Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych

Grafy i sieci
“KP5 - Sprawiedliwe kolorowanie grafu”

Skład grupy:
inż. Anna Kotkowska
inż. Alicja Sowińska

Warszawa 2020

1. Treść zadania projektowego.

Problem kolorowania wierzchołków grafu polega na takim przyporządkowaniu kolorów do wierzchołków grafu, przy którym wierzchołki sąsiednie (połączone krawędzią) mają przypisane różne kolory. Pokolorowanie jest sprawiedliwe jeżeli różnice pomiędzy liczbami wierzchołków pomalowanych poszczególnymi kolorami nie są większe niż 1. W ramach projektu opracowano algorytm i program do sprawiedliwego kolorowania wierzchołków grafu przy użyciu jak najmniejszej liczby kolorów.
Problem sprawiedliwego kolorowania wierzchołków znajduje zastosowanie przy wyznaczaniu „zrównoważonego" sposobu rozdziału różnych zasobów w harmonogramach, np. przy poszukiwaniu jak najbardziej wyrównanego obciążenia w planach zajęć, wykorzystania sal itp.

2. Język programowania i użyte biblioteki.

Jako język programowania został wybrany Python. Skorzystano z bibliotek: numpy, matplotlib, netxX.

3. Wielkość grafów.

Program będzie obsługuje dowolne grafy nieskierowane posiadające dowolną liczbę wierzchołków. Zgodnie z sugestią prowadzącego, testy programu przeprowadzone zostały również dla grafów o liczbie wierzchołków sięgającej minimum setki.

4. Przewidywane sposoby testowania.

Na podstawie artykułu M. Kubale (red.): Modele i metody kolorowania grafów, WNT, 2002 określone zostały oczekiwane rozwiązania problemu optymalnego sprawiedliwego kolorowania grafów. Rozwiązania te zostały porównane z wynikami uzyskanymi przez nasz program.

Z implementacyjnego punktu widzenia wykorzystane zostały funkcje biblioteki netxx umożliwiające wygenerowanie losowego grafu o zadanej liczbie wierzchołków oraz prawdopodobieństwie wystąpienia krawędzi. Test składał się z licznika iteracji oraz pętli realizującej 30 wywołań programu. Podczas każdej iteracji generowany był nowy graf podawany na wejście programu, a jego plik źródłowy z przedrostkiem „generated_*" został zapisany w folderze odpowiadającym klasie generowanego grafu wśród plików źródłowych „./res". W liście „test_result"

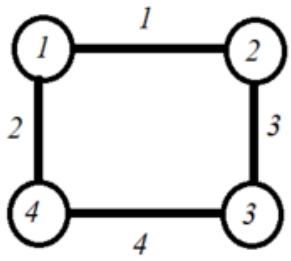
zapisane zostały wyrazy „true” oraz „false” oznaczające odpowiednio powodzenie oraz niepowodzenie w uzyskaniu sprawiedliwego kolorowania wierzchołków grafu.

5. Szczegółowy opis wejścia programu.

Na wejście programu możliwe jest wczytanie pliku .txt z przykładowym grafem nie pokolorowanym lub wygenerowanie grafu za pomocą generatora z biblioteki netxx.

Struktura pliku wejściowego przedstawiona została poniżej. W linii zapisywane będą odpowiednio wierzchołki które połączone są krawędzią, co oznacza, że w pierwszej linii będą informacje dotyczące pierwszej krawędzi, w drugiej linii drugiej krawędzi itd. Pomiędzy etykietami wierzchołków, w przypadku poniżej liczbami, znajduje się odstęp umożliwiający odróżnienie wierzchołka pierwszego od drugiego np.:

1 2
1 4
2 3
3 4



Rys.1

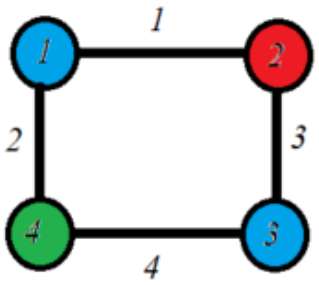
6. Szczegółowy opis wyjścia programu.

Na wyjściu programu otrzymany został plik .txt o nazwie z przedrostkiem *„output_naiwny_final.txt”*, w którym znajdują się informacje na temat pokolorowanych sprawiedliwie wierzchołków. Plik ten należy traktować jako ostateczne rozwiązanie. Dodatkowo, w pliku pomocniczym „*output_SmallestLast.txt*” umieszczono kopię grafu w stadium pośrednim – po kolorowaniu zachłannym metodą Smallest Last, ale przed ponownym pokolorowaniem, tym razem sprawiedliwym.

Finalnie każdemu wierzchołkowi został przypisany kolor. W pierwszej kolumnie znajduje się etykieta wierzchołka, czyli jego numer, a w kolejnej numer koloru np.:

1 kolor: 1
2 kolor: 2
3 kolor: 1
4 kolor: 3

gdzie:
1- niebieski
2 - czerwony
3 - zielony



Rys.2

Graf został przedstawiony graficznie za pomocą biblioteki matplotlib oferującej podstawowe narzędzia do rysowania grafów w Python - pyplot. Pliki graficzne w formacie .png z grafem nie pokolorowanym, pokolorowanym zachłannie oraz pokolorowanym sprawiedliwie zostały zapisane odpowiednio pod nazwą *„input_graph.png”, „output_SmallestLast.png” oraz „output_Naiwny_final.png”*.

7. Opis algorytmu.

W projekcie w celu wstępnego pokolorowania wierzchołków użyta została metoda *SL (Smallest Last)* opracowana przez Matulę w 1972 r., charakteryzująca się złożonością obliczeniową O(n+m). Metoda ta wykorzystuje algorytm *Zachłannego kolorowania* wierzchołków grafu. W kolejnym kroku, sprawiedliwe kolorowanie wierzchołków zostało uzyskane przez użycie algorytmu *Naiwny* przedstawionego w [1] o pesymistycznej złożoności rzędu O(n^4).

Ogólna koncepcja algorytmu:

1. Posortuj wierzchołki według nierosnących stopni wierzchołków.
2. Znajdź wierzchołek o minimalnym stopniu.
3. Usuń znaleziony wierzchołek z grafu G.
4. Zapisz usunięty wierzchołek na początku listy.
5. Wróć do punktu 1 i wykonuj do momentu aż graf będzie pusty.
6. Pokoloruj wierzchołki zachłannie rozpoczynając od wierzchołków usuniętych najpóźniej.
7. Zlicz liczbę wystąpień kolorów.
8. Jeśli pokolorowanie niesprawiedliwe, to dla każdego wierzchołka pokolorowanego kolorem o najwyższej liczbie wystąpień sprawdź czy można pokolorować go kolorem o najniższej liczbie wystąpień. Jeśli tak, zrób to.
9. Powtórz kroki *3-4,* aż pokolorowanie sprawiedliwe.
10. Jeśli pokolorowanie sprawiedliwe to zwróć pokolorowany graf.

8. Wewnętrzne struktury danych.

1) Lista sąsiedztw.
   W pliku .txt początkowo zadeklarowana została liczba par wierzchołków, czyli liczba linijek do wczytania z pliku, ale w kolejnej wersji implementacji zrezygnowano z tego rozwiązania na rzecz prostszego, polegającego na wczytywaniu pliku aż do momentu wyczerpania się zasobu linii w pliku tekstowym.
   Jedna para to jedna krawędź między wierzchołkami v_i i v_j. Za pomocą algorytmu znajdującego się w bibliotece netX, wygenerowane zostały listy sąsiedztw dla zapisanego w pliku grafu G.
2) Słownik.
   Biblioteka netX daje możliwość zapisu grafów w postaci słownikowej czyli zapisu wierzchołków (bądź krawędzi) wraz z odpowiadającymi im atrybutami. Zgodnie z [6] biblioteka ta generuje też automatycznie macierz sąsiedztw (również w postaci słowników) oraz słownika stopni wierzchołków.
   Docelowo w projekcie wykorzystana została macierz sąsiedztw, pozwalająca przez zrzutowanie odpowiednich wierszy na stworzenie list sąsiedztw, oraz wierzchołki w postaci słowników zawierających atrybut ‘kolor’.
3) Listy.
   Wykorzystane zostały dwa typy list – pierwsze powstałe ze zrzutowanych na typ „list” słowników oraz listy ‘pomocnicze’ tworzone na początku jako puste listy i stopniowo wypełniane wartościami.

9. Wyniki przeprowadzonych badań.

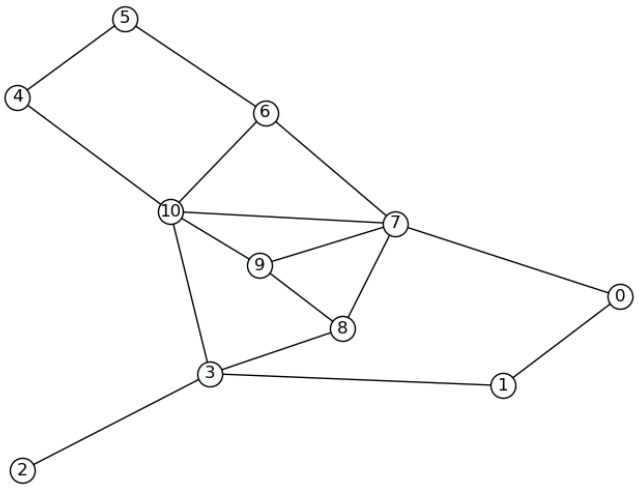
Testy poprawności działania przedstawionego algorytmu przeprowadzone zostały dla przykładowego grafu wczytanego z pliku „*input.txt”* oraz dla wygenerowanych losowo grafów, w tym dla grafów pełnych, drzew, grafów z samotnymi wierzchołkami itp. Dzięki walidacji na podstawie grafu z samotnym wierzchołkiem wykryta została luka, która została w pełni naprawiona.

Łącznie wykonano ok. 100 testów (w tym 30 dla losowych grafów pełnych, 30 dla grafów losowych w każdym możliwym aspekcie) , przy czym wszystkie testy przebiegły pomyślnie. Wszystkie z pokolorowań okazały się sprawiedliwe, co zostało ocenione na podstawie wyjściowych postaci listy *test_result.*
Poniżej przykład składni *test_result* dla pięciu scenariuszy testowych:

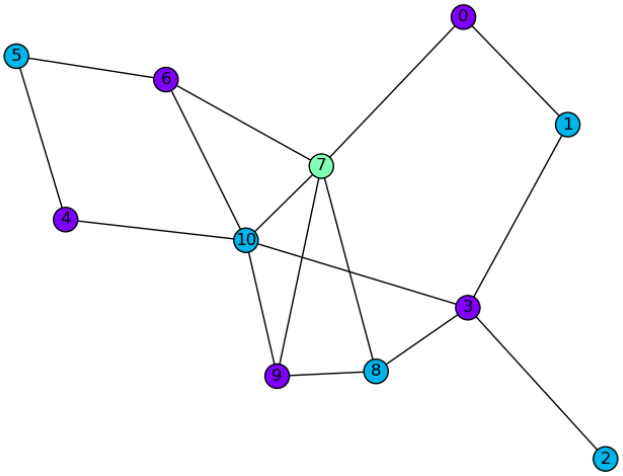
[‘true’, ‘true’, ‘true’, ‘true’, ‘true’]

W przypadku pojawienia się wartości ‘false’ przykładowo na trzeciej pozycji, oznaczałoby to nie otrzymanie sprawiedliwego kolorowania wierzchołków.

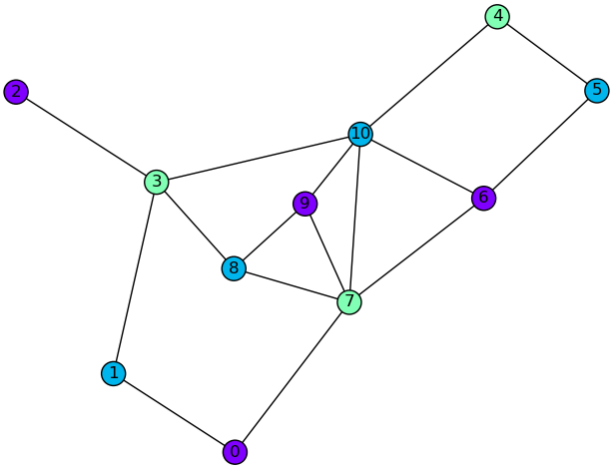
Poniżej przedstawiona została grafika obrazująca trzy etapy kolorowania grafu G – stan nie pokolorowany, stan pokolorowany zachłannie oraz stan pokolorowany sprawiedliwie.



Rys. 3. Graf nie pokolorowany



Rys. 4. Graf pokolorowany zachłannie



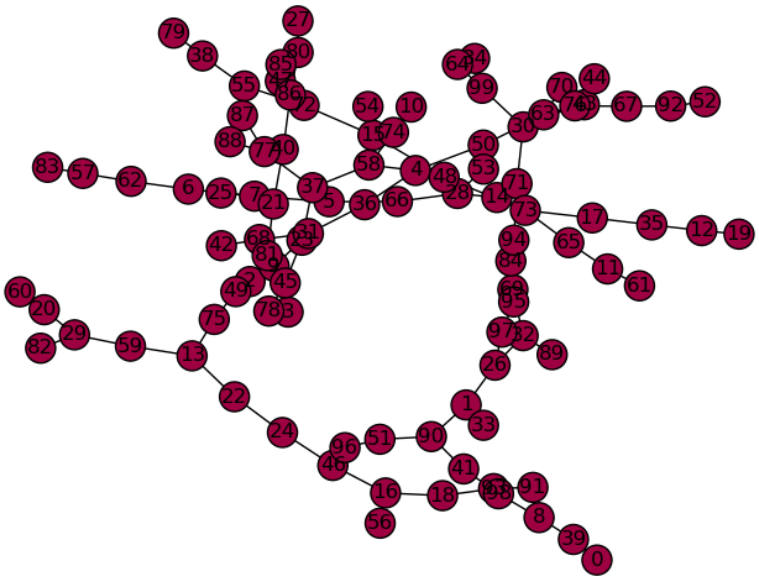
Rys. 5. Graf pokolorowany sprawiedliwie.

Na rys. 5 można zauważyć, że w porównaniu do rys. 4 kolor jasnozielony został przyporządkowany do wierzchołków 4 i 3 , tym samym odbierając te wierzchołki kolorowi fioletowemu. Różnica pomiędzy liczbą wystąpień koloru najczęstszego i najrzadszego

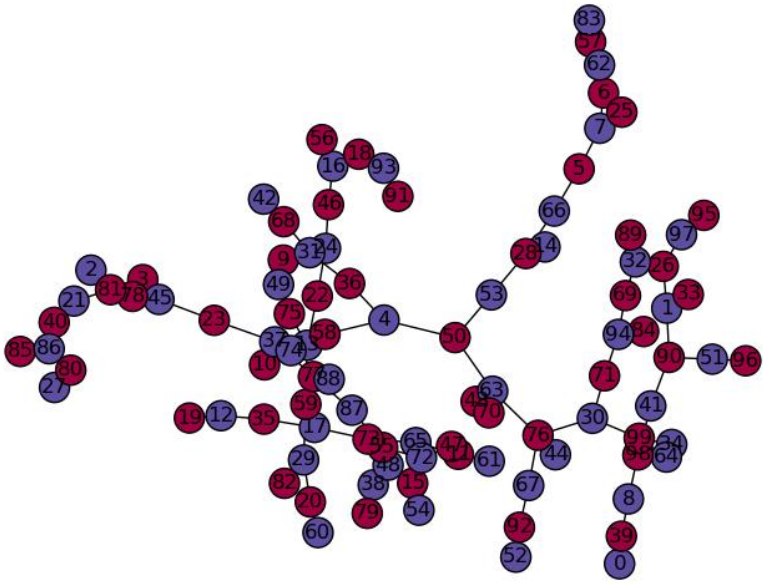
wynosi teraz nie więcej niż jeden (równo jeden), co jest wymogiem sprawiedliwego pokolorowania wierzchołków grafu.

Ciekawy w analizie pozostaje problem kolorowania optymalnego i sprawiedliwego. Kolorowanie optymalne oznacza jak najmniejszą liczbę kolorów potrzebnych do pokolorowania grafu tak by dwa incydentne wierzchołki miały inne kolory. Z badań wyniknęło, że zaimplementowany algorytm zawsze koloruje sprawiedliwie, ale nie zawsze optymalnie. Przykładem jest graf będący drzewem o 100 wierzchołkach przedstawiony poniżej. Drzewa, z minimum dwoma wierzchołkami, ze względu na swój acykliczny charakter, można pokolorować dwoma kolorami, co spełnia warunek jak najmniejszej liczby użytych kolorów. Mogłoby się więc wydawać, że takie rozwiązanie jest najlepsze, bo optymalne, jednak w przypadku dodania kolejnego warunku na kolorowanie sprawiedliwe, mogą się wydarzyć przypadki, w których dwa kolory nie wystarczą. Posłużyć się w tym miejscu można pojęciem maksymalnego zbioru niezależnego, czyli maksymalnej liczby wierzchołków, które nie są połączone ze sobą krawędzią. Dla przedstawionego przykładu, maksymalny zbiór niezależny został przedstawiony przez kolor czerwony (pokolorowanie przez algorytm zachłanny) na rys. 7 i liczy sobie 52 wierzchołki. Dla każdej sytuacji dla której te zbiory niezależne będą różniły się liczbą przydzielonych wierzchołków o więcej niż jeden, algorytm Naiwny utworzy nowy kolor by w rzeczywistości podzielić zbiór wierzchołków na mniejsze zbiory niezależne o zbliżonej liczbie wierzchołków. Tworzenie kolejnych zbiorów niezależnych będzie trwało dopóki program nie uzna, że graf jest pokolorowany sprawiedliwie.

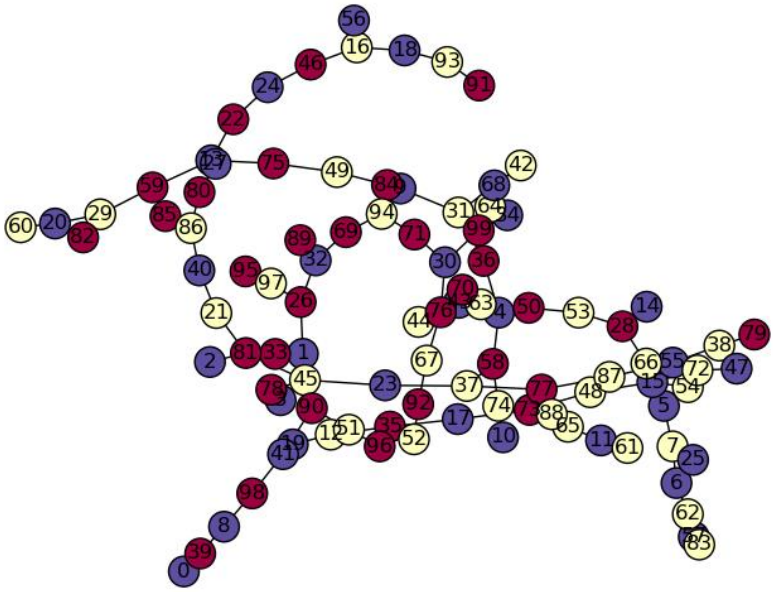
Zbierając informacje zawarte powyżej, po ‘pierwszym kolorowaniu’ algorytmem zachłannym z użyciem metody Smallest Last otrzymane zostanie kolorowanie optymalne, czyli dwoma kolorami. Kolejny krok to algorytm Naiwny, który po przez zamianę kolorów niektórych wierzchołków zapewni kolorowanie sprawiedliwe.



Rys. 6 Nie pokolorowany graf drzewo o 100 wierzchołkach.



Rys. 7 Pokolorowany zachłannie graf drzewo o 100 wierzchołkach. Rozwiązanie optymalne (2 kolory).



Rys. 8 Pokolorowany sprawiedliwie graf drzewo o 100 wierzchołkach (3 kolory).

Na przedstawionym przykładzie należy wnioskować, że zaimplementowany algorytm przynajmniej większość grafów nieskierowanych jest w stanie pokolorować sprawiedliwie czyli równomiernie, jednak nie zawsze kolorowanie to przebiega najmniejszą możliwą liczbą kolorów, co jest definicją kolorowania optymalnego.

Literatura

[1] M. Kubale (red.): Modele i metody kolorowania grafów cz.1, WNT, 2002.
[2] K. Pieńkosz: Grafy i sieci (GIS) - wykład: Kolorowanie grafów, 2020
[3] M. Kubale (red.): Optymalizacja dyskretna. Modele i metody kolorowania grafów, rozdział 4, WNT, Warszawa, 2002.
[4] https://zagalski.pl/blog/2015/08/26/jak-otworzyc-plik-i-czytac-kazda-linie-po-kolei-python/
[5] https://eduinf.waw.pl/inf/alg/001_search/0142.php
[6] https://networkx.github.io/documentation/stable/index.html