Bartosz Durys 229869 229869@edu.p.lodz.pl
Szymon Klewicki 229911 229911@edu.p.lodz.pl

Zadanie 2 - Przeszukiwanie wszerz i w głąb

1. Cel

Celem zadania było stworzenie programu z algorytmami BFS i DFS w celu znalezienia połączeń między węzłami grafu. Do tego porównanie obu metod w zależności od sposobu reprezentacji struktury danych oraz zmierzyć czas działania algorytmów w zależności od liczby węzłów.

2. Opis implementacji

Przy implementacji skorzystaliśmy z języka Python. Do rozróżniania klas grafów skierowanych i nieskierowanych przyjęliśmy poniższą konwencję:
- **Dir*** - graf skierowany,
- **Und*** - graf nieskierowany.

I do reprezentacji grafu:
- ***List** - z użyciem tablicy,
- ***AdjacencyMatrix** - z macierzą sąsiedztwa,
- ***IncidenceMatrix** - z macierzą incydencji,
- ***AdjacencyList** - z listą sąsiedztw.

Wszystkie oferują te same funkcjonalności dzięki klasie bazowej **GraphRepr**. Umożliwiło to stworzenie klasy **GraphSolver**, która współgra z każdą zaimplementowaną reprezentację grafu.

Do stworzenia struktury danych możemy zarówno wczytać dane z odpowiednio przygotowanego pliku jak i wpisać je manualnie przez terminal.

3. Badania

Cel poszukiwań jest taki sam dla każdego badanego grafu - musimy znaleźć drogę od pierwszego do ostatniego węzła. Stworzyliśmy skrypt, który generuje pliki do wczytania z grafami o podanych charakterystykach:

1. Pierwszy generuje graf z 1000 wierzchołków połączonych kolejno ze sobą.
2. Drugi różni się od pierwszego liczbą wierzchołków - 2000.
3. W kolejnym, pierwszy węzeł ma 100 połączeń z kolejnymi węzłami. Te natomiast losowo dobierane są z następnymi 400. Ostatnie 499 również łączymy losowo z poprzednimi. Zatem między początkiem a końcem są dwa węzły pośredniczące, ale liczba sąsiadów na tej drodze jest spora.
4. Analogicznie do poprzedniego, ale mamy 2000 węzłów. Czyli patrząc na kolejne poziomy drzewa mamy: 1, 200, 800, 999.
5. Analogicznie do poprzedniego, ale dodajemy kolejną warstwę i mamy po kolei: 1, 200, 466, 667, 666.

W każdej reprezentacji grafu wykonujemy metody:

- **add_vertex()** - dodanie węzła,
- **add_edge(idx1, idx2)** - dodanie gałęzi między dwoma węzłami,
- **get_neighbors(idx)** - zwracanie sąsiadów danego węzła.

Różnice w przedstawianiu grafów powodują różnice w implementacji metod. Możemy się domyślać, że to przekłada się na różny stopień złożoności obliczeniowej. Sprawdzimy to podczas analizy wyników. Skupimy się tutaj na czasowej złożoności. Dlatego w badaniach zmierzymy następujące procesy:

- **add_vertices** - dodania wszystkich wierzchołków,
- **add_edges** - dodania wszystkich gałęzi,
- **bfs** - szukania połączenia algorytmem przeszukiwania w szerz,
- **dfs** - szukania połączenia algorytmem przeszukiwania wgłąb,
- **all** - sumaryczny czas przetwarzania.

Metodę **get_neighbors(idx)** używamy w obu algorytmach przeszukiwania. Podczas implementacji uzyskaliśmy taką samą kolejność operacji bez względu na wybrany sposób reprezentacji. Możemy więc czas procesu dla poszczególnych poszczególnych algorytmów utożsamić z czasową złożonością obliczeniową poruszonej wcześniej metody i porównać dla każdej reprezentacji grafu.

Warto dodać, że dokonujemy wszędzie 5 razy tych samych pomiarów, a potem uśredniamy czas przetwarzania grupując względem po kolei procesu, pliku i metod reprezentacji.

Dla estetyki wykresów zmieniliśmy nazwy klas odpowiednio:

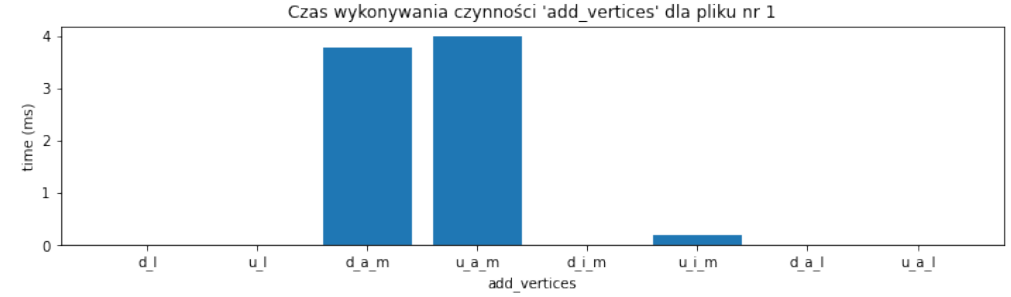
Tabela 1. Aliasy do nazw klas reprezentacji grafu

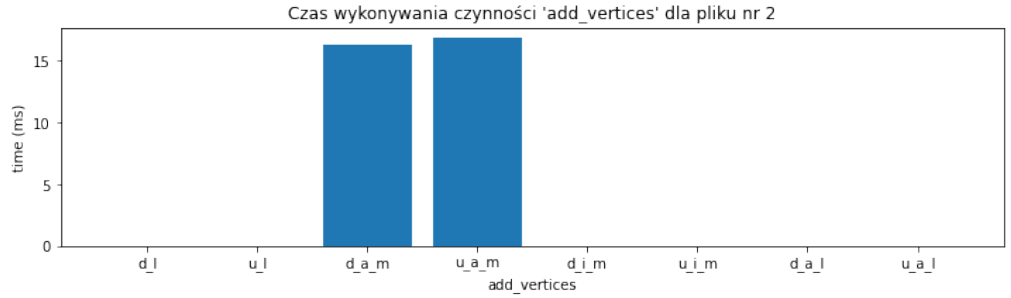
Nazwa klasy	**Alias**
DirList	d_l
UndList	u_l
DirAdjacencyMatrix	d_a_m
UndAdjacencyMatrix	u_a_m
DirIncidenceMatrix	d_i_m
UndIncidenceMatrix	u_i_m
DirAdjacencyList	d_a_l
UndAdjacencyList	u_a_l

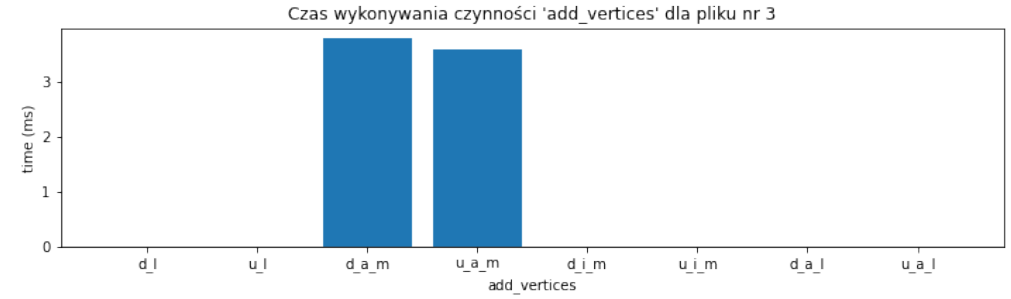
4. Wyniki

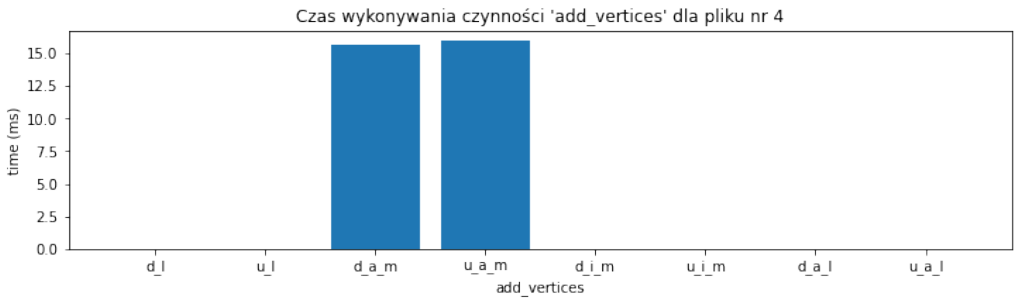
Przedstawimy tutaj wyniki naszych badań w postaci wykresów.

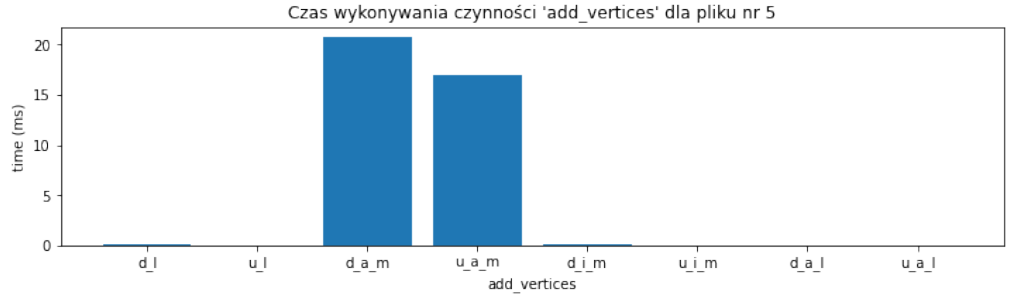
Wykresy 1.1.-1.5. 'add_vertices'



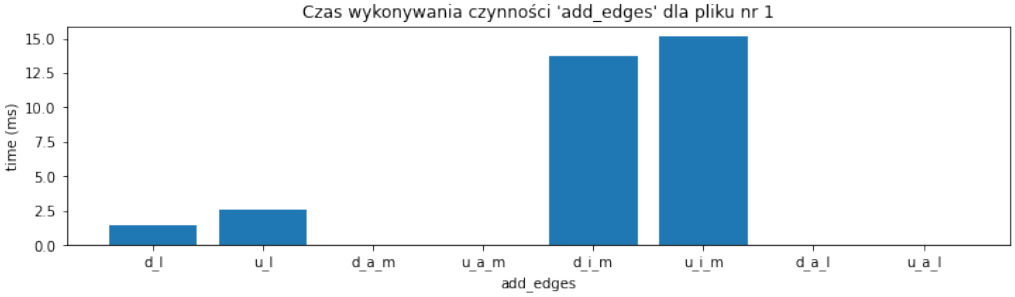


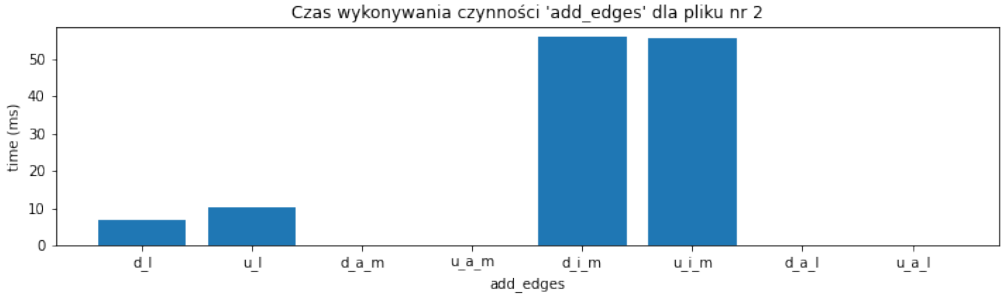


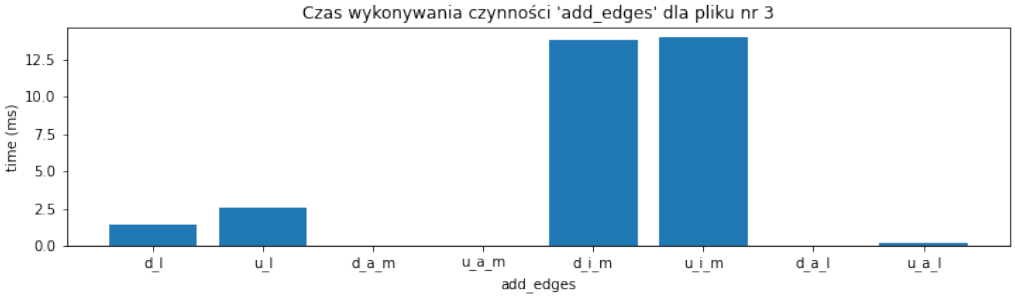


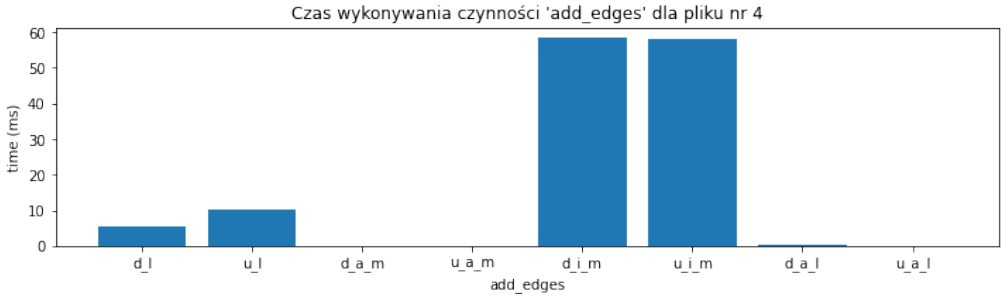


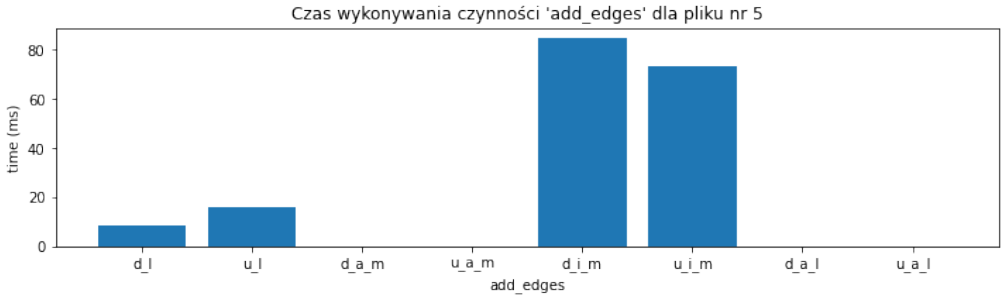
Wykresy 2.1.-2.5. ’add_edges’



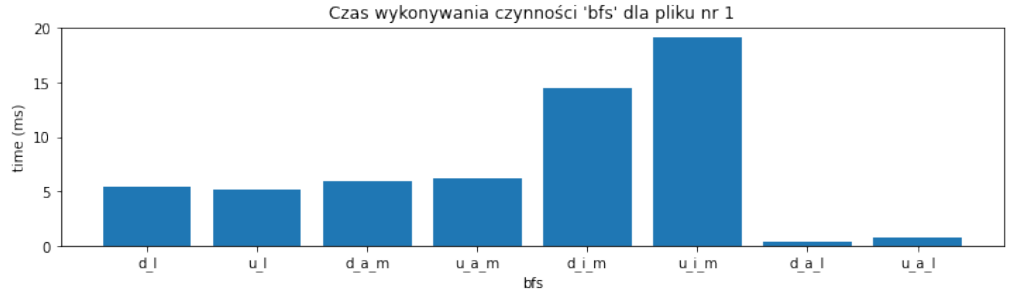


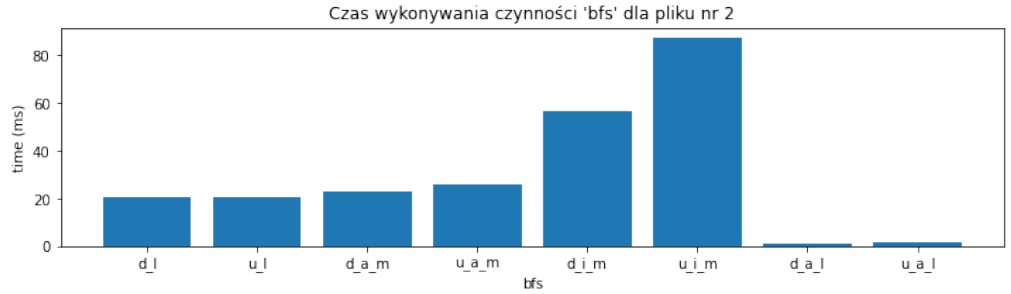


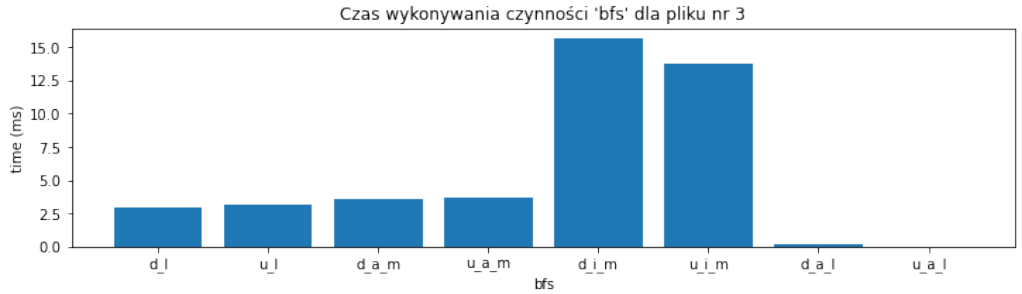


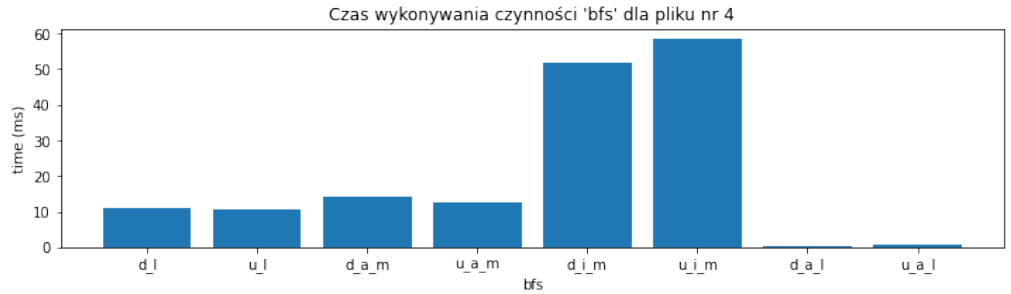


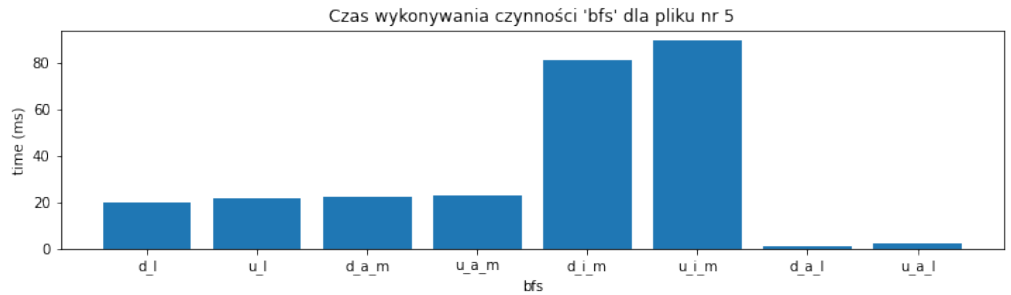
Wykresy 3.1.-3.5. ’bfs’



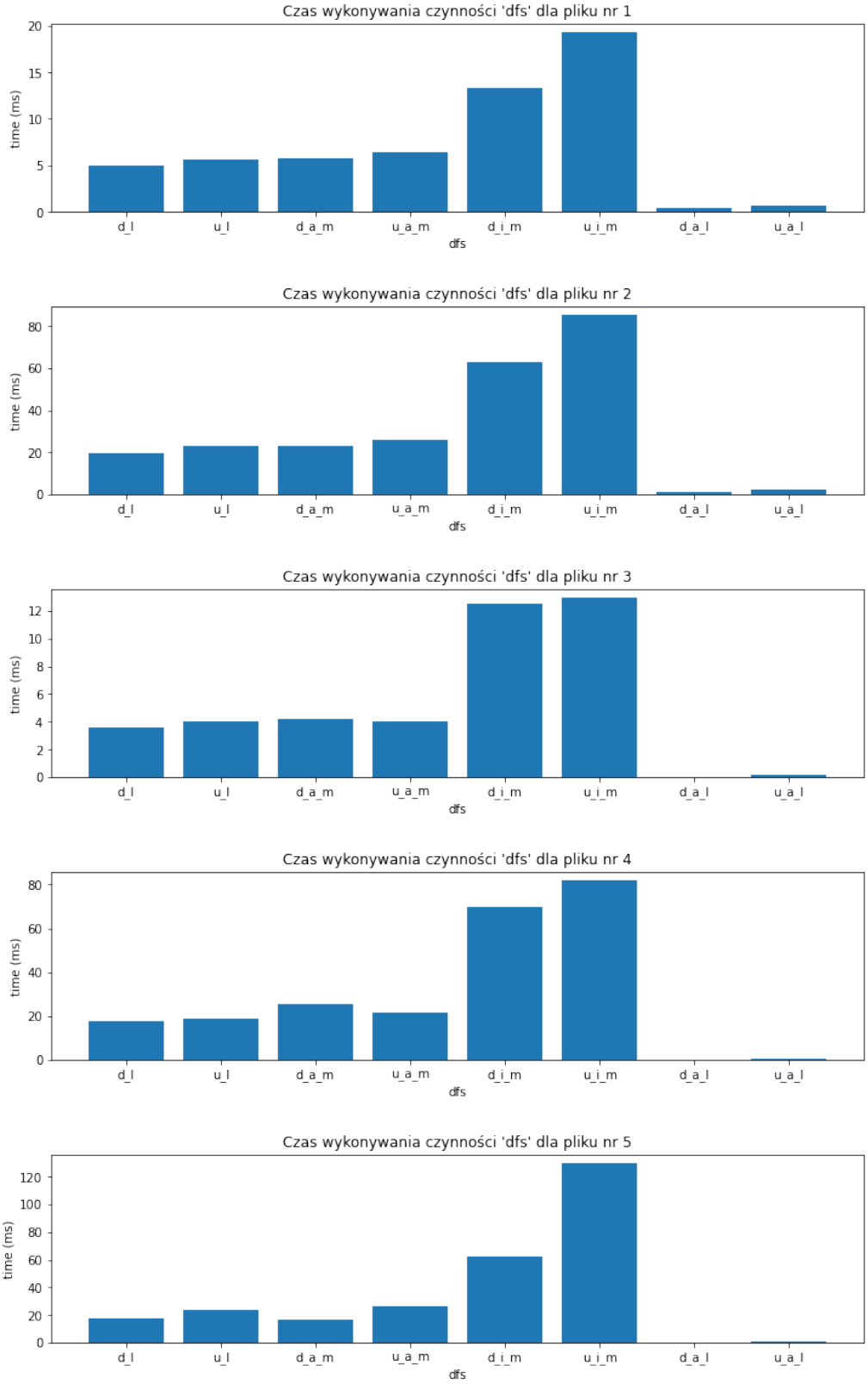




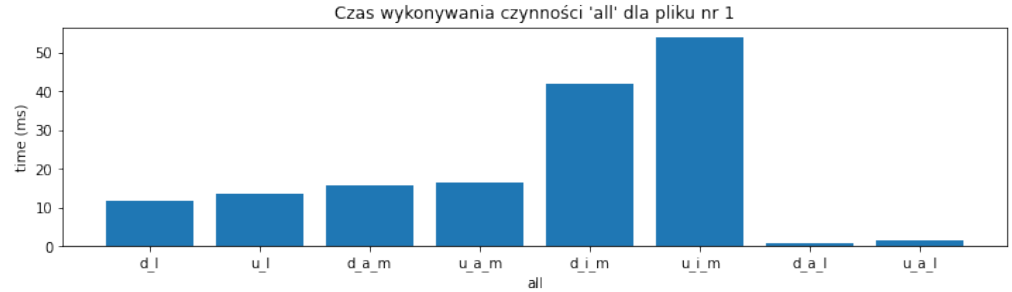


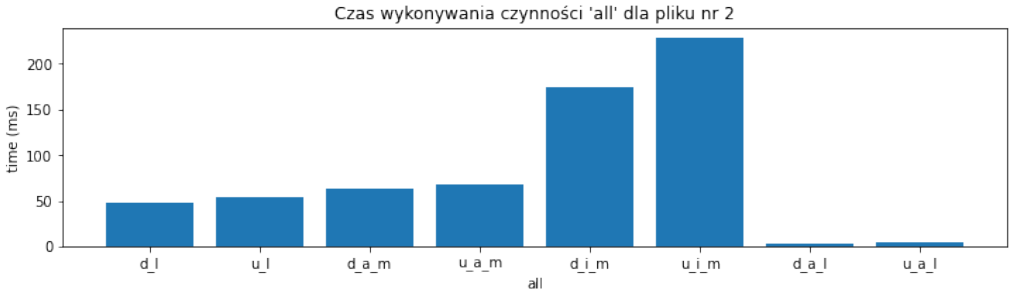


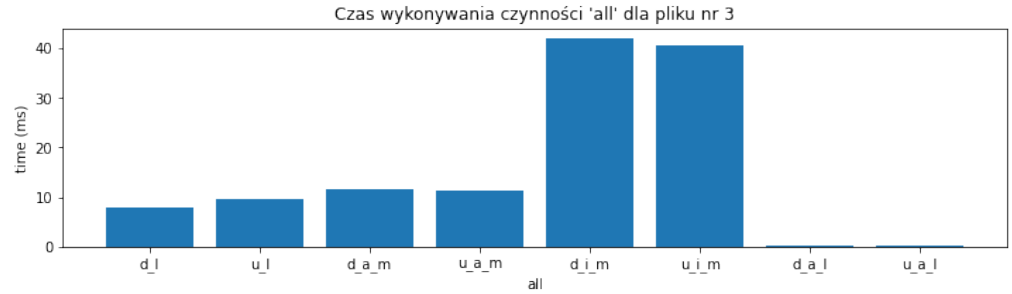
Wykresy 4.1.-4.5. 'dfs'

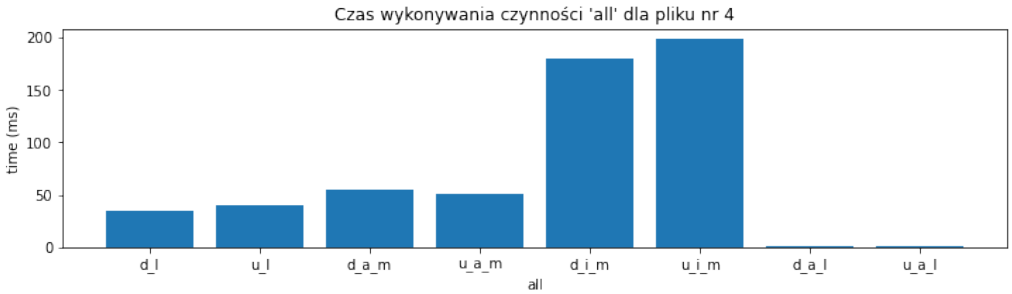


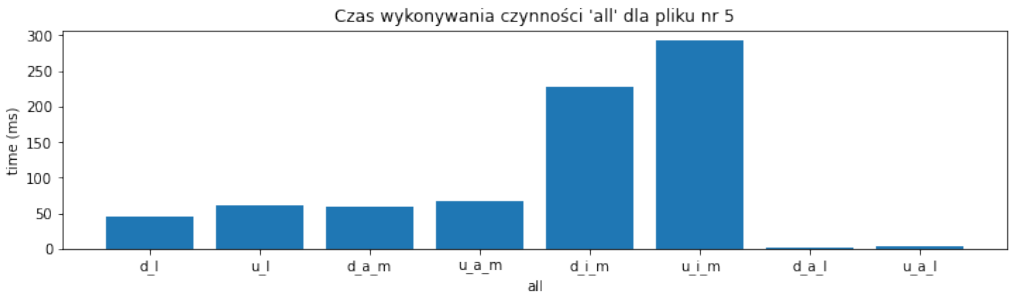
Wykresy 5.1.-5.5. ’all’











5. Analiza

Pierwszy zbiór wykresów bada proces 'add_vertices'. Bez żadnych wątpliwości, implementacja z wykorzystaniem macierzy sąsiedztwa potrzebuje tutaj więcej czasu od innych reprezentacji. W plikach nr 2, 4 i 5 wykorzystujemy dwa razy więcej węzłów niż w numerach 1 i 3. Odczytując wartości z wykresów możemy zauważyć zależność podobną do kwadratowej. Struktura kodu programu potwierdza wyniki tych badań

Następny proces to 'add_edges'. Tym razem najwięcej czasu konsumuje sposób z macierzą incydencji. Tutaj nie ma zależności kwadratowej. Możemy stwierdzić również na podstawie implementacji, że wraz ze wzrostem liczby wierzchołków zwiększa się liczba iteracji, żeby mogła powstać nową gałąź. Z drugiej strony widzimy też reprezentację powstałą za pomocą tablicy. Dodatkowo, dla grafu nieskierowanego widzimy ciągłą przewagę w mierzonej wartości. Po weryfikacji z kodem możemy zauważyć, że realnym powodem takiego stanu jest sprawdzanie czy nie ma już takiego połączenia w liście. Przy grafie nieskierowanym sprawdzamy drugi raz ze zmienioną kolejnością elementów w tablicy. Złożoność rośnie zatem wraz z liczbą gałęzi.

Pora na algorytm przeszukiwania w szerz (wykresy 3.1.-3.5.). Macierz incydencji pozostaje najwyżej. W tym akapicie już możemy powiedzieć, która metoda wygląda na lepszą od innych. Jest to implementacja z listą sąsiedztwa. Wygląda jakby wykonywała się z czasem stałym. Pozostałe dwie metody wyglądają podobnie pod względem złożoności czasowej.

Płynnie przechodzimy do algorytmu DFS. Pozycje względne zostały, upraszczając, bez zmian. Możemy zauważyć, że czas przetwarzania pierwszych dwóch plików jest identyczny z poprzednim algorytmem. Bardzo one się nie różnią jeżeli mają cały czas tylko jednego sąsiada do wyboru. Zestawiając pliki 3 i 4 w obu algorytmach, spodziewana jest gorsza efektywność DFS w wyniku szukania zbyt głęboko w miejscach z daleka od celu.

Ostatni zestaw wykresów pokazuje wszystko co już wiemy o poszczególnych reprezentacjach jeżeli chodzi o ogólną złożoność czasową.

6. Wnioski

Implementacja oraz badanie zagadnień dała parę wniosków:

1. Dla badanych metod najbardziej opłacalne okazało się korzystanie z list sąsiedztw, a najmniej z macierzy incydencji w ujęciu ogólnym.
2. Pod względem dodawania nowych wierzchołków, macierz sąsiedztwa okazała się znacznie wolniejsza.
3. Dodatkowe sprawdzanie warunków może okazać się obciążające tak jak w przypadku reprezentacji opartej na tablicy przy metodzie 'add_edges'.
4. Liczba węzłów może zmienić czas przetwarzania w różnym stopniu.
5. Liczba gałęzi zwiększa rozmiar struktury danych grafu gdzie wykorzystuje się tablicę lub macierz incydencji. Dla tych metod wartość ta może zwiększyć złożoność algorytmu.
6. Algorytm DFS posiada ryzyko tracenia czasu podczas głębszych poszukiwań gdzie cel jest w innej gałęzi.