Zadanie laboratoryjne: raport

Michał Rusinek

1 grudnia 2024

Spis treści

Wprowadzenie

Celem zadania było zrealizowanie i omówienie analizy grafów poprzez implementację programów komputerowych oraz przedstawienie sprawozdania zawierającego opis użytych metod.

Wybranym wariantem zadania było stworzenie programu rozwiązującego polecenie dla grafów skierowanych.

Zespół składa się z jednej osoby.

1 Opis techniczny programu

Program został napisany w języku Python z wykorzystaniem biblioteki NetworkX do analizy grafów. Pozwala ona na łatwe tworzenie, manipulację i analizę grafów, a także zawiera wiele algorytmów do analizy sieci.

Instrukcja kompilacji i uruchomienia programu znajduje się w pliku `instrukcja_uruchomienia.pdf`.

2 Opis danych wejściowych

Dane wejściowe składają się z pliku tekstowego zawierającego opisy jednego lub więcej grafów. Każdy opis grafu rozpoczyna się od liczby wierzchołków, a następnie zawiera wiersze macierzy sąsiedztwa, które reprezentują krawędzie między wierzchołkami. Każdy graf w pliku jest oddzielony pustą linią. Pierwsza linia pliku zawiera liczbę grafów, ale w przypadku kiedy plik zawiera jeden graf w pierwszej linii znajduje się od razu liczba wierzchołków.

3 Opis algorytmów

3.1 Rozmiar grafu

Definicja

Rozmiar grafu jest definiowany jako liczba wszystkich krawędzi w grafie. Dla macierzy sąsiedztwa grafu skierowanego, rozmiar grafu obliczamy jako sumę wszystkich elementów macierzy. [5]

Algorytm

Jako iż zgodnie z założeniami rozpatrywane są wyłącznie grafy skierowane - przejście przez każdy wiersz macierzy i sumowanie wartości.

Kod rozwiązania

```
def calculate_size(graph):
    return sum([sum(row) for row in graph])
```

Złożoność obliczeniowa

$O(V^2)$, gdzie V to liczba wierzchołków w grafie.

3.2 Metryki w zbiorze wszystkich grafów

3.2.1 Gęstość grafu

Definicja

Gęstość grafu to stosunek liczby krawędzi w grafie do liczby wszystkich możliwych krawędzi. Dla grafu skierowanego, liczba wszystkich możliwych krawędzi wynosi V^2, gdzie V to liczba wierzchołków. [1]

Algorytm

Algorytm oblicza gęstość grafu jako stosunek rozmiaru grafu do liczby wszystkich możliwych krawędzi.

Kod rozwiązania

```
def density(graph):
    g = nx.DiGraph()
    for row_idx, row in enumerate(graph):
        for col_idx, cell in enumerate(row):
            if cell == 1:
                g.add_edge(row_idx, col_idx)
    return nx.density(g)
```

Złożoność obliczeniowa

$O(V^2)$, gdzie V to liczba wierzchołków w grafie.

3.2.2 S-metryka grafu

Definicja

S-metryka mierzy, jak mocno połączone są kluczowe węzły (huby) w sieci. Pozwala ona lepiej różnicować sieci o podobnym rozkładzie stopni poprzez ocenę, jak dobrze połączone są główne węzły o wysokim stopniu. [4]

Algorytm

Algorytm wykorzystuje podejście polegające na zliczaniu trójkątów (czyli cykli o długości 3) w grafie. S-metryka mierzy liczbę trójkątów w stosunku do możliwej liczby trójkątów w grafie losowym o tej samej liczbie wierzchołków i krawędzi.

Kod rozwiązania

```
def s_metric(graph):
    g = nx.DiGraph()
    for row_idx, row in enumerate(graph):
        for col_idx, cell in enumerate(row):
            if cell == 1:
                g.add_edge(row_idx, col_idx)
    return nx.s_metric(g)
```

Opis szczegółowy

1. Tworzenie grafu skierowanego `g` przy użyciu biblioteki NetworkX.
2. Dodawanie krawędzi do grafu `g` na podstawie macierzy sąsiedztwa.
3. Obliczanie s-metryki grafu za pomocą funkcji `nx.s_metric`, która:
    - Iteruje przez wszystkie możliwe trójkąty w grafie.
    - Dla każdego trójkąta sprawdza, czy jest zamknięty (czyli czy wszystkie trzy krawędzie są obecne).
    - Oblicza stosunek liczby trójkątów do liczby wszystkich możliwych trójkątów.

Złożoność obliczeniowa

Obliczenie s-metryki ma złożoność $O(V^3)$ dla grafów gęstych, gdzie V to liczba wierzchołków.

3.3 Maksymalny cykl w grafie, liczba maksymalnych cykli

Definicja

Cykl w grafie to ścieżka, która zaczyna się i kończy w tym samym wierzchołku. Maksymalny cykl w grafie to cykl, który ma największą liczbę wierzchołków. Liczba maksymalnych cykli to liczba cykli o tej samej długości. [2]

Algorytm

Algorytm wykorzystuje funkcję `nx.simple_cycles` z biblioteki NetworkX do znajdowania wszystkich prostych cykli w grafie. Następnie oblicza długość maksymalnego cyklu oraz liczbę maksymalnych cykli.

Proste cykle, znane również jako elementarne cykle, to zamknięte ścieżki, w których każdy węzeł odwiedzany jest tylko raz. W grafach skierowanych dwa cykle są różne, jeśli nie są cyklicznymi permutacjami siebie nawzajem. W grafach nieskierowanych dwa cykle są różne, jeśli nie są ani cyklicznymi permutacjami siebie nawzajem, ani odwróceniem jednego z nich.

W przypadku, gdy długość cykli nie jest ograniczona, stosujemy nierekurencyjną wersję algorytmu Johnsona, wykorzystującą iteratory i generatory. Natomiast dla cykli o ograniczonej długości, używamy algorytmu Gupty i Suzumury. Istnieją również inne algorytmy, które mogą być bardziej efektywne w niektórych przypadkach.

Aby poprawić efektywność tych algorytmów, stosujemy techniki przetwarzania wstępnego. W grafach skierowanych skupiamy się na silnie spójnych składowych grafu, generując wszystkie proste cykle zawierające dany węzeł. Następnie usuwamy ten węzeł i dzielimy resztę grafu na silnie spójne składowe. W grafach nieskierowanych koncentrujemy się na dwuspójnych składowych, generując proste cykle zawierające daną krawędź. Po usunięciu tej krawędzi, również dzielimy resztę grafu na dwuspójne składowe.

W ten sposób możemy skutecznie znaleźć i analizować proste cykle w grafie, co pozwala lepiej zrozumieć strukturę analizowanej sieci.[3]

Kod rozwiązania

```
def find_max_cycles(graph):
    g = nx.DiGraph()
    for row_idx, row in enumerate(graph):
        for col_idx, cell in enumerate(row):
            if cell == 1:
```

```
            g.add_edge(row_idx, col_idx)
    cycles = list(nx.simple_cycles(g))
    max_cycle_len = max(
        [len(cycle) for cycle in cycles]
    ) if cycles else 0
    max_cycles = [
        cycle for cycle in cycles if len(
            cycle
        ) == max_cycle_len
    ]
    return max_cycle_len, max_cycles
```

Opis szczegółowy

1. Tworzenie grafu skierowanego `g` przy użyciu biblioteki NetworkX.
2. Dodawanie krawędzi do grafu `g` na podstawie macierzy sąsiedztwa.
3. Znajdowanie wszystkich prostych cykli w grafie `g` za pomocą funkcji `nx.simple_cycles`, która:
   - Iteruje przez wszystkie możliwe cykle w grafie.
   - Sprawdza każdy cykl, czy jest prosty (czyli nie zawiera podcykli).
   - Zwraca listę wszystkich prostych cykli.
4. Obliczanie długości maksymalnego cyklu oraz liczby maksymalnych cykli.

Złożoność obliczeniowa

Złożoność algorytmu do znajdowania prostych cykli wynosi $O((V+E)(C+1))$, gdzie V to liczba wierzchołków, E to liczba krawędzi, a C to liczba prostych cykli. W najgorszym przypadku złożoność wynosi $O(V^3)$.

4 Wyniki testów obliczeniowych

Testy zostały przeprowadzone na kilku przykładowych grafach. Poniżej przedstawiono przykładowe wyniki dla jednego z grafów:

4.1 Przykładowy test 1

Dane wejściowe

```
4
0 1 0 0
0 0 1 0
0 0 0 1
0 1 0 0
```

Wyniki

```
Graph 1
  -> Nodes: 4
  -> Size: 4
  -> Density:  0.33
  -> S-metric: 19.00
  -> Max cycle length: 3
  -> Max cycle count: 1
```

Czas wykonania

Czas wykonania programu dla powyższego grafu wyniósł 0.177 sekundy.

4.2 Przykładowy test 2

Dane wejściowe

```
2
13
0 0 1 1 1 1 0 0 1 0 0 1 0
0 0 0 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 0 0 1 1 0 1 0 1
0 0 1 0 0 1 0 1 0 0 1 0 0
1 1 1 0 0 0 0 1 1 0 0 0 0
1 0 0 1 1 0 1 0 1 0 0 0 1
0 0 1 0 0 1 0 0 0 0 1 1 1
1 0 0 0 1 1 0 0 1 1 1 0 1
1 1 1 0 1 0 1 0 0 0 0 0 0
1 0 1 0 1 0 1 1 0 0 1 1 0
1 0 1 0 1 0 0 1 0 0 0 0 1
0 1 1 1 1 0 1 0 0 1 0 0 0
0 0 0 1 1 0 1 0 0 0 0 0 0
```

```
13
0 0 1 0 0 1 1 1 1 0 0 0 0
1 0 1 1 1 1 0 0 1 0 1 1 0
1 1 0 1 0 1 0 1 1 0 1 1 0
1 1 1 0 1 1 1 1 0 0 1 0 0
1 0 1 1 0 0 0 0 1 0 0 0 1
1 1 0 1 0 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 0 0 0 1 1 1
1 0 1 0 1 0 1 0 0 0 1 0 1
1 1 0 0 0 0 1 1 0 1 0 1 0
1 1 0 0 1 0 1 1 1 0 1 0 0
0 1 0 1 1 0 1 1 0 1 0 0 0
0 1 0 0 0 0 0 1 0 0 1 0 1
0 0 0 1 1 1 0 0 1 0 0 1 0
```

Wyniki

```
Graph 1
 -> Nodes: 13
 -> Size: 71
 -> Density:  0.46
 -> S-metric: 8956.00
 -> Max cycle length: 13
 -> Max cycle count: 10058

 -----------------

Graph 2
 -> Nodes: 13
 -> Size: 81
 -> Density:  0.52
 -> S-metric: 13269.00
 -> Max cycle length: 13
 -> Max cycle count: 67039
```

Czas wykonania

Czas wykonania programu dla powyższych grafów wyniósł 1.83 sekundy.

4.3 Przykładowy test 3

Dane wejściowe

```
2
4
0 1 0 0
0 0 1 0
0 0 0 1
0 1 0 0

3
0 1 0
0 0 1
0 0 0
```

Wyniki

```
Graph 1
  -> Nodes: 4
  -> Size: 4
  -> Density:  0.33
  -> S-metric: 19.00
  -> Max cycle length: 3
  -> Max cycle count: 1

-----------------

Graph 2
  -> Nodes: 3
  -> Size: 2
  -> Density:  0.33
  -> S-metric:  4.00
  -> Max cycle length: 0
  -> Max cycle count: 0
```

Czas wykonania

Czas wykonania programu dla powyższego grafu wyniósł 0.243 sekundy.

Wnioski

Na podstawie przeprowadzonych testów można wyciągnąć następujące wnioski:

- Program poprawnie analizuje grafy skierowane, obliczając ich rozmiar, gęstość, s-metrykę oraz maksymalne cykle.
- Wyniki testów pokazują, że program jest w stanie przetwarzać zarówno małe, jak i duże grafy, zachowując przy tym akceptowalny czas wykonania.
- Algorytmy zaimplementowane w programie mają złożoność obliczeniową odpowiednią do analizy grafów o różnej wielkości, co czyni program skalowalnym.
- Biblioteka NetworkX okazała się być efektywnym narzędziem do analizy grafów, umożliwiając szybkie i łatwe implementowanie zaawansowanych algorytmów grafowych.
- Program może być rozszerzony o dodatkowe funkcjonalności, takie jak analiza multigrafów czy implementacja algorytmów do znajdowania cykli Hamiltona, co zwiększyłoby jego użyteczność w różnych dziedzinach.

Podsumowując, zrealizowane zadanie umożliwia efektywną analizę grafów skierowanych i może być użyte w różnych zastosowaniach, takich jak analiza sieci społecznych, struktury biologiczne czy optymalizacja sieci.

Literatura

[1] Baeldung. Graph density. https://www.baeldung.com/cs/graph-density. Dostęp: 2024-11-16.

[2] Baeldung. Path vs cycle vs circuit. https://www.baeldung.com/cs/path-vs-cycle-vs-circuit. Dostęp: 2024-11-30.

[3] NetworkX. Simple cycles. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.cycles.simple_cycles.html.networkx.algorithms.cycles.simple_cycles. Dostęp: 2024-11-30.

[4] ResearchGate. Measuring the effectiveness of the s-metric to produce better network models. https://www.researchgate.net/publication/224366095_Measuring_the_effectiveness_of_the_s-metric_to_produce_better_network_models. Dostęp: 2024-11-21.

[5] D3 Graph Theory. Order and size of a graph. https://d3gt.com/unit.html?order-and-size. Dostęp: 2024-11-09.