Grafy

Programowanie funkcyjne

Kowal E. Bocul R. Saribekyan A.

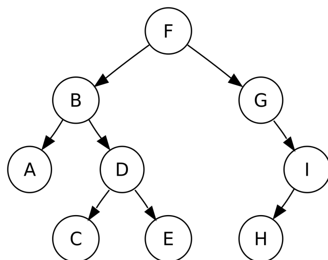
29 stycznia 2022

Spis treści

1. Przedstawienie grafu za pomocą list krawędzi
2. Przedstawienie grafu za pomocą list sąsiedztwa
3. Sortowanie topologiczne przeprowadzone na grafie
4. Badanie grafu metodą przeszukiwania w głąb (DFS)
5. Badanie grafu metodą przeszukiwana w szerz (BFS)
6. Przedstawienie grafu przy pomocy Graphviz
7. Stosowanie acyklicznych skierowanych grafów słów (DAWG)
8. Praca z sieciami sześciokątnymi i kwadratowymi
9. Znajdowanie maksymalnych klik na wykresie.
10. Ustalanie izomorficzności dwóch wykresów.

Wstęp

Grafy są podstawową strukturą danych do reprezentowania sieci. Pozwalają na większą dowolność w przedstawianiu struktur niż drzewa. Przedstawimy przykładowe implementacje kodów angażujących grafy w języku programowania *Haskell*, który posiada wiele funkcji umożliwiających pracę z nimi.

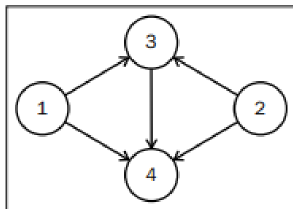


Rysunek: Przykładowy graf

Przedstawienie grafu za pomocą list krawędzi

Graf może być zdefiniowany za pomocą **list krawędzi**. Jest to struktura danych używana do reprezentowania grafu jako list jego krawędzi, gdzie krawędzie są skończonymi listami składającymi się z wierzchołka początkowego oraz końcowego.

Do stworzenia kodu używamy biblioteki *Data.Graph*. Wierzchołek (węzeł) to *Inc.*, a *buildG*, to funkcja służąca do skonstruowania struktury danych grafu z listy krawędzi.

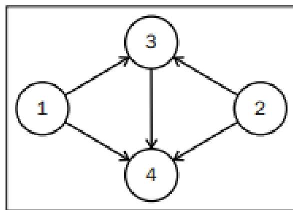


Rysunek: Graf wykorzystywany w programie listofedges.hs.

Przedstawienie grafu za pomocą list sąsiedztwa

Kolejnym sposobem jest użycie **list sąsiedztwa**. Jest to zbiór nieuporządkowanych list służących do reprezentowania skończonego grafu.

Tutaj również korzystamy z paczki *Data.Graph*. Posłuży nam do czytania mapowania wierzchołka do listy połączonych wierzchołków. Wykorzystamy również funkcję *graphFromEdges'*, która zwraca skończoną listę złożoną z elementów- struktury danych grafu i mapowania numeru wierzchołka do jego klucza.



Rysunek: Graf wykorzystywany w programie adjencylist.hs.

Sortowanie topologiczne przeprowadzone na grafie

Jeśli graf jest skierowany (jego krawędzie mają kierunek), to jednym z naturalnych jego porządków jest **sortowanie topologiczne**, gdzie każdy wierzchołek poprzedza wszystkie te wierzchołki, do których prowadzą wychodzące od niego krawędzie.

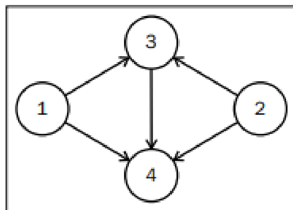
W podejściu opartym na sieciach zależności, topologiczne sortowanie pokaże możliwe kolejności wyliczenia wszystkich wierzchołków, które spełniają takie zależności.

Tutaj użyjemy wbudowanej funkcji Haskella *topSort*, która pozwala na ten rodzaj sortowania elementów grafu. Stworzymy graf zależności i wybierzemy kolejność za pomocą sortowania topologicznego.

Badanie grafu metodą przeszukiwania w głąb (DFS)

Kolejnym rozwiązaniem jest przeszukiwanie grafu za pomocą metody **DFS**, czyli **przeszukiwania w głąb**. Polega ona na przebadaniu każdej krawędzi, która wychodzi z danego wierzchołka, po czym wraca do wierzchołka "początkowego", czyli takiego, z którego przeszliśmy do wspomnianego wyżej, danego wierzchołka.

Implementacja sortowania topologicznego, algorytmu rozwiązywania labiryntu, czy znajdowania połączonych komponentów są przykładami algorytmów wykorzystujących przeszukiwanie w głąb.



Rysunek: Graf wykorzystywany w programie depth-first.hs.

Badanie grafu metodą przeszukiwana w szerz (BFS)

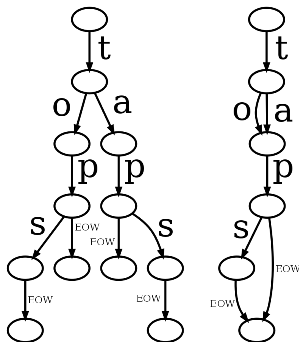
Metoda przeszukiwania grafu w szerz jest jednym z najprostszych algorytmów przeszukiwania grafu. Przeszukiwanie grafu rozpoczyna się od zadanego przez Nas wierzchołka i polega na odwiedzeniu wszystkich osiągalnych z niego wierzchołków. Za pomocą przeszukiwania grafu w szerz można wyznaczyć najkrótsze pod względem liczby krawędzi (ale nie wag) ścieżki między wierzchołkami źródłowymi, a pozostałymi wierzchołkami.

Przedstawienie grafu przy pomocy Graphviz

Przy pomocy biblioteki **graphvic** możemy łatwo narysować interesujące nas grafy. W świecie analizy danych wizualna interpretacja może ujawnić szczegóły, których ludzkie oko nie byłoby w stanie dostrzec.

Stosowanie acyklicznych skierowanych grafów słów (DAWG)

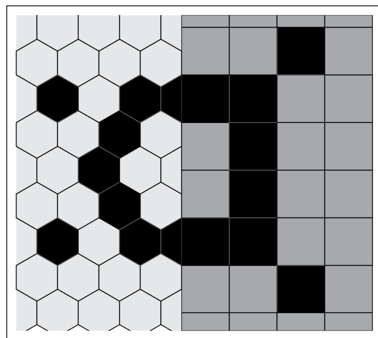
Skierowany
acykliczny wykres słów (DAWG)
to struktura danych reprezentująca zestaw ciągów, często używana jako kompaktowy sposób przechowywania słownika. Każde słowo zaczyna się od węzła źródłowego i kończy w węźle ujścia. Pomiędzy źródłem, a ujściem znajduje się sieć węzłów, połączonych skierowanymi krawędziami oznaczonymi literą lub jako EOW (koniec słowa) podczas łączenia się z ujściem.



Każdy węzeł ma co najwyżej jedną krawędź wychodzącą dla każdej litery lub EOW. Podążanie każdą możliwą ścieżką od węzła źródłowego do węzła ujścia daje listę słów w słowniku.

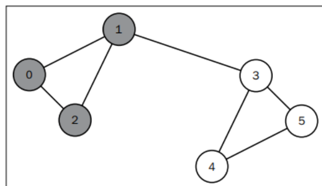
Praca z sieciami sześciokątnymi i kwadratowymi

Czasami wykres, z którym mamy do czynienia ma konkretną strukturę taką jak siatka kwadratowa lub heksagonalna. Wiele gier wideo używa heksagonalnego układu siatki, aby ułatwić ruch po przekątnej, ponieważ poruszanie się po przekątnej w kwadratowej siatce bardziej komplikuje wartości przebytego dystansu. Z drugiej strony kwadratowe struktury siatki są często używane w algorytmach manipulacji obrazem np. wypełnianie powodziowe.



Znajdowanie maksymalnych klik na wykresie

Haskell oferuje takie udogodnienia jak biblioteki grafów witalnych, z których jedna z nich to biblioteka wykrywania z **Data.Algoritm.Maximual Cliques**. Klika na wykresie to podgraf, w którym wszystkie węzły mają połączenia między sobą i jest przedstawiony następująco:

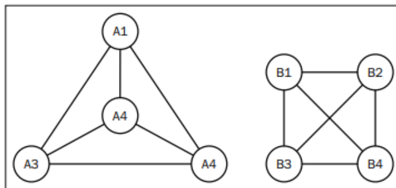


Rysunek: Graf przedstawiający dwie połączone ze sobą kliki.

Wykres ten zawiera dwie kliki. Może on przedstawiać połączone ze sobą strony internetowe, co możemy wywnioskować ze względu na strukturę wykresu. Wraz ze wzrastającą liczbą połączeń, trudniejsze staje się znalezienie największej kliki.

Ustalanie izomorficzności dwóch wykresów.

Analizując różne sieci graficzne jesteśmy w stanie stwierdzić czy są one izomorficzne, czyli czy ich połączenia mają identyczny wzór. To pomaga nam odkryć, kiedy dwa pozornie różne grafiki sieci kończą tym samym mapowaniem sieci.



Rysunek: Przykładowe grafy, którymi będziemy się zajmować.

Do wykrycia izomorficzności grafów użyjemy funkcji **isIsomorphic** z **Data.Graph.Automorphism**. Biblioteka ta pomoże nam zbadać ewentualną identyczność połączeń powyższych grafów.

Bibliografia

[1] https://pl.wikipedia.org/wiki/Wierzcho

[2] https://pl.wikipedia.org/wiki/Przeszukiwaniewszerz

[3] http://algorytmy.ency.pl/artykul/przeszukiwaniewszerz

[4] http://www.algorytm.edu.pl/grafy/bfs.html

[5] https://en.wikipedia.org/wiki/Adjacencylist

[6] https://pl.wikipedia.org/wiki/Sortowanietopologiczne

[7] https://en.wikipedia.org/wiki/Dependencynetwork

[8] https://pl.wikipedia.org/wiki/Przeszukiwaniewg

[9] Chapter "*Graph Fundamentals*"

Dziękujemy za uwagę!