiar tablicy samodzielnie? (y/n)
y
Wprowadz rozmiar:
█
```

C:\Users\lenovo\Desktop\DKruhlov\_48720\DKruhlov\_48720\bin\Debug\netcoreapp3.1\DKruhlov\_48720.exe

```
(2) wierzcholek -->[ 0 0 0 0 0 1 0 1 1 1]
(3) wierzcholek -->[ 0 1 0 1 1 1 1 1 1 1]
(4) wierzcholek -->[ 1 1 0 0 1 0 1 1 1 1]
(5) wierzcholek -->[ 1 0 0 0 0 0 1 0 0 1]
(6) wierzcholek -->[ 0 0 0 1 0 0 0 0 0 1]
(7) wierzcholek -->[ 1 1 0 0 0 1 0 0 0 0]
(8) wierzcholek -->[ 0 0 1 1 0 0 1 0 0 0]
(9) wierzcholek -->[ 1 1 0 0 1 1 0 1 0 0]
(10) wierzcholek -->[ 1 1 1 1 1 1 0 1 1 0]
Rozpoczynamy przechodzenie od 10 wierzchołka
Przeszliśmy do węzła 10
Dodaliśmy do kolejki węzeł 1
Dodaliśmy do kolejki węzeł 2
Dodaliśmy do kolejki węzeł 3
Dodaliśmy do kolejki węzeł 4
Dodaliśmy do kolejki węzeł 5
Dodaliśmy do kolejki węzeł 6
Dodaliśmy do kolejki węzeł 8
Dodaliśmy do kolejki węzeł 9
Przeszliśmy do węzła 1
Przeszliśmy do węzła 2
Przeszliśmy do węzła 3
Dodaliśmy do kolejki węzeł 7
Przeszliśmy do węzła 4
Przeszliśmy do węzła 5
Przeszliśmy do węzła 6
Przeszliśmy do węzła 8
Przeszliśmy do węzła 9
Przeszliśmy do węzła 7
Czas: 00:00:00.02
Ilość pamięci bieżącego procesu (w bajtach): 16695296
Zakończyc program? (y/n)
```

## Wniosek

Podczas pracy nad projektem zapoznałem się bliżej z algorytmem BFS i jego implementacji w C#. Projekt nie był skomplikowany, więc nie powinno być błędów. Oceniam pracę na 5.