Matematyka Dyskretna

Wykład 13

Komputerowe reprezentacje grafów

Wojciech Kordecki
wojciech.kordecki@collegiumwitelona.pl

Collegium Witelona
Wydział Nauk Technicznych i Ekonomicznych
Zakład Informatyki

Semestr letni 2023/24

Macierz incydencji grafu nieskierowanego

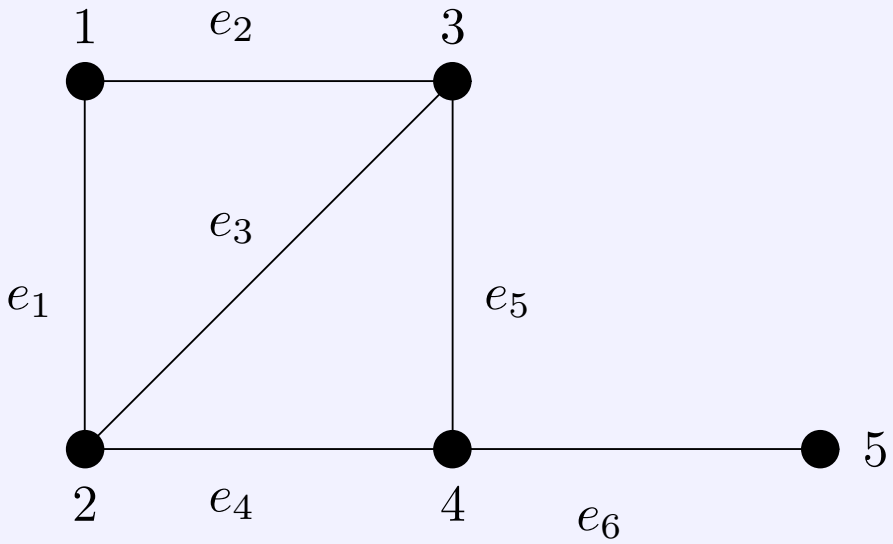
$G = (V, E)$ graf nieskierowany:
$V = \{1, \ldots, n\}$, $E = \{e_1, e_2, \ldots, e_m\} \subseteq \{\{i, j\} : i, j \in V\}$

$$A(G) = \begin{bmatrix} a_{11} & \ldots & a_{1m} \\ & \ldots\ldots & \\ a_{n1} & \ldots & a_{nm} \end{bmatrix},$$

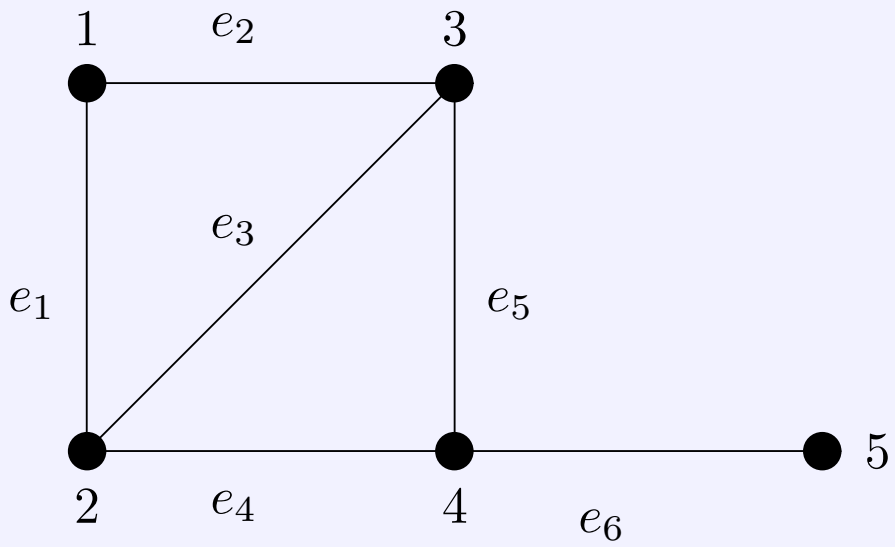
gdzie:

$$a_{ij} = \begin{cases} 1, & \text{gdy } v_i \in e_j\,, \\ 0 & \text{w przeciwnym przypadku.} \end{cases}$$

Przykład



Przykład



$$V = \{1, 2, 3, 4, 5\}$$

$$\begin{aligned} E &= \{e_1, e_2, e_3, e_4, e_5, e_6\} \\ &= \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}. \end{aligned}$$

Przykład c.d.

$$V = \{1, 2, 3, 4, 5\}$$

$$\begin{aligned} E &= \{e_1, e_2, e_3, e_4, e_5, e_6\} \\ &= \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}. \end{aligned}$$

Przykład c.d.

$$V = \{1, 2, 3, 4, 5\}$$

$$\begin{aligned} E &= \{e_1, e_2, e_3, e_4, e_5, e_6\} \\ &= \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}\}. \end{aligned}$$

$$A(G) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Macierz incydencji – własności

- Macierz incydencji składa się tylko z elementów 0 oraz 1 – jest macierzą binarną.
- Krawędź grafu jest incydentna z dokładnie dwoma wierzchołkami – każda kolumna macierzy ma dokładnie dwie jedynki.
- Wiersz, w którym wszystkie elementy są zerami, reprezentuje wierzchołek izolowany.
- Krawędzie równoległe mają identyczne kolumny.
- Liczba jedynek w każdym wierszu jest równa stopniowi odpowiadającego mu wierzchołka.

Macierz incydencji grafu skierowanego

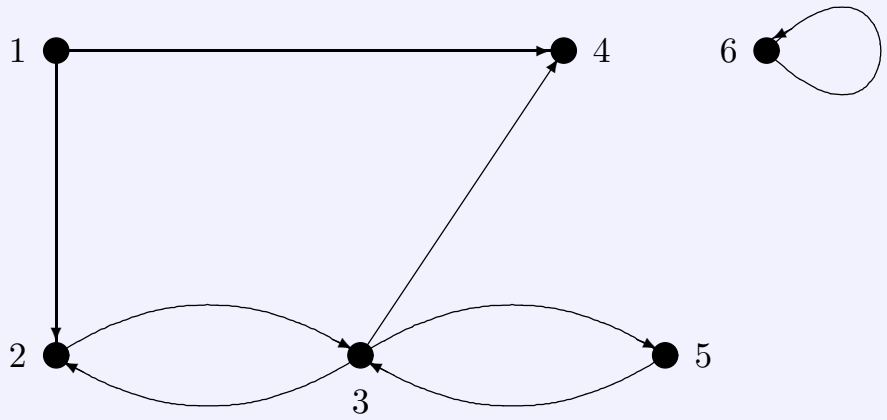
$G = (V, E)$ graf skierowany bez pętli: $V = \{1, 2, \ldots, n\}$,
$E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$

$$A(G) = \begin{bmatrix} a_{11} & \ldots & a_{1m} \\ & \ldots\ldots & \\ a_{n1} & \ldots & a_{nm} \end{bmatrix},$$

gdzie:

$$a_{ij} = \begin{cases} -1, & \text{gdy } e_j = (j, i), \\ 1, & \text{gdy } e_j = (i, j), \\ 0 & \text{w przeciwnym przypadku.} \end{cases}$$

Przykład (1)



$V = \{1, 2, 3, 4, 5, 6\}$,
$E = \{(1, 2), (1, 4), (2, 3), (3, 2), (3, 4), (3, 5), (5, 3), (6, 6)\}$

Przykład (2)

$V = \{1, 2, 3, 4, 5, 6\}$,
$E = \{(1, 2), (1, 4), (2, 3), (3, 2), (3, 4), (3, 5), (5, 3), (6, 6)\}$

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierz sąsiedztwa

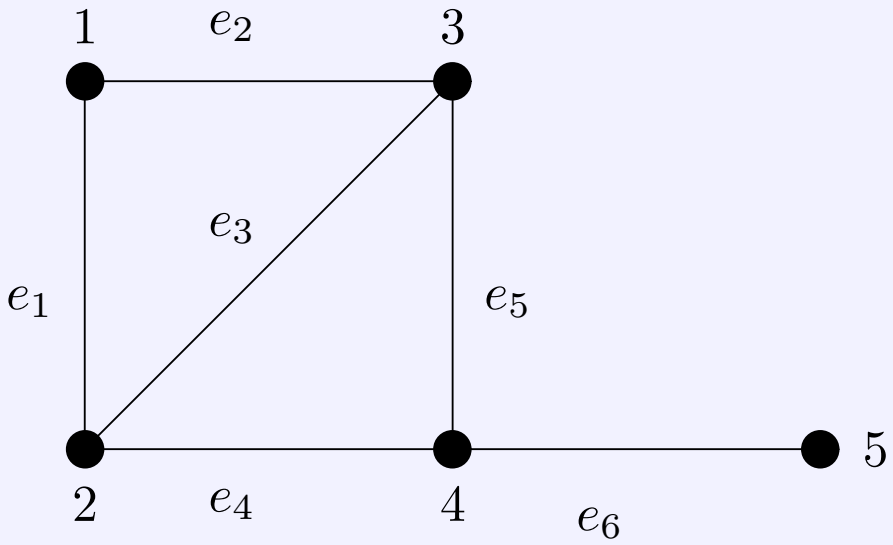
Graf $G = (V, E)$:
$V = \{1, 2, \ldots, n\}$,
$E = \{e_1, e_2, \ldots, e_m\} \subseteq \{\{i, j\} : i, j \in V\}$.

$$B(G) = \begin{bmatrix} b_{11} & \ldots & b_{1n} \\ & \ldots\ldots & \\ b_{n1} & \ldots & b_{nn} \end{bmatrix},$$

gdzie:

$$b_{ij} = b_{ji} = \begin{cases} 1, & \text{gdy } \{i, j\} \in E, \\ 0 & \text{w przeciwnym przypadku.} \end{cases}$$

Przykład



$$B(G) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Wady reprezentacji macierzowej

Macierze są bardzo dużych rozmiarów.

Wady reprezentacji macierzowej

Macierze są bardzo dużych rozmiarów.

Macierz incydencji ma rozmiar $n \times m$.
Składa się głównie z zer.

Wady reprezentacji macierzowej

Macierze są bardzo dużych rozmiarów.

Macierz incydencji ma rozmiar $n \times m$.
Składa się głównie z zer.

Macierz sąsiedztwa ma rozmiar $n \times n$.
Może mieć bardzo dużo zer.

Lista krawędzi lub łuków

Lista zbiorów dwuelementowych (multizbiorów dwelementowych) – grafy nieskierowane.
Lista par – grafy nieskierowane.

Lista krawędzi lub łuków

Lista zbiorów dwuelementowych (multizbiorów dwelementowych) – grafy nieskierowane.
Lista par – grafy nieskierowane.

Reprezentacja oszczędna pamięciowo.

Lista krawędzi lub łuków

Lista zbiorów dwuelementowych (multizbiorów dwelementowych) – grafy nieskierowane.
Lista par – grafy nieskierowane.

Reprezentacja oszczędna pamięciowo.

Listy w większości języków są proste w implementacji.

W pakiecie *Maxima* i w języku *Python* są wbudowane.

Lista krawędzi lub łuków

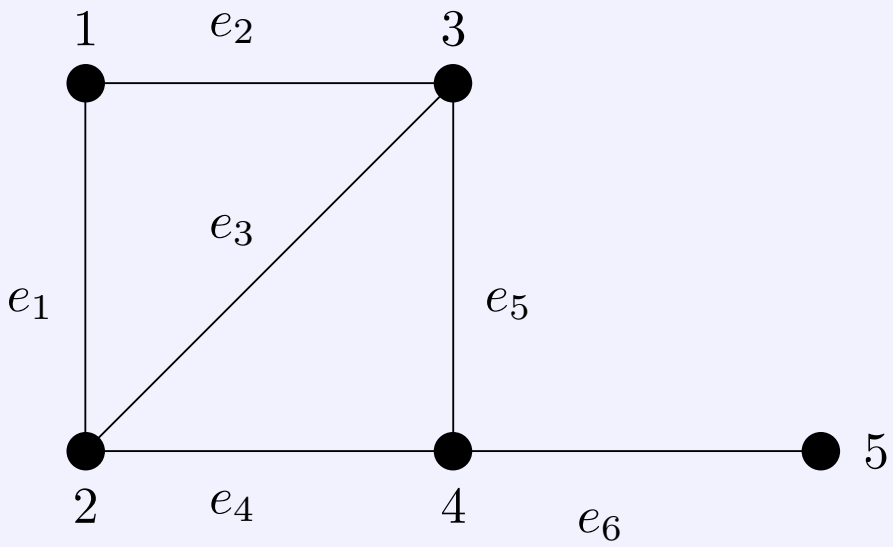
$G = (V, E)$ graf nieskierowany lub skierowany:
$V = \{1, \ldots, n\}$,
$E = \{e_1, e_2, \ldots, e_m\} \subseteq \{\{i, j\} : i, j \in V\}$
$e_j = \{v_{i_j}, u_{i_j}\}$.
lub
$E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$
$e_j = (v_{i_j}, u_{i_j})$.

Lista krawędzi lub łuków

$G = (V, E)$ graf nieskierowany lub skierowany:
$V = \{1, \ldots, n\}$,
$E = \{e_1, e_2, \ldots, e_m\} \subseteq \{\{i, j\} : i, j \in V\}$
$e_j = \{v_{i_j}, u_{i_j}\}$.
lub
$E = \{e_1, e_2, \ldots, e_m\} \subseteq V \times V$
$e_j = (v_{i_j}, u_{i_j})$.

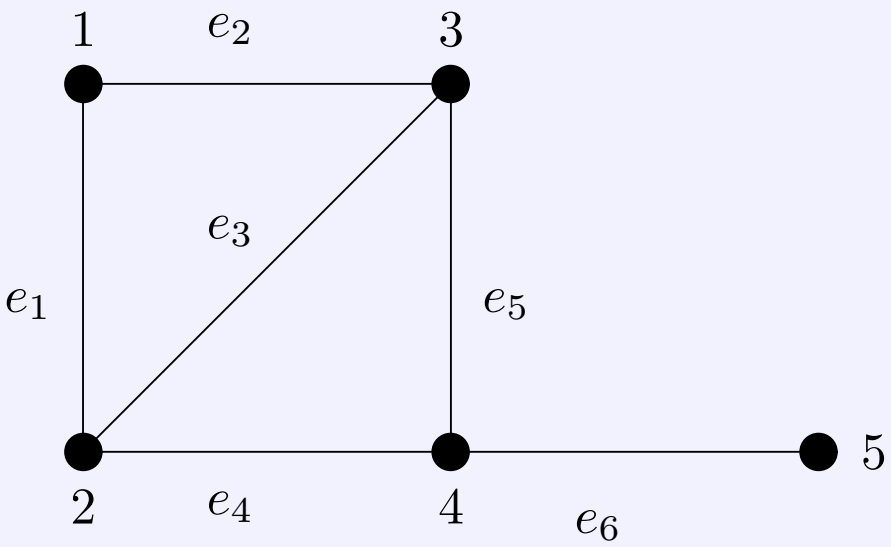
v_{i_1}	u_{i_1}
v_{i_2}	u_{i_2}
$\ldots$	$\ldots$
v_{i_m}	u_{i_m}

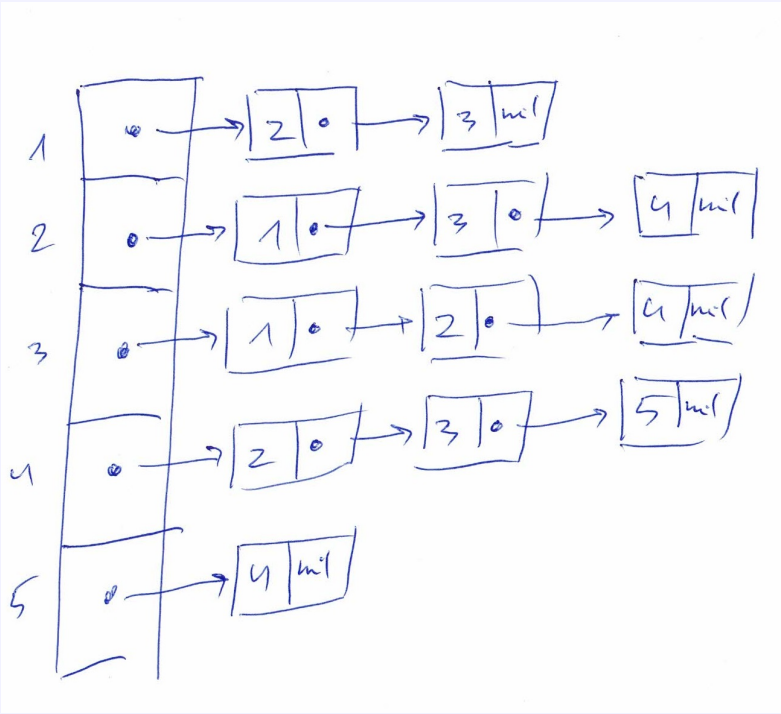
Przykład



1	2
1	3
2	3
2	4
3	4
4	5

Lista sąsiedztwa



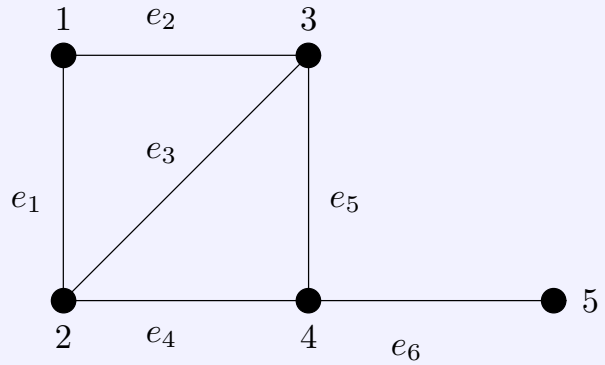


Reprezentacje macierzowe

Reprezentacje macierzowe łatwo jest zaimplementować w pakiecie *Scilab*, który jest darmowym odpowiednikiem pakietu *Matlab*.

Również wygodna jest implementacja reprezentacji macierzowej w pakiecie *Maxima*, który jest darmowym odpowiednikiem pakietu *Mathematica*.
Mniej jest jednak możliwości operacji na macierzach.

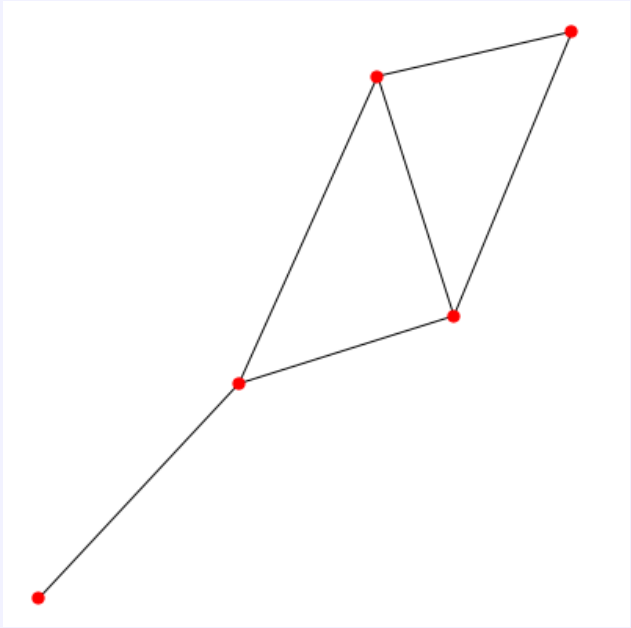
Maxima – tworzenie grafów




```
(%i1) load (graphs)$
(%i2) g : create_graph([1,2,3,4,5], [[1,2],[1,3],[2,3], [2,
(%i3) print_graph(g)$
Graph on 5 vertices with 6 edges.
Adjacencies:
  1 :  3  2
  2 :  4  3  1
  3 :  4  2  1
  4 :  5  3  2
  5 :  4
```

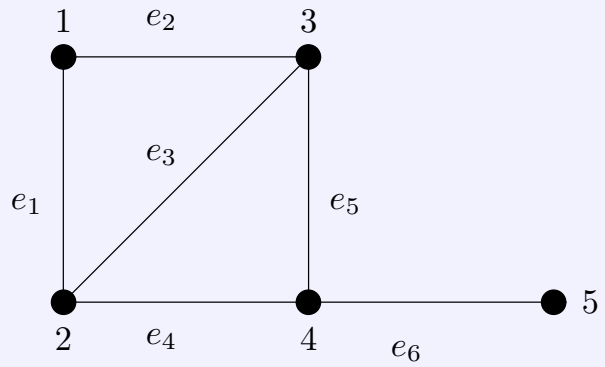
Maxima – rysowanie grafów

```
(%i4)  draw_graph(g);
(%t4)  (Graphics)
(%o4) done
```



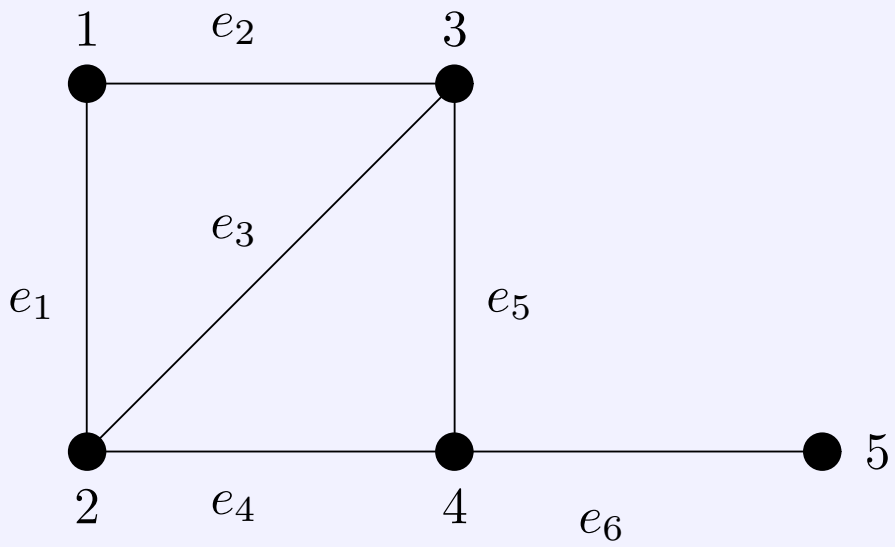
Listy sąsiedztwa

W pakiecie *Maxima* można też łatwo zaprogramować grafy jako listy.




```
(%i1) G:[[2,3],[1,3,4],[1,2,4],[2,3,5],[4]];
(%o1) [[2,3],[1,3,4],[1,2,4],[2,3,5],[4]]
(%i2) delete([4],G);
(%o2) [[2,3],[1,3,4],[1,2,4],[2,3,5]]
(%i3) for i:1 thru 4 do G[i]: delete(5,G[i]);
(%o3) done
(%i4) G;
(%o4) [[2,3],[1,3,4],[1,2,4],[2,3],[4]]
```

Lista krawędzi (1)



Graf G – Lista krotek:

```
>>> G=[(1,2),(1,3),(2,3),(2,4),(3,4),(4,5)]
>>> print(G)
[(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 5)]
```

Lista krawędzi (2)

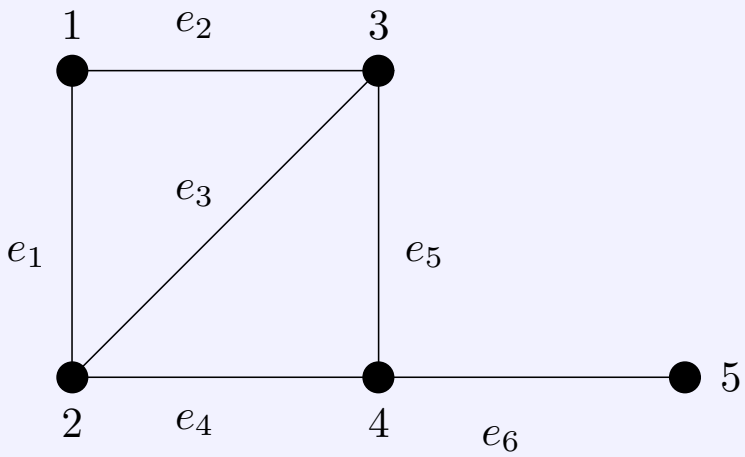
Graf G – Lista krotek:

```
>>> G=[(1,2),(1,3),(2,3),(2,4),(3,4),(4,5)]
>>> print(G)
[(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 5)]
```

Usuwamy krawędź $(4,5)$ i dodajemy $(1,4)$, otrzymując graf K_4.

```
>>> del G[5]
>>> print(G)
[(1, 2), (1, 3), (2, 3), (2, 4), (3, 4)]
>>> G+=[(1,4)]
>>> print(G)
[(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (1, 4)]
```

Listy sąsiedztwa



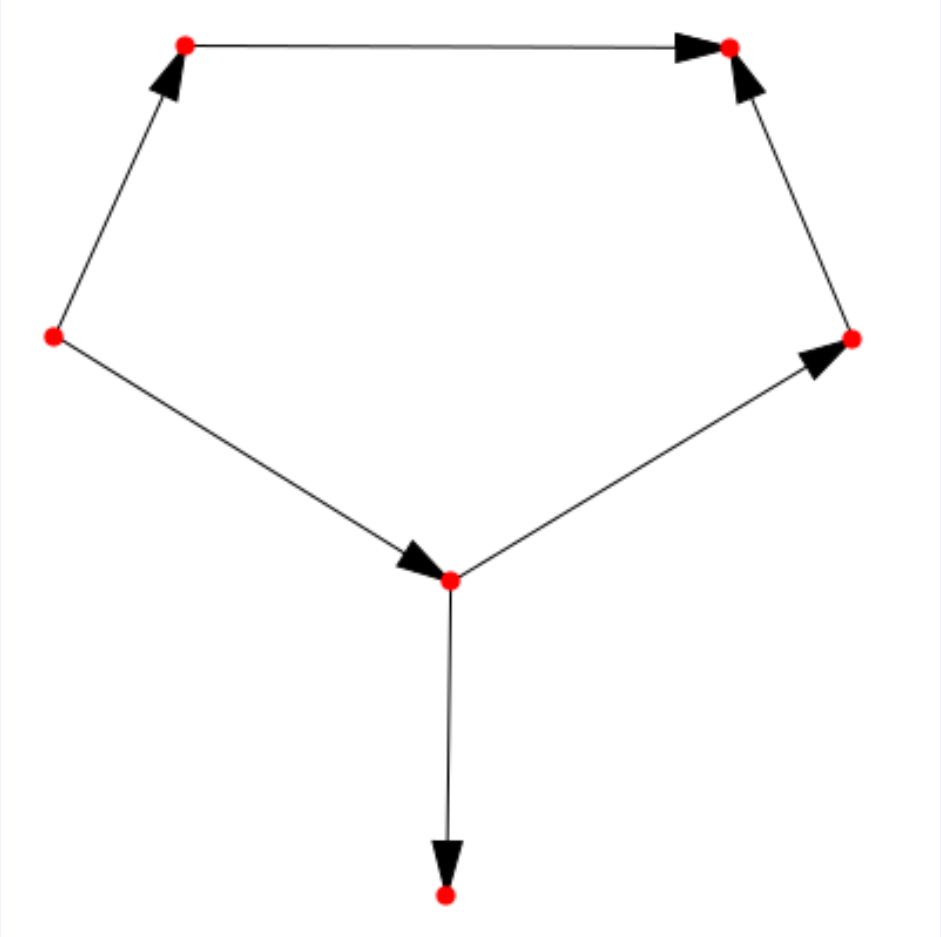

```
>>> G=[[1,2],[0,2,3],[0,1,3],[1,2,4],[3]]
>>> print(G)
[[1, 2], [0, 2, 3], [0, 1, 3], [1, 2, 4], [3]]
```

Graf skierowany jako słownik

```
# Using a Python dictionary to act as an adjacency list

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
```

Rysunek

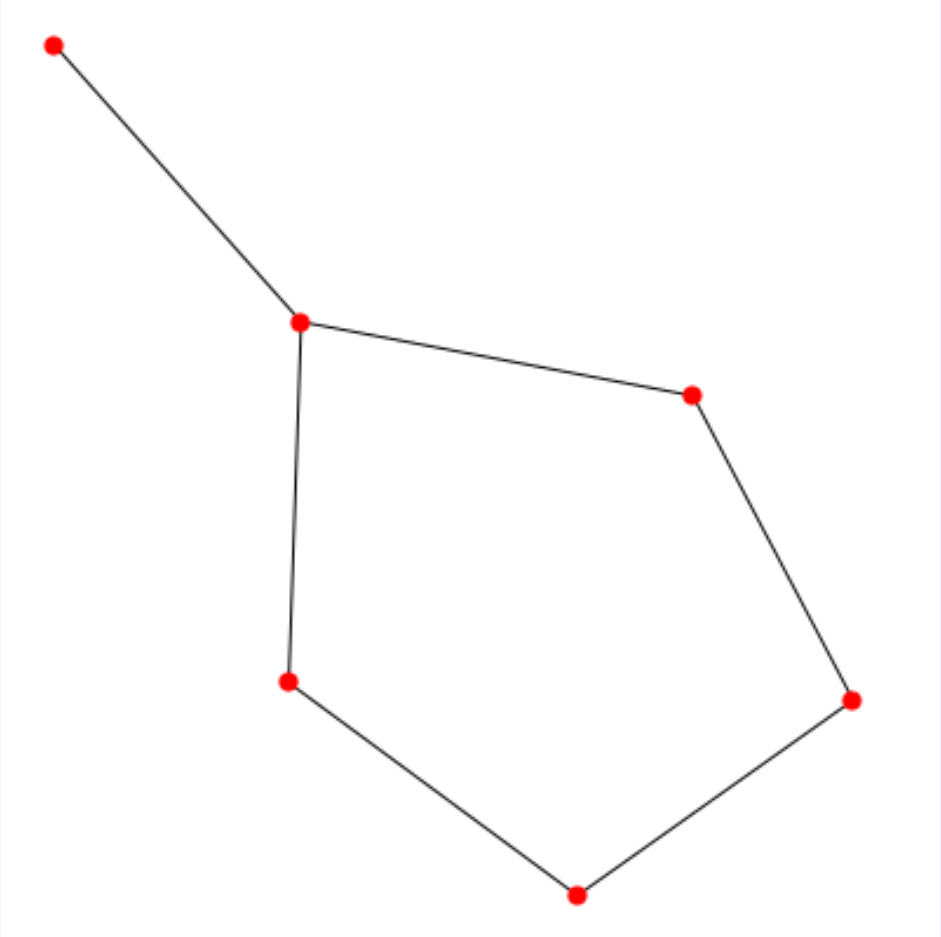


Graf nieskierowany jak słownik

```
# Using a Python dictionary to act as an adjacency list

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4', '5'],
  '7' : ['5', '8'],
  '2' : ['3'],
  '4' : ['3', '8'],
  '8' : ['4', '7']
}
```

Rysunek



Porównanie

```
# Directed

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
```

```
# Undirected

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4', '5'],
  '7' : ['5', '8'],
  '2' : ['3'],
  '4' : ['3', '8'],
  '8' : ['4', '7']
}
```

DFS dla grafu skierowanego

```
visited = set()
# Set to keep track of visited nodes of graph.
def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')

# https://favtutor.com/blogs/depth-first-search-python
```

DFS dla grafu nieskierowanego

```
visited = set() # Set to keep track of visited nodes of gra

def dfs(visited, graph, node):  #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

BFS dla grafu skierowanego

```
visited = [] # List for visited nodes.
queue = []     #Initialize a queue
def bfs(visited, graph, node): #function for BFS
  visited.append(node)
  queue.append(node)

  while queue:          # Creating loop to visit each node
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
```

BFS dla grafu skierowanego c.d.

```
# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')    # function calling

# https://favtutor.com/blogs/breadth-first-search-python
```