Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie
AGH University of Science
and Technology

AGH

Grafy i ich zastosowania

Zestaw 2

Elzbieta.Strzalka@fis.agh.edu.pl
p. 232/D-10

Zestaw 2, zadanie 1

Napisać program do sprawdzania, czy dana sekwencja liczb naturalnych jest ciągiem graficznym, i do konstruowania grafu prostego o stopniach wierzchołków zadanych przez ciąg graficzny.

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach:

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n  OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
    if $\forall_i\ A[i] = 0$ **then**
        return TRUE
    end if
    if $A[0] \geqslant n$ OR $\exists_i\ A[i] < 0$ **then**
        return FALSE
    end if
    for $(i \leftarrow 1;\ i \leqslant A[0];\ i \leftarrow i+1)$ **do**
        $A[i] \leftarrow A[i] - 1$
    end for
    $A[0] \leftarrow 0$
    Posortuj tablicę A nierosnąco
end while

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
    if $\forall_i \; A[i] = 0$ **then**
        return TRUE
    end if
    if $A[0] \geqslant n$ OR $\exists_i \; A[i] < 0$ **then**
        return FALSE
    end if
    for $(i \leftarrow 1; \; i \leqslant A[0]; \; i \leftarrow i + 1)$ **do**
        $A[i] \leftarrow A[i] - 1$
    end for
    $A[0] \leftarrow 0$
    Posortuj tablicę A nierosnąco
end while

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	**3**	**3**	**2**	**2**	1	1
0	**2**	**2**	**1**	**1**	1	1

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
  if $\forall_i\ A[i] = 0$ **then**
    return TRUE
  end if
  if $A[0] \geqslant n$ OR $\exists_i\ A[i] < 0$ **then**
    return FALSE
  end if
  for $(i \leftarrow 1;\ i \leqslant A[0];\ i \leftarrow i + 1)$ **do**
    $A[i] \leftarrow A[i] - 1$
  end for
  $A[0] \leftarrow 0$
  Posortuj tablicę A nierosnąco
end while

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	2	1	1	1	1	0

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	**2**	**1**	1	1	1	0
0	**1**	**0**	1	1	1	0

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ≥ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ≤ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	**1**	1	1	0	0	0
0	**0**	1	1	0	0	0

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0
1	1	0	0	0	0	0

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0
1	1	0	0	0	0	0
0	**0**	0	0	0	0	0

Ciąg graficzny

- **Ciąg graficzny** – ciąg stopni wierzchołków pewnego **grafu prostego**.
- **Stopień wierzchołka** v: $\deg(v)$ – liczba jego sąsiadów.
- Liczba wierzchołków o nieparzystym stopniu w grafie prostym jest parzysta.
- **Suma stopni** w grafie o n wierzchołkach i k krawędziach: $\sum_{i=0}^{n-1} \deg(v_i) = 2k$.

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 1

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0
1	1	0	0	0	0	0
0	0	0	0	0	0	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	**3**	**3**	**3**	**2**	1	1
0	**2**	**2**	**2**	**1**	1	1

Ciąg graficzny

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
    if $\forall_i \ A[i] = 0$ **then**
        return TRUE
    end if
    if $A[0] \geqslant n$ OR $\exists_i \ A[i] < 0$ **then**
        return FALSE
    end if
    for $(i \leftarrow 1; \ i \leqslant A[0]; \ i \leftarrow i + 1)$ **do**
        $A[i] \leftarrow A[i] - 1$
    end for
    $A[0] \leftarrow 0$
    Posortuj tablicę A nierosnąco
end while

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	**2**	**2**	1	1	1	0
0	**1**	**1**	1	1	1	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
  if $\forall_i \; A[i] = 0$ **then**
    return TRUE
  end if
  if $A[0] \geqslant n$ OR $\exists_i \; A[i] < 0$ **then**
    return FALSE
  end if
  for $(i \leftarrow 1; \; i \leqslant A[0]; \; i \leftarrow i+1)$ **do**
    $A[i] \leftarrow A[i] - 1$
  end for
  $A[0] \leftarrow 0$
  Posortuj tablicę A nierosnąco
end while

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	**1**	1	1	1	0	0
0	**0**	1	1	1	0	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
    if $\forall_i \; A[i] = 0$ **then**
        return TRUE
    end if
    if $A[0] \geqslant n$ OR $\exists_i \; A[i] < 0$ **then**
        return FALSE
    end if
    for $(i \leftarrow 1; \; i \leqslant A[0]; \; i \leftarrow i + 1)$ **do**
        $A[i] \leftarrow A[i] - 1$
    end for
    $A[0] \leftarrow 0$
    Posortuj tablicę A nierosnąco
end while

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0
1	**1**	1	0	0	0	0
0	**0**	1	0	0	0	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	0	0	0	0	0	0

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	**0**	0	0	0	0	0
0	**-1**	0	0	0	0	0

Ciąg graficzny

AGH

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n  OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	-1

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 2

$$A = \{1, 3, 3, 4, 2, 3, 1\}.$$

4	3	3	3	2	1	1
2	2	2	1	1	1	0
1	1	1	1	1	0	0
1	1	1	0	0	0	0
1	0	0	0	0	0	0
0	0	0	0	0	0	-1

Uwaga

Liczba nieparzystych stopni musi być parzysta, żeby suma była parzysta.

Ciąg graficzny

Algorithm: degree_seq(A, n)

```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n  OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 3

$$A = \{1, 3, 3, 7, 2, 3, 1\}.$$

7	3	3	3	2	1	1

Ciąg graficzny

Algorithm: degree_seq(A, n)

Posortuj tablicę A nierosnąco
while TRUE **do**
    if $\forall_i\ A[i] = 0$ **then**
        return TRUE
    end if
    if $A[0] \geqslant n$ **OR** $\exists_i\ A[i] < 0$ **then**
        return FALSE
    end if
    for $(i \leftarrow 1;\ i \leqslant A[0];\ i \leftarrow i + 1)$ **do**
        $A[i] \leftarrow A[i] - 1$
    end for
    $A[0] \leftarrow 0$
    Posortuj tablicę A nierosnąco
end while

Przykład nr 3

$$A = \{1, 3, 3, 7, 2, 3, 1\}.$$

7	3	3	3	2	1	1

Uwaga

Każdy stopień musi być mniejszy niż liczba wierzchołków n.

Ciąg graficzny

Algorithm: degree_seq(A, n)

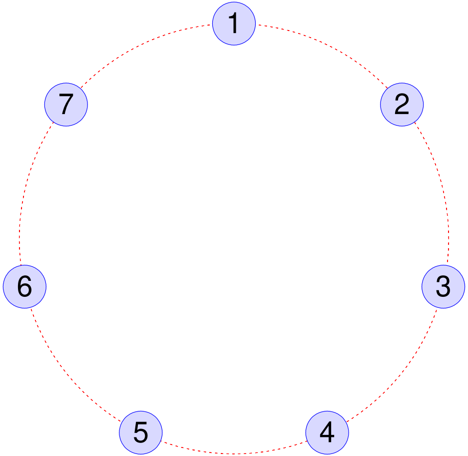
```
Posortuj tablicę A nierosnąco
while TRUE do
    if ∀_i A[i] = 0 then
        return TRUE
    end if
    if A[0] ⩾ n  OR ∃_i A[i] < 0 then
        return FALSE
    end if
    for (i ← 1; i ⩽ A[0]; i ← i + 1) do
        A[i] ← A[i] − 1
    end for
    A[0] ← 0
    Posortuj tablicę A nierosnąco
end while
```

Przykład nr 4

$$A = \{2, 2, 6, 4, 4, 6, 6\}.$$

6	6	6	4	4	2	2
5	5	3	3	1	1	0
4	2	2	0	0	0	0
1	1	0	0	0	-1	-1

Konstrukcja grafu na podstawie ciągu graficznego

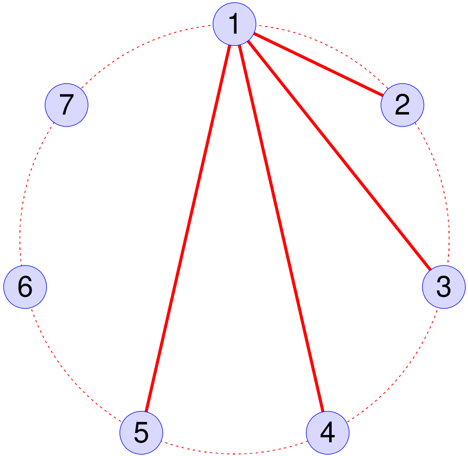


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[1]	[2]	[3]	[4]	[5]	[6]	[7]
4	3	3	2	2	1	1

Konstrukcja grafu na podstawie ciągu graficznego

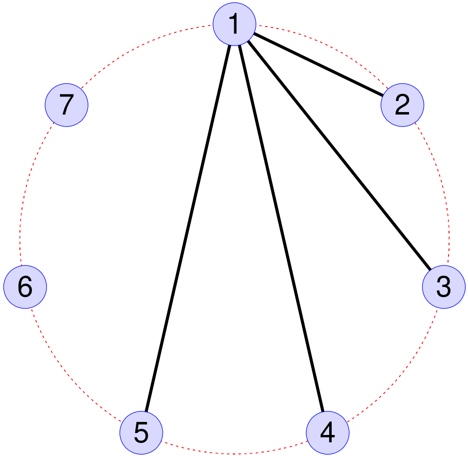


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[1]	[2]	[3]	[4]	[5]	[6]	[7]
4	**3**	**3**	**2**	**2**	1	1
0	**2**	**2**	**1**	**1**	1	1

Konstrukcja grafu na podstawie ciągu graficznego

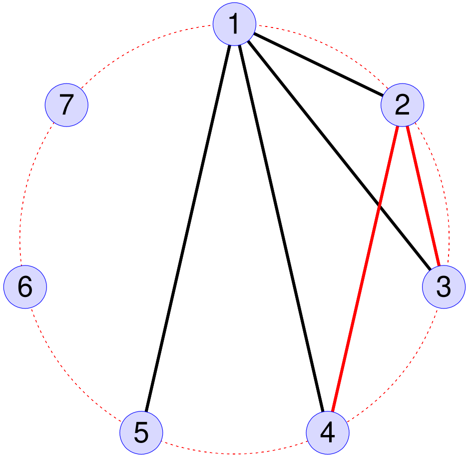


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[2]	[3]	[4]	[5]	[6]	[7]	[ ]
4	3	3	2	2	1	1
2	2	1	1	1	1	0

Konstrukcja grafu na podstawie ciągu graficznego

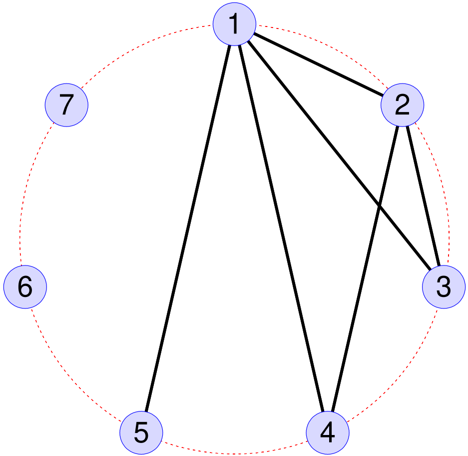


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[2]	[3]	[4]	[5]	[6]	[7]	[ ]
4	3	3	2	2	1	1
2	**2**	**1**	1	1	1	0
0	**1**	**0**	1	1	1	0

Konstrukcja grafu na podstawie ciągu graficznego

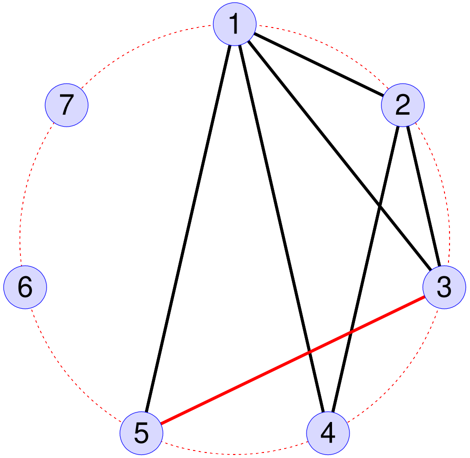


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[3]	[5]	[6]	[7]	[ ]	[ ]	[ ]
4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0

Konstrukcja grafu na podstawie ciągu graficznego

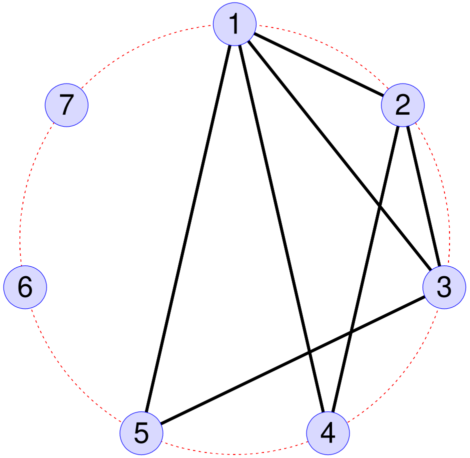


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[3]	[5]	[6]	[7]	[ ]	[ ]	[ ]
4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	**1**	1	1	0	0	0
0	**0**	1	1	0	0	0

Konstrukcja grafu na podstawie ciągu graficznego

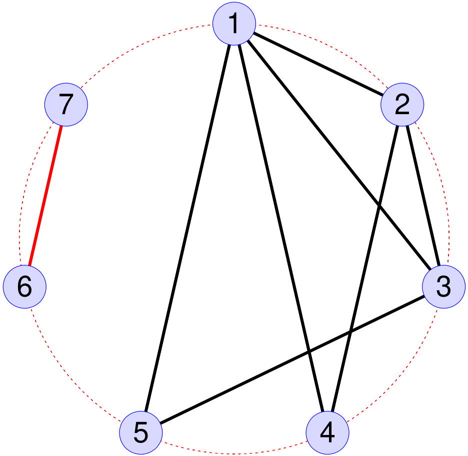


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[6]	[7]	[ ]	[ ]	[ ]	[ ]	[ ]
4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0
1	1	0	0	0	0	0

Konstrukcja grafu na podstawie ciągu graficznego

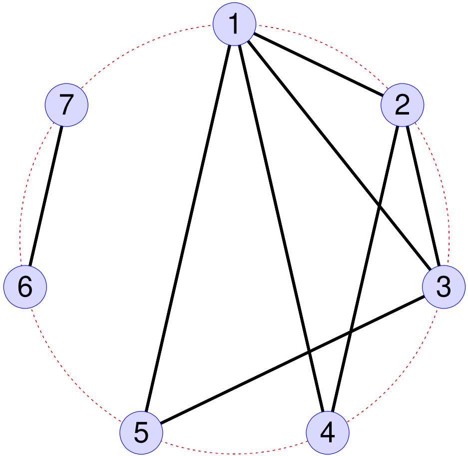


Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[6]	[7]	[ ]	[ ]	[ ]	[ ]	[ ]
4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0
1	1	0	0	0	0	0
0	**0**	0	0	0	0	0

Konstrukcja grafu na podstawie ciągu graficznego



Przykład nr 1 – cd.

$$A = \{1, 3, 2, 3, 2, 4, 1\}.$$

[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]
4	3	3	2	2	1	1
2	2	1	1	1	1	0
1	1	1	1	0	0	0
1	1	0	0	0	0	0
0	0	0	0	0	0	0

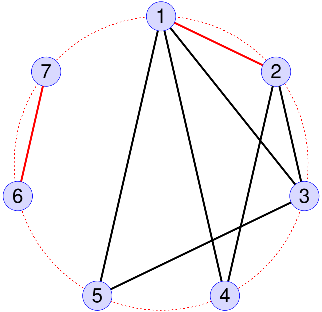
AGH

Zestaw 2, zadanie 2

Napisać program do randomizacji grafów prostych o zadanych stopniach wierzchołków. Do tego celu wielokrotnie powtórzyć operację zamieniającą losowo wybraną parę krawędzi: (a, b) i (c, d) na parę (a, d) i (b, c).

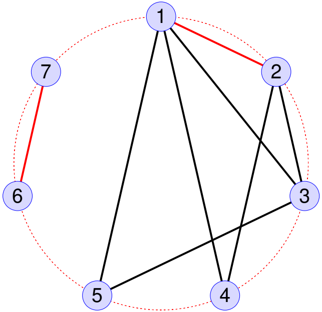
Randomizacja grafów prostych: $(a, b), (c, d) \Rightarrow (a, d), (b, c)$

- Stopnie $\{4, 3, 3, 2, 2, 1, 1\}$
- Np. krawędzie $(1, 2), (6, 7)$



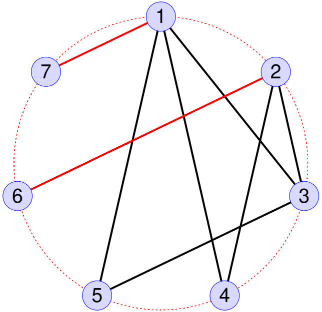
Randomizacja grafów prostych: $(a, b), (c, d) \Rightarrow (a, d), (b, c)$

- Stopnie $\{4, 3, 3, 2, 2, 1, 1\}$
- Np. krawędzie $(1, 2), (6, 7)$



$\Longrightarrow$

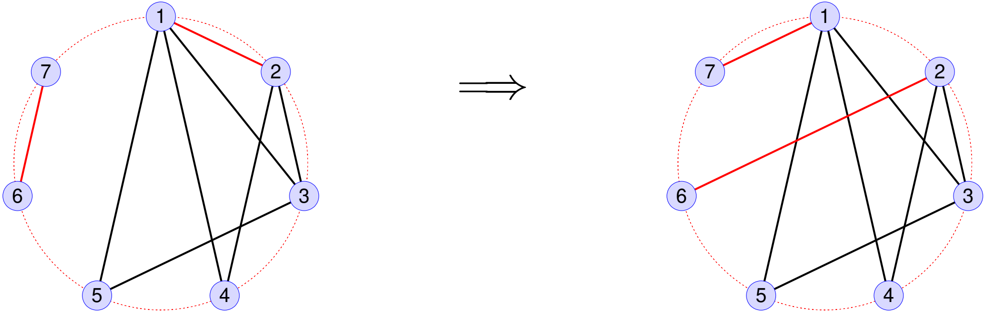
- Stopnie $\{4, 3, 3, 2, 2, 1, 1\}$
- Krawędzie $(1, 7), (2, 6)$



Randomizacja grafów prostych: $(a, b), (c, d) \Rightarrow (a, d), (b, c)$

- Stopnie $\{4, 3, 3, 2, 2, 1, 1\}$
- Np. krawędzie $(1, 2), (6, 7)$

- Stopnie $\{4, 3, 3, 2, 2, 1, 1\}$
- Krawędzie $(1, 7), (2, 6)$



Uwaga

- **Randomizacja nie powoduje zmiany stopni wierzchołków.**
- Nie wylosować tych samych wierzchołków ani nie utworzyć krawędzi wielokrotnych!

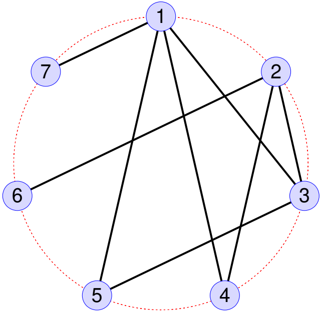
Zestaw 2, zadanie 3

Napisać program do znajdowania największej spójnej składowej na grafie.

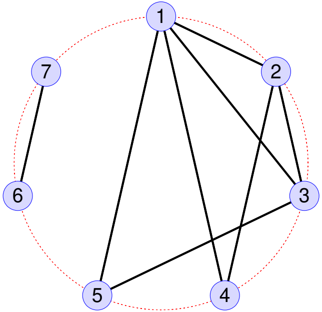
Graf spójny

- **Graf jest spójny** ⇔ między każdą parą wierzchołków istnieje ścieżka.
- **Ścieżka** składa się z kolejnych (różnych) krawędzi.
- **Niespójny** graf składa się z osobnych **składowych**, które są **spójne**.

Graf spójny



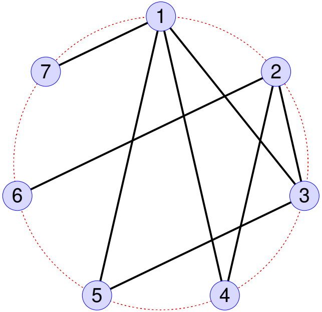
Graf niespójny



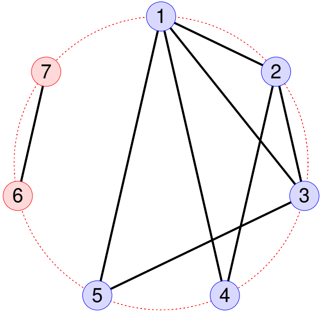
Graf spójny

- **Graf jest spójny** $\Leftrightarrow$ między każdą parą wierzchołków istnieje ścieżka.
- **Ścieżka** składa się z kolejnych (różnych) krawędzi.
- **Niespójny** graf składa się z osobnych **składowych**, które są **spójne**.

Graf spójny

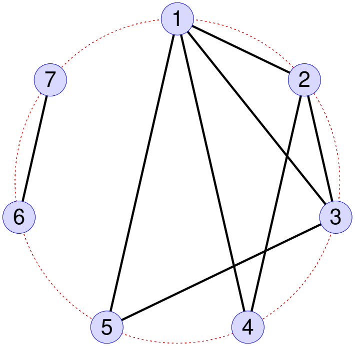


Graf niespójny



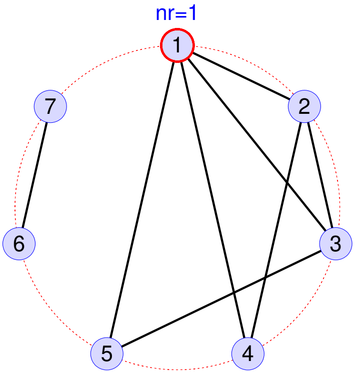
Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.



Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
    - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

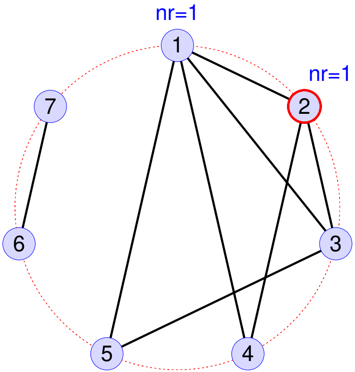


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: 1,

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

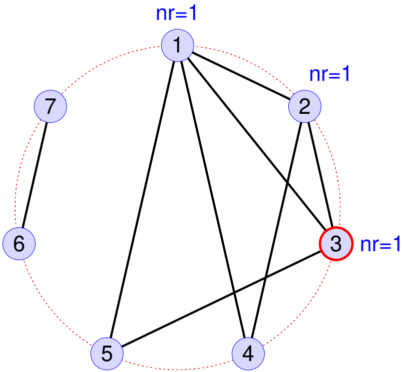


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2,$

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
    - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

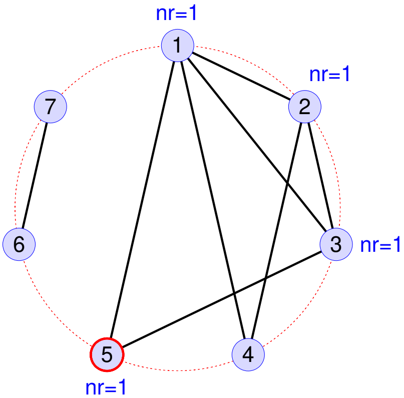


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3,$

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

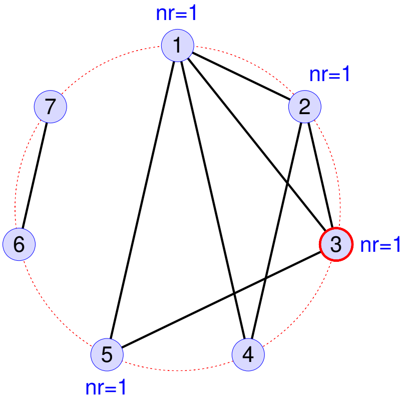


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5,$

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

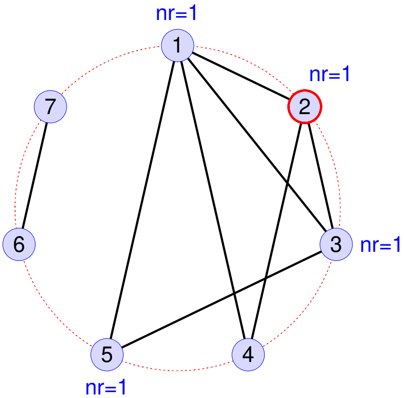


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5,$

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

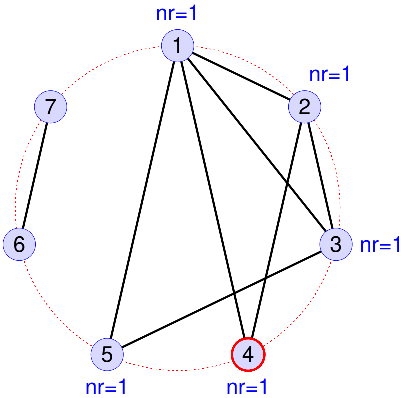


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5,$

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

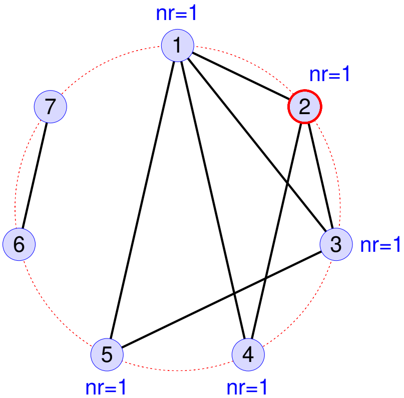


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5, 4$.

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

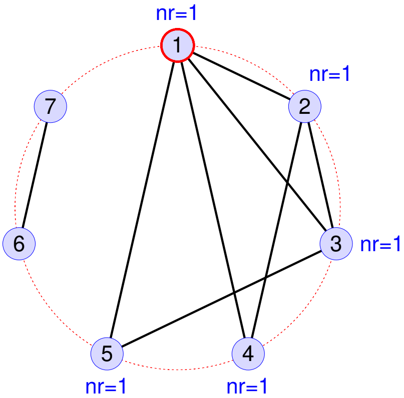


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5, 4$.

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

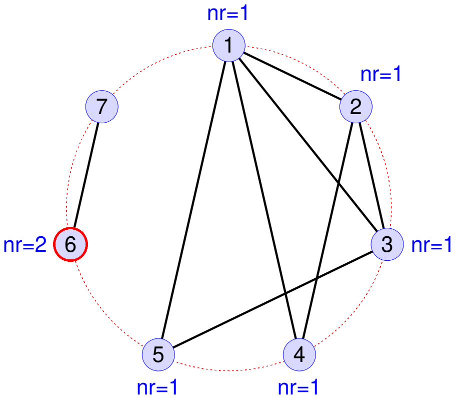


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5, 4$.

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

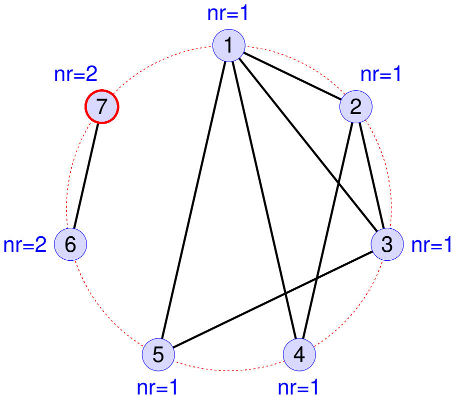


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: 1, 2, 3, 5, 4.
- Składowa numer 2, wierzchołki: 6,

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.

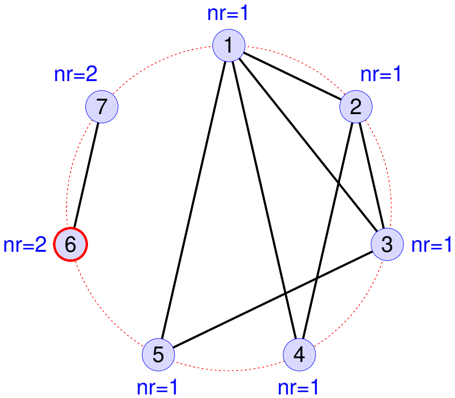


Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5, 4$.
- Składowa numer 2, wierzchołki: $6, 7$.

Algorytm oznaczania spójnych składowych

- **Przeszukiwanie w głąb** (*depth-first search*, DFS).
- Zaczynamy od dowolnego wierzchołka, oznaczamy go jako **odwiedzonego**.
  - Rekurencyjnie odwiedzamy jego **sąsiadów** i oznaczamy ich tym samym numerem składowej.



Znalezione spójne składowe

- Składowa numer 1, wierzchołki: $1, 2, 3, 5, 4$.
- Składowa numer 2, wierzchołki: $6, 7$.

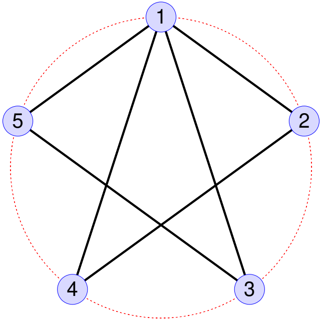
Zestaw 2, zadanie 4

Używając powyższych programów, napisać program do tworzenia losowego grafu eulerowskiego i znajdowania na nim cyklu Eulera.

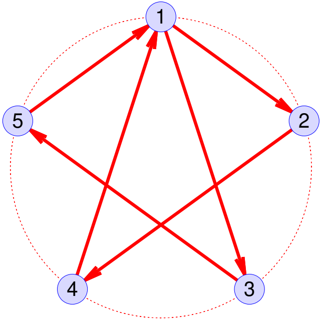
Graf eulerowski

- **Graf eulerowski** ⇔ istnieje w nim cykl Eulera.
- **Cykl Eulera** – zamknięta ścieżka zawierająca każdą krawędź dokładnie 1 raz.
- Graf eulerowski ⇔ **spójny**, a stopień każdego wierzchołka jest **parzysty**.

Graf eulerowski



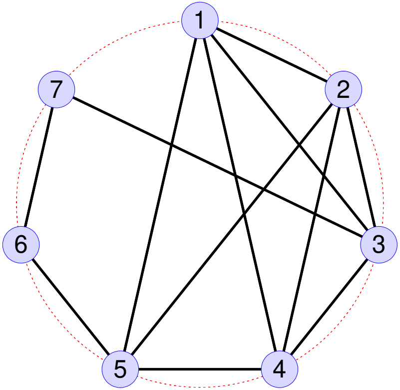
Cykl Eulera



Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

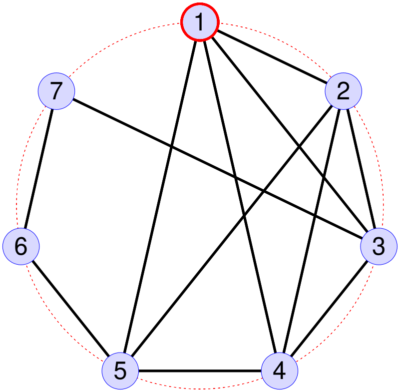
$\Downarrow$

Graf jest eulerowski.

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

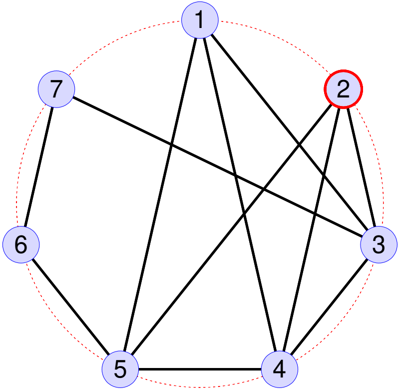
Zawartość stosu:

1

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

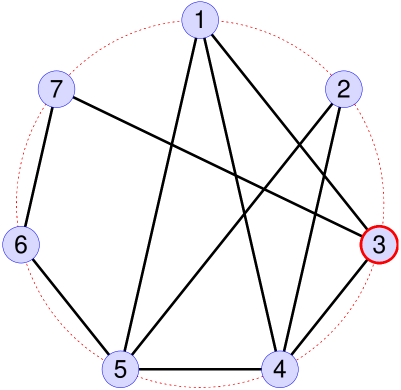
Zawartość stosu:

1 – 2

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

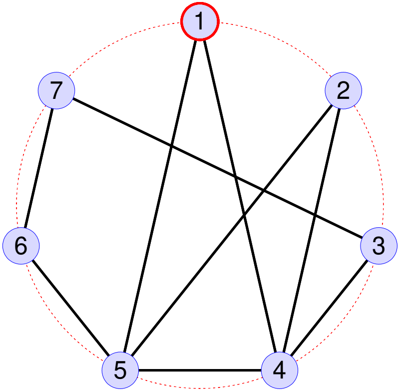
Zawartość stosu:

1 – 2 – 3

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

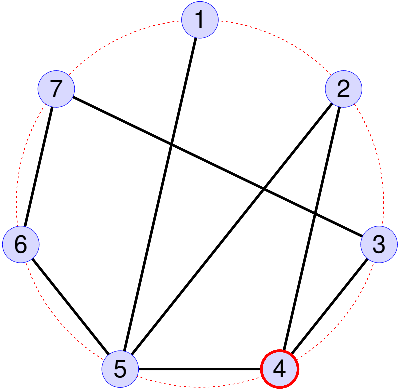
Zawartość stosu:

1 – 2 – 3 – 1

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

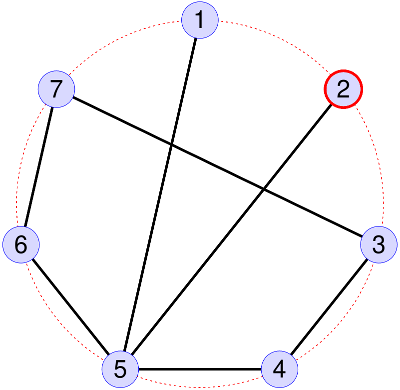
Zawartość stosu:

1 – 2 – 3 – 1 – 4

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

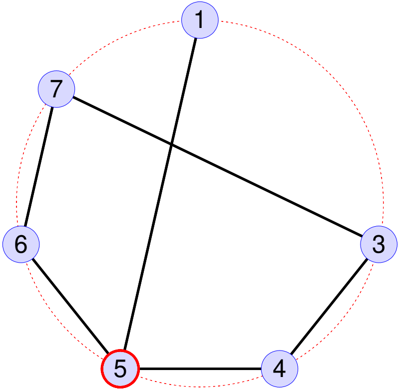
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

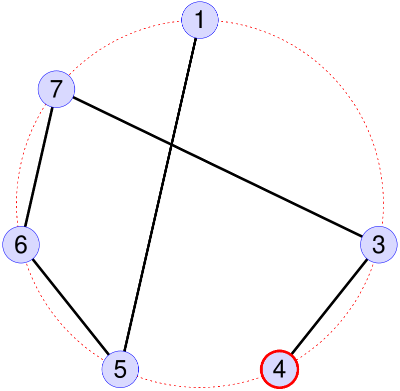
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

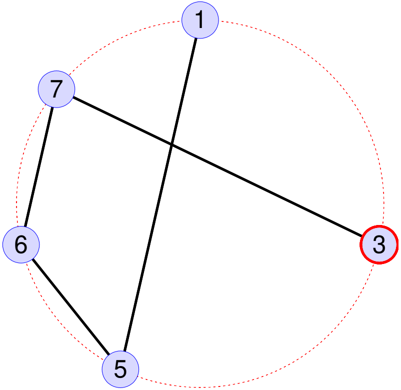
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5 – **4**

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

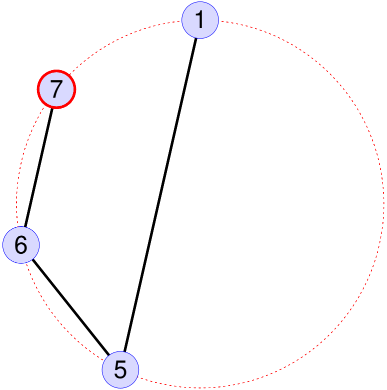
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5 – 4 – 3

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

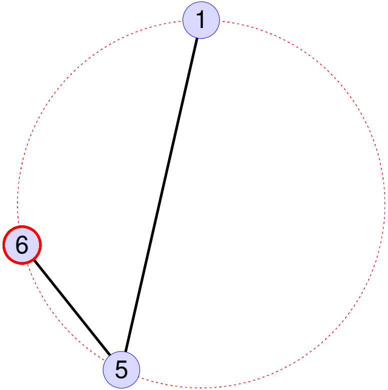
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5 – 4 – 3 – 7

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

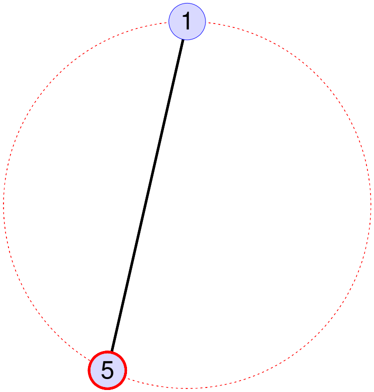
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5 – 4 – 3 – 7 – 6

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

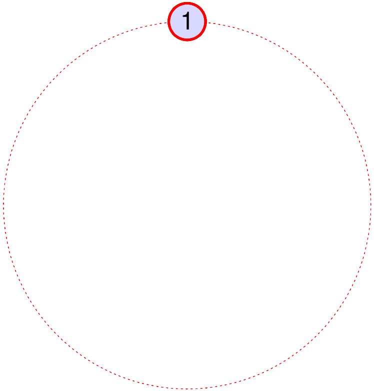
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5 – 4 – 3 – 7 – 6 – 5

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

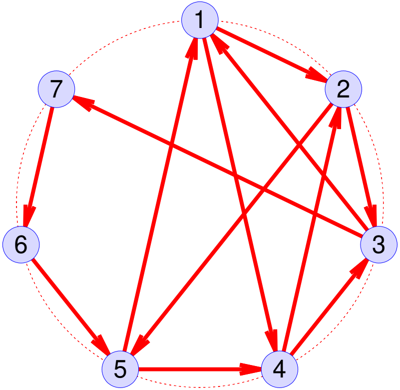
Zawartość stosu:

1 – 2 – 3 – 1 – 4 – 2 – 5 – 4 – 3 – 7 – 6 – 5 – 1

Algorytm poszukiwania cyklu Eulera

Algorytm Fleury'ego (dla grafu eulerowskiego)

- Przechodzimy po krawędziach w dowolnej kolejności, odkładając odwiedzane **wierzchołki na stos**, a **krawędzie i wierzchołki izolowane usuwając**, ale **nie** przechodzimy przez **most** – chyba że nie ma innego wyjścia z wierzchołka.
- **Most** – krawędź, której usunięcie spowoduje rozspójnienie grafu.



Ciąg stopni wierzchołków:

4, 4, 4, 4, 4, 2, 2

$\Downarrow$

Graf jest eulerowski.

Cykl Eulera:

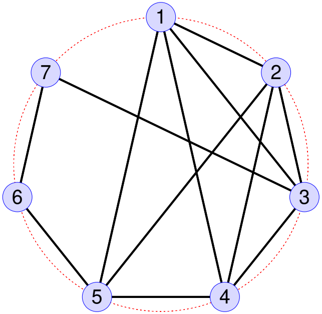
1 – 2 – 3 – 1 – 4 – 2 – 5 – 4 – 3 – 7 – 6 – 5 – 1

Poszukiwanie mostów

Dygresja: jak znaleźć **wszystkie** mosty w grafie?

- **Podejście naiwne**: zastosowanie algorytmu oznaczania spójnych składowych do grafu przed i po usunięciu każdej z krawędzi $\Rightarrow$ **2 przejścia DFS dla** k **krawędzi** $\Rightarrow O(k(n+k))$.
- **Algorytm Tarjana**[a]: **1 przejście DFS** znajduje wszystkie mosty $\Rightarrow O(n+k)$.

[a] Podstawa działania: most nie może należeć do cyklu, musi należeć do drzewa rozpinającego.

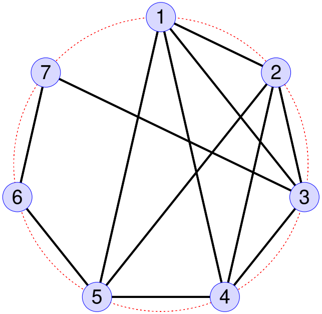


Poszukiwanie mostów

Dygresja: jak znaleźć **wszystkie** mosty w grafie?

- **Podejście naiwne**: zastosowanie algorytmu oznaczania spójnych składowych do grafu przed i po usunięciu każdej z krawędzi $\Rightarrow$ **2 przejścia DFS dla** k **krawędzi** $\Rightarrow O(k(n+k))$.
- **Algorytm Tarjana**[a]: **1 przejście DFS** znajduje wszystkie mosty $\Rightarrow O(n+k)$.

[a] Podstawa działania: most nie może należeć do cyklu, musi należeć do drzewa rozpinającego.

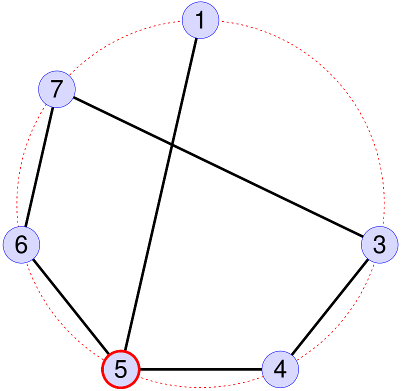


Brak mostów?

Poszukiwanie mostów

Jak sprawdzić, czy **dana krawędź** jest mostem?

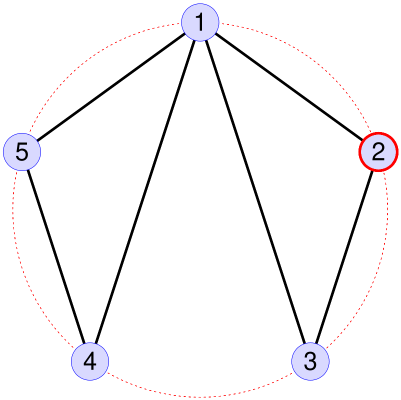
- **Podejście naiwne**: **2 przejścia DFS** $\Rightarrow O(n+k)$ (ale w algorytmie Fleury'ego, wybierając krawędź do przejścia w danym kroku, maks. razy:...?).
- **Algorytm Tarjana**: **1 przejście DFS** $\Rightarrow O(n+k)$.



Poszukiwanie mostów

Jak sprawdzić, czy **dana krawędź** jest mostem?

- **Podejście naiwne**: **2 przejścia DFS** $\Rightarrow O(n+k)$ (ale w algorytmie Fleury'ego, wybierając krawędź do przejścia w danym kroku, maks. razy: 2).
- **Algorytm Tarjana**: **1 przejście DFS** $\Rightarrow O(n+k)$.



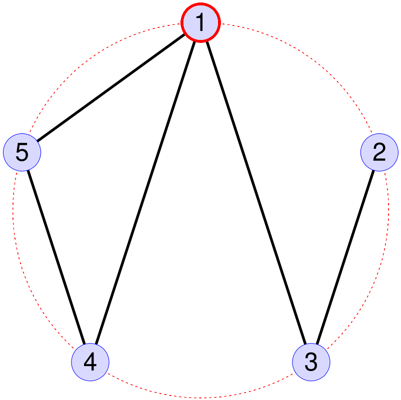
Uwaga

- Przejście: 2 – 1 – ?
- **Usunięcie mostu, przez który nie można przejść, nie musi być powiązane z utworzeniem wierzchołka izolowanego.**

Poszukiwanie mostów

Jak sprawdzić, czy **dana krawędź** jest mostem?

- **Podejście naiwne**: **2 przejścia DFS** $\Rightarrow O(n+k)$ (ale w algorytmie Fleury'ego, wybierając krawędź do przejścia w danym kroku, maks. razy: 2).
- **Algorytm Tarjana**: **1 przejście DFS** $\Rightarrow O(n+k)$.



Uwaga

- Przejście: 2 – 1 – ?
- **Usunięcie mostu, przez który nie można przejść, nie musi być powiązane z utworzeniem wierzchołka izolowanego.**

Zestaw 2, zadanie 5

Napisać program do generowania losowych grafów k-regularnych.

Grafy k-regularne

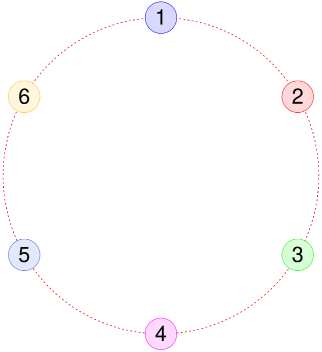
AGH

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

Grafy k-regularne

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.

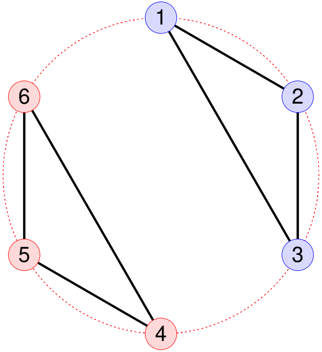


$n = 6, k = 0$

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

Grafy k-regularne

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.

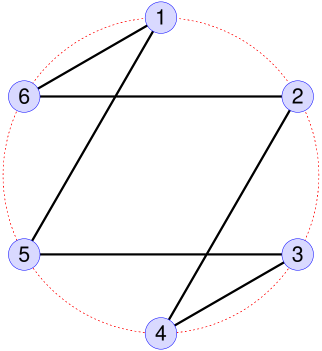


$n = 6, k = 2$

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

Grafy k-regularne

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.

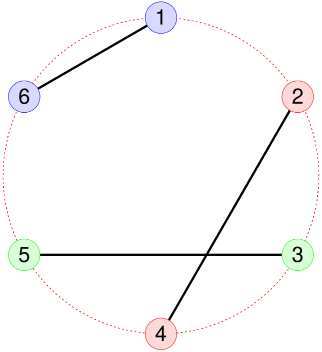


$n = 6, k = 2$

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

Grafy k-regularne

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.

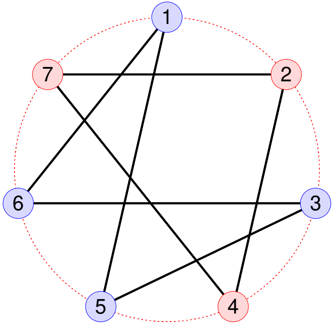


$n = 6, k = 1$

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

Grafy k-regularne

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.

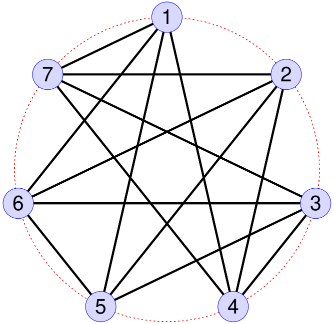


$n = 7, k = 2$

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

Grafy k-regularne

- **Graf k-regularny** – graf, którego każdy wierzchołek ma stopień równy k.
- Może być **niespójny**.
- **Wejście**: liczba wierzchołków n oraz stopień k, spełniające **warunki**:
  - $n > k$,
  - jeżeli k nieparzyste, to n parzyste.



$n = 7, k = 4$

1. **Kontrola warunków**.
2. **Utworzenie grafu** (np. jak przy ciągu graficznym).
3. Wielokrotna **randomizacja** $\Rightarrow$ losowy graf.

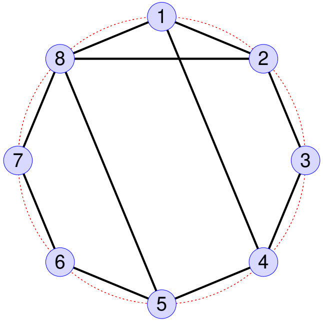
Zestaw 2, zadanie 6

Napisać program do sprawdzania (dla małych grafów), czy graf jest hamiltonowski.

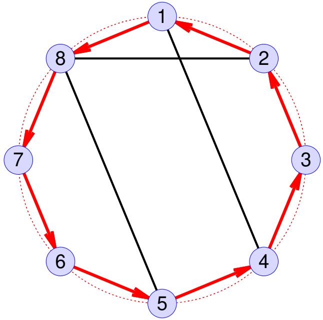
Graf hamiltonowski

- **Graf hamiltonowski** $\Leftrightarrow$ istnieje w nim cykl Hamiltona.
- **Cykl Hamiltona** – zamknięta ścieżka zawierająca każdy wierzchołek dokładnie 1 raz (poza startowym, który występuje dwukrotnie).

Graf hamiltonowski



Cykl Hamiltona



Graf hamiltonowski

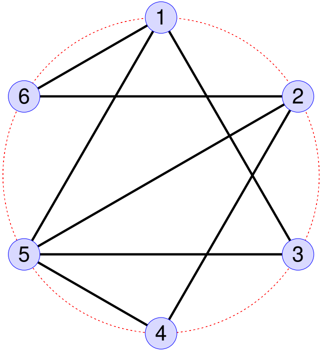
Jak sprawdzić, czy graf jest hamiltonowski?

- „Jeśli graf ma wystarczająco dużo krawędzi w stosunku do wierzchołków, to jest hamiltonowski".
- Np. **twierdzenie Diraca**: jeżeli w grafie prostym $n \geqslant 3 \wedge \forall_v \deg(v) \geqslant \frac{n}{2} \Rightarrow$ jest hamiltonowski.
- **Warunek konieczny i wystarczający**: nie istnieje $\Rightarrow$ **szukamy cyklu Hamiltona do skutku**; jeśli znaleziony, to graf jest hamiltonowski.
- Testowanie programu: np. grafy k-regularne spełniające twierdzenie Diraca.

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.

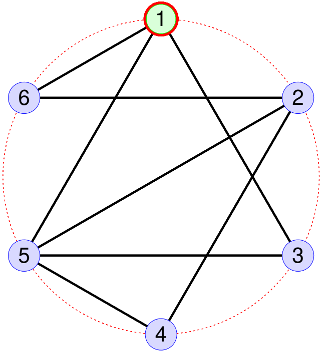


Zawartość stosu:

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



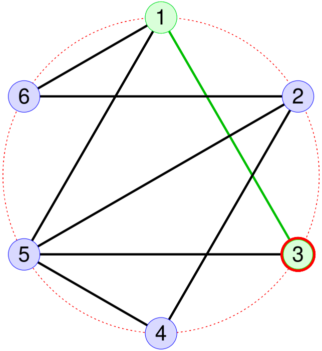
Zawartość stosu:

1

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie ⇒ kontrola połączenia końcowego z początkowym.



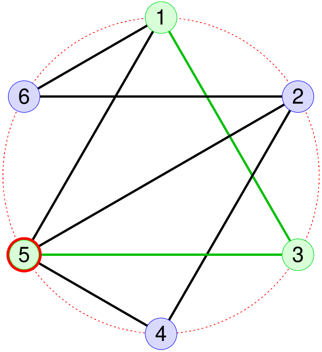
Zawartość stosu:

1 – 3

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



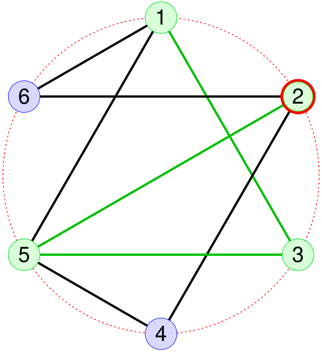
Zawartość stosu:

1 – 3 – 5

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



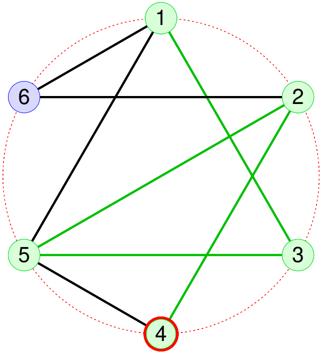
Zawartość stosu:

1 – 3 – 5 – 2

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



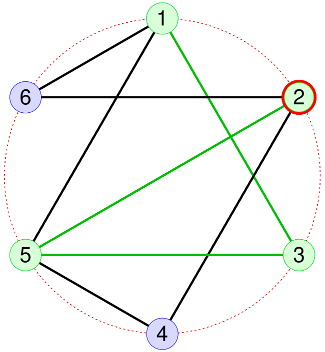
Zawartość stosu:

1 – 3 – 5 – 2 – 4

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

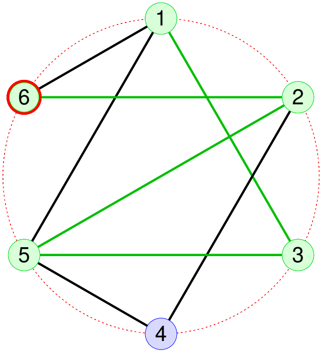
$1 - 3 - 5 - 2 - \cancel{4}$

$1 - 3 - 5 - 2$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

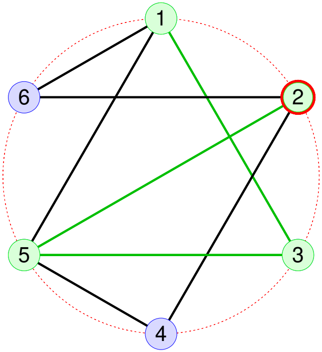
$1 - 3 - 5 - 2 - \cancel{4}$

$1 - 3 - 5 - 2 - 6$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

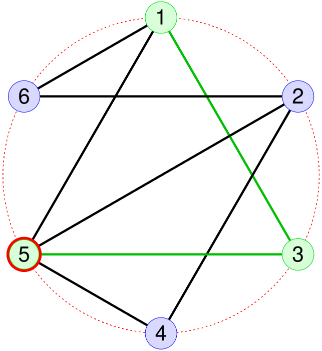
$1 - 3 - 5 - 2 - \cancel{4}$

$1 - 3 - 5 - 2 - \cancel{6}$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie ⇒ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

$1 - 3 - 5 - 2 - \cancel{4}$

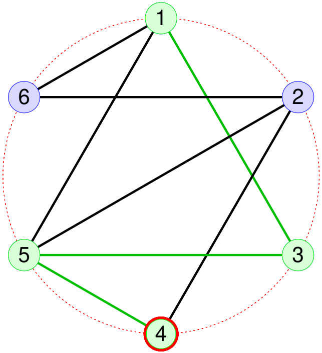
$1 - 3 - 5 - \cancel{2} - \cancel{6}$

$1 - 3 - 5$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie ⇒ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

$1 - 3 - 5 - 2 - \cancel{4}$

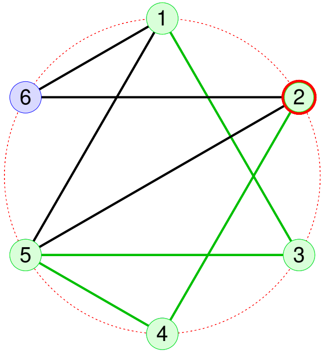
$1 - 3 - 5 - \cancel{2} - \cancel{6}$

$1 - 3 - 5 - 4$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie ⇒ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

$1 - 3 - 5 - 2 - \not 4$

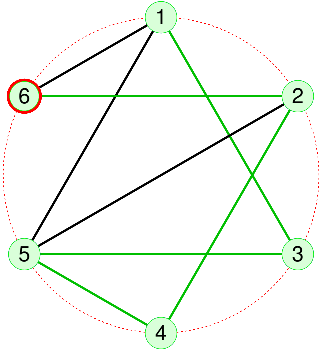
$1 - 3 - 5 - \not 2 - \not 6$

$1 - 3 - 5 - 4 - 2$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

$1 - 3 - 5 - 2 - \cancel{4}$

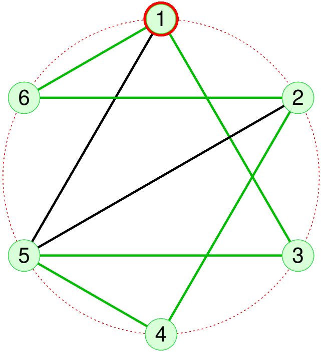
$1 - 3 - 5 - \cancel{2} - \cancel{6}$

$1 - 3 - 5 - 4 - 2 - 6$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie $\Rightarrow$ kontrola połączenia końcowego z początkowym.



Zawartość stosu:

$1 - 3 - 5 - 2 - \cancel{4}$

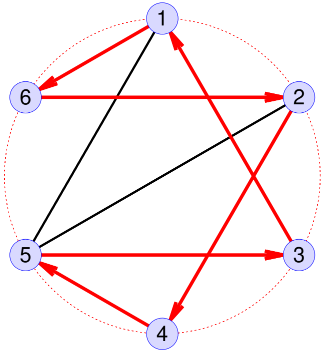
$1 - 3 - 5 - \cancel{2} - \cancel{6}$

$1 - 3 - 5 - 4 - 2 - 6 - 1$

Poszukiwanie cyklu Hamiltona

Algorytm siłowy

- **Przeszukiwanie w głąb** (DFS); odwiedzone wierzchołki odkładamy na stos, ściąganie w przypadku niepowodzenia i dalsze odwiedzanie.
- Wszystkie wierzchołki na stosie ⇒ kontrola połączenia końcowego z początkowym.



Cykl Hamiltona

1 – 3 – 5 – 4 – 2 – 6 – 1