

Poszukiwanie największej kliki w grafie

Anna Stępień
Adam Stelmaszczyk

Spis treści

1	Zadanie	2
2	Założenia	2
2.1	Dane wejściowe	2
2.2	Dane wyjściowe	3
2.3	Sytuacje wyjątkowe	3
3	Algorytm	3
3.1	Pseudokod	3
3.2	Opis działania	4
3.3	Analiza złożoności	6
4	Struktury danych	6
5	Testy	6
5.1	Badanie poprawności zwracanych wyników	6
5.2	Badanie czasu wykonania dla różnych typów grafów	7

1 Zadanie

Kliką grafu nazywamy podgraf, w którym każde dwa wierzchołki są ze sobą połączone. Maksymalną kliką nazywamy klikę, do której nie można dodać ani jednego wierzchołka więcej, tak aby razem z nią nadal tworzył klikę. Największą kliką nazywamy klikę o największej liczbie wierzchołków. Celem zadania jest implementacja wybranego algorytmu znajdującego największą klikę w grafie oraz analiza otrzymanych wyników.

2 Założenia

Realizowana aplikacja będzie pracowała w trybie konsolowym, z ewentualną możliwością specyfikacji dodatkowych parametrów. W projekcie zostanie wykorzystany zmodyfikowany algorytm Brona–Kerboscha [1], dokładniej opisany w sekcji 3. Do implementacji zadania wykorzystany zostanie język Java.

2.1 Dane wejściowe

Wejściem dla algorytmu jest graf nieskierowany dany macierzą o n wierszach i n kolumnach:

$$\begin{array}{cccc} q_{0,0} & q_{1,0} & \cdots & q_{n-1,0} \\ q_{0,1} & q_{1,1} & \cdots & q_{n-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{0,n-1} & q_{1,n-1} & \cdots & q_{n-1,n-1} \end{array}, q_{i,j} \in \{0, 1\}, 0 \leq i, j < n$$

$q_{i,j}$ równe 0 oznacza, że wierzchołki i oraz j nie są połączone krawędzią. W przeciwnym razie, wierzchołki są połączone.

Macierz jest dana w pliku tekstowym, w formacie FullMatrix¹, w którym pierwszy wiersz specyfikuje liczbę wierszy macierzy, następny informuje o początku danych, natomiast kolejne wiersze definiują właściwą macierz sąsiedztwa grafu.

Kolejne $q_{i,j}$ w wierszu j są oddzielone co najmniej jednym znakiem białym. Przez znak biały rozumiemy spację lub tabulator. $q_{i,j}$ różne od 0 będą traktowane jak 1.

¹<http://www.analytictech.com/networks/dataentry.htm>

2.2 Dane wyjściowe

Wyjściem jest niepusty zbiór numerów wierzchołków, które tworzą największą klikę w podanym grafie. Wierzchołki numerujemy od 0 do $n - 1$. W grafie może istnieć więcej niż jedna największa klika. W takim przypadku algorytm zwróci pierwszą ze znalezionych klik.

2.3 Sytuacje wyjątkowe

Problemami, które mogą wystąpić podczas działania aplikacji są:

- błędny format danych wejściowych,
- przepełnienie stosu spowodowane zbyt głębokim poziomem rekurencji.

W przypadku, gdy algorytm otrzyma na wejściu błędne dane np. liczba wierszy macierzy będzie niezgodna z zadeklarowaną na początku pliku z danymi, użytkownik zostanie poinformowany o zaistniałej sytuacji a dalsze działanie programu zostanie przerwane.

Ze względu na rekurencyjny charakter algorytmu Brona-Kerboscha może się zdarzyć, iż dla pewnych danych wejściowych algorytm nie będzie w stanie zwrócić wyniku ze względu na ograniczoną pojemność stosu. Próba rozwiązania tego problemu mogłaby być iteracyjna implementacja algorytmu.

3 Algorytm

Algorytm Brona-Kerboscha jest rekurencyjnym algorytmem z nawrotami, który umożliwia poszukiwanie maksymalnych klik w zadanym grafie nieorientowanym.

Domyślnie algorytm zwraca wszystkie maksymalne kliki. W algorytmie wprowadzona zostanie zmiana, dzięki której zwracana będzie największa ze znalezionych maksymalnych klik, charakteryzująca się największą liczbą wierzchołków.

3.1 Pseudokod

Na poniższym listingu przedstawiona została podstawowa wersja algorytmu Brona-Kerboscha.

Algorithm 1 Algorytm Brona–Kerboscha (wersja podstawowa)

```
1:  $compsub \leftarrow \emptyset$ 
2:  $candidates \leftarrow V(G)$ 
3:  $not \leftarrow \emptyset$ 
4:  $cliques \leftarrow \emptyset$ 
5: function BRON_KERBOSCH( $compsub, candidates, not$ )
6:   if  $candidates = \emptyset$  and  $not = \emptyset$  then
7:      $cliques \leftarrow cliques \cup \{compsub\}$  ▷ Maksymalna klika
8:   else
9:     for each  $v$  in  $candidates$  do
10:       $candidates \leftarrow candidates \setminus \{v\}$ 
11:       $new\_compsub \leftarrow compsub \cup \{v\}$ 
12:       $new\_candidates \leftarrow candidates \cap neighbors(v)$ 
13:       $new\_not \leftarrow not \cap neighbors(v)$ 
14:      BRON_KERBOSCH( $new\_compsub, new\_candidates, new\_not$ )
15:       $compsub \leftarrow compsub \cup \{v\}$ 
16:   end for
17: end if
18: end function
```

3.2 Opis działania

Istotą działania przedstawionego algorytmu jest utrzymywanie trzech rozłącznych zbiorów: $compsub$, $candidates$ oraz not .

Algorytm Brona–Kerboscha znajduje maksymalne kliki składające się ze wszystkich wierzchołków należących do zbioru $compsub$, niektórych należących do zbioru $candidates$, i z żadnego, który należy do zbioru not .

Poniżej przedstawiona została charakterystyka każdego ze zbiorów wykorzystywanych przez algorytm:

- $compsub$
do zbioru należą wszystkie wierzchołki grafu, które tworzą powstającą klikę.
- $candidates$
do zbioru należą wierzchołki grafu, które mogą posłużyć do rozszerzenia zbioru $compsub$.
- not
do zbioru należą te wierzchołki, które były już wcześniej wykorzystane do rozszerzenia zbioru $compsub$.

Algorithm 2 Algorytm Brona–Kerboscha (wersja rozszerzona)

```
1:  $compsub \leftarrow \emptyset$ 
2:  $candidates \leftarrow V(G)$ 
3:  $not \leftarrow \emptyset$ 
4:  $biggest\_clique \leftarrow \emptyset$ 
5: function BRON_KERBOSCH( $candidates, not$ )
6:   if  $candidates = \emptyset$  and  $not = \emptyset$  then
7:     if  $size(biggest\_clique) < size(compsub)$  then
8:        $biggest\_clique \leftarrow compsub$  ▷ Największa klika
9:     end if
10:  else
11:     $pivot \leftarrow vertex\_with\_maxdeg(candidates \cup not)$ 
12:     $candidates\_to\_check \leftarrow candidates \setminus neighbors(pivot)$ 
13:    for each  $v$  in  $candidates\_to\_check$  do
14:       $compsub \leftarrow compsub \cup \{v\}$ 
15:       $candidates \leftarrow candidates \setminus \{v\}$ 
16:       $new\_candidates \leftarrow candidates \cap neighbors(v)$ 
17:       $new\_not \leftarrow not \cap neighbors(v)$ 
18:      BRON_KERBOSCH( $new\_candidates, new\_not$ )
19:       $compsub \leftarrow compsub \setminus \{v\}$ 
20:       $not \leftarrow not \cup \{v\}$ 
21:    end for
22:  end if
23: end function
```

Należy zauważyć, iż wszystkie wierzchołki, które są połączone z każdym wierzchołkiem należącym do zbioru $compsub$ znajdują się albo w zbiorze $candidates$ albo not .

Zmodyfikowana wersja algorytmu Brona–Kerboscha wprowadza pojęcie wierzchołka zwrotnego (dalej oznaczanego $pivot$), który wybierany jest ze zbioru $candidates \cup not$ jako wierzchołek o największym stopniu.

W każdym rekurencyjnym wywołaniu algorytmu rozważane są wierzchołki należące do zbioru $candidates$. Jeśli zbiory $candidates$ i not są puste, sprawdzane jest czy znaleziona maksymalna klika (oparta na wierzchołkach ze zbioru $compsub$) jest większa od największej dotychczas znalezionej kliki. Jeśli tak, to znaleziona klika staje się największą, w przeciwnym wypadku największa klika pozostawiana jest bez zmian.

W przypadku, gdy zbiory $candidates$ i not nie są puste, dla każdego wierzchołka ze zbioru $candidates \setminus neighbors(pivot)$ następuje rekurencyjne wy-

wołanie algorytmu, w którym bieżący wierzchołek v dodawany jest do zbioru *compsub* i usuwany ze zbioru *candidates*, a w zbiorach *candidates* i *not* pozostawiane są tylko te wierzchołki grafu, które są sąsiadami wierzchołka v . Następnie, wierzchołek v jest dodawany do zbioru *not* jako już wykorzystany do rozszerzenia kliku oraz usuwany ze zbioru *compsub*.

Wynikiem działania algorytmu jest zbiór *biggest_clique*, który początkowo inicjowany jest jako zbiór pusty. W przypadku, gdy znaleziona zostanie największa klika, zbiór ten zawiera wierzchołki ją tworzące.

3.3 Analiza złożoności

4 Struktury danych

Graf Do reprezentacji grafu zostanie wykorzystana macierz sąsiedztwa, zaimplementowana jako dwuwymiarowa tablica wartości boolowskich.

Zbiory wierzchołków (*compsub*, *candidates*, *not*, *biggest_clique*) Zbiory przechowujące wierzchołki zostaną zaimplementowane jako klasa *Vertices* dziedzicząca po klasie *TreeSet* języka Java.

5 Testy

Istotną częścią realizowanego zadania jest przeprowadzenie testów związanych zarówno z poprawnością zwracanych wyników jak również wpływem danych wejściowych na czas wykonania algorytmu.

5.1 Badanie poprawności zwracanych wyników

Do weryfikacji poprawności zwracanych przez algorytm wyników zostanie wykorzystana biblioteka *igraph*², która udostępnia m.in funkcję wyznaczającą maksymalne klikę w zadanym grafie. Podczas testowania planujemy wykorzystać dane zwrócone przez bibliotekę *igraph* jako rozwiązania referencyjne, które następnie posłużą do porównania z wynikami otrzymanymi przez zaimplementowany algorytm. Rozwiązanie, a więc największa klika zwrócona przez algorytm jest poprawna wtedy, gdy znajduje się na liście rozwiązań referencyjnych.

Proces generowania rozwiązań referencyjnych oraz porównywania wyników zostanie zautomatyzowany.

²<http://igraph.sourceforge.net/>

5.2 Badanie czasu wykonania dla różnych typów grafów

Z punktu widzenia analizy zaimplementowanego algorytmu istotne jest zbadanie jego zachowania dla różnych typów grafów. W szczególności przeprowadzone zostaną eksperymenty na zestawach grafów o zróżnicowanej gęstości.

Referencje

- [1] Coen Bron, Joep Kerbosch, *Algorithm 457: finding all cliques of an undirected graph*, Communications of the ACM, 16(9): 575–577, 1973.