

# Poszukiwanie największej kliki w grafie

Anna Stępień  
Adam Stelmaszczyk

## Spis treści

<b>1</b>	<b>Zadanie</b>	<b>2</b>
<b>2</b>	<b>Założenia</b>	<b>2</b>
<b>3</b>	<b>Algorytm</b>	<b>2</b>
3.1	Pseudokod . . . . .	3
3.2	Opis działania . . . . .	4
<b>4</b>	<b>Struktury danych</b>	<b>5</b>
<b>5</b>	<b>Testy</b>	<b>6</b>
5.1	Badanie poprawności zwracanych wyników . . . . .	6
5.2	Badanie czasu wykonania dla różnych typów grafów . . . . .	6

# 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

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 którym 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.

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 dowolną z nich.

Realizowana aplikacja będzie pracowała w trybie konsolowym, z ewentualną możliwością specyfikacji dodatkowych parametrów. W projekcie zostanie wykorzystany algorytm Brona–Kerboscha [1].

# 3 Algorytm

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

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

---

**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

```

---

---

**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
```

---

### 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*.

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 wywołanie 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 kliki 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.

## 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

### 5.1 Badanie poprawności zwracanych wyników

### 5.2 Badanie czasu wykonania dla różnych typów grafów

## Referencje

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