Projekt – nr 2

Maciej Leśniak[1]

[1]**Faculty of Physics and Applied Computer Science,, AGH University of Science and Technology**

ABSTRACT

Keywords:

ZADANIE 1, 2: SPRAWDZANIE, CZY CIAG JEST GRAFICZNY ORAZ RANDOMIZACJA KRAWEDZI

Zaimplementowano algorytm Havel-Hakimi, który sprawdza, czy dany ciag liczb naturalnych może być ciagiem stopni grafu prostego nieskierowanego. Działanie algorytmu polega na wielokrotnym sortowaniu ciagu w porzadku nierosnacym oraz odejmowaniu 1 od kolejnych elementów, zgodnie z wartościa pierwszego elementu.

Jeśli w pewnym momencie którykolwiek element stanie sie ujemny lub wartość pierwszego elementu przekroczy długość ciagu, ciag nie jest graficzny. Jeśli natomiast wszystkie elementy stana sie równe zeru, ciag jest graficzny.

Algorithm 1 Algorytm Havel-Hakimi

Input: Tablica A długości n reprezentujaca ciag stopni
Output: TRUE jeśli ciag jest graficzny, FALSE w przeciwnym razie
Posortuj tablice A nierosnaco **while** *TRUE* **do**
    if $\forall i \, A[i] = 0$ **then**
        return *TRUE*
    end
    if $A[0] \geq n$ ***lub*** $\exists i \, A[i] < 0$ **then**
        return *FALSE*
    end
    for $i = 1$ **to** $A[0]$ **do**
        $A[i] \leftarrow A[i] - 1$
    end
    $A[0] \leftarrow 0$ Posortuj tablice A nierosnaco
end

W oparciu o wynik działania algorytmu Havel-Hakimi skonstruowano graf prosty odpowiadajacy konkretnemu ciagowi graficznemu. Nastepnie zastosowano procedure randomizacji krawedzi przy zachowaniu stopni wierzchołków.

Zastosowany algorytm randomizacji wielokrotnie wykonuje operacje przekształcenia dwóch losowych krawedzi ab i cd na ad i bc, pod warunkiem, że nie powoduje to powstania wielokrotnych krawedzi ani petli. Taka operacja nie zmienia stopni wierzchołków, ale pozwala uzyskać różne losowe realizacje tego samego ciagu graficznego.

Algorithm 2 Algorytm randomizacji krawedzi przy zachowaniu stopni

Input: Graf prosty $G = (V, E)$

Output: Zrandomizowany graf G' o tych samych stopniach

for *liczba iteracji* **do**

  Wybierz losowo dwie różne krawedzie: ab i cd z E, gdzie a, b, c, d sa parami różne **if** $ad \notin E$ *i* $bc \notin E$ *i* $a \neq d$ *i* $b \neq c$ **then**

    Zamień ab, cd na ad, bc

  end

end

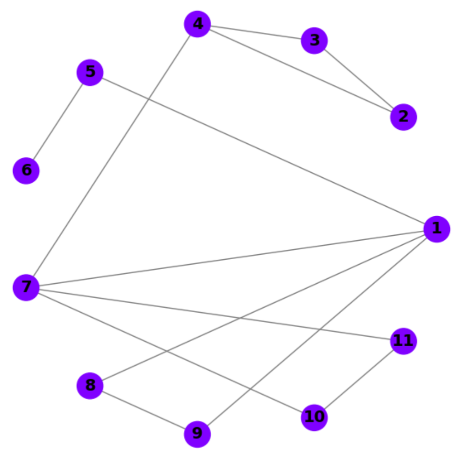

Figure 1. Graf skonstruowany z ciagu graficznego: [4 2 2 3 2 1 4 2 2 2 2]

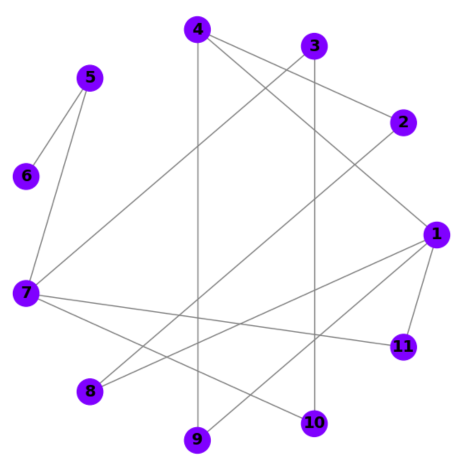

Figure 2. Graf po randomizacji przy zachowaniu stopni wierzchołków

ZADANIE 3: ZNAJDOWANIE NAJWIEKSZEJ SPÓJNEJ SKŁADOWEJ GRAFU

W tym zadaniu zaimplementowano algorytm do oznaczania składowych spójnych w grafie nieskierowanym, oparty na przeszukiwaniu w głab (DFS). Na jego podstawie określono, które wierzchołki należa do jakiej składowej, a nastepnie wybrano najwieksza z nich.

Zasada działania:

- Wierzchołki grafu sa poczatkowo oznaczane jako nieodwiedzone.
- Dla każdego nieodwiedzonego wierzchołka uruchamiana jest funkcja, która rekurencyjnie przeszukuje cała składowa, przypisujac jej unikalny numer.
- Po zakończeniu działania algorytmu wszystkie wierzchołki maja przypisana etykiete odpowiadajaca numerowi ich składowej spójnej.
- Zliczajac rozmiary poszczególnych składowych, można wyznaczyć najwieksza.

Algorithm 3 components(G)

Input: Graf G
Output: Tablica *comp*: dla każdego wierzchołka numer składowej spójnej

```
nr ← 0 ;                                          // nr - numer spójnej sk␣ladowej
for każdy wierzchołek v należacy do grafu G do
    comp[v] ← −1 ;                      // Wszystkie wierzcho␣lki sa nieodwiedzone
end
for każdy wierzchołek v należacy do grafu G do
    if comp[v] = −1 then
        nr ← nr + 1  comp[v] ← nr ;                  // Oznaczamy v jako odwiedzony
        components_R(nr, v, G, comp) ;                         // Dalsze odwiedzanie
    end
end
return comp
```

Algorithm 4 components_R(nr, v, G, $comp$)

Input: Numer składowej *nr*, wierzchołek *v*, graf *G*, tablica *comp*

```
for każdy wierzchołek u ∈ G bedacy sasiadem v do
    if comp[u] = −1 then
        comp[u] ← nr  components_R(nr, u, G, comp)
    end
end
```

Zastosowanie algorytmu: Po wykonaniu algorytmu na grafie, utworzono mapowanie numerów składowych na liczbe ich wierzchołków i pokolorowano wierzchołki zgodnie z numerem spójnej składowej.

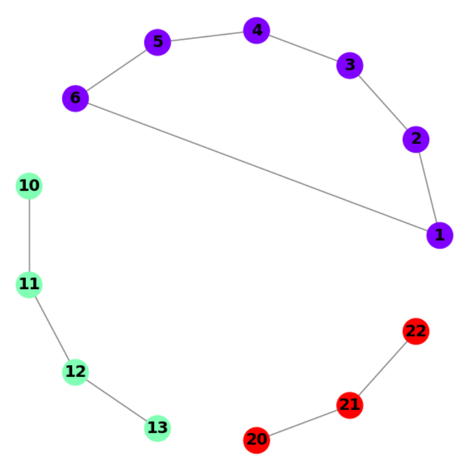

Figure 3. Graf z oznaczonymi składowymi spójnymi - najwieksza (1)

ZADANIE 4: LOSOWY GRAF EULEROWSKI I CYKL EULERA

Celem zadania było wygenerowanie losowego grafu nieskierowanego, który spełnia warunki bycia grafem eulerowskim, a nastepnie znalezienie cyklu Eulera metoda Fleury'ego.

Definicja: Cykl Eulera to zamknieta ścieżka zawierajaca każda krawedź grafu dokładnie jeden raz. Graf spójny jest eulerowski wtedy i tylko wtedy, gdy stopień każdego wierzchołka jest parzysty.

Tworzenie grafu eulerowskiego

Wygenerowano graf losowy o zadanej liczbie wierzchołków i dodawano krawedzie w taki sposób, by:

- każda krawedź była unikalna,
- graf pozostał spójny,
- wszystkie wierzchołki miały parzysty stopień.

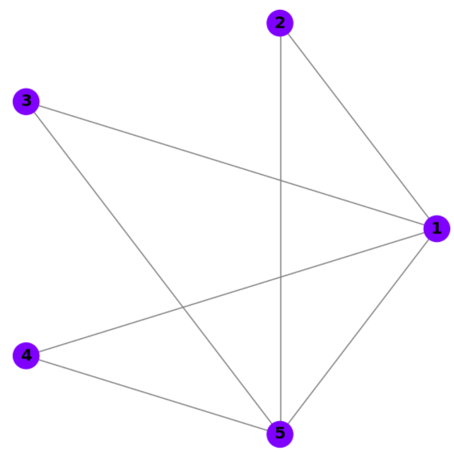

Figure 4. Wygenerowany losowy graf eulerowski - Cykl Eulera: [1, 5, 2, 1, 4, 5, 3, 1]

Algorytm Fleury'ego

Algorytm polega na przechodzeniu krawedzi grafu z zachowaniem zasady unikania mostów, o ile istnieje alternatywna ścieżka.

Algorithm 5 Algorytm Fleury'ego do znajdowania cyklu Eulera

Input: Graf eulerowski G
Output: Cykl Eulera w postaci listy wierzchołków
Wybierz dowolny wierzchołek startowy v **while** *istnieja nieodwiedzone krawedzie wychodzace z v* **do**
  Wybierz krawedź $e = (v, u)$ niebedaca mostem (jeśli sa inne możliwości) Dodaj e do cyklu Usuń e z grafu Jeśli v jest izolowany, usuń go z grafu $v \leftarrow u$
end
return *cykl*

ZADANIE 5: GENEROWANIE LOSOWEGO GRAFU k-REGULARNEGO

Celem zadania było wygenerowanie grafu k-regularnego, tzn. takiego, w którym każdy wierzchołek ma dokładnie k sasiadów. Grafy takie moga być niespójne, a ich generacja wymaga spełnienia określonych warunków.

Warunki wejściowe

Dla podanych parametrów: liczba wierzchołków n oraz stopień k, graf k-regularny istnieje tylko wtedy, gdy:

- $n > k$
- jeśli k jest nieparzyste, to n musi być parzyste

Etapy generowania grafu

1. **Kontrola warunków** dla n i k
2. **Utworzenie grafu**: na podstawie ciagu graficznego złożonego z n wartości k
3. **Randomizacja**: wykonano wielokrotna zamiane par krawedzi (jak w zadaniu 2), aby uzyskać losowy układ przy zachowaniu regularności

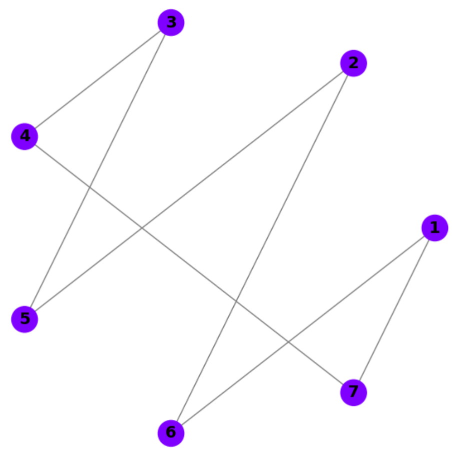

Figure 5. Przykładowy losowy graf k-regularny ($n = 7$, $k = 2$)

ZADANIE 6: SPRAWDZANIE, CZY GRAF JEST HAMILTONOWSKI

Celem zadania było sprawdzenie, czy dany nieskierowany graf zawiera cykl Hamiltona, czyli taki cykl prosty, który przechodzi przez każdy wierzchołek dokładnie jeden raz i wraca do wierzchołka poczatkowego.

Ponieważ problem jest NP-zupełny, zastosowano algorytm siłowy oparty na przeszukiwaniu w głab (DFS), który sprawdza wszystkie możliwe ścieżki odwiedzajace każdy wierzchołek dokładnie raz.

Zasada działania:

- Algorytm rozpoczyna sie od wybranego wierzchołka i eksploruje wszystkie możliwe ścieżki, odkładajac odwiedzone wierzchołki na stos.
- Gdy stos zawiera wszystkie wierzchołki, sprawdzane jest połaczenie miedzy ostatnim i pierwszym wierzchołkiem — jeśli istnieje, cykl Hamiltona został znaleziony.
- W przypadku braku dalszych możliwości, algorytm cofa sie i próbuje innych ścieżek (backtracking).

Ograniczenia: Ze wzgledu na złożoność obliczeniowa, algorytm został przetestowany jedynie na małych grafach (np. $n \leq 10$).

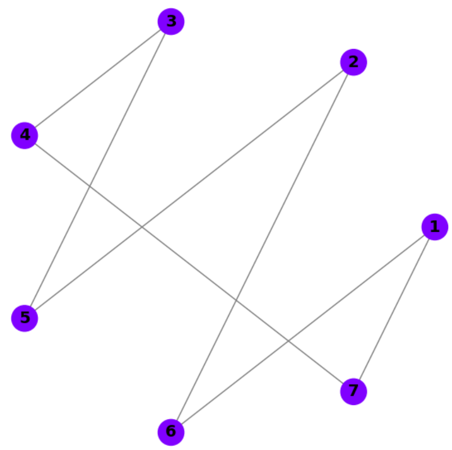

Figure 6. Przykładowy graf testowany pod katem cyklu - Cykl Hamiltona: [1, 6, 2, 5, 3, 4, 7, 1]

PODSUMOWANIE

W ramach projektu zrealizowano szereg zadań zwiazanych z analiza i generowaniem grafów w teorii grafów, opierajac sie na algorytmach klasycznych oraz autorskiej implementacji w jezyku Python.

- Zaimplementowano algorytm Havel-Hakimi do weryfikacji, czy ciag liczb naturalnych jest ciagiem graficznym, a nastepnie na jego podstawie generowano odpowiadajace grafy proste.
- Przeprowadzono randomizacje krawedzi z zachowaniem stopni wierzchołków, co umożliwiło uzyskiwanie różnych reprezentacji tego samego ciagu graficznego.
- Za pomoca algorytmu DFS wyznaczono składowe spójne grafu, identyfikujac najwieksza z nich.
- Wygenerowano graf eulerowski i odnaleziono w nim cykl Eulera z wykorzystaniem algorytmu Fleury'ego.
- Opracowano metode do generowania losowych grafów *k*-regularnych oraz przeprowadzono ich wizualizacje.
- W przypadku problemu NP-zupełnego, jakim jest wykrycie cyklu Hamiltona, zastosowano pełne przeszukiwanie (DFS bruteforce) dla małych grafów, co potwierdziło poprawność koncepcji.

REFERENCES

[1] NetworkX: Python package for complex networks. `https://networkx.org/`
[2] Matplotlib: 2D plotting library in Python. `https://matplotlib.org/`
[3] `https://pl.wikipedia.org/wiki/Algorytm_Fleuryego`
[4] `https://pl.wikipedia.org/wiki/Cykl_Eulera`
[5] `https://pl.wikipedia.org/wiki/Cykl_Hamiltona`