PAMSI - Projekt 1

GRAFY

Prowadzący	mgr inż. Marta Emirsajłow
Autor	Patryk Szydlik ind 248949 Wydział Elektroniki W4
Termin zajęć	Piątek 7:30 - 9:00
Data oddania sprawozdania	3 Maja 2020 r

Spis treści

1 Wprowadzenie

Celem projektu było zaimplementowanie i analiza efektywności wybranego algorytmu odnajdywania najkrótszej ścieżki w grafie skierowanym. Implementacje grafu należało wykonać na dwa sposoby: przy pomocy macierzy sąsiedztwa oraz przy pomocy listy sąsiedztwa. Testy efektywności zostały przeprowadzone dla 100 różnych grafów o następującej ilości krawędzi:

- 20
- 40
- 80
- 160
- 320

Wykonano eksperymenty dla następujących gęstości grafu:

- 25 %
- 50 %
- 75 %
- Graf pełny

2 Algorytm Dijkstry

2.1 Opis

Algorytm służy do wyznaczania najkrótszych ścieżek w grafie. Wyznacza najkrótsze ścieżki z jednego wierzchołka startowego do pozostałych wierzchołków. Algorytm wymaga, aby wagi krawędzi grafu nie były ujemne.
W trakcie wykonywania algorytmu dla każdego wierzchołka zostają wyznaczone dwie wartości: koszt dotarcia do tego wierzchołka oraz poprzedni wierzchołek na ścieżce. Na początku działania algorytmu dla wierzchołka startowego koszt dotarcia wynosi 0, a dla każdego innego wierzchołka nieskończoność (w implementacji użyłem maksymalnej wartości zmiennej intiger). Złożoność czasowa algorytmu przy implementacji w postaci listy sąsiedztwa wynosi $O((v + e)log(v))$natomiast w wypadku macierzy sąsiedztwa będzie to złożoność $O(v^2 + e)$. Wpływ na nią ma również sposób implementacji kolejki priorytetowej w algorytmie, implementacja z wykorzystaniem kopca pozwala zmniejszyć czas pracy algorytmu.

3 Przebieg eksperymentu

3.1 Argumenty wywołania programu

W programie zaimplementowane zostało uruchamianie programu z wykorzystaniem argumentów podanych przy wywołaniu programu. Program automatycznie analizuje użyte flagi argumentów , a następnie zgodnie z wybranym ustawieniem realizuje testy. Poniżej spis flag do uruchamiania programu:

- Parametr "-g" włącza generowanie grafu do testu algorytmu. Wraz z tą opcją należy podać, oddzielone białym znakiem, jeszcze 4 zmienne:
  - ilość wierzchołków generowanego grafu
  - numer wierzchołka startowego

  - procent wypełnienia grafu
  - nazwę pliku do zapisu wygenerowanego grafu
- Parametr "-i" pozwala wczytać graf z pliku. Należy podać dodatkowo nazwę pliku do wczytania.
- Parametr "-r" pozwala ustalić ilość powtórzeń testu. Należy podać liczbę powtórzeń.
- Parametr "-o" pozwala ustawić końcówkę nazwy plików wynikowych ze ścieżką. Nazwy posiadają zawsze początek z kodem określającym rodzaj implementacji oraz numer powtórzenia. Końcówkę nazwy pliku należy podać jako dodatkowy argument do ej opcji.
- Parametr "-t" pozwala wybrać typ implementacji badanych grafów. Wartość 0 bada tylko grafy na bazie macierzy, wartość 1 bada grafy na bazie listy, wartość 2 bada oba rodzaje grafów.

3.2 Skrypt do przeprowadzania testów

Stworzony został skrypt "runTest.sh" do automatycznego wykonywania testów algorytmu Dijkstry. Skrypt uruchamia wielokrotnie program zmieniając argumenty wywołania oraz przekierowuje dane wynikowe do odpowiednich plików w katalogach /log/ oraz /routes/. Wewnątrz skryptu można zmienić wartości parametrów aby edytować przebieg testów:

- IS_RAW - pozwala wybrać format wyświetlania ścieżki między wierzchołkami (0- oznacza opis słowny , 1 - oznacza surowy wygląd)
- GRAPH_TYPE - pozwala wybrać typ badanego grafu (parametr -t)
- STARTING_VERTEX - pozwala wybrać startowy wierzchołek
- REPETITIONS - pozwala wybrać ilość powtórzeń dla każdego przypadku

3.3 Makefile

Projekt posiada plik makefile umożliwiający kompilace przy pomocy polecenia "make build" , pośród pozostałych komend "make run" buduje i uruchamia program, "make clean" usuwa poprzednio skompilowany kod, "make cleanlog" usuwa wszystkie logi oraz ścieżki route.

4 Wyniki

Poniżej zaprezentowane są wyniki testów. Na tabelach oraz wykresach czas podany jest w ms i jest on wartością uśrednioną dla jednego przejścia algorytmu ze 100 pomiarów.

4.1 Tabela pomiarowa

TYPE	MATRIX			
DENSITY / VERTICES	25,00%	50,00%	75,00%	100,00%
20	3,559	4,078	5,163	7,235
40	9,315	17,021	51,352	55,373
80	50,816	122,006	196,436	280,556
160	218,455	578,837	1306,500	1971,960
320	1157,470	4045,900	8548,260	14871,000
640	8743,110	29486,600	64840,000	105596,000

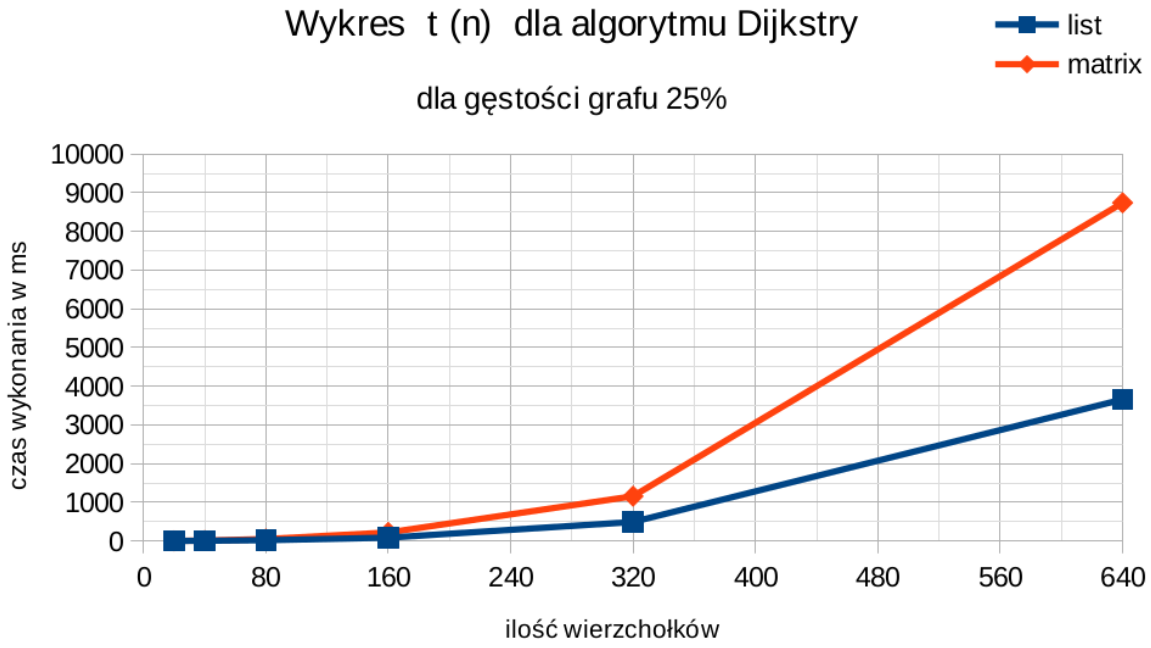
Rysunek 1: Tabela pomiaru algorytmu Dijkstry na grafie opartym o macierz sąsiedztwa. Czas podany w ms

TYPE	LIST			
DENSITY / VERTICES	25,00%	50,00%	75,00%	100,00%
20	1,708	1,924	2,260	2,964
40	4,149	7,257	15,958	23,515
80	19,992	48,501	82,063	127,347
160	84,547	260,144	577,363	815,847
320	491,542	1613,070	3771,420	6638,890
640	3658,820	13106,300	30615,000	58306,400

Rysunek 2: Tabela pomiaru algorytmu Dijkstry na grafie opartym o liście sąsiedztwa. Czas podany w ms

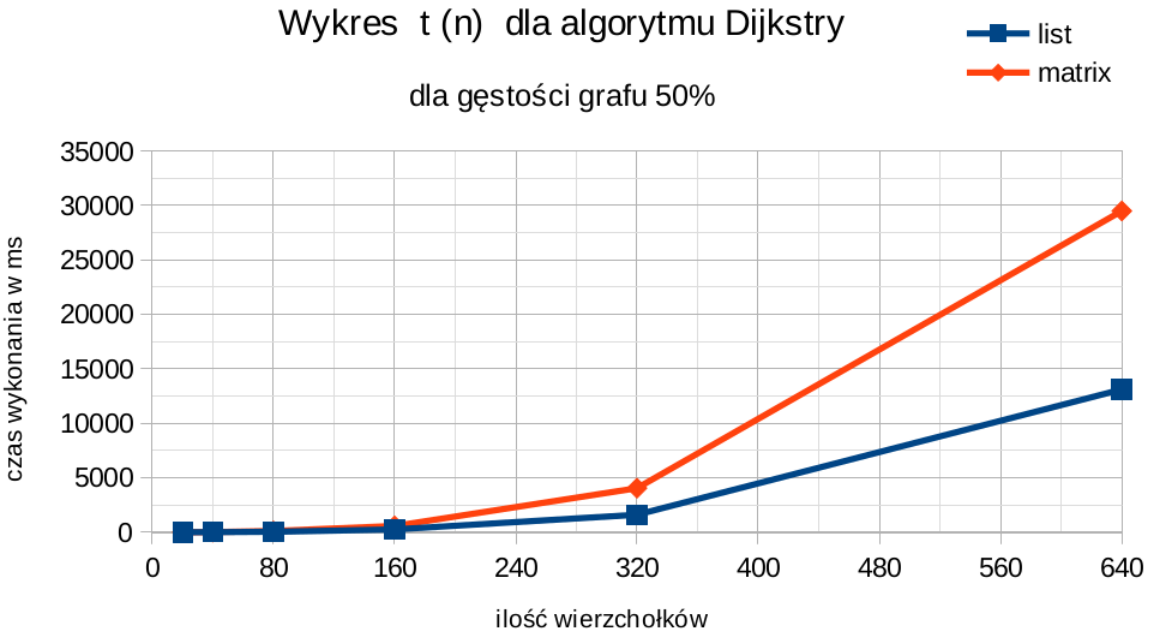
4.2 Wykresy w zależności od gęstości

4.2.1 Dla gęstości 25 %



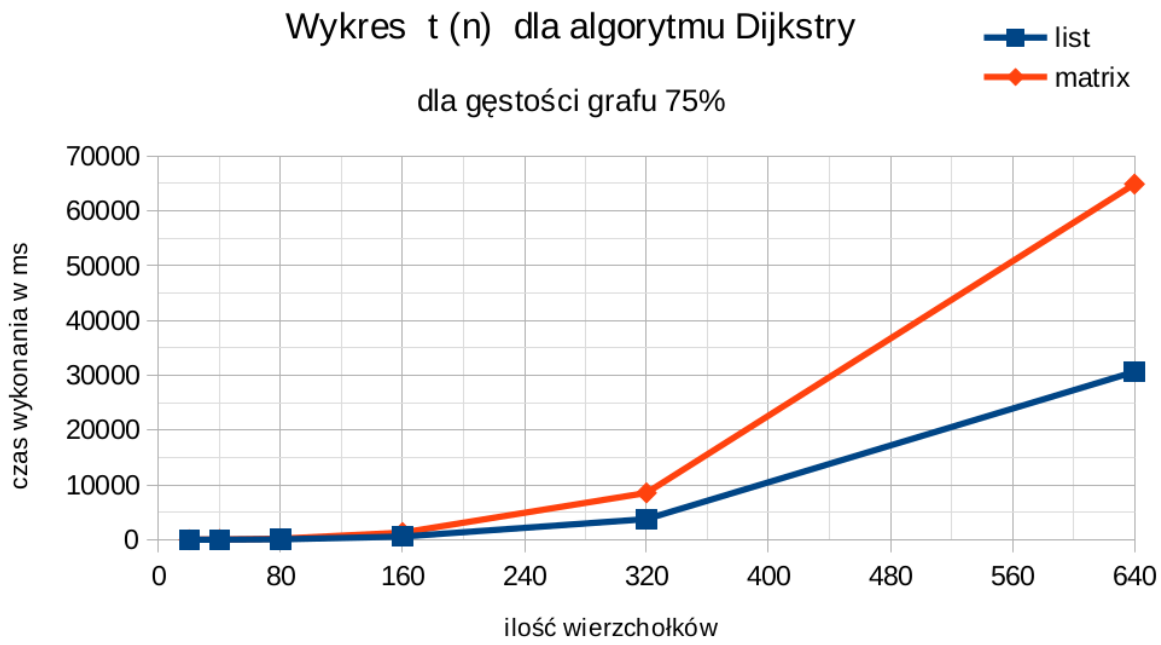
Rysunek 3: Wykres zależności czasu od ilości wierzchołków grafu o wypełnieniu 25%

4.2.2 Dla gęstości 50 %



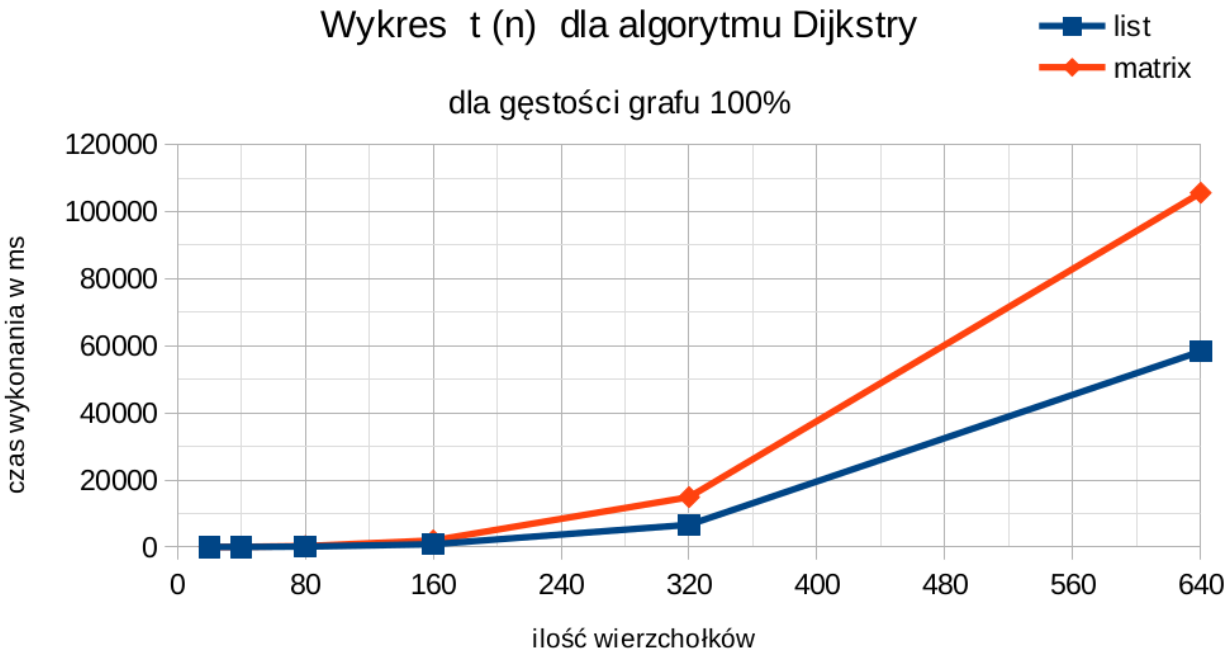
Rysunek 4: Wykres zależności czasu od ilości wierzchołków grafu o wypełnieniu 50%

4.2.3 Dla gęstości 75 %



Rysunek 5: Wykres zależności czasu od ilości wierzchołków grafu o wypełnieniu 75%

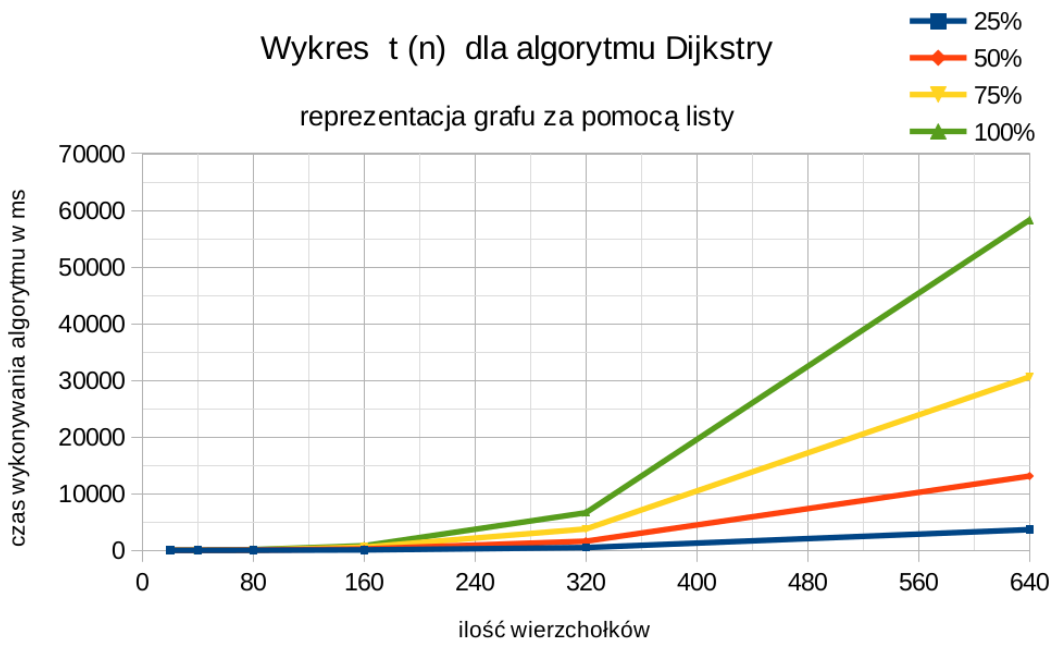
4.2.4 Dla gęstości 100 %



Rysunek 6: Wykres zależności czasu od ilości wierzchołków grafu o wypełnieniu 100%

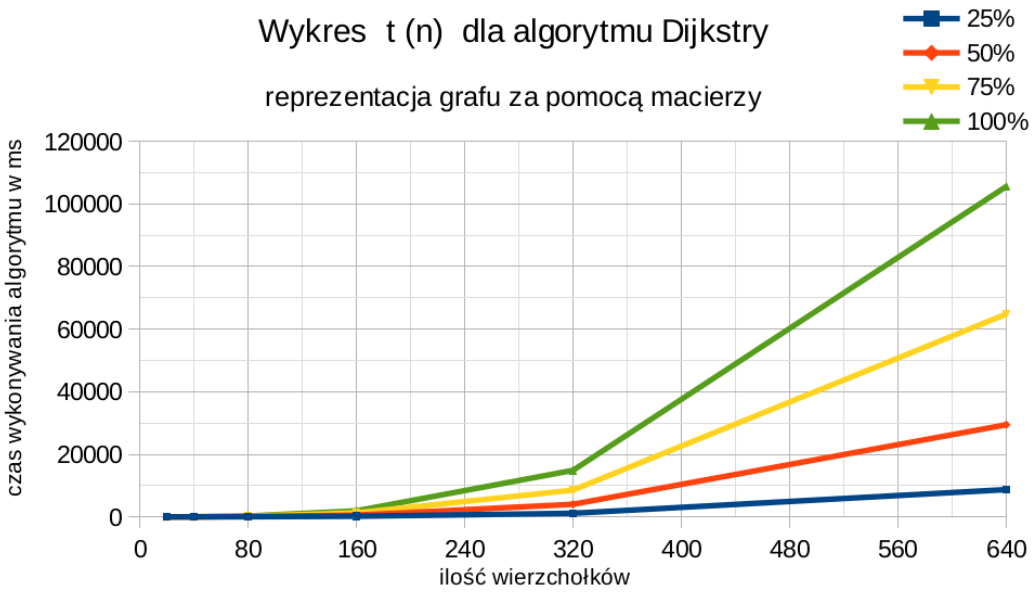
4.3 Wykresy w zależności od reprezentacji

4.3.1 Graf na podstawie listy



Rysunek 7: Wykres zależności czasu od ilości wierzchołków grafu bazowanego na liście

4.3.2 Graf na podstawie macierzy



Rysunek 8: Wykres zależności czasu od ilości wierzchołków grafu bazowanego na macierzy

5 Wnioski

Jak widać na podstawie wyników eksperymentu algorytm użyty na grafie zaimplementowanym na podstawie listy sąsiedztwa sprawdził się dużo lepiej niż na grafie wykorzystującym macierz sąsiedztwa. Spowodowane może być to alokowaniem nadmiarowej ilości struktur, które w przypadku macierzy wynoszą zawsze $v^2 + v$ natomiast w wypadku listy wynoszą $2 * v + e$. Powodem znacznie dłuższego czasu pracy algorytmu dla macierzy jest również złożoność metod zwracających listy krawędzi incydentnych, oraz przeszukujące zbiór krawędzi, które w wypadku macierzy muszą przejrzeć zawsze 2*v elementów w macierzy, natomiast w wypadku listy przeglądają one tylko tyle elementów ile krawędzi jest poszukiwanych.

Analizując wpływ gęstości grafów na czas pracy algorytmu możemy wyciągnąć wnioski na temat złożoności czasowej algorytmu. Ponieważ ilość wierzchołków jest stała, a ilość krawędzi skierowanych w grafie zależy od wierzchołków oraz gęstości i wynosi $v*(v-1)*dens$. W związku z tym podwojenie gęstości powoduje podwojenie ilości krawędzi, które są kwadratowo zależne od ilości wierzchołków. Dlatego też oczekiwana złożoność algorytmu dla listy sąsiedztwa wynosząca $O((v+e)log(v))$ przekształca się w $O((v + dens * (v^2 - v))log(v))$. Można zauważyć zależność w wynikach, iż przy zmianie gęstości z 25% na 50% czas działania wzrastał o współczynnik mniejszy niż 3.6 (dla największej ilości wierzchołków) podczas gdy zmiana gęstości z 50% na 100% powodowała wzrost o współczynnik 4.5. Podczas gdy dla implementacji grafu z użyciem macierzy, współczynnik wzrostu przy podwojeniu ilości krawędzi wynosił zawsze około 3.5, pomimo, że ogólny czas działania algorytmu dla macierzy był i tak wolniejszy. Potwierdza to, że złożonośc czasową dla implementacji macierzowej nie zależała od gęstości grafu, a głównie od ilości wierzchołków. Natomiast w wypadku listy sąsiedztwa gęstość grafu ma znaczący wpływ na złożoność czasową algorytmu.

6 Kod programu

Zdalne repozytorium z kodem programu dostępne jest pod linkiem:

https://bitbucket.org/patryk_sz/pamsi/src/master/Project_2/

7 źródła

Podczas wykonywania projektu poza materiałami wykładowcy wspierałem się nastepującymi stronami:

1. Teoria grafów eduinf.waw
2. szukanie najkrótszej ścieżki eduinf.waw
3. Encyklopedia algorytmów
4. Dijkstra wiki