Warszawa, 2010-04-30

Bartosz Barczyński (bbarczynski@gmail.com)
Mateusz Czubak (mczubak@gmail.com)

Grafy i Sieci

Sprawozdanie 2

Generacja największych klik w grafie

Spis treści

1. Wstęp

Celem projektu jest napisanie programu, który w danym grafie znajdzie i wyświetli wszystkie największe kliki. Klikę definiujemy jako podgraf pełny danego grafu, czyli taki podgraf, w którym każde dwa wierzchołki są ze sobą połączone. Największa klika w danym grafie, to klika o największej możliwej liczbie wierzchołków, natomiast kliką maksymalną nazywamy klikę do której nie można dołączyć już nowego wierzchołka. Klika największa jest zawsze maksymalna.
Do realizacji zadania został wybrany algorytm opracowany przez Shuji Tsukiyama, Miko Ide, Hiromu Ariyoshi i Isao Shirakawa w 1977 roku i opisany po raz pierwszy w SIAM Journal on Computing [1]. Z powodu niedostępności ww. źródła byliśmy zmuszeni posłużyć się artykułami do niego nawiązującymi [2][3]. Algorytm ten znajduje maksymalne kliki w grafie. Aby wyznaczyć te największe, należy przejrzeć wyniki i wybrać kliki o najwyższym stopniu wierzchołka.

2. Algorytm

2.1 Założenia

Graf G=(V, E) składa się z n wierzchołków $V=\{u_1,...,u_n\}$ oraz z k krawędzi $E=\{e_1,...,e_k\}$. $N(u_j)$ oznacza zbiór wszystkich sąsiadów wierzchołka u_j. Algorytm jest uruchamiany rekurencyjne dla każdego kolejnego wierzchołka. W danym kroku j rekurencyjnego wywołania, M oznacza maksymalną klikę w podgrafie składającym się z $\{u_1,...,u_{j-1}\}$ wierzchołków.

2.2 Pseudokod

Poniżej pokazujemy psuedokod przedstawiony w [2]:

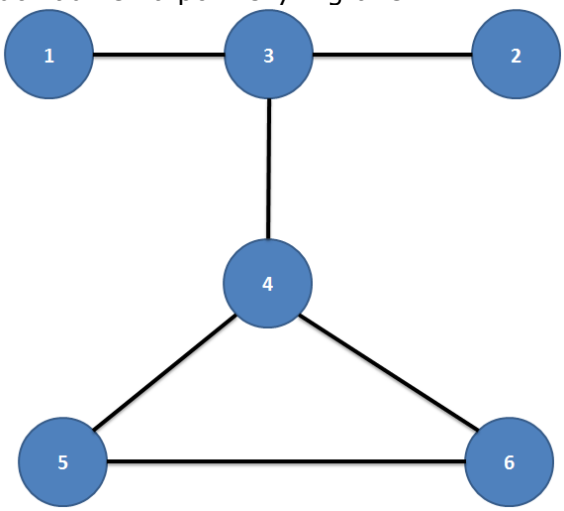
```
Algorithm call: TIAS(ϕ, 1)

function TIAS (M, j) {
     if j = n + 1 then
          Report M to be a maximal clique
     else {
          N(uj) denotes the set of neighbors of uj
          if M ⊂ N(uj) then
               TIAS(M ⋃ {uj}, j + 1)
          else {
               TIAS(M, j + 1)
               Let M' = M ⋂ N(uj)
               if M is the lexicographically smallest maximal clique
               of {u1,...,uj-1} that contains M' then
                    TIAS(M' ⋃ {uj}, j + 1)
          }
     }
}
```

2.3 Opis działania

Algorytm jest uruchamiany dostając na wejście M = ϕ (maksymalna klika znaleziona na danym etapie algorytmu jest zbiorem pustym) oraz j = 1 (pierwszy wierzchołek). W każdym kolejnym rekurencyjnym wywołaniu j jest jest zwiększane o 1, czyli analizowany jest następny wierzchołek. Wierzchołki przeglądane są w dowolnej kolejności, np. w takiej jak zostały wczytane do programu. Istotne jest jedynie przeanalizowanie wszystkich.
W ramach każdego uruchomienia funkcji TIAS możliwe jest jedno lub dwa rekurencyjne wywołania. Oznacza to, że w ramach przetwarzania algorytmu tworzone jest drzewo rekurencyjnych wywołań, w którym każdy węzeł posiada 1 lub 2 dzieci. Decyzja czy przetwarzanie algorytmu zostanie rozdwojone jest podejmowana na podstawie warunku if M $\subset$ N(uj), czyli sprawdzeniu czy wierzchołek u_j ma połączenie ze wszystkimi wierzchołkami aktualnie znalezionej kliki M:

- Jeśli tak to można go bezpiecznie dołączyć do M, ponieważ razem będą dalej tworzyć klikę. Następnie uruchamiana jest funkcja TIAS dla kolejnego wierzchołka z powiększoną kliką M. W tym przypadku następuje tylko i wyłącznie jedno wywołanie TIAS.
- W przeciwnym przypadku nie można dołączyć u_j do M, ale zostaje uruchomiony TIAS dla niezmienionej kliki M i kolejnego wierzchołka u_{j+1}. Oznacza to, że algorytm będzie próbował dodać kolejny wierzchołek do M, aż do znalezienia w kolejnych rekurencyjnych wywołaniach takiego, z którym uda mu się utworzyć większą klikę. Po każdym powrocie z rekurencji następuje sprawdzenie czy wierzchołek u_j tworzy klikę z podzbiorem wierzchołków należących do M. Aby to zweryfikować wybierane są wierzchołki wspólne między M i N(u_j), powstaje zbiór M'. Suma zbiorów M' i u_j jest na pewno kliką, ponieważ wybraliśmy wierzchołki z M (mają połączenie między sobą) oraz połączony z nimi u_j. Tak powstałą nową klikę analizujemy w kolejnym kroku rekurencji sprawdzając czy można do niej dołączyć wierzchołek j + 1. W tym przypadku następują 2 rekurencyjne wywołania TIAS, co powoduje, że drzewo rekurencyjnych wywołań rozchodzi się w dwie strony.

Niestety klika M' $\cup$ u_j może zostać utworzona na podstawie różnych klik M, gdzie M' = M $\cap$ N(u_j). Gdybyśmy wszystkie takie przypadki analizowali dalej, to algorytm byłby nieefektywny. W wyniku na liściach drzewa przetwarzania algorytmu otrzymalibyśmy kilka identycznych największych klik. Widać to dokładnie na poniższym grafie:



W pierwszych wywołaniach rekurencyjnych zostają znalezione M={1,3} oraz M={2,3}. Jeśli kolejnym analizowanym wierzchołkiem będzie 4 to dla obu

przypadków zostanie utworzona kilka M' $\cup u_j$ ={3,4}, z której następnie zostanie wyliczona największa klika w całym grafie, czyli {4,5,6}. Algorytm zwróciłby 2 identyczne wyniki i zbędnie wykonał 2 razy tą samą pracę podczas sprawdzania wierzchołków 5 i 6. Dlatego do dalszego przetwarzanie powinna zostać wybrana klika M' $\cup u_j$ postała tyko z pierwszego M. Zostanie to zapewnione przez sprawdzenie czy M jest leksykograficznie najmniejszą maksymalną kliką w zbiorze $\{u_1,...,u_{j-1}\}$ (czyli wśród wierzchołków, które mogą potencjalnie należeć do M) zawierającą M'. Leksykograficznie najmniejsza klika oznacza klikę składającą się z węzłów o najniższych numerach (alfabetyczne sortowanie). Wynika stąd, że do M' będą dołączane po kolei wierzchołki $\{u_1,...,u_{j-1}\}$, a następnie będzie sprawdzane, czy powstały w ten sposób graf jest kliką. Jeśli otrzymana w ten sposób klika jest równa M to należy kontynuować algorytm dla TIAS(M' $\cup \{u_j\}$, j + 1), w przeciwnym przypadku należy zakończyć wywołanie algorytmu dla danego wierzchołka.

Ciekawym przypadkiem jest sytuacja gdy M'=ϕ. Oznacza to, że analizowany wierzchołek nie ma połączenia z aktualnym M. Wtedy przetwarzanie algorytmu zaczyna się de facto od początku, bo kliką maksymalną staje się analizowany wierzchołek. Dzięki takiemu rozwiązaniu algorytm znajdzie wszystkie maksymalne kliki nawet w grafach niespójnych.

Warunkiem stopu dla rekurencji w algorytmie jest przejście przez wszystkie wierzchołki, a dokładniej próba analizy wierzchołka o zbyt dużym numerze (j = n + 1). Wtedy zgodnie z założeniami M jest maksymalną kliką w całym grafie. Na wszystkich liściach drzewa rekurencyjnych wywołań będą znajdować się maksymalne kliki.

Ostatnim krokiem algorytmu (nie zaznaczonym w pseudokodzie) jest wybranie największej (lub największych) klik ze zbioru wyznaczonych klik maksymalnych. Zostanie to wykonana po prostu przez liniowe przejrzenie wszystkich wyników algorytmu TIAS.

2.4 Analiza złożoności

Każdy liść drzewa rekurencyjnych wywołań zawiera unikalną maksymalną klikę. Ilość rekurencyjnych wywołań wymaganych do wyznaczenia takiej kliki jest równa wysokości drzewa, czyli złożoność obliczeniowa wynosi O(n) - po kolei dodajemy wierzchołki. W każdym wywołaniu TIAS są wykonywane następujące operacje:

- **if** M $\subset$ N(u_j) - aby sprawdzić zawieranie się dwóch zbiorów należy dla każdego elementu z pierwszego zbioru sprawdzić czy istnieją one w drugim zbiorze. Biorąc po uwagę, że liczności tych zbiorów są wprost proporcjonalne do n to złożoność takiej operacji wynosi $O(n^2)$.
- M' = M $\cap$ N(u_j) - wyznaczenie części wspólnej 2 zbiorów również wymaga sprawdzenia czy dla każdego wierzchołka z pierwszego zbioru istnietnieją wierzchołki w drugim. Analogicznie jak poprzednio daje to złożoność $O(n^2)$.
- **if** M is the lexicographically smallest maximal clique of $\{u_1,...,u_{j-1}\}$ that contains M' - wyznaczenie tego warunku również ma złożoność $O(n^2)$ ponieważ jak zostało to napisane wcześniej zostanie wykorzystany algorytm zachłanny, który do kliki M' będzie próbował dołączyć kolejno każdy z przeanalizowanych już wierzchołków.

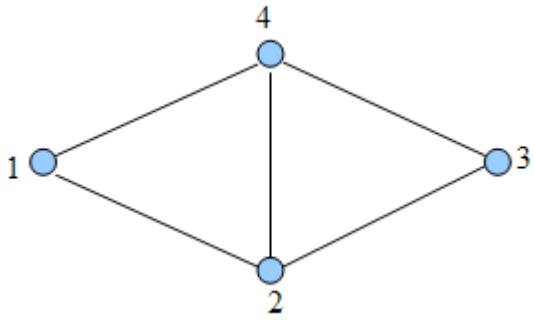
Niezależnie ile opisanych wyżej operacji zostanie wykonanych (1 czy wszystkie 3) to pojedyncze uruchomienie TIAS dąży do $O(n^2)$.
Z powyżej analizy wynika, że aby znaleźć 1 maksymalną klikę należy wykonać $O(n)*O(n^2)=O(n^3)$ operacji. Zakładając, że algorytm znajdzie C maksymalnych klik to złożoność obliczeniowa będzie wynosić $O(n^3)*C=O(Cn^3)$.

Ostatnim etapem algorytmu jest wybór największej (lub największych) klik ze zbioru wyznaczonych maksymalnych klik. Aby to wykonać potrzeba O(C*n) operacji, ponieważ dla każdej maksymalnej kliki należy obliczyć stopień dowolnego wierzchołka.

Podsumowując złożoność obliczeniowa całego algorytmu wynosi $O(Cn^3)+O(C*n)=O(Cn^3)$.

2.5 Przykład działania

Działanie algorytmu przedstawimy na przykładzie poniższego grafu:



Krok 1:

M = {0}, j = 1

Warunek j = n + 1 jest oczywiście niespełniony => $N(u_1) = \{2, 4\}$.
Warunek $M \subset N(u_1)$ jest spełniony (zbiór pusty zawsze zawiera się w dowolnym zbiorze) => dodajemy wierzchołek nr 1 do M i uruchamiamy procedurę dla kolejnego wierzchołka, kończąc jej wywołanie dla wierzchołka nr 1.

Krok 2:

M = {1}, j = 2

Warunek j = n + 1 jest niespełniony => $N(u_2) = \{1, 3, 4\}$.
Warunek $M \subset N(u_2)$ jest spełniony => dodajemy wierzchołek nr 2 do M i uruchamiamy procedurę dla kolejnego wierzchołka, kończąc jej wywołanie dla wierzchołka nr 2.

Krok 3:

M = {1, 2}, j = 3

Warunek j = n + 1 jest niespełniony => $N(u_3) = \{2, 4\}$.
Warunek $M \subset N(u_3)$ jest niespełniony, więc uruchamiamy procedurę dla wierzchołka nr 4, nie kończąc jej jednak (wrócimy do tego miejsca w kroku 6).

Krok 4:

M = {1, 2}, j = 4

Warunek j = n + 1 jest niespełniony => $N(u_4) = \{1, 2, 3\}$. Warunek $M \subset N(u_4)$ jest spełniony, więc dodajemy wierzchołek nr 4 do M i uruchamiamy procedurę dla kolejnego wierzchołka, kończąc ją dla wierzchołka nr 4.

Krok 5:

M = {1, 2, 4}, j = 5

Warunek j = n + 1 jest spełniony => dodajemy M do zbioru największych klik. Wywołanie algorytmu się na tym nie kończy – należy dokończyć wywołanie procedury z kroku 3.

Krok 6:

M = {1, 2}, j = 3

$N(u_3) = \{2, 4\}$, $M' = M \cap N(u_3) \Rightarrow M' = \{2\}$
Leksykograficznie najmniejszą maksymalną kliką dla wierzchołków 1, 2 zawierającą M' jest klika M'' = {1, 2}. Ponieważ M = M'' to naszą kolejną badaną kliką będzie M' + {3}. Wywołujemy algorytm dla następnego wierzchołka dla M = {2, 3}.

Krok 7:

M = {2, 3}, j = 4

Warunek j = n + 1 jest niespełniony => $N(u_4) = \{1, 2, 3\}$.
Warunek $M \subset N(u_4)$ jest spełniony => dodajemy wierzchołek nr 4 do M i uruchamiamy procedurę dla kolejnego wierzchołka, kończąc ją dla wierzchołka nr 4.

Krok 8:

M = {2, 3, 4}, j = 5

Warunek j = n + 1 jest spełniony => dodajemy M do zbioru największych klik i kończymy działanie algorytmu.

Po wykonaniu algorytmu otrzymamy dwie największe kliki – {1, 2, 4} i {2, 3, 4}

2.6 Optymalizacja

Algorytm dla każdego rekurencyjnego wywołania wykonuje $O(n^2)$ lub $3*O(n^2)$ operacji. Aby zmniejszyć jego złożoność należy zoptymalizować każdą z tych operacji. Niestety jest to niemożliwe. Podczas implementacji zostaną wykonane następujące usprawnienia, które nie będą w stanie obniżyć złożoności obliczeniowej całego algorytmu a jedynie przyczynią się do jego lepszego i szybszego działania programu:

- Wyszukiwanie leksykograficznie najmniejszej maksymalnej kliki w przypadku gdy $M'=\phi$ zostanie uproszczona do sprawdzenia czy M zawiera pierwszy wierzchołek. M jest zawsze kliką maksymalną, a gdy będzie zawierać u_1 to od razu można stwierdzić, że jest również alfabetycznie najmniejsza.
- Sprawdzenie czy $M \subset N(u_j)$ można zakończyć niepowodzeniem po znalezieniu pierwszego wierzchołka należącego to M, a nie należącego do $N(u_j)$. Dzięki temu nie trzeba do końca przeglądać obu list.

- Graf wczytany do pamięci będzie reprezentowany jako lista sąsiedztwa. Dzięki temu często wykonywana operacja N(u_j) będzie wymagała przejścia po wierzchołkach, z którymi faktycznie jest połączony u_j, a nie po wszystkich możliwych, czyli n.

3. Założenia programu

3.1 Ogólne

- Program zostanie napisany w języku Java
- Do wszystkich operacji na grafach wykorzystamy bibliotekę JUNG - Java Universal Network/Graph Framework [4]
- Wczytywany graf jest zwyczajny (niezorientowany, bez pętli i krawędzi równoległych)
- Jeśli graf zawiera więcej niż jedną największą klikę, to zostaną znalezione wszystkie
- Warunkiem stopu aplikacji będzie zakończenie działania algorytmu. Przy dużej liczbie wierzchołków, czas obliczeń może się okazać zbyt długi, dlatego planujemy wprowadzenie dodatkowego parametru stopu - maksymalnego czasu wykonania programu.

3.2 Dane wejściowe

Dane wejściowe zostaną wczytane do programu z pliku tekstowego zawierającego macierz sąsiedztwa grafu:

```
0 1 0 0 1 0
1 0 1 0 1 0
0 1 0 1 0 0
0 0 1 0 1 1
1 1 0 1 0 0
0 0 0 1 0 0
```

Do utworzenia grafu posłużymy się klasą *MatrixFile* z pakietu edu.uci.ics.jung.io [4], która na podstawie podanego pliku utworzy obiekt klasy *UndirectedSparseGraph* z pakietu edu.uci.ics.jung.graph [4].

3.3 Dane wyjściowe

Dane wyjściowe będą miały postać zbioru wszystkich maksymalnych klik danego grafu. Spośród nich zostanie wybrana klika (lub kliki) o największym rozmiarze. Każda klika zostanie przedstawiona jako zbiór wierzchołków wchodzących w jej skład.
Wyniki przedstawimy graficznie za pomocą klasy *BasicVisualizationServer* z pakietu edu.uci.ics.jung.visualization [4]. Posługując się zbiorem klas edu.uci.ics.jung.algorithms.layout [4] pokażemy graf wejściowy w jednym z wielu możliwych układów, a każdą znalezioną największą klikę dodatkowo oznaczymy innym kolorem.

3.4 Struktury danych

Każdą znalezioną maksymalną klikę i aktualną listę sąsiadów badanego wierzchołka będziemy przechowywać w obiekcie klasy *HashSet* [5], której wydajność operacji na zbiorach jest największa z dostępnych kolekcji języka Java. Kolejne maksymalne kliki będą dołączane do obiektu klasy *TreeSet* [5] w kolejności od największej do najmniejsze.

3.5 Testy

- W pierwszym kroku będziemy sprawdzać poprawność danych wejściowych, w szczególności czy macierz sąsiedztwa jest symetryczna i czy na jej opadającej przekątnej są same zera. W przypadku wykrycia błędu, program się zakończy a użytkownikowi zostanie przedstawiony odpowiedni komunikat.
- Z uwagi na brak dostępności głównego źródła algorytmu, nie jesteśmy w stanie dowieść jego całkowitej poprawności. Dlatego posłużymy się testem, w którym dla różnych grafów wejściowych znajdziemy rozwiązanie metodą brute-force i porównamy ich jakość z wynikami algorytmu [1].
- Przetestujemy zachowanie aplikacji zarówno dla grafów spójnych jak i niespójnych.

4. Literatura

1. Shuji Tsukiyama, Miko Ide, Hiromu Ariyoshi and Isao Shirakawa - "A New Algorithm for Generating All the Maximal Independent Sets", SIAM J. Comput., Sep. 1977, strony 505 - 517.
2. Frédéric Cazals, Chinmay Karande - "Reporting maximal cliques: new insights into an old problem", TIAS algorytm strony 11-13 (ftp://ftp-sop.inria.fr/abs/fcazals/papers/rr_mbk_v1.pdf)
3. Kazuhisa Makino, Takeaki Uno - "New Algorithms for Enumerating All Maximal Cliques" (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.705&rep=rep1&type=pdf)
4. JUNG - Java Universal Network/Graph Framework - http://jung.sourceforge.net/
5. Dokumentacja Java SE 6 - http://java.sun.com/javase/6/docs/api