# ЛЕКЦІЯ12

## Основні алгоритми розфарбування графів

# БАЗОВІ ВІДОМОСТІ

1. Графи, розглянуті в даній лекції, є ***неорієнтованими*** і такими, що ***не мають петель***.

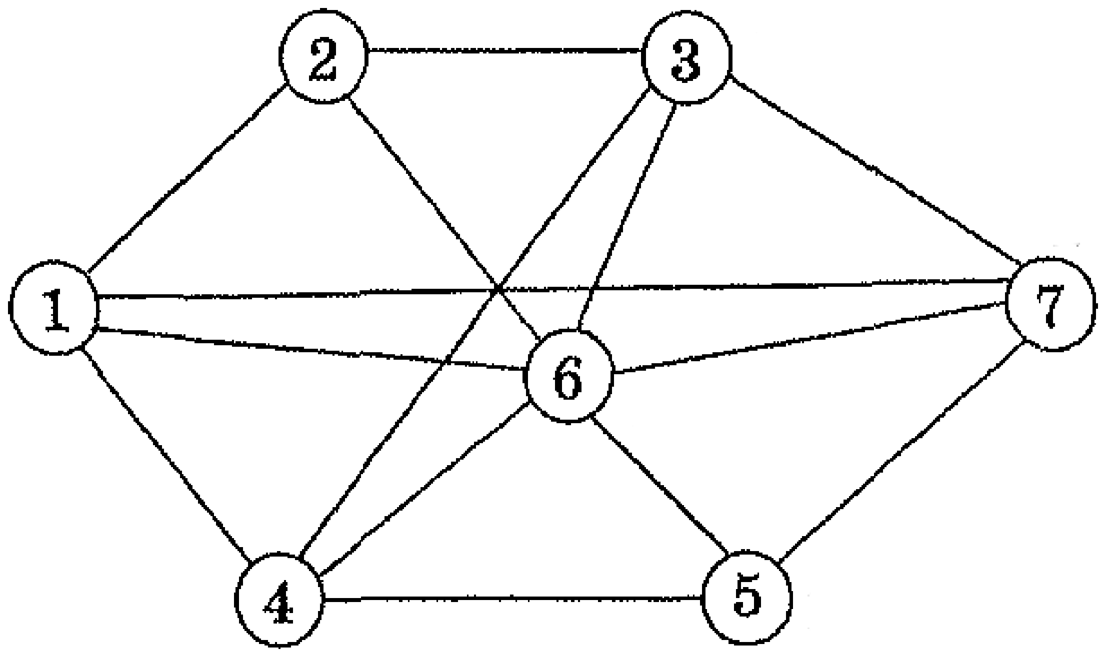
2. Граф $G$ називають r-*хроматичним*, якщо його вершини можуть бути розфарбовані з використанням *r* кольорів (фарб) так, що не найдеться двох суміжних вершин одного кольору. Найменше число *r*, таке, що граф є хроматичним, називають хроматичним числом графа $G$ і позначають $\gamma\left(G\right)$.

3. Задачу знаходження хроматичного числа графа називають задачею про розфарбування (або задачею розфарбування) графа.

4. Повний граф $K_n$ завжди розфарбовується у $n$ кольорів, тобто кількість кольорів дорівнює кількості його вершин.

## ПРИКЛАД АГОРИТМУ ПРЯМОГО НЕЯВНОГО ПЕРЕБОРУ

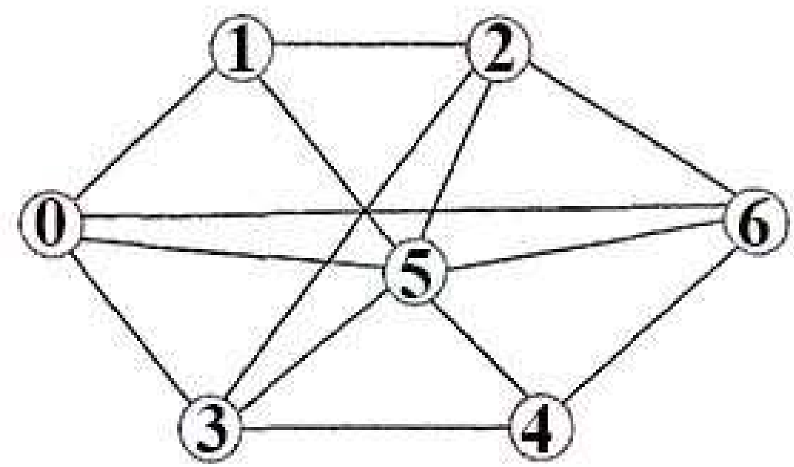
1. Розглянемо граф $G\left(V, E\right)$, який показаний на рисунку.



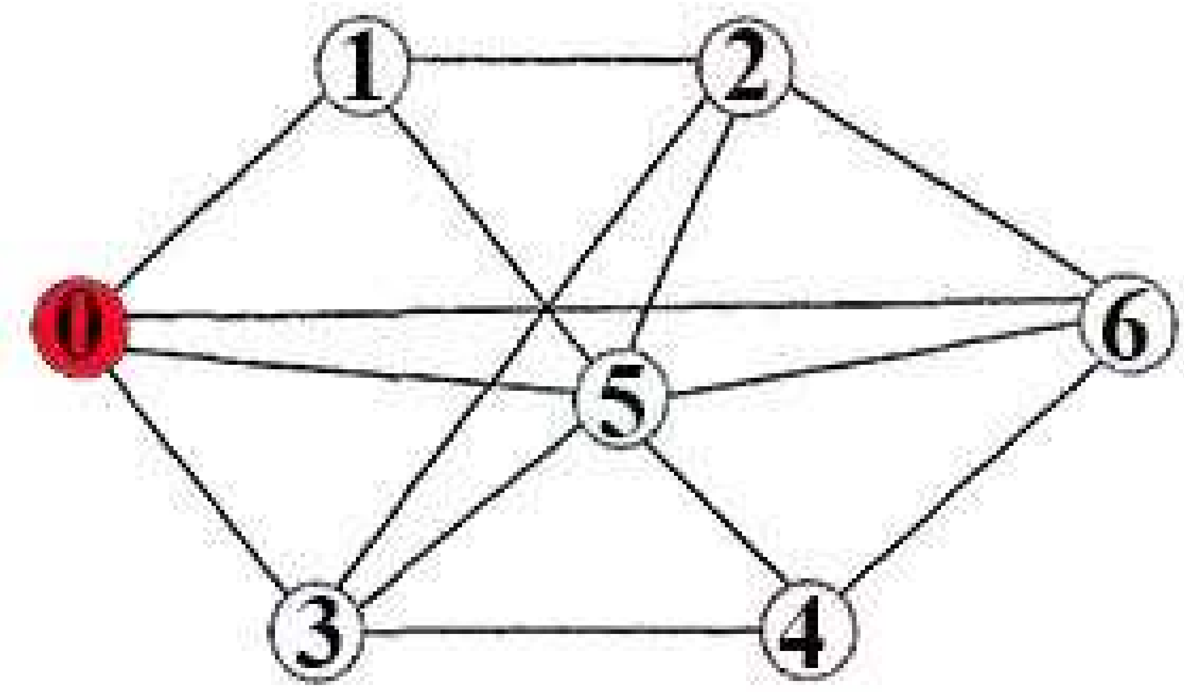
Множину вершин графа $V = \left\{1, 2, 3, 4, 5, 6, 7\right\}$ потрібно розфарбувати з використанням алгоритму послідовного розфарбування.

Сформуємо матрицю суміжності $A$ :

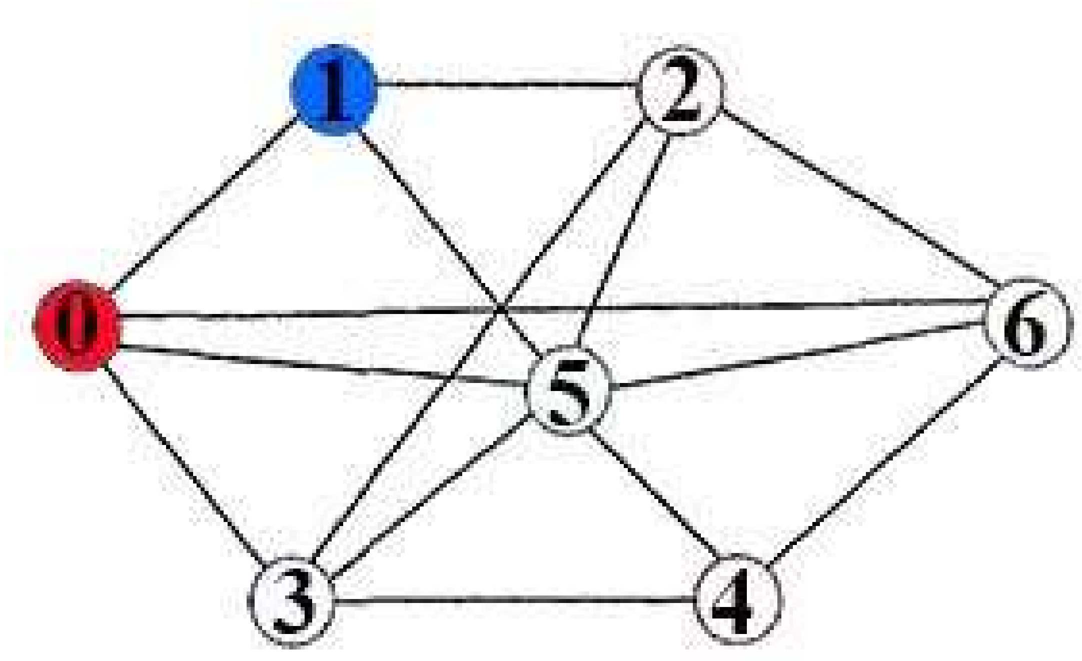
$$A = \begin{array}{c|ccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 2 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 3 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 6 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}.$$



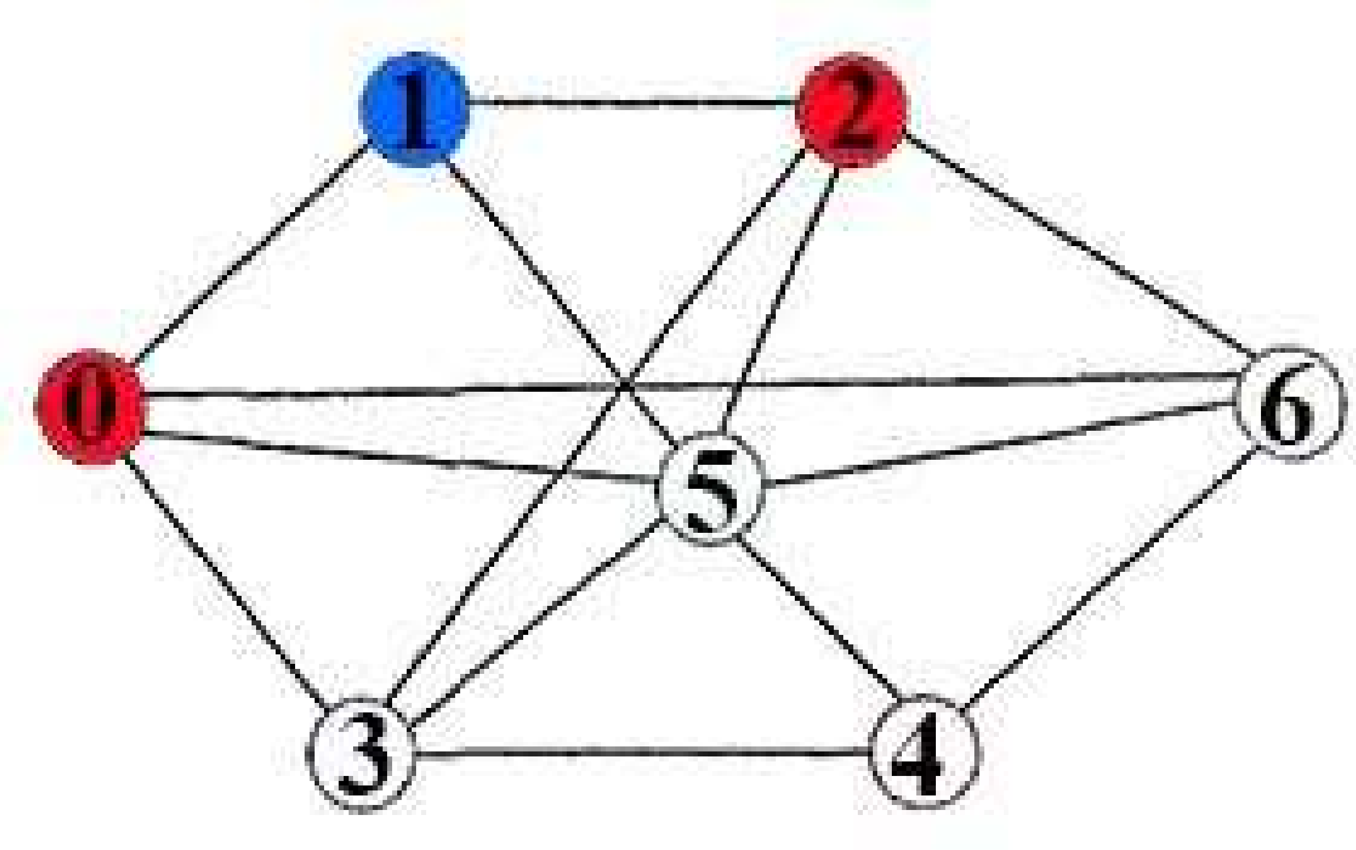
***Крок0***. Розглядаємо вершину 0. Множина розфарбованих суміжних вершин w містить колір 0. Тому функція Color(0) повертає 1. Нехай колір 1- червоний.



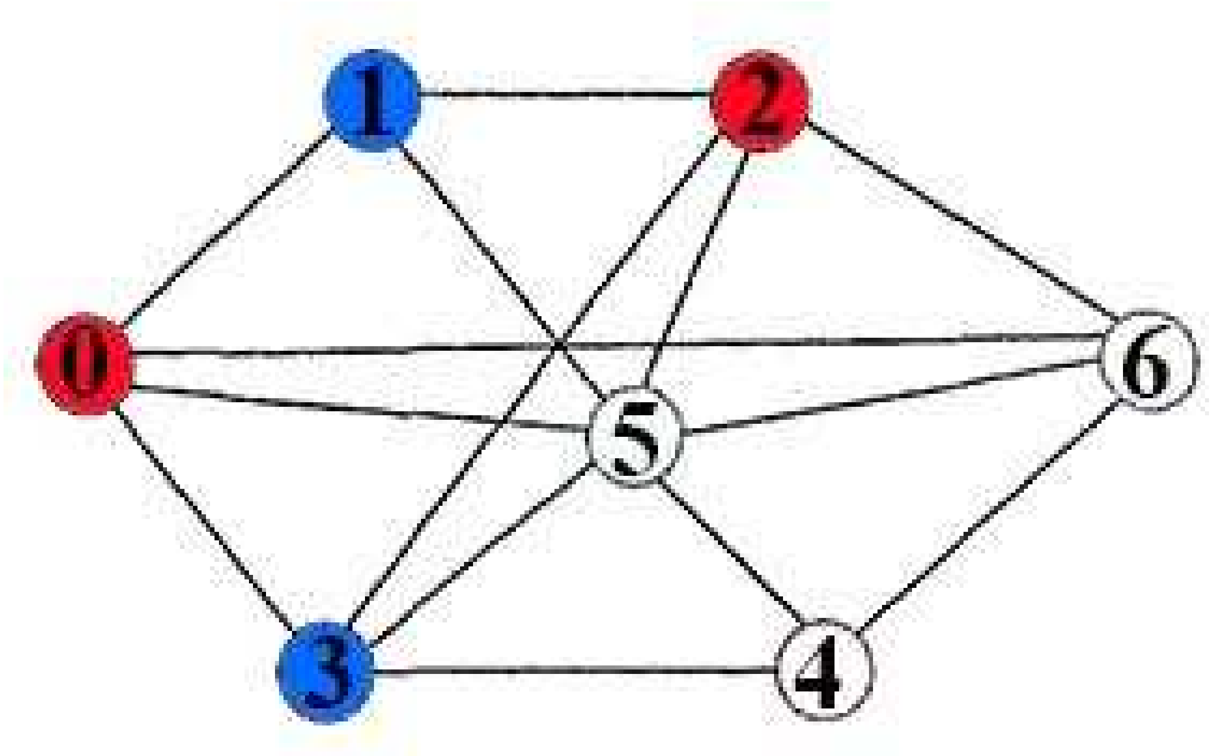
***Крок1***. Розглянемо вершину 1. Єдиною меншою за номером суміжною вершиною є вершина 0, яка уже червона. Тому множина w містить елементи {0,1}. Функція Color(1) повертає наступну за номером фарбу синього кольору: curcol=2.



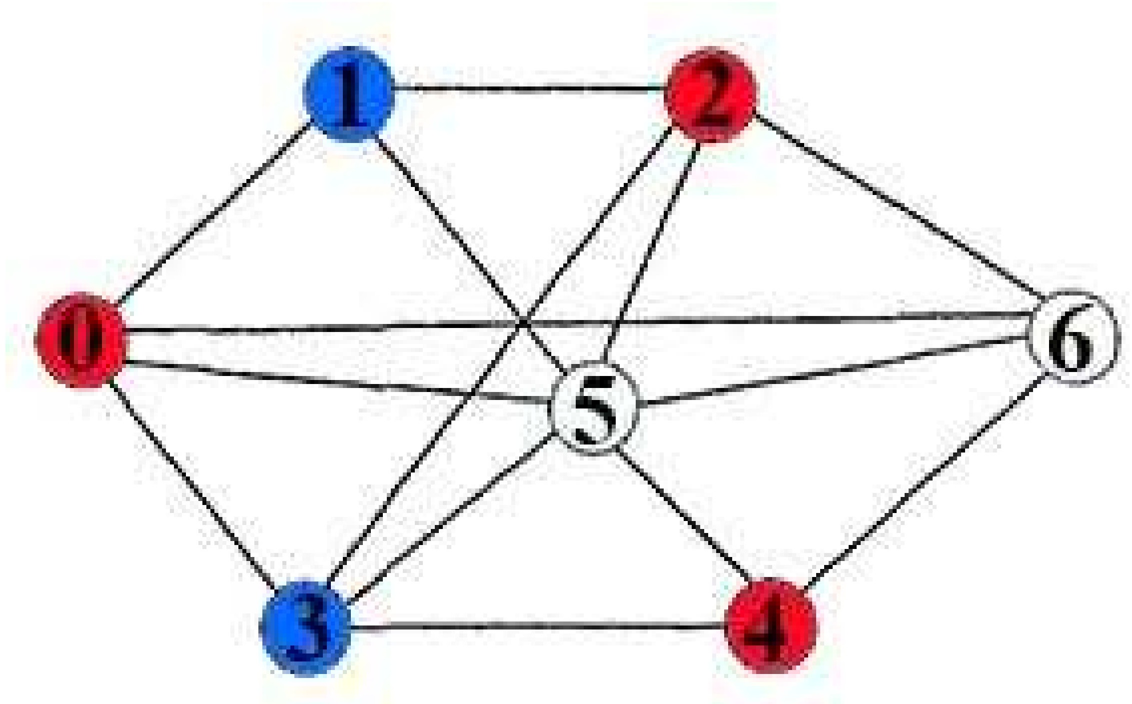
***Крок2***. Вершина 2 має єдину суміжну вершину 1 з меншим номером. Множина w містить елементи {0,2}. Функція Color(2) повертає фарбу з номером 1 червоного кольору.



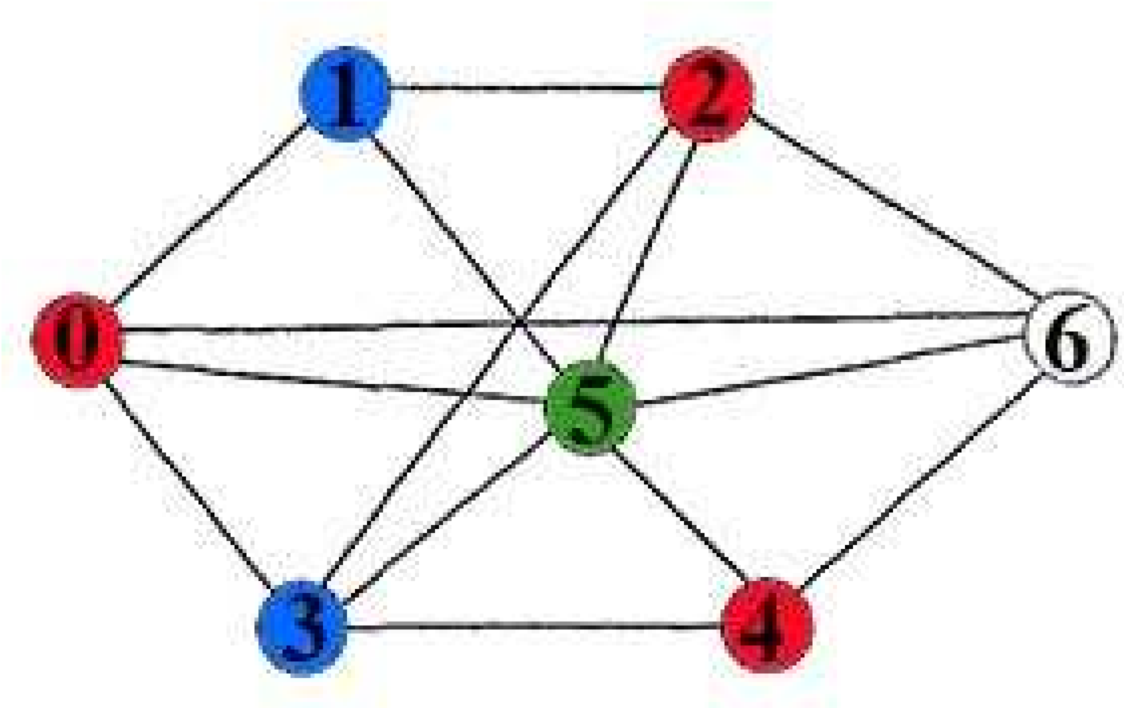
***Крок3***. Вершина 3 має дві суміжні вершини з меншими номерами:0 і 2. Оскільки обидві вершини розфарбовані в колір 1, то множина w містить елементи {0,1}. Тому функція Color(3) повертає наступну за номером фарбу 2 синього кольору.



***Крок4***. Вершина 4 має єдину суміжну вершину з меншим номером. Це вершина 3. Множина w містить елементи {0,2}. Тому функція Color(4) повертає фарбу з номером 1 червоного кольру.

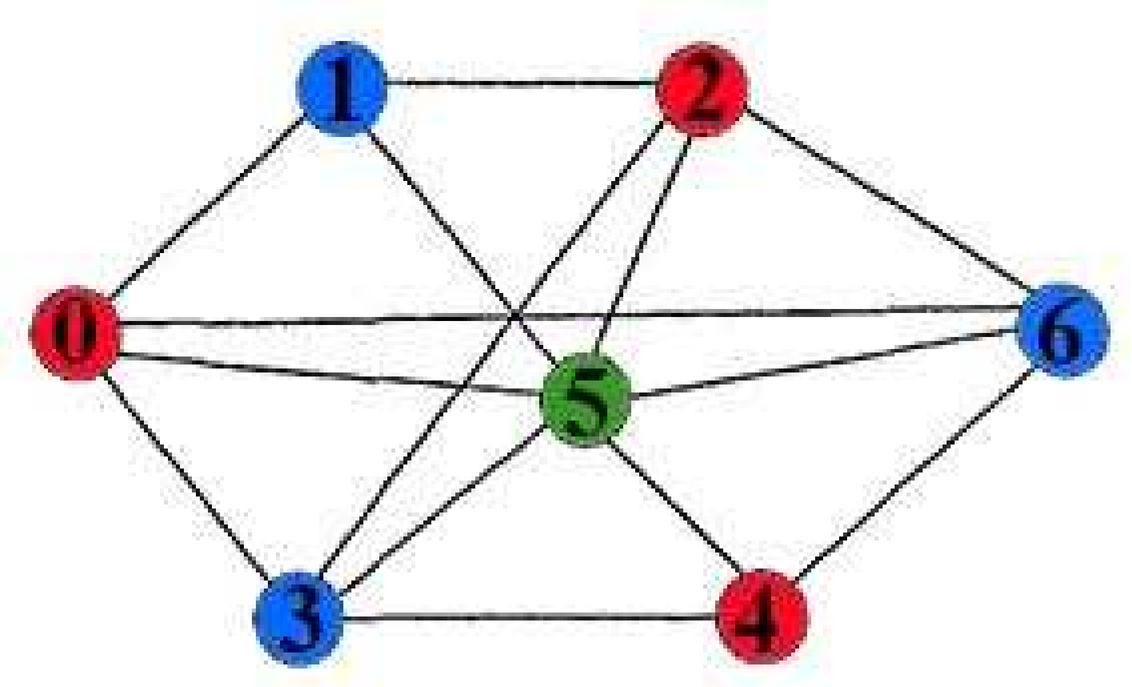


***Крок5***. Вершина 5 має такі суміжні вершини з меншими номерами:0,1,2, 3 і 4. Ці вершини розфарбовані в колір 1 та колір 2. Отже, множина містить елементи: {0,1,2}. Тому функція Color(5) повертає наступну за номером фарбу 3 зеленого кольору.



***Крок6***. Вершина 6 має такі суміжні вершини з меншими номерами:0,2, 4 і 5. Ці вершини розфарбовані в колір 1 та колір 3. Отже, множина w містить два елементи: {0,1,2}. Тому функція Color(6) повертає фарбу 2 синього кольору.

В результаті роботи данного алгоритму одержуємо правильно розфарбований граф, що показаний на рисунку.



## Алгоритм прямого неявного перебору

Алгоритм дозволяє реалізувати правильне розфарбування графа з вибором мінімальної в рамках даного алгоритму кількості фарб.

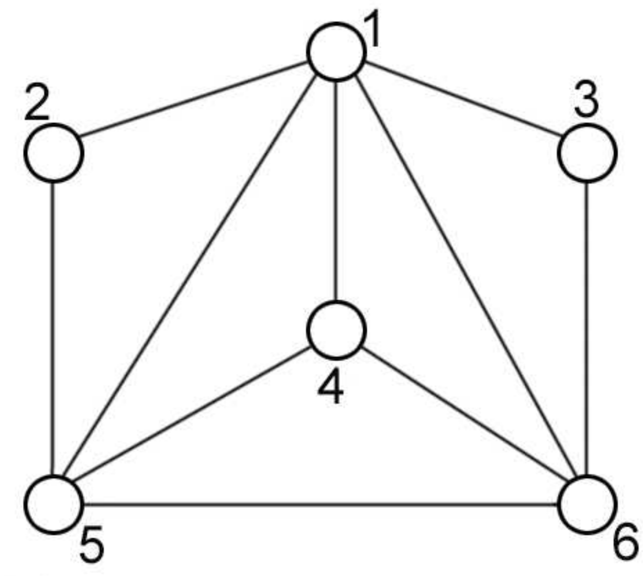
```
from random import *
n = 7  # максимальна кількість вершин графа
# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
# Формування протилежних напрямів
for r in range(n):
    for c in range(n):
        if r < c :
            a[c][r] = randint(0,1)
        else:
            a[c][r]=a[r][c]
```

```
print("Матриця суміжності")
for r in range(n):
    print(a[r])
"""
a=[[0,1,0,1,0,1,1],
   [1,0,1,0,0,1,0],
   [0,1,0,1,0,1,1],
   [1,0,1,0,1,1,0],
   [0,0,0,1,0,1,1],
   [1,1,1,1,1,1,0],
   [1,0,1,0,1,1,0]]
"""
colarr = [0 for i in range(n)]
```

```python
def color(i):
    # Функція вибору фарби для розфарбування вершини з номером i
    w = {0}
    for j in range(i):
        if a[j][i] > 0: w.add(colarr[j])
    curcol = 0
    while True:
        curcol += 1
        if curcol not in w: break
    return curcol

print("Список кольорів")
for i in range(n):
    colarr[i] = color(i)
print(colarr)
```

# ПРИКЛАД ЕВРИСТИЧНОГО АЛГОРИТМУ РОЗФАРБУВАННЯ

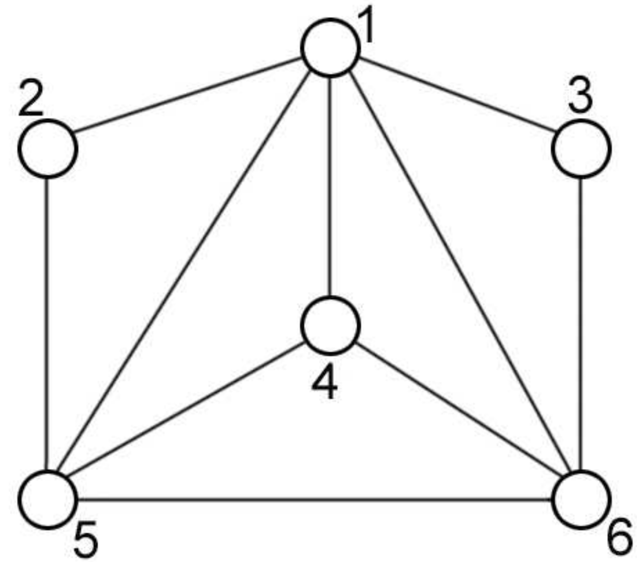
Дано граф $G$, зображений на рисунку.




Множину вершин графа $V = \{1, 2, 3, 4, 5, 6\}$ потрібно розфарбувати з використанням евристичного алгоритму розфарбування.

Сформуємо матрицю суміжності $A$ :

$$A = \begin{array}{cc} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$



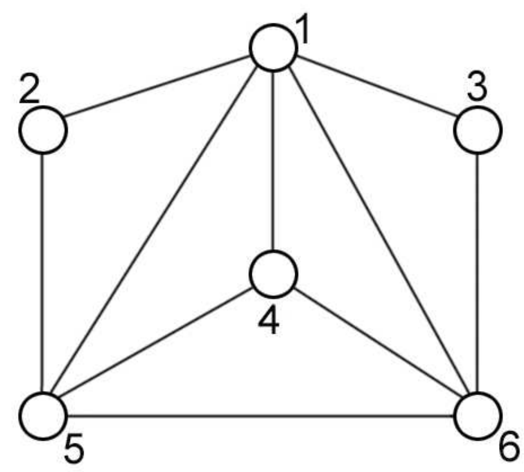

Сформуємо список списків. Перший елемент вкладеного списку відповідає степеню вершини, а другий містить номер вершини. Посортуємо за першими елементами вкладеного списку по спаданню степенів вершин.

| Index | | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| SortArr | 0 | 5 | 4 | 4 | 3 | 2 | 2 |
| | 1 | 1 | 5 | 6 | 4 | 2 | 3 |

1. Рухаючись по номерах вершин вкладених списків *SortArr* першій знайденій нерозфарбованій вершині надаємо черговий новий колір.
2. Всім несуміжним зі знайденою вершинам надаємо цей же колір.

У першому рядку таблиці розташуємо степені вершин зі списків SortArr , у другому – номери відповідних вершин. Наступні рядки відображають вміст вектора розфарбування.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Степені вершин DegArr | | 5 | 4 | 4 | 3 | 2 | 2 |
| Номери вершин SortArr | | 1 | 5 | 6 | 4 | 2 | 3 |
| $\mathrm{CurCol} = 1$ | | 1 | - | - | - | - | - |
| $\mathrm{CurCol} = 2$ | | 1 | 2 | - | - | - | 2 |
| $\mathrm{CurCol} = 3$ | | 1 | 2 | 3 | - | 3 | 2 |
| $\mathrm{CurCol} = 4$ | | 1 | 2 | 3 | 4 | 3 | 2 |



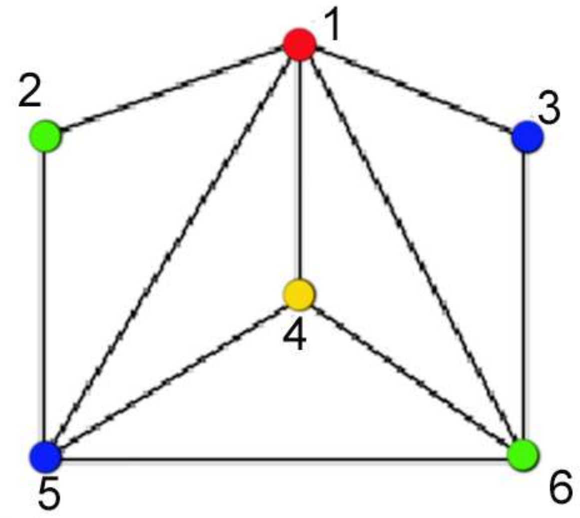
**Крок 1.** Першою SortArr стоїть вершина 1, яку фарбуємо червоним кольором 1. Несуміжних з 1 немає.

**Крок 2**. Другою в SortArr стоїть вершина 5, яку фарбуємо синім кольором 2. Несуміжна з 5 вершина 3, яку процедура dyer(curcol,5) фарбує також синім кольором 2.

**Крок 3.** Третьою в SortArr стоїть вершина 6, яку фарбуємо зеленим кольором 3. Несуміжна з 6 вершина 2, яку процедура dyer(curcol,6) фарбує також зеленим кольором 3.

**Крок 4.** Четвертою в SortArr стоїть вершина 4, яку фарбуємо жовтим кольором 4. Всі несуміжні вершини з 4 вже розфарбовані

$$A = \begin{array}{cc} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$



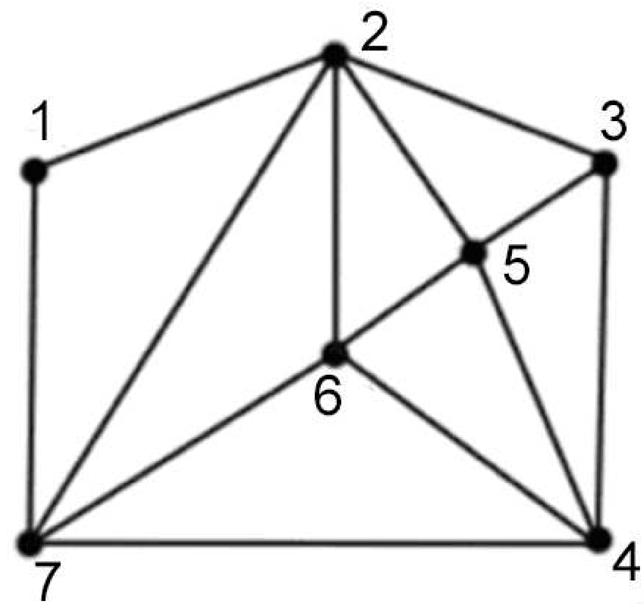
```python
# Код евристичного алгоритму

from random import *

n = 5 # максимальна кількість вершин графа

# Початковий колір
curcol=1

# Список кольорів виршин
colarr=[0 for i in range(n)]

# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
```

```python
# Формування списку за степенями
def degforming():
    def getkey(item):
        return item[0]

    # Список кортежей (степінь, номер вершини)
    degarr=[[0 for i in range(2)] for j in range(n)]
    for i in range(n):
        for j in range(n):
            degarr[i][0] += a[i][j]
            degarr[i][1] = i


    # Сортуємо кортежі за спаданням степенів
    degarr.sort(key=getkey, reverse=True)
    return degarr
```

```
# Розфарбовка вершин
def dyer(curcol,node):
    for k in range(n):
        if a[node][k]==0:
            if colarr[k]==0:colarr[k]= curcol

# Основний код
sortarr=degforming()

for i in range(n):
    if not colarr[sortarr[i][1]]:
        colarr[sortarr[i][1]]=curcol
        dyer(curcol,sortarr[i][1])
        curcol+=1

              for r in range(n): print(a[r])
                      print(sortarr)
                       print(colarr)
```

## Модифікований евристичний алгоритм розфарбування.

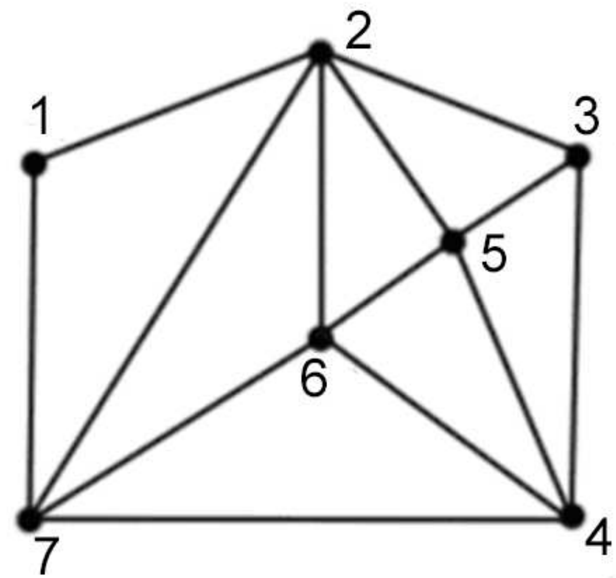
Дано граф $G$, зображений на рисунку




Множину вершин графа $V = \{1, 2, 3, 4, 5, 6, 7\}$ потрібно розфарбувати з використанням модифікованого евристичного алгоритму розфарбування.

Сформуємо матрицю суміжності $A$ :

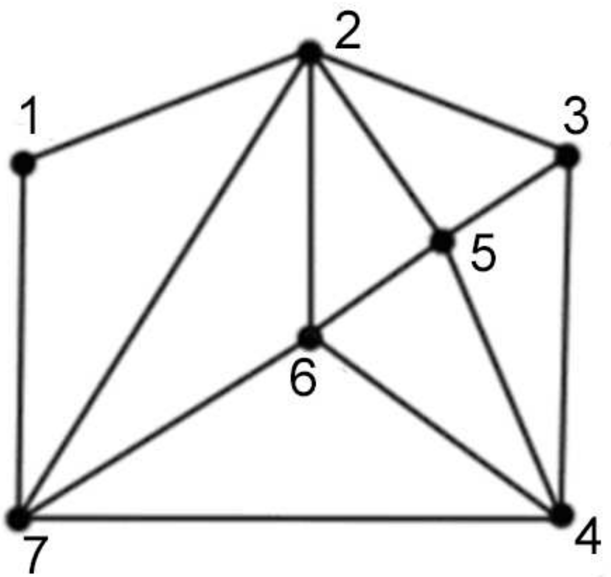
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$




1. Модифікуємо процедуру формування списку степенів вершин degarr для врахування двокрокових степенів.
2. Відсортуємо вершини графа за незростанням їх перших та других степенів. Як результат, отримуємо вектор відсортованих вершин sortarr

Вектор степенів відсортованих вершин має наступний вигляд:

$$D = (5, 4, 4, 4, 4, 3, 2)$$




Вектор других степенів відсортованих вершин має наступний вигляд:

$$D^2 = (17, 17, 16, 15, 15, 13, 09)$$

Сформуємо список списків. Перший елемент вкладеного списку відповідає степеню вершини і формується за формулою:

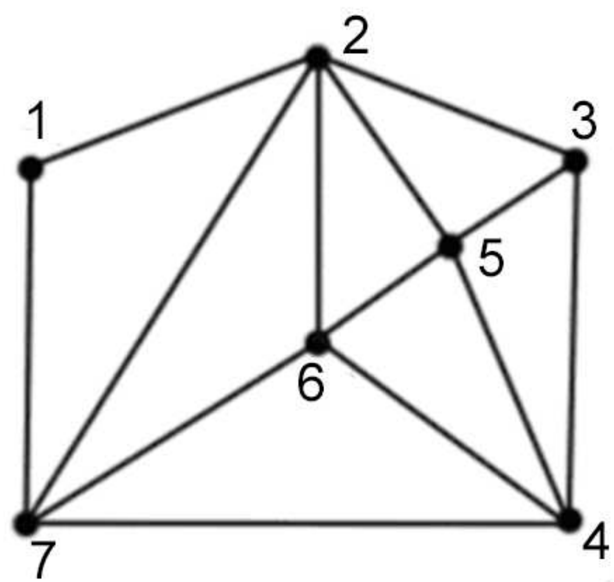
sortarr[i][0]:=$D$[i]*100+$D^2$[i],

а другий містить номер вершини.

Посортуємо за першими елементами вкладеного списку по спаданню степенів вершин.

У результаті отримуємо вектор відсортованих вершин у вигляді других елементів списку sortarr

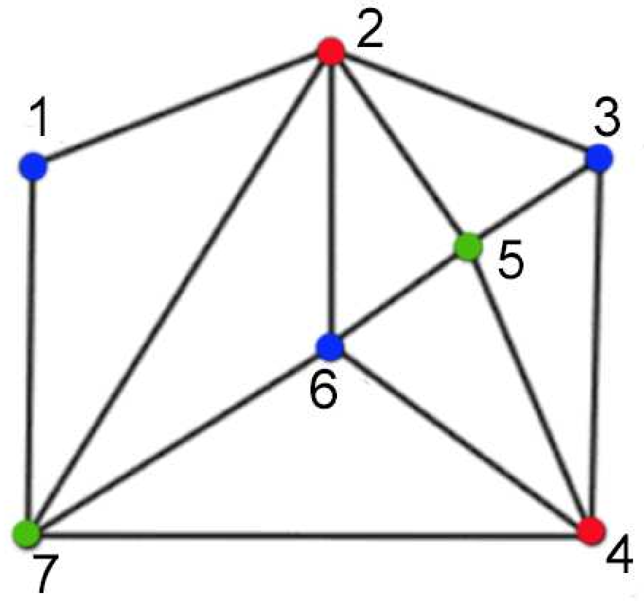
| Номери вершин $X^*$ | 2 | 6 | 5 | 4 | 7 | 3 | 1 |
|---|---|---|---|---|---|---|---|
| Степінь вершин $D$ | 5 | 4 | 4 | 4 | 4 | 3 | 2 |
| Двокроковий степінь $D^2$ | 17 | 17 | 16 | 15 | 15 | 13 | 9 |
| sortarr[i,0] | 517 | 417 | 416 | 415 | 415 | 313 | 209 |
| curcol=1 | 1 | - | - | 1 | - | - | - |
| curcol=2 | 1 | 2 | - | 1 | - | 2 | 2 |
| curcol=3 | 1 | 2 | 3 | 1 | 3 | 2 | 2 |



**Крок 1.** sortarr[0][1]==2 фарбуємо червоним кольором 1. Несуміжна вершина 4. Також фарбуємо червоним.

**Крок 2**. sortarr[1][1]==6 фарбуємо синім кольором 2. Несуміжні 1 та 3 фарбуємо також синім кольором 2.

**Крок 3.** sortarr[2][1]==5 фарбуємо зеленим кольором 3. Несуміжна вершина 7. Фарбуємо також зеленим кольором 3.



## #Код модифікованого евристичного алгоритму

```
from random import *

n = 5 # максимальна кількість вершин графа
#Початковий номер кольору
curcol=1

#Список кольорів вершин
colarr=[0 for i in range(n)]

# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
```

```python
# Функція формування списку упорядкованих степенів
def degforming():
    def getkey(item):
        return item[0]
    def degcount(d): #степень вкршини d
        degnum=0
        for k in range(n):
            degnum += a[k][d]
        return degnum
    degarr=[[0 for i in range(2)] for j in range(n)]
  # Формування degarr
    for j in range(n):
     degarr[j][0]=degcount(j)*100
     degarr[j][1] = j
     for i in range(n):
         if a[i][j]==1:
             degarr[j][0]+=degcount(i)
    # Сортування за спаданням степенів
    degarr.sort(key=getkey, reverse=True)
    return degarr
```

```python
# Функція розфарбування
def dyer(curcol,node):
    for k in range(n):
        if a[node][k]==0:
            if colarr[k]==0:colarr[k]= curcol
# Основний код програми
sortarr=degforming()

for i in range(n):
    if not colarr[sortarr[i][1]]:
        colarr[sortarr[i][1]]=curcol
        dyer(curcol,sortarr[i][1])
        curcol+=1

for r in range(n): # Вивід результатів
    print(a[r])
print(sortarr)
print(colarr)
```

**Рекурсивна процедура послідовного розфарбування**

1. Фіксуємо порядок обходу вершин. Нумеруємо кольори.

2. Ідемо по вершинах, використовуючи такий найменший номер кольору, який не викликає конфліктів.

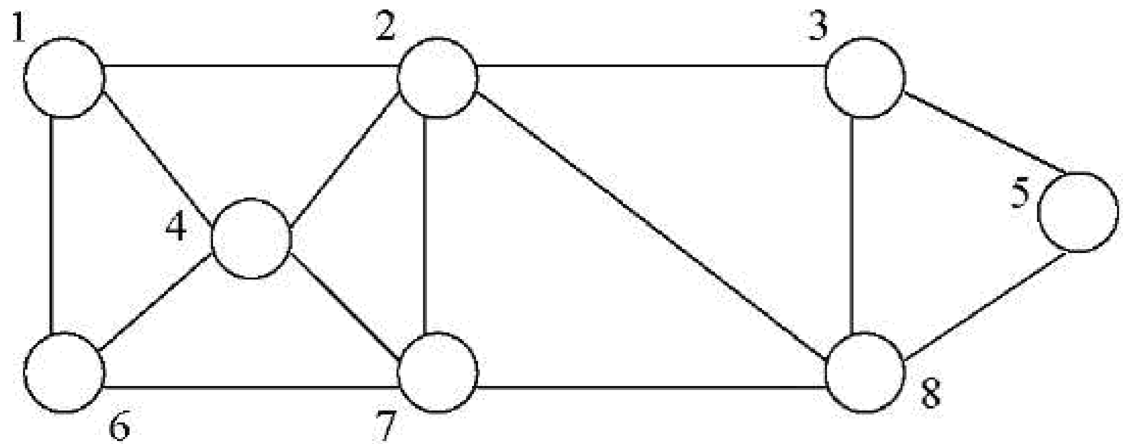
3. Якщо використаний колір вибрати не виходить, то тільки тоді застосовуємо новий колір.

**Особливість даного методу розфарбування:**

У процедурі використовується рекурсивний виклик процедури фарбування наступної вершини у випадку успішного фарбування попередньої вершини.

## ПРИКЛАД РОБОТИ РЕКУРСИВНОЇ ПРОЦЕДУРИ ПОСЛІДОВНОГО РОЗФАРБУВАННЯ

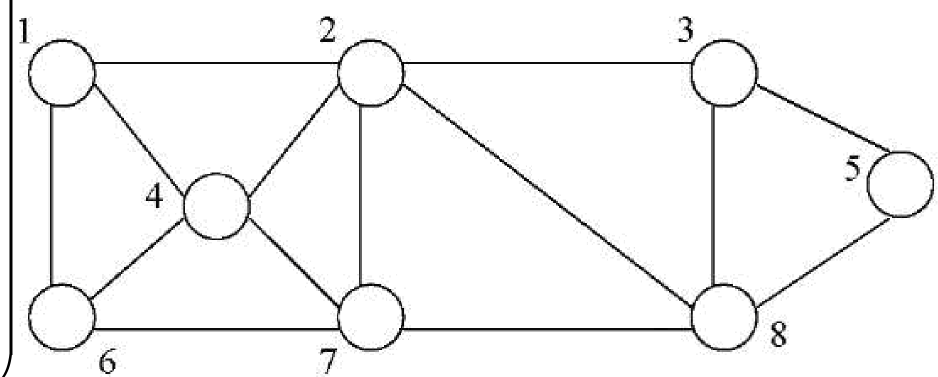
Розглянемо граф $G$:




Множину вершин графа $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ потрібно розфарбувати з використанням алгоритму рекурсивної процедури послідовного розфарбування.

Сформуємо матрицю суміжності $A$ :

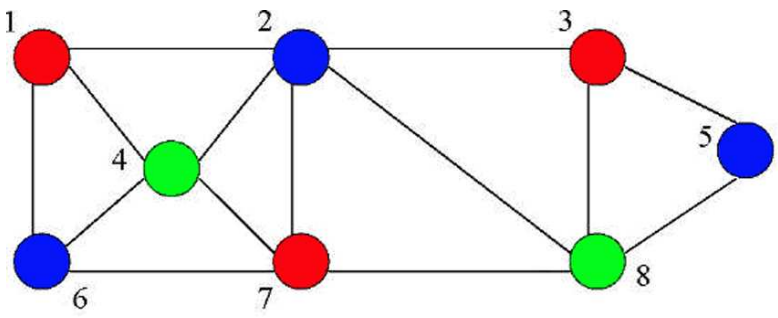
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$




1. Процедуру visit(i) викликаємо рекурсивно для вершин графа.
2. Для кожної вершини знаходимо неконфліктний колір з мінімальним номером за допомогою функції Nicecolor.

Перший стовпчик містить виклики процедури Visit, а решта показує, яка фарба була прийнята, а яка відхилена.

| | Червоний | Синій | Зелений |
|---|---|---|---|
| Visit(1) | + | | |
| Visit(2) | - | + | |
| Visit(3) | + | | |
| Visit(4) | - | - | + |
| Visit(5) | - | + | |
| Visit(6) | - | + | |
| Visit(7) | + | | |
| Visit(8) | - | - | + |



```
#Код рекурсивного алгоритму

from random import *
# Максимально допустима кількість кольорів
cmax=10

n = 5 # максимальна кількість вершин графа

#Список кольорів вершин
color=[0 for i in range(n)]

# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
```

```
def visit (i):

# Функція вибору фарби для розфарбування вершини з номером i

    def nicecolor():
        w = {0}
        newcol=0
        for j in range(n):
           if a[i][j] > 0: w.add(color[j])
        for cm in range(1,cmax):
           if cm not in w:
               newcol=cm
               break
        return newcol
```

```
 # код функції visit
    if  i == n:
       print("FINAL") #Якщо всі вершини розфарбовані, те 
виводимо результат
    else:
       if color[i]==0: #Якщо поточна вершина не розфарбована
        curcol=nicecolor()
        if curcol >0:
            color [i] = curcol #Якщо неконфліктний, то 
розфарб. вершину і фарбою с
            visit (i + 1) #Рекурсивно викликаємо для 
наступної вершини


#Основний код програми
visit (0)
for r in range(n):
    print(a[r])
print()
print(color)
```

## «Жадібний» алгоритм розфарбування

Нехай даний зв'язний граф $G(V,E)$ .

1. Задамо множину $monochrom := \varnothing$, куди будемо записувати всі вершини, які можна пофарбувати одним кольором.

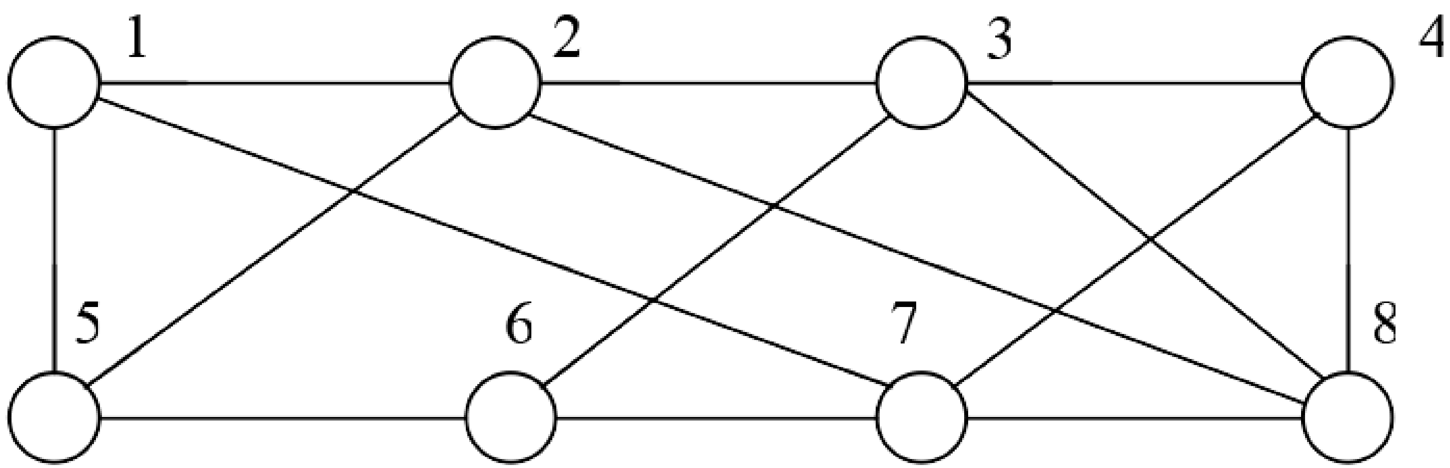
2. Переглядаємо всі вершини й виконуємо такий «жадібний» алгоритм:

**Procedure** Greedy;
**For** ( *Для кожної нерозфарбованої вершини* $v \in V$ ) **do**
**If** $v$ *не суміжна з вершинами з* $monochrom$ **then**
**begin**
$color(v) :=$колір;
$monochrom := monochrom \cup \{v\}$
**end;**

Розглянемо докладніше програмну реалізацію даного алгоритму за умови, що для представлення графа використовують матрицю суміжності.

## ПРИКЛАД РОБОТИ «ЖАДІБНОГО» АЛГОРИТМУ РОЗФАРБУВАННЯ

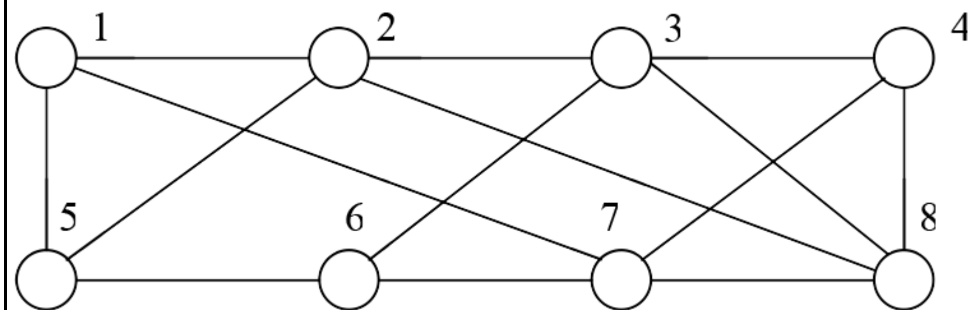
Розглянемо граф $G$:




Множину вершин графа $V = \left\{1,2,3,4,5,6,7,8\right\}$ потрібно розфарбувати з використанням «ЖАДІБНОГО» алгоритму розфарбування.

Сформуємо матрицю суміжності $A$ :

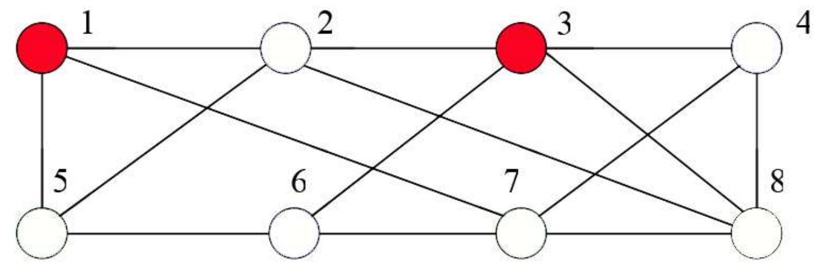
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$



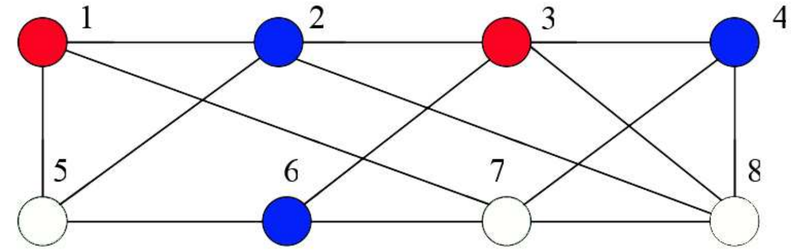

1. Знаходимо нерозфарбовану вершину і встановлюємо для неї новий колір.
2. Запускаємо процедуру «жадібного» розфарбування, яка розфарбовує в цей колір всі вершини, які тільки можливо.
3. Якщо не всі вершини розфарбовані, то переходимо до п.1.

**Крок 1.** Вибираємо вершину 1 і фарбуємо її в червоний колір 1.

Крок 1.1.
2-конфліктна, оскільки суміжна з 1
3-неконфліктна. Фарбуємо в червоний.
4- конфліктна, оскільки суміжна з 3.
5- конфліктна, оскільки суміжна з 1.
6- конфліктна, оскільки суміжна з 3.
7- конфліктна, оскільки суміжна з 1.
8- конфліктна, оскільки суміжна з 3.



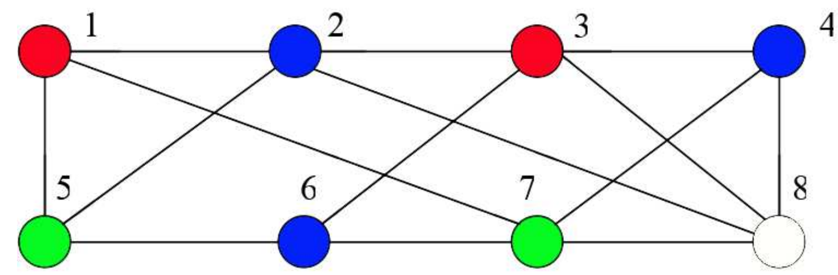

**Крок 2.** Вибираємо вершину 2 і фарбуємо її в синій колір 2.
Крок 2.1.
1-конфліктна, має колір.
3-конфліктна, має колір.
4- неконфліктна. Фарбуємо в синій.
5- конфліктна, оскільки суміжна з 2.
6- неконфліктна. Фарбуємо в синій.
7- конфліктна, оскільки суміжна з 4.
8- конфліктна, оскільки суміжна з 4.




**Крок 3.** Вибираємо вершину 5 і фарбуємо її в зелений колір 3.

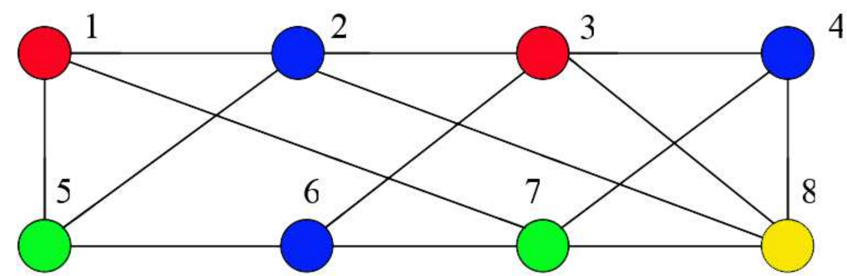
Крок 3.1.

1-конфліктна, має колір.
2-конфліктна, має колір.
3-конфліктна, має колір.
4-конфліктна, має колір.
6-конфліктна, має колір.
7- неконфліктна. Фарбуємо в зелений.
8- конфліктна, оскільки суміжна з 7.




**Крок 4.** Вибираємо вершину 8 і фарбуємо її в жовтий колір 4.

1-конфліктна, має колір.
2-конфліктна, має колір.
3-конфліктна, має колір.
4-конфліктна, має колір.
5-конфліктна, має колір.
6-конфліктна, має колір.
7-конфліктна, має колір.
Кінець алгоритму.



```
# код жадібного алгоритму
from random import *
n=5

allcolored = False #Ознака того, що всі вершини не розфарбовані
color=0
#Список кольорів вершин
colarr=[0 for i in range(n)]

# Генерація симетричної матриці
a = [[0 for i in range(n)] for j in range(n)]
for r in range(n): # n рядків
    for c in range(n): # в кожному рядку по n елементів
        if r > c:
            a[r][c]= randrange(0,2) # додаємо ребро
            a[c][r]=a[r][c] # протилежний напрям
```

```
def  avid (i,color):  #Функція вибору фарби для
розфарбування вершини з номером і
   def  check (i): #Перевірка кольору суміжних вершин
       ch = True
       for j in range(n):
           if a [i][j] == 1: #Якщо вершина j суміжна з
тією, що підлягає перевірці
               if (j in w): ch = False
       return ch
   w = set()  # Очищаємо множину одноколірних вершин
   colarr [i]=color #Розфарбовуємо першу вершину новою
фарбою
   w.add(i) #Доповнюємо множину одноколірних вершин
вершиною
   #Перевіряємо інші вершини на можливість
розфарбування цією фарбою *}
   for k in range(n):
       if colarr[k] == 0:
          if check(k):
```

```
              colarr [k] = color
              w.add(k)


# Головний код
while not allcolored: #Цикл по вершинах графа
   allcolored = True
   for i in range(n):
       if colarr [i] == 0: #Знайшли не розфарбовану вершину
           color += 1  # Встановлюємо новий колір
           allcolored = False
           avid (i,color) #процедура жадібного розфарбування

for r in range(n):  # 6 строк
   print(a[r])
print()
print(colarr)
```

# Ще один жадібний алгоритм

```
gr = [[0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0],
     [0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0],
     [1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
     [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
     [1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0],
     [1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0],
     [0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
     [0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0],
     [0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0],
     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
def GreedyColor():
    color = [[]] # список монохромів
    numc = 0  # номер кольору
    colored = []  # розфарбовані вершини
    n1 =-1
    # перевіряємо, чи підходить поточний клір
    def check(n):
        for i in color[numc]:
```

```
            if gr[i][n] == 1:
                return False
        return True

    for j in gr:
        n1 += 1
        if n1 not in colored:  # переглядаємо нерозфарбовані 
вершини
            if not check(n1):  # якщо не підходить то змінюємо
                color.append([])
                numc += 1
                colored.append(n1)
                color[numc].append(n1)
            n2 = n1
            for k in j[n1:]:
                if k == 0 and n2 not in colored and check(n2):
                    colored.append(n2)
                    color[numc].append(n2)
                n2 += 1
    return color

print(GreedyColor())
```

## Розфарбування з використанням пакета NetworkX

Пакет NetworkX дозволяє виконати правильне розфарбування графів з використанням різних стратегій. Найчастіше використовують стратегії на основі жадібних алгоритмів.
Розглянемо приклад розфарбування графа з використанням пакету NetworkX

```
import networkx as nx
import matplotlib.pyplot as plt
G=nx.Graph()

colors = ['Red', 'Blue', 'Green', 'Yellow',  'Black',
'Pink', 'Orange', 'White', 'Gray', 'Purple', 'Brown',
'Navy']
```

```
G.add_nodes_from([1,2,3,4,5,6])
G.add_edges_from([(1,5),(1,3),(1,2),(1,4),(4,5),(2,3),(1,6),
(6,5),(6,3),(6,2),(4,3)])
colors_of_nodes={}


def coloring(node, color):
      for neighbor in G.neighbors(node):
          color_of_neighbor = colors_of_nodes.get(neighbor,
None)
          if color_of_neighbor == color:
             return False

      return True

def get_color_for_node(node):
       for color in colors:
          if coloring(node, color):
             return color
```

```
def main():
       color_map=[]
       for node in G.nodes():
           colors_of_nodes[node] = get_color_for_node(node)
           color_map.append(colors_of_nodes[node])
       print(colors_of_nodes)
       nx.draw(G, node_color=color_map, node_size=800,
with_labels=True)
       plt.show()

main()
```