



**Algorytmy  
i SD**

**Struktury  
danych**

**-**

**Grafy  
w MATLABie**

---

## przykład 1

SimBiology® model of a  
Repressilator oscillatory network



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- załaduj graf:

```
>> load oscillatorgraph
```

- podejrzuj zmienne:

```
>> whos g names
```

Name	Size	Bytes	Class	Attributes
g	65x65	2544	double	sparse
names	65x1	8340	cell	



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- załaduj graf:

```
>> load oscillatorgraph
```

- podejrzuj zmienne:

```
>> g
```

```
g = (39,1)      1  
     (41,1)      1  
     (52,1)      1  
     (53,1)      1  
     (60,1)      1  
     (31,2)      1  
     (33,2)      1  
     (43,2)      1  
     (45,2)      1  
     (61,2)      1
```



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- załaduj graf:

```
>> load oscillatorgraph
```

- podejrzyj zmienne:

```
>> names
```

```
names = 'pA'
        'pB'
        'pC'
        'mA'
        'mB'
        'mC'
        'OpA'
        'OpB'
        ...
        'Reaction1'
        'Reaction2'
        ...
```



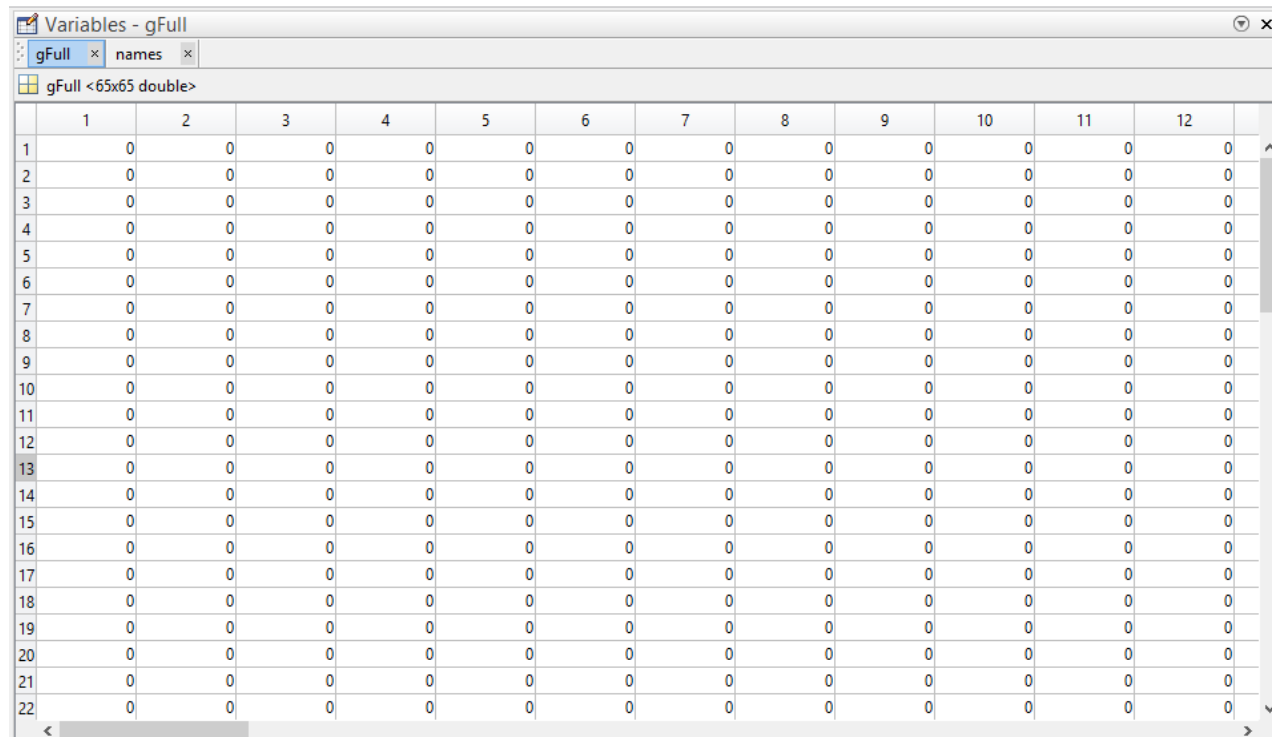
# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

- zamień macierz rzadką g na macierz zwykłą:

```
>> gFull = full ( g ) ;
```

- i obejrzyj w MATLABie:



The image shows a MATLAB Variables window titled "Variables - gFull". It displays a variable named "gFull" of type "double" with dimensions "65x65". The matrix is visualized as a grid of 65 rows and 65 columns. The first 12 columns are labeled 1 through 12, and the first 12 rows are labeled 1 through 12. The matrix is sparse, with most cells containing the value 0. The non-zero elements are concentrated in the first 12 rows and columns, forming a pattern that suggests a circulant-like structure. The rest of the matrix (rows 13-65 and columns 13-65) is entirely zero.

	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0



(cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo>

- zamień macierz rzadką g na

```
>> gFull = full ( g )
```

- i obejrzyj w MATLABie  
- albo lepiej - w Excelu:

[illegible]

# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- utwórz obiekt *biograph* (Bioinformatics Toolbox):

```
>> graphObject = biograph ( g , names )
```

```
Biograph object with 65 nodes and 123 edges.
```

- obejrzyj graf:

```
>> graphFigure = view ( graphObject ) ;
```

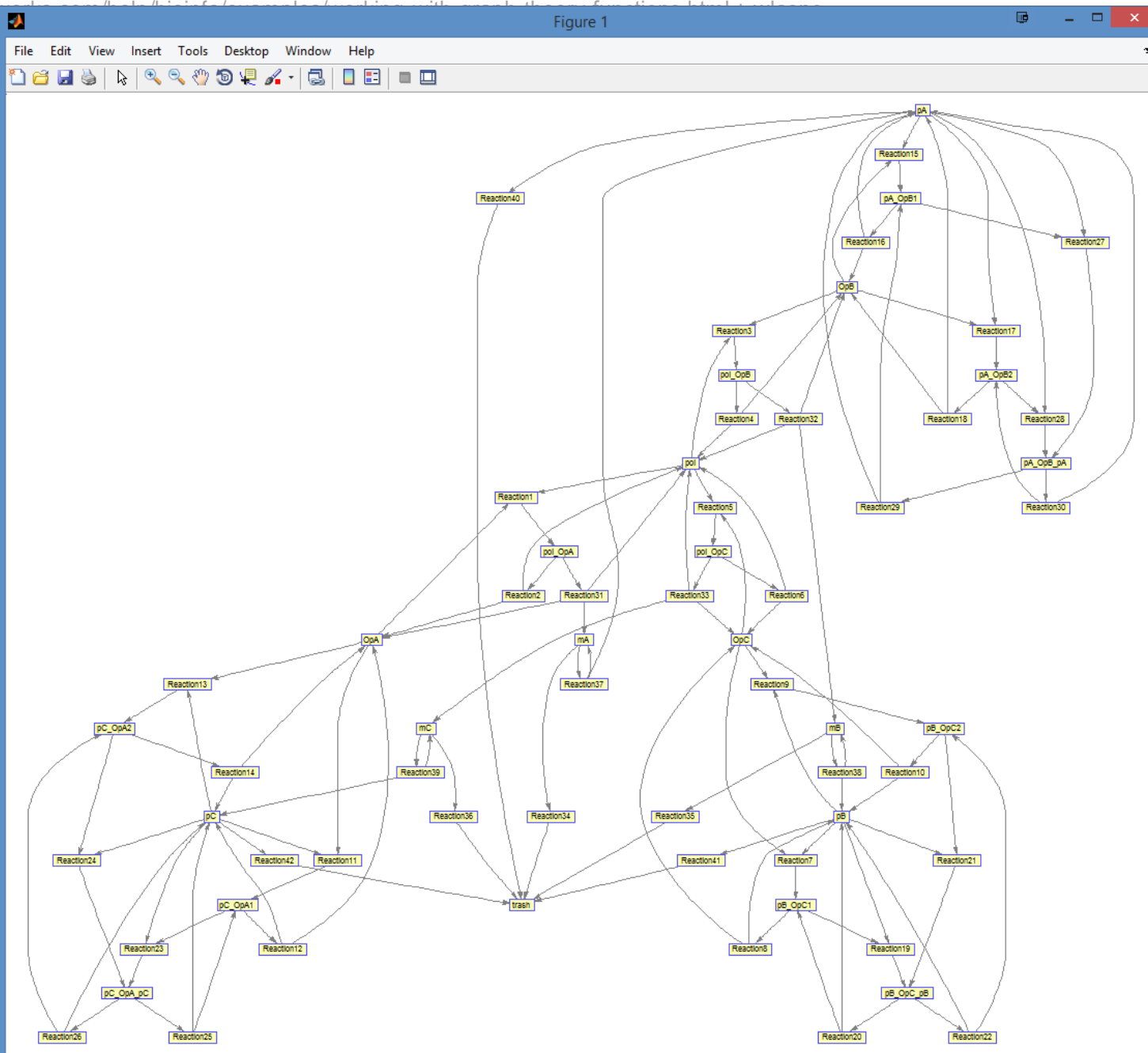




źródło: <http://www.math>

Figure 1

- ```
>> graphE
```



Wyższa Szkoła  
we Wrocławiu

# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- znajdź wierzchołki pA, pB i pC:

```
>> pAnode = find ( strcmp ('pA',names) ) ;  
>> pBnode = find ( strcmp ('pB',names) ) ;  
>> pCnode = find ( strcmp ('pC',names) ) ;
```

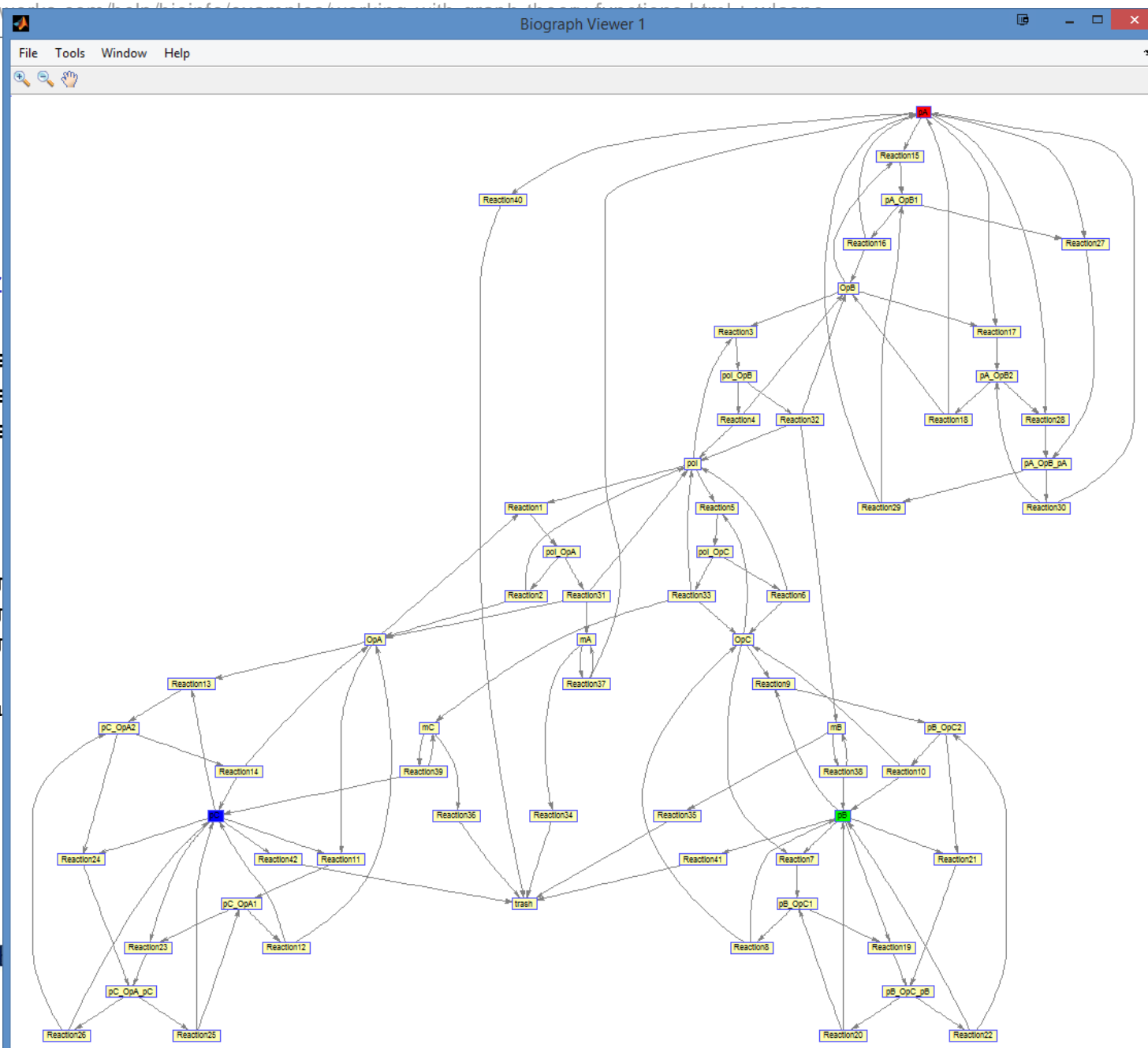
- więc chodź, pomaluj je na: czerwono, zielono i na niebieeeskoo

```
>> set ( graphFigure.nodes(pAnode) , 'Color' , [ 1 0 0 ] , 'size' , [ 40 30 ] ) ;  
>> set ( graphFigure.nodes(pBnode) , 'Color' , [ 0 1 0 ] , 'size' , [ 40 30 ] ) ;  
>> set ( graphFigure.nodes(pCnode) , 'Color' , [ 0 0 1 ] , 'size' , [ 40 30 ] ) ;  
  
>> dolayout ( graphFigure ) ;
```



źródło: <http://www.math>

źródło: <http://www.math>



- ```
>> delayou
```



Wyższa Szkoła  
we Wrocławiu

# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

- znajdź najkrótszą ścieżkę między węzłami pA i pC:  
(wszystkie krawędzie mają długość 1)

```
>> [ dist , path , pred ] = shortestpath ( graphObject, pAnode , pCnode )
```

```
dist =      14
```

```
path =      1      38      14      39      8      26      12      27      10      28      13      56      6      62      3
```

```
pred =      0      61      62      54      55      56      54      39      56      27      24      26      28      38      30
      34      50      40      32      36      42      47      63      10      11      8      12      10      13      2
      15      2      19      7      16      7      20      1      14      1      18      2      21      2      21
      16      20      22      22      1      1      17      17      11      12      13      4      5      6      4
      5      6      1      2      3
```



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- znajdź najkrótszą ścieżkę między węzłami pA i pC:  
(wszystkie krawędzie mają długość 1)

```
>> [ dist , path , pred ] = shortestpath ( graphObject, pAnode , pCnode )
```

- pokoloruj ją:

```
>> set ( graphFigure.nodes(path) , 'Color' , [ 1 0.4 0.4 ] )
```

```
>> edges = getedgesbynodeid ( graphFigure , ...  
                                get ( graphFigure.nodes(path) , 'ID' ) )  
    % get edges by node id
```

```
>> set ( edges , 'LineColor' , [ 1 0 0 ] )
```

```
>> set ( edges , 'LineWidth' ,      1.5      )
```



źródło: [http://www.mathworks.com/help/matlab/creating\\_plots/creating-a-3d-surface-plot.html](http://www.mathworks.com/help/matlab/creating_plots/creating-a-3d-surface-plot.html)

- ```
>> set (
```



Wyższa Szkoła  
we Wrocławiu

# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- stawersuj graf, począwszy od węzła pA:

```
>> traverseOrder = traverse ( graphObject , pAnode )
```

```
traverseOrder =      1      38      14      39       8      26      12      27      10
                   24      11      25       7      34      16      35       3      36
                   20      37      47      22      48      49      46      65      23
                   54       4      57      60      28      13      29       9      30
                   15      31       2      32      19      33      44      21      43
                   45      42      64      56       6      59      62      55       5
                   58      61      40      18      41      51      17      52      53
                   50      63
```



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- strawersuj graf, począwszy od węzła pA:

```
>> traverseOrder = traverse ( graphObject , pAnode )  
    % domyślnie: 'Method' , 'DFS' = Depth-first search
```

- znajdź alternatywną drogę z węzła pA do węzła pC:

```
>> altPathAC = traverseOrder ( 1 : find ( traverseOrder == pCnode ) ) ;  
  
>> set ( graphFigure.nodes(altPathAC) , 'Color' , [ 0.4 0.4 1 ] )  
  
>> edges = getedgesbynodeid ( graphFigure , ...  
                                get(graphFigure.nodes(altPathAC), 'ID') )  
    % get edges by node id  
  
>> set ( edges , 'LineColor' , [ 0 0 1 ] )  
  
>> set ( edges , 'LineWidth' ,      1.5      )
```







# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

- strawersuj graf, począwszy od węzła pA – inną metodą:

```
>> traverseOrder = traverse ( graphObject , pAnode )  
% poprzednio - domyślnie: 'Method' , 'DFS' = Depth-first search
```

```
traverseOrder = 1    38    14    39     8    26    12    27    10    24    11    25     7    34  
                16    35     3    36    20    37    47    22    48    49    46    65    23    54  
                4    57    60    28    13    29     9    30    15    31     2    32    19    33  
                44    21    43    45    42    64    56     6    59    62    55     5    58    61  
                40    18    41    51    17    52    53    50    63
```

```
>> traverseOrder2 = traverse ( graphObject , pAnode , 'Method, 'BFS' )  
% teraz - umyślnie: 'Method' , 'BFS' = Breadth-first search
```

```
traverseOrder2 = 1    38    40    50    51    63    14    18    17    23    39    41    52    53  
                 8    26    12    27    55    10     5    24    28    58    61    11    13     2  
                25    54    29    56    30    32    42    44    64     7     4     9     6    15  
                19    21    34    36    57    60    59    62    31    33    43    45    16    20  
                 3    35    46    37    47    65    22    48    49
```



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- znajdź drugą alternatywną drogę z węzła pA do węzła pC – wypisz nazwy węzłów:

```
>> nuberOfNodes = length(altPathACDC) ; % 57

>> altPathACDCnames{1,1} = 'pA';
>> altPathACDCnames{numberOfNodes,1} = 'pC';

>> for i = 1:numberOfNodes ,
    altPathACDCnames{i,1} = names { altPathACDC(i) , 1 } ;
end
```



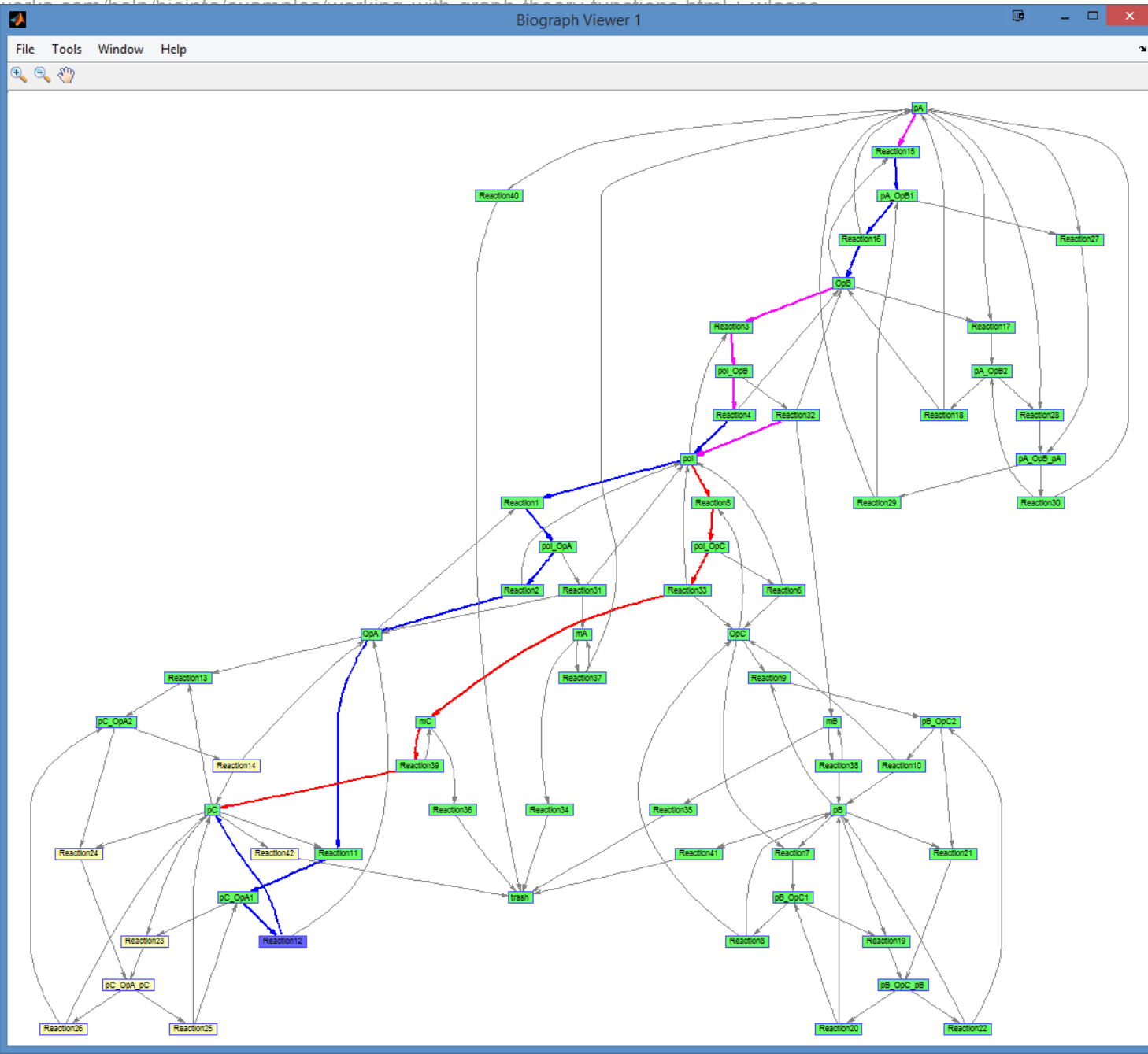
## Repressilator oscillatory network

(cokolwiek to znaczy ;-)

źródło: <http://www.mathy>

```
{ } altPathACDCnames <57x1 cell>
```

|    | 1          | 2 |  |
|----|------------|---|--|
| 1  | pA         |   |  |
| 2  | Reaction15 |   |  |
| 3  | Reaction17 |   |  |
| 4  | Reaction27 |   |  |
| 5  | Reaction28 |   |  |
| 6  | Reaction40 |   |  |
| 7  | pA_OpB1    |   |  |
| 8  | pA_OpB2    |   |  |
| 9  | pA_OpB_pA  |   |  |
| 10 | trash      |   |  |
| 11 | Reaction16 |   |  |
| 12 | Reaction18 |   |  |
| 13 | Reaction29 |   |  |
| 14 | Reaction30 |   |  |
| 15 | OpB        |   |  |
| 16 | Reaction3  |   |  |
| 17 | pol_OpB    |   |  |
| 18 | Reaction4  |   |  |
| 19 | Reaction32 |   |  |
| 20 | pol        |   |  |
| 21 | mB         |   |  |
| 22 | Reaction1  |   |  |



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graph-theory-functions.html> + własne

---

- poprzednia droga – niebieska – nazwy węzłów:

```
>> nuberOfNodes = length(altPathAC) ; % 17

>> altPathACnames{1,1} = 'pA';
>> altPathACnames{numberOfNodes,1} = 'pC';

>> for i = 1:numberOfNodes ,
    altPathACnames{i,1} = names { altPathAC(i) , 1 } ;
end
```



# Repressilator oscillatory network (cokolwiek to znaczy ;-)

źródło: <http://www.mathworks.com/help/bioinfo/examples/working-with-graphs-functions.html#examples/working-with-graphs-functions/working-with-graphs-functions-1>

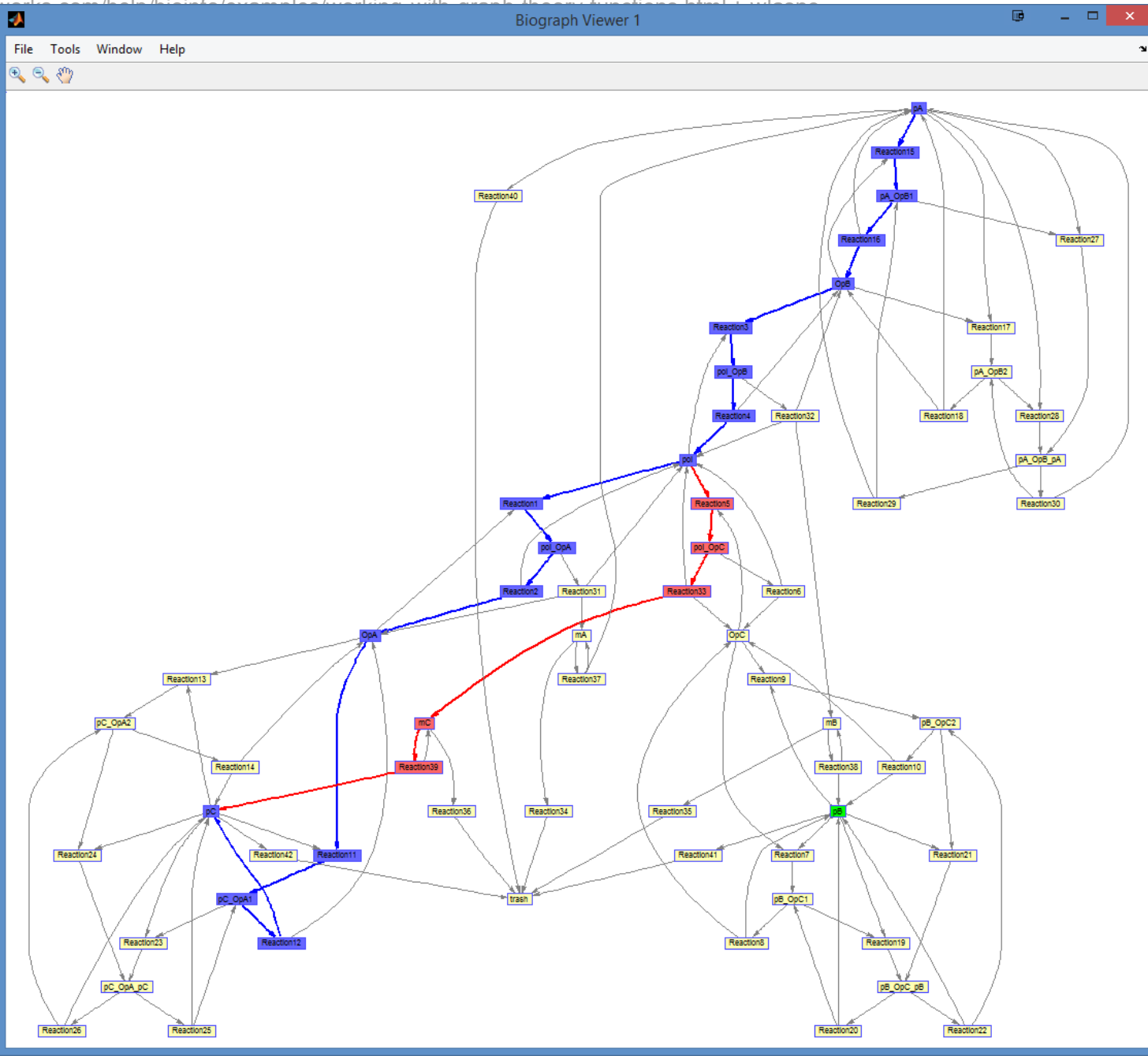
• strawersuj g

>> travers

% domys

altPathACNames <17x1 cell>

|    | 1          | 2 |
|----|------------|---|
| 1  | pA         |   |
| 2  | Reaction15 |   |
| 3  | pA_OpB1    |   |
| 4  | Reaction16 |   |
| 5  | OpB        |   |
| 6  | Reaction3  |   |
| 7  | pol_OpB    |   |
| 8  | Reaction4  |   |
| 9  | pol        |   |
| 10 | Reaction1  |   |
| 11 | pol_OpA    |   |
| 12 | Reaction2  |   |
| 13 | OpA        |   |
| 14 | Reaction11 |   |
| 15 | pC_OpA1    |   |
| 16 | Reaction12 |   |
| 17 | pC         |   |



---

## przykład 2

Minimalne drzewo rozpinające



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne



Bioinformatics Toolbox

High-Throughput Sequencing

Network Analysis and Visualization

## graphminspantree

R2014a

Find minimal spanning tree in graph

[expand all in page](#)

### Syntax

```
[Tree, pred] = graphminspantree(G)
```

```
[Tree, pred] = graphminspantree(G, R)
```

```
[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)
```

```
[Tree, pred] = graphminspantree(..., 'Weights', WeightsValue, ...)
```

### Arguments



|     |                                                                                                                             |
|-----|-----------------------------------------------------------------------------------------------------------------------------|
| $G$ | N-by-N sparse matrix that represents an undirected graph. Nonzero entries in matrix $G$ represent the weights of the edges. |
| $R$ | Scalar between 1 and the number of nodes.                                                                                   |



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- funkcja `graphminspantree`:

```
[Tree, pred] = graphminspantree(G)
```

```
[Tree, pred] = graphminspantree(G, R)
```

```
[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)
```

```
[Tree, pred] = graphminspantree(..., 'Weights', WeightsValue, ...)
```

- **wybór metody:** `[Tree, pred] = graphminspantree(..., 'Method', MethodValue, ...)`
  - 'Kruskal' — Grows the minimal spanning tree (MST) one edge at a time by finding an edge that connects two trees in a spreading forest of growing MSTs  
— Time complexity is  $O(E + X \cdot \log(N))$
  - 'Prim' — Default algorithm. Grows the minimal spanning tree (MST) one edge at a time by adding a minimal edge that connects a node in the growing MST with any other node  
— Time complexity is  $O(E \cdot \log(N))$



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- **zrób porządek:**

```
>> clear all
```

```
>> close all
```

```
>> clc
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- utwórz graf z 6 węzłami i 11 krawędziami:

```
>> W = [ 0.41  0.29  0.51  0.32  0.50  0.45  ...
         0.38  0.32  0.36  0.29  0.21  ] ;
        % wagi krawędzi - np. odległości między węzłami

>> DG = sparse ( [ 1 1 2 2 3 4 4 5 5 6 6 ] , ...
                  [ 2 6 3 5 4 1 6 3 4 2 5 ] , W ) ;
        %                                     wagi krawędzi
        %                                     - elementy macierzy
        % krawędzie - od
        %               do - indeksy elementów macierzy
        %
        % DG - directed graph
        %     - graf skierowany
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

- utwórz graf z 6 węzłami i 11 krawędziami:

```
>> W = [ 0.41  0.29  0.51  0.32  0.50  0.45  ...  
         0.38  0.32  0.36  0.29  0.21    ] ;  
        % wagi krawędzi - np. odległości między węzłami
```

```
>> DG = sparse ( [ 1 1 2 2 3 4 4 5 5 6 6 ] , ...  
                 [ 2 6 3 5 4 1 6 3 4 2 5 ] , W )
```

```
DG =  
    (4,1)    0.4500  
    (1,2)    0.4100  
    (6,2)    0.2900  
    (2,3)    0.5100  
    (5,3)    0.3200  
    (3,4)    0.5000  
    (5,4)    0.3600  
    (2,5)    0.3200  
    (6,5)    0.2100  
    (1,6)    0.2900  
    (4,6)    0.3800
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- obejrzyj macierz wag:

```
>> DGfull = full ( DG )
```

```
DGfull =
```

|        |        |        |        |        |        |
|--------|--------|--------|--------|--------|--------|
| 0      | 0.4100 | 0      | 0      | 0      | 0.2900 |
| 0      | 0      | 0.5100 | 0      | 0.3200 | 0      |
| 0      | 0      | 0      | 0.5000 | 0      | 0      |
| 0.4500 | 0      | 0      | 0      | 0      | 0.3800 |
| 0      | 0      | 0.3200 | 0.3600 | 0      | 0      |
| 0      | 0.2900 | 0      | 0      | 0.2100 | 0      |



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- w grafie nieskierowanym macierz wag musi być symetryczna:

```
>> DGsumka = DG + DG'
```

```
DGsumka = (2,1) 0.4100
           (4,1) 0.4500
           (6,1) 0.2900
           (1,2) 0.4100
           (3,2) 0.5100
           (5,2) 0.3200
           (6,2) 0.2900
           (2,3) 0.5100
           (4,3) 0.5000
           (5,3) 0.3200
           (1,4) 0.4500
           (3,4) 0.5000
           (5,4) 0.3600
           (6,4) 0.3800
           (2,5) 0.3200
           (3,5) 0.3200
           (4,5) 0.3600
           (6,5) 0.2100
           (1,6) 0.2900
           (3,6) 0.2900
           (4,6) 0.3800
           (5,6) 0.2100
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- w grafie nieskierowanym macierz wag musi być symetryczna:

```
>> DGsumka = DG + DG'
```

- obejrzyj ją:

```
>> DGsumkaFull = full ( DGsumka )
```

```
DGsumkaFull =      0      0.4100      0      0.4500      0      0.2900
      0.4100      0      0.5100      0      0.3200      0.2900
      0      0.5100      0      0.5000      0.3200      0
      0.4500      0      0.5000      0      0.3600      0.3800
      0      0.3200      0.3200      0.3600      0      0.2100
      0.2900      0.2900      0      0.3800      0.2100      0
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- w grafie nieskierowanym macierz wag musi być symetryczna:

```
>> DGsumka = DG + DG'
```

- obejrzyj ją:

```
>> DGsumkaFull = full ( DGsumka )
```

- wyciągnij z niej macierz trójkątną dolną:

```
>> UG = tril ( DGsumka ) ;  
    % UG - undirected graph  
    %      graf nieskierowany
```





# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- w grafie nieskierowanym macierz wag musi być symetryczna:

```
>> DGsumka = DG + DG'
```

- obejrzyj ją:

```
>> DGsumkaFull = full ( DGsumka )
```

- wyciągnij z niej macierz trójkątną dolną:

```
>> UG = tril ( DGsumka ) ;
```

- obejrzyj ją:

```
>> UGfull = full ( UG )
```

- obejrzyj graf:



Wyższa Szkoła Bankowa

```
>> view ( biograph ( UG , [] , 'ShowArrows' , 'off' , ...  
    'ShowWeights' , 'on' ) )
```

[www.wsb.pl](http://www.wsb.pl)

# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

- w grafie nieskierowanym macierz

```
>> DGsumka = DG + DG'
```

- obejrzyj ją:

```
>> DGsumkaFull = full
```

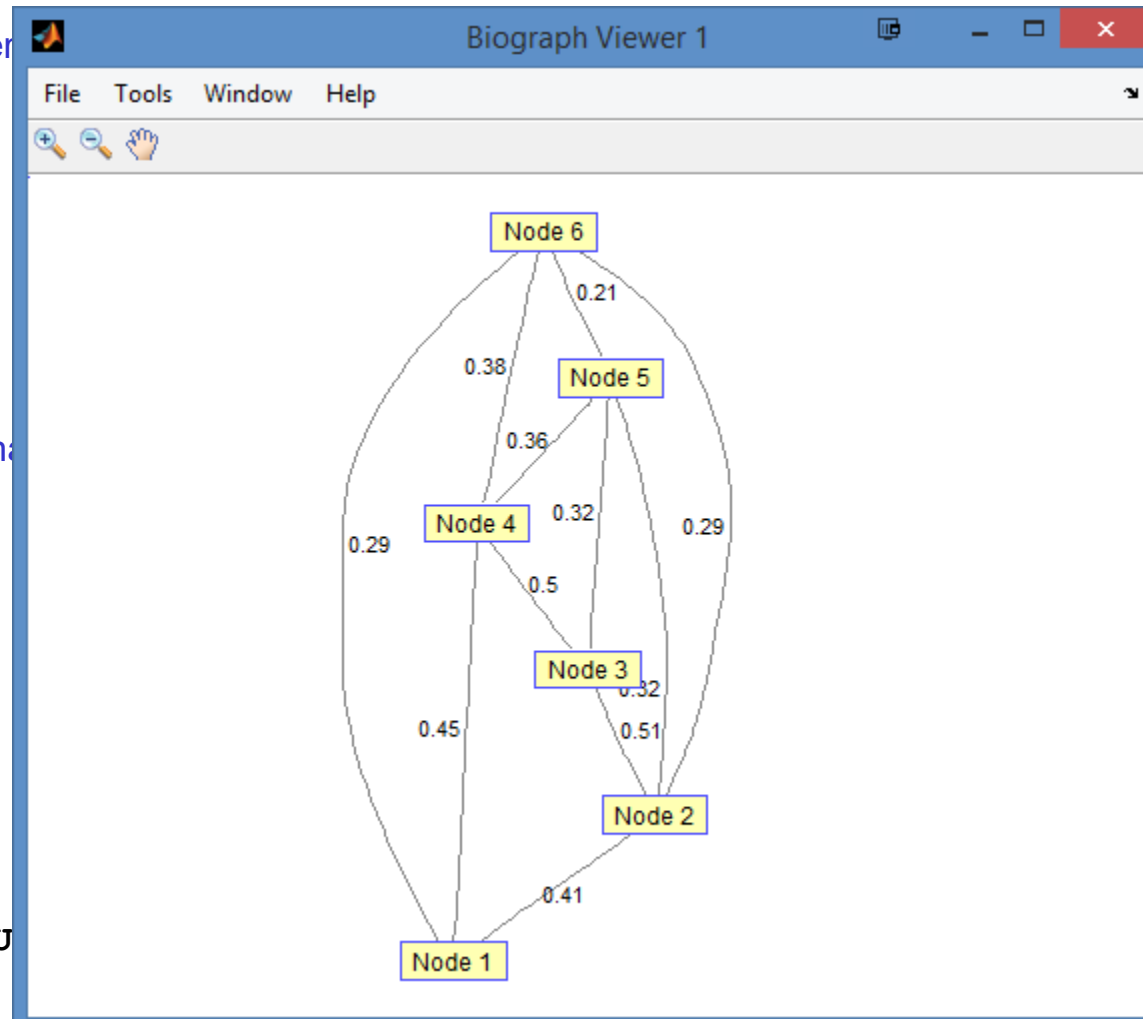
- wyciągnij z niej macierz trójkątną

```
>> UG = tril ( DG ) ;
```

- obejrzyj ją:

```
>> UGfull = full ( UG
```

- obejrzyj graf:



Wyższa Szkoła Bankowa

wrocław we Wrocławiu

```
>> view ( biograph ( UGfull
```

# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- rozeplnij drzewo:

```
>> [ ST , pred ] = graphminspantree ( UG )  
      % graph min span tree
```

```
ST =      (6,1)      0.2900  
          (6,2)      0.2900  
          (5,3)      0.3200  
          (5,4)      0.3600  
          (6,5)      0.2100
```

```
pred =      0      6      5      5      6      1
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- rozeplnij drzewo:

```
>> [ ST , pred ] = graphminspantree ( UG )  
      % graph min span tree
```

- obejrzyj drzewo – jako macierz:

```
>> STfull = full ( ST )
```

```
STfull =  
      0      0      0      0      0      0  
      0      0      0      0      0      0  
      0      0      0      0      0      0  
      0      0      0      0      0      0  
      0      0      0.3200      0.3600      0      0  
0.2900      0.2900      0      0      0.2100      0
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

- rozepnij drzewo:

```
>> [ ST , pred ] = graphminspantree ( UG )  
      % graph min span tree
```

- obejrzyj drzewo – jako macierz:

```
>> STfull = full ( ST )
```

- obejrzyj drzewo – jako drzewo:

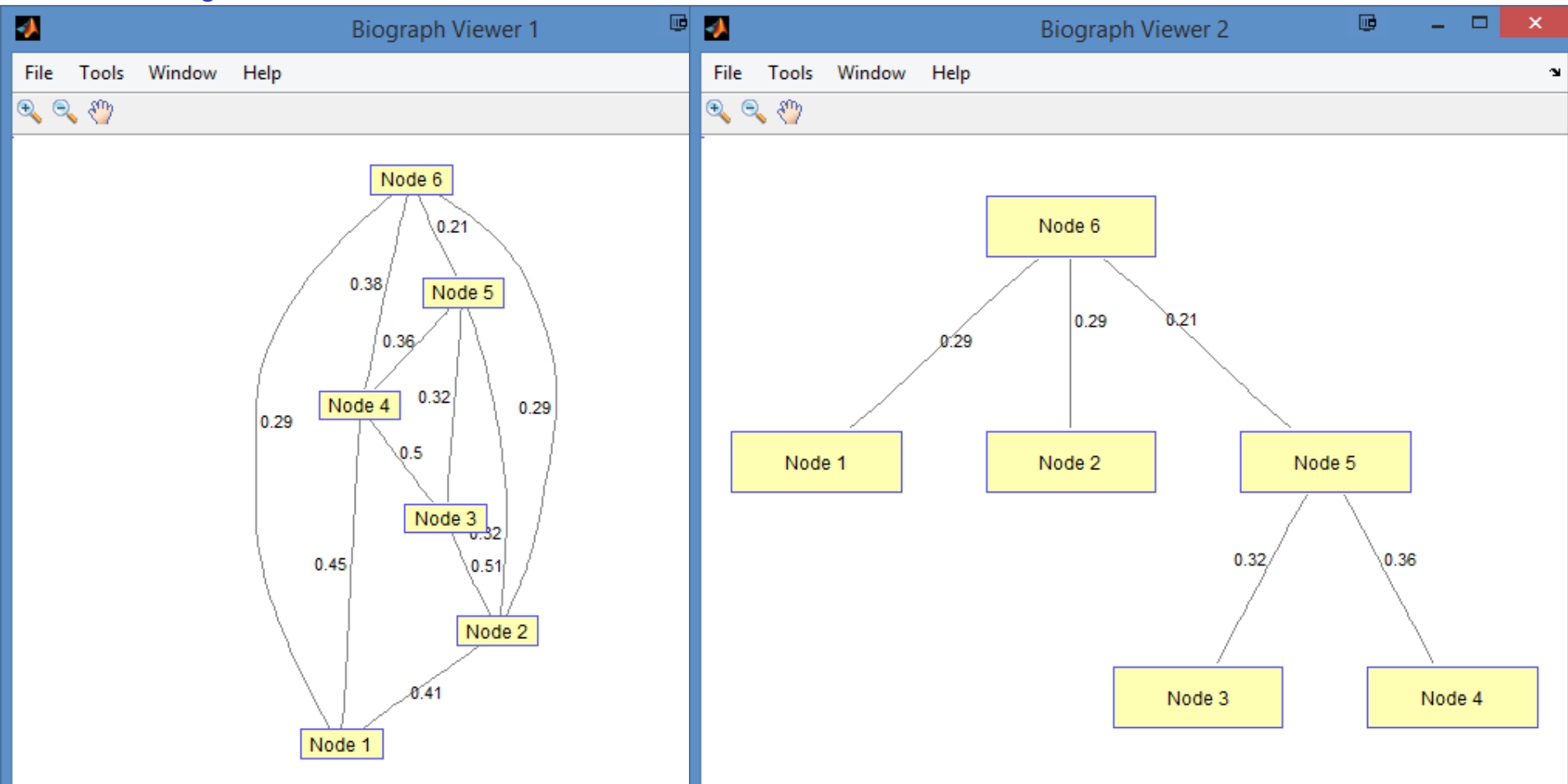
```
>> view ( biograph ( ST , [] , 'ShowArrows' , 'off' , ...  
      'ShowWeights' , 'on' ) )
```



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

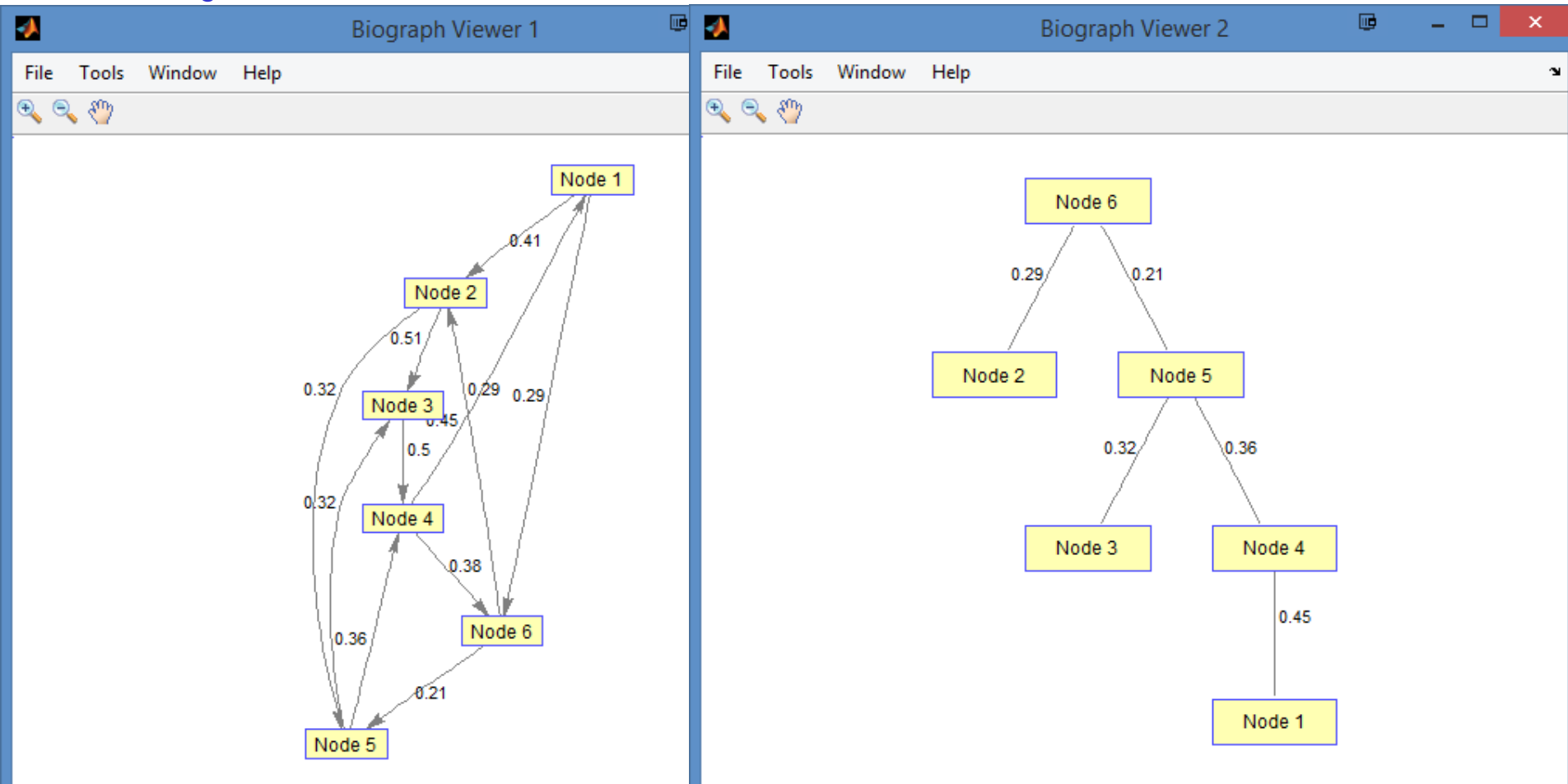
- graf i drzewo – nieskierowane:



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

- graf i drzewo – skierowane:



# Minimalne drzewo rozpinające

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphminspantree.html> + własne

---

## References

---

- [1] Kruskal, J.B. (1956). On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. Proceedings of the American Mathematical Society 7, 48-50.
- [2] Prim, R. (1957). Shortest Connection Networks and Some Generalizations. Bell System Technical Journal 36, 1389-1401.





---

## przykład 3

Najkrótsza ścieżka – graf skierowany





Bioinformatics Toolbox

High-Throughput Sequencing

Network Analysis and Visualization

## graphshortestpath

R2014a

Solve shortest path problem in graph

[expand all in page](#)

### Syntax

```
[dist, path, pred] = graphshortestpath(G, S)
```

```
[dist, path, pred] = graphshortestpath(G, S, T)
```

```
[...] = graphshortestpath(..., 'Directed', DirectedValue, ...)
```

```
[...] = graphshortestpath(..., 'Method', MethodValue, ...)
```

```
[...] = graphshortestpath(..., 'Weights', WeightsValue, ...)
```

### Arguments

|                 |                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $G$             | N-by-N sparse matrix that represents a graph. Nonzero entries in matrix $G$ represent the weights of the edges.                                                                                                              |
| $S$             | Node in $G$ .                                                                                                                                                                                                                |
| $T$             | Node in $G$ .                                                                                                                                                                                                                |
| $DirectedValue$ | Property that indicates whether the graph is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> . |



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

- funkcja `graphshortestpath`:

```
[dist, path, pred] = graphshortestpath(G, S)
[dist, path, pred] = graphshortestpath(G, S, T)
```

```
[...] = graphshortestpath(..., 'Directed', DirectedValue, ...)
[...] = graphshortestpath(..., 'Method', MethodValue, ...)
[...] = graphshortestpath(..., 'Weights', WeightsValue, ...)
```

- wyбір metody: `[Tree, pred] = graphshortestpath(..., 'Method', MethodValue, ...)`

- 'Bellman-Ford' — Assumes weights of the edges to be nonzero entries in sparse matrix G  
— Time complexity is  $O(N \cdot E)$
- 'BFS' — Breadth-first search  
Assumes all weights to be equal, nonzero entries in sparse matrix G to represent edges  
— Time complexity is  $O(N + E)$
- 'Acyclic' — Assumes G to be a directed acyclic graph  
and that weights of the edges are nonzero entries in sparse matrix G  
— Time complexity is  $O(N + E)$
- 'Dijkstra' — Default algorithm. Assumes weights of the edges to be positive values in sparse matrix G  
— Time complexity is  $O(\log(N) \cdot E)$



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

- **zrób porządek:**

```
>> clear all
```

```
>> close all
```

```
>> clc
```



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

- utwórz graf z 6 węzłami i 11 krawędziami:

```
>> W = [ 0.41  0.99  0.51  0.32  0.15  0.45  
         0.38  0.32  0.36  0.29  0.21   ] ;
```

```
>> DG = sparse ( [ 6 1 2 2 3 4 4 5 5 6 1 ] , ...  
                 [ 2 6 3 5 4 1 6 3 4 3 5 ] , W )
```

```
DG =  
    (4,1)    0.4500  
    (6,2)    0.4100  
    (2,3)    0.5100  
    (5,3)    0.3200  
    (6,3)    0.2900  
    (3,4)    0.1500  
    (5,4)    0.3600  
    (1,5)    0.2100  
    (2,5)    0.3200  
    (1,6)    0.9900  
    (4,6)    0.3800
```



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

- utwórz graf z 6 węzłami i 11 krawędziami:

```
>> W = [ 0.41  0.99  0.51  0.32  0.15  0.45  
         0.38  0.32  0.36  0.29  0.21   ] ;
```

```
>> DG = sparse ( [ 6 1 2 2 3 4 4 5 5 6 1 ] , ...  
                 [ 2 6 3 5 4 1 6 3 4 3 5 ] , W )
```

- obejrzyj go:

```
>> h = view ( biograph ( DG , [] , 'ShowWeights' , 'on' ) )
```

```
Biograph object with 6 nodes and 11 edges.
```



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

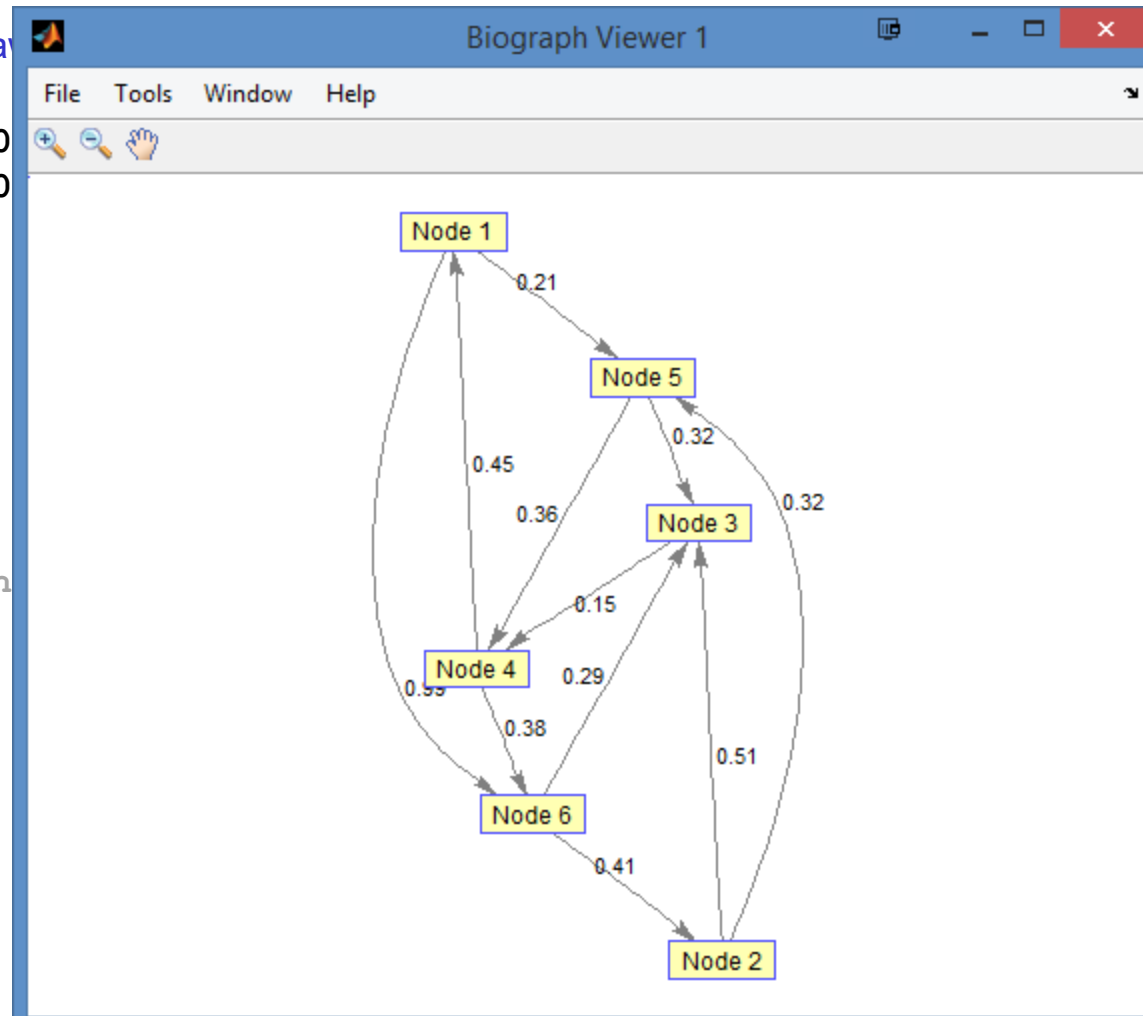
- utwórz graf z 6 węzłami i 11 krawędziami

```
>> W = [ 0.41  0.99  0  
        0.38  0.32  0
```

```
>> DG = sparse ( [ 6 1  
                  [ 2 6
```

- obejrzyj go:

```
>> h = view ( biograph  
              Biograph object with
```



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

- znajdź najkrótszą ścieżkę z węzła 1 do węzła 6:

```
>> [ dist , path , pred ] = graphshortestpath ( DG , 1 , 6 )  
      % graph shortest path
```

```
dist =      0.9500
```

```
path =      1      5      4      6
```

```
pred =      0      6      5      5      1      4
```





# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

- znajdź najkrótszą ścieżkę z węzła 1 do węzła 6:

```
>> [ dist , path , pred ] = graphshortestpath ( DG , 1 , 6 )  
      % graph shortest path
```

- znajdź najkrótszą ścieżkę z węzła 1 do węzła 6 – na grafie:

```
>> set ( h.Nodes(path) , 'Color' , [ 1 0.4 0.4 ] )
```

```
>> edges = getedgesbynodeid ( h , get ( h.Nodes(path) , 'ID' ) ) ;  
      % get edges by node id
```

```
>> set ( edges , 'LineColor' , [ 1 0 0 ] )
```

```
>> set ( edges , 'LineWidth' , 1.5 )
```



# Najkrótsza ścieżka – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

- znajdź najkrótszą ścieżkę z węzła

```
>> [ dist , path , pred
```

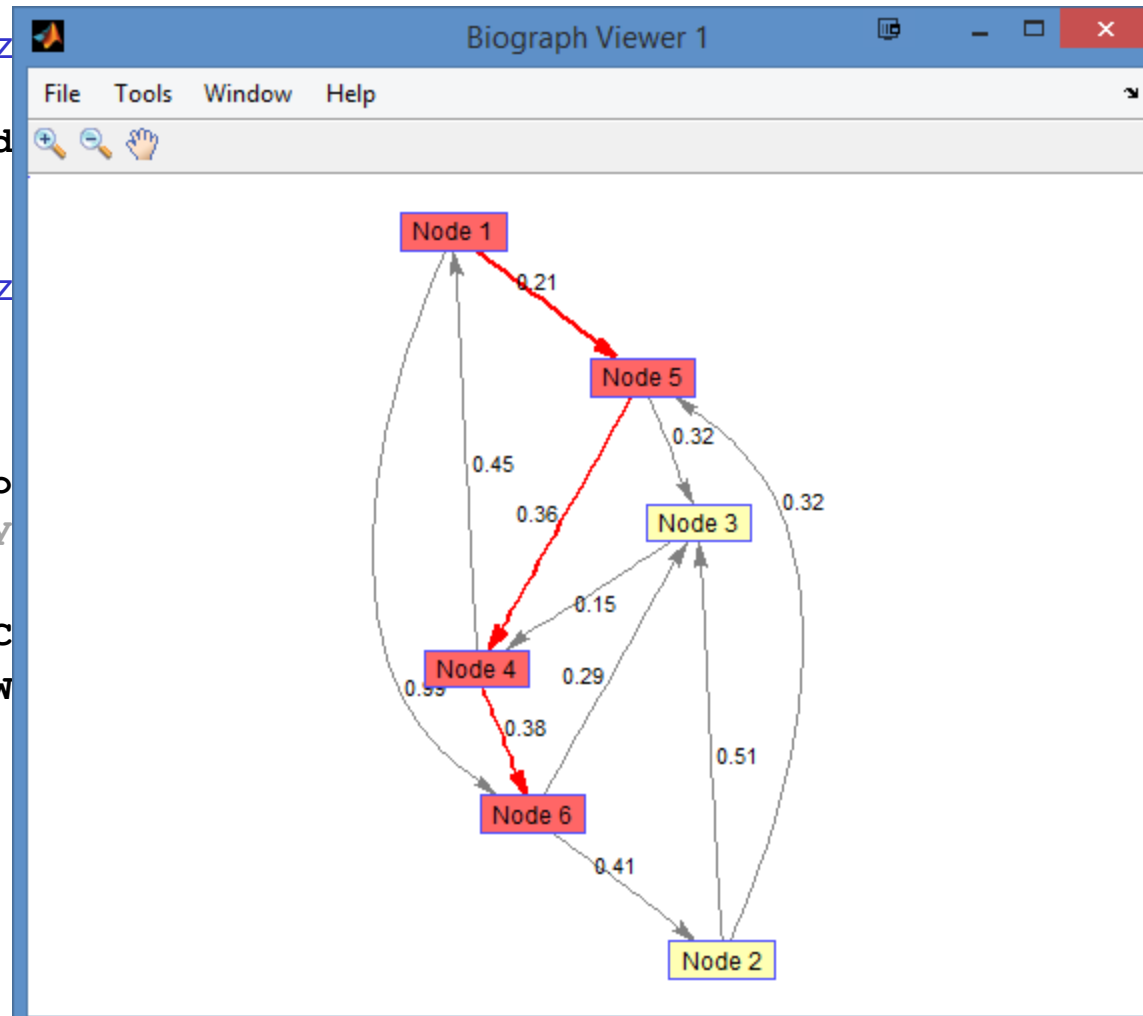
- znajdź najkrótszą ścieżkę z węzła

```
>> set ( h.Nodes(path)
```

```
>> edges = getedgesbyno  
% get edges by
```

```
>> set ( edges , 'LineC
```

```
>> set ( edges , 'LineW
```



---

## przykład 4

Najkrótsza ścieżka – graf Nieskierowany



# Najkrótsza ścieżka – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

- utwórz graf nieskierowany z poprzedniego (skierowanego):

```
>> UG = tril ( DG + DG' )
```

```
UG =      (4,1)      0.4500  
          (5,1)      0.2100  
          (6,1)      0.9900  
          (3,2)      0.5100  
          (5,2)      0.3200  
          (6,2)      0.4100  
          (4,3)      0.1500  
          (5,3)      0.3200  
          (6,3)      0.2900  
          (5,4)      0.3600  
          (6,4)      0.3800
```



# Najkrótsza ścieżka – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

- utwórz graf nieskierowany z poprzedniego (skierowanego):

```
>> UG = tril ( DG + DG' )
```

- obejrzyj go:

```
>> h = view ( biograph ( UG , [] , 'ShowArrows' , 'off' , ...  
                  'ShowWeights' , 'on' , ) )
```



# Najkrótsza ścieżka – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

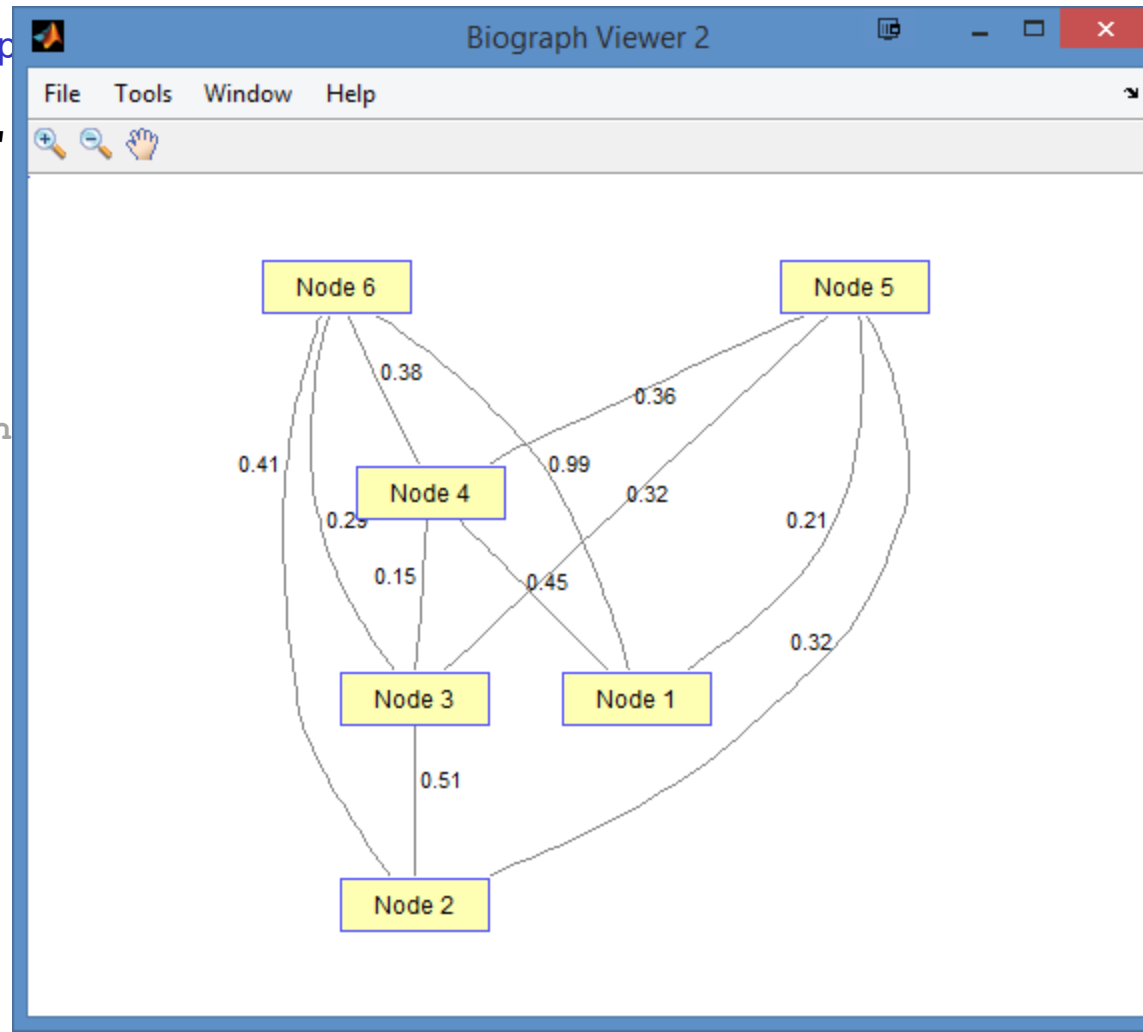
- utwórz graf nieskierowany z pop

```
>> UG = tril ( DG + DG'
```

- obejrzyj go:

```
>> h = view ( biograph
```

Biograph object with



# Najkrótsza ścieżka – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

- znajdź najkrótszą ścieżkę z węzła 1 do węzła 6:

```
>> [ dist , path , pred ] = graphshortestpath ( UG , 1 , 6 , ...  
  'directed' , false )  
                                % graph shortest path  
  
dist =      0.8200  
path  =      1      5      3      6  
pred  =      0      5      5      1      1      3
```

- pokoloruj ją:

```
>> set ( h.Nodes(path) , 'Color' , [ 1 0.4 0.4 ] )  
  
>> fowEdges = getedgesbynodeid ( h , get ( h.Nodes(path) , 'ID' ) );  
>> revEdges = getedgesbynodeid ( h , get ( h.Nodes(fliplr(path)) , 'ID' ) );  
                                % get edges by node id  
  
>> edges = [ fowEdges ; revEdges ] ;  
  
>> set ( edges , 'LineColor' , [ 1 0 0 ] )  
  
>> set ( edges , 'LineWidth' , 1.5 )
```



# Najkrótsza ścieżka – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

- znajdź najkrótszą ścieżkę z węzła

```
>> [ dist , path , pred
```

```
dist =      0.8200
path =      1      5
pred =      0      5
```

- pokoloruj ją:

```
>> set ( h.Nodes(path)
```

```
>> fowEdges = getedgesb
```

```
>> revEdges = getedgesb
           % get edges
```

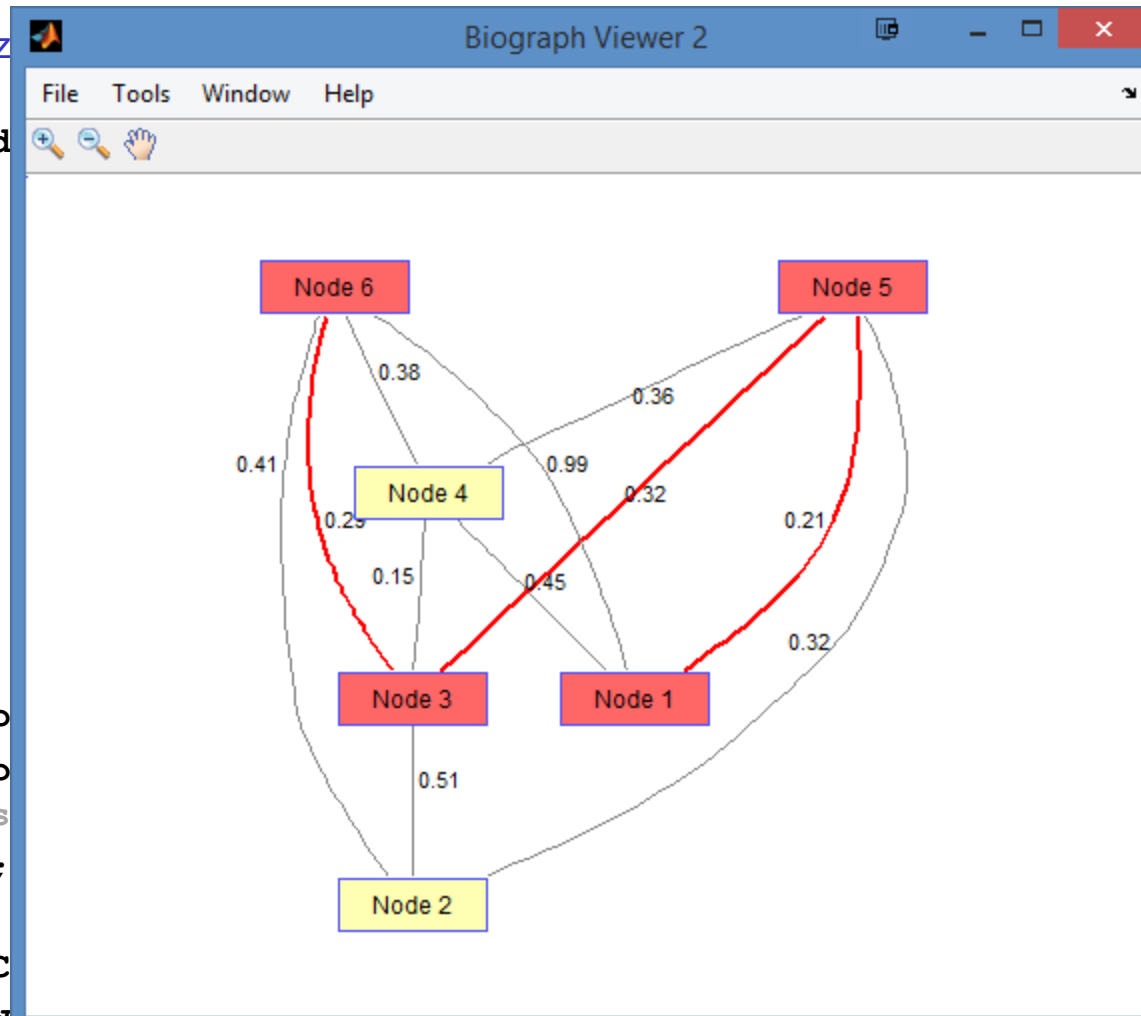
```
>> edges = [ fowEdges ;
```

Wyższa Szkoła Bankowa

we Wrocławiu

```
>> set ( edges , 'LineColor
```

```
>> set ( edges , 'LineStyle
```





# Najkrótsza ścieżka – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/minspantreebiograph.html> + własne

---

## References

---

- [1] Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik* 1, 269-271.
- [2] Bellman, R. (1958). On a Routing Problem. *Quarterly of Applied Mathematics* 16(1), 87-90.



---

## przykład 5

Wszystkie najkrótsze ścieżki  
– graf skierowany



# Wszystkie najkrótsze ścieżki

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne



Bioinformatics Toolbox

High-Throughput Sequencing

Network Analysis and Visualization

## graphallshortestpaths

Find all shortest paths in graph

R2014a

[expand all in page](#)

### Syntax

```
[dist] = graphallshortestpaths(G)
```

```
[dist] = graphallshortestpaths(G, ...'Directed', DirectedValue, ...)
```

```
[dist] = graphallshortestpaths(G, ...'Weights', WeightsValue, ...)
```

### Arguments

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>G</i>             | N-by-N sparse matrix that represents a graph. Nonzero entries in matrix <i>G</i> represent the weights of the edges.                                                                                                                                                                                                                                                                                                                                                         |
| <i>DirectedValue</i> | Property that indicates whether the graph is directed or undirected. Enter <i>false</i> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <i>true</i> .                                                                                                                                                                                                                                                             |
| <i>WeightsValue</i>  | Column vector that specifies custom weights for the edges in matrix <i>G</i> . It must have one entry for every nonzero value (edge) in matrix <i>G</i> . The order of the custom weights in the vector must match the order of the nonzero values in matrix <i>G</i> when it is traversed column-wise. This property lets you use zero-valued weights. By default, <code>graphallshortestpaths</code> gets weight information from the nonzero entries in matrix <i>G</i> . |



# Wszystkie najkrótsze ścieżki – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

---

- nie rób porządku, wykorzystaj grafy z poprzedniego przykładu:

```
>> clear all  
>> close all  
>> clc
```

- dla nadgorliwych - utwórz graf z 6 węzłami i 11 krawędziami:

```
>> W = [ 0.41  0.99  0.51  0.32  0.15  0.45  
        0.38  0.32  0.36  0.29  0.21   ] ;  
  
>> DG = sparse ( [ 6 1 2 2 3 4 4 5 5 6 1 ] , ...  
                 [ 2 6 3 5 4 1 6 3 4 3 5 ] , W )
```

- dla wszystkich – obejrzyj graf:

```
>> view ( biograph ( DG , [] , 'ShowWeights' , 'on' ) )
```



# Wszystkie najkrótsze ścieżki – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

- nie rób porządku, wykorzystaj `graphallshortestpaths`

```
>> clear all  
>> close all  
>> clc
```

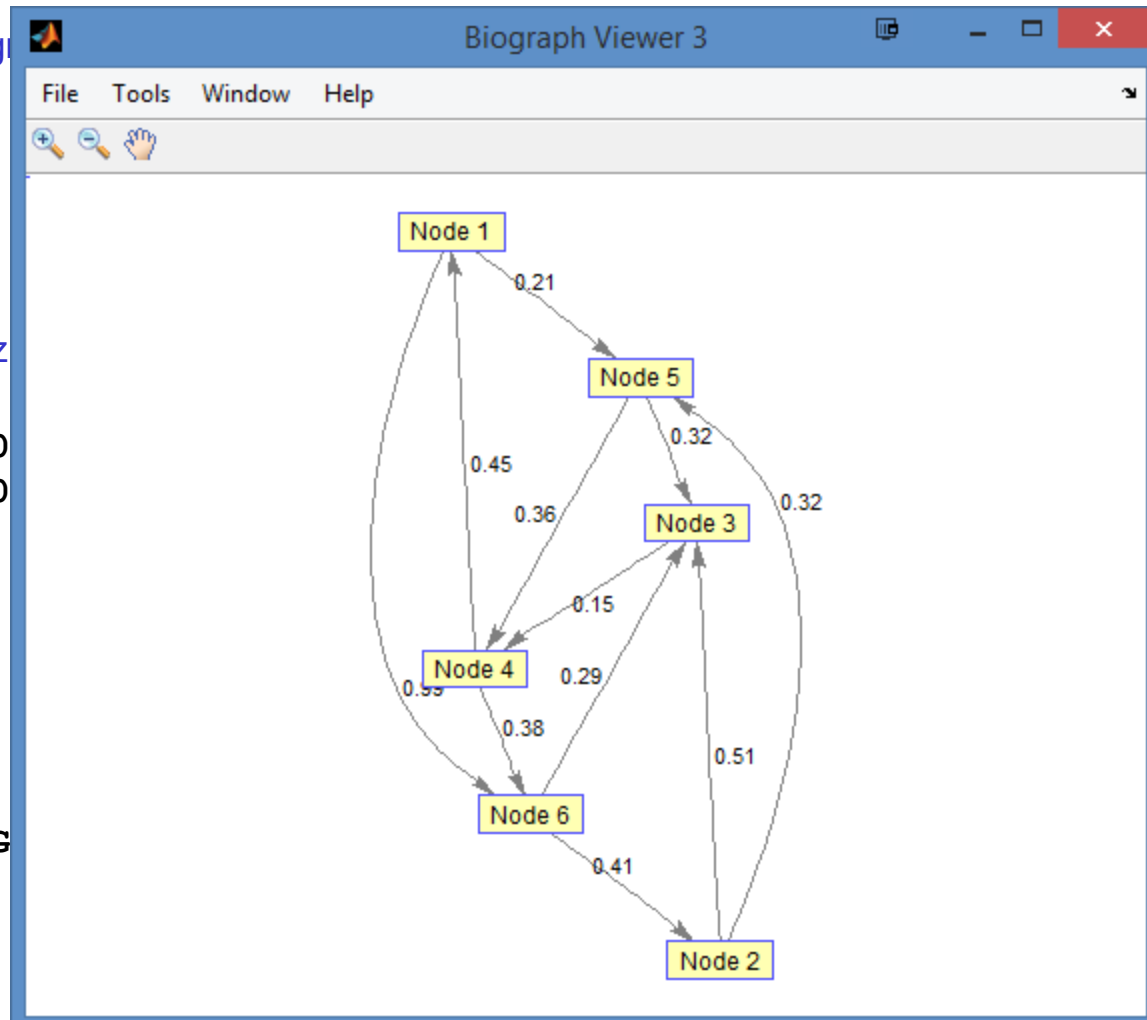
- dla nadgorliwych - utwórz graf z

```
>> W = [ 0.41  0.99  0  
        0.38  0.32  0
```

```
>> DG = sparse ( [ 6 1  
                  [ 2 6
```

- dla wszystkich – obejrzyj graf:

```
>> view ( biograph ( DG
```



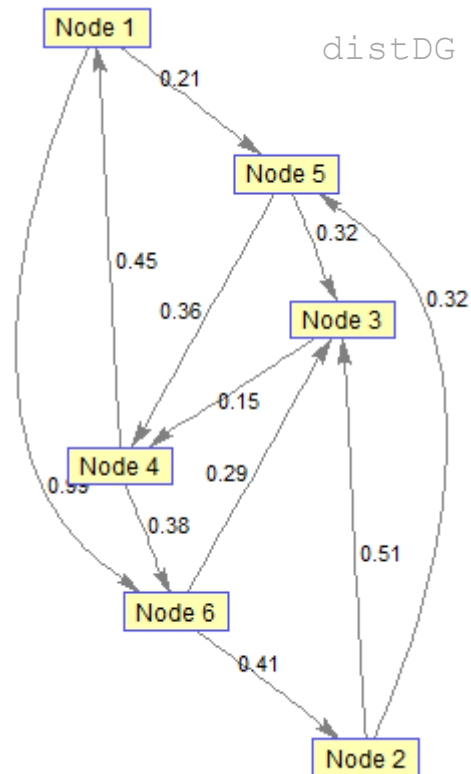
# Wszystkie najkrótsze ścieżki – graf skierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

- znajdź najkrótsze odległości między wszystkimi parami węzłów:

```
>> distDG = graphallshortestpaths ( DG )  
% graph all shortest paths
```

|          |        |        |        |        |        |               |
|----------|--------|--------|--------|--------|--------|---------------|
| distDG = | 0      | 1.3600 | 0.5300 | 0.5700 | 0.2100 | <b>0.9500</b> |
|          | 1.1100 | 0      | 0.5100 | 0.6600 | 0.3200 | 1.0400        |
|          | 0.6000 | 0.9400 | 0      | 0.1500 | 0.8100 | 0.5300        |
|          | 0.4500 | 0.7900 | 0.6700 | 0      | 0.6600 | 0.3800        |
|          | 0.8100 | 1.1500 | 0.3200 | 0.3600 | 0      | 0.7400        |
|          | 0.8900 | 0.4100 | 0.2900 | 0.4400 | 0.7300 | 0             |



nkowa

---

## przykład 6

Wszystkie najkrótsze ścieżki  
– graf nieskierowany



# Wszystkie najkrótsze ścieżki – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

- nic nie rób – wykorzystamy graf nieskierowany (UG) z poprzedniego przykładu

```
>> UG
```

- dla nadgorliwych – utwórz graf nieskierowany o 6 węzłach i 11 krawędziach:

```
>> UG = tril ( DG + DG' )
```

```
UG =      (4,1)      0.4500
          (5,1)      0.2100
          (6,1)      0.9900
          (3,2)      0.5100
          (5,2)      0.3200
          (6,2)      0.4100
          (4,3)      0.1500
          (5,3)      0.3200
          (6,3)      0.2900
          (5,4)      0.3600
          (6,4)      0.3800
```





# Wszystkie najkrótsze ścieżki – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

---

- nic nie rób – wykorzystamy graf nieskierowany (UG) z poprzedniego przykładu

```
>> UG
```

- dla nadgorliwych – utwórz graf nieskierowany o 6 węzłach i 11 krawędziach:

```
>> UG = tril ( DG + DG' )
```

- dla wszystkich – obejrzyj graf nieskierowany:

```
>> view ( biograph ( UG , [] , 'ShowArrows' , 'off' , ...  
                    'ShowWeights' , 'on' ) )
```



# Wszystkie najkrótsze ścieżki – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

- nic nie rób – wykorzystamy graf

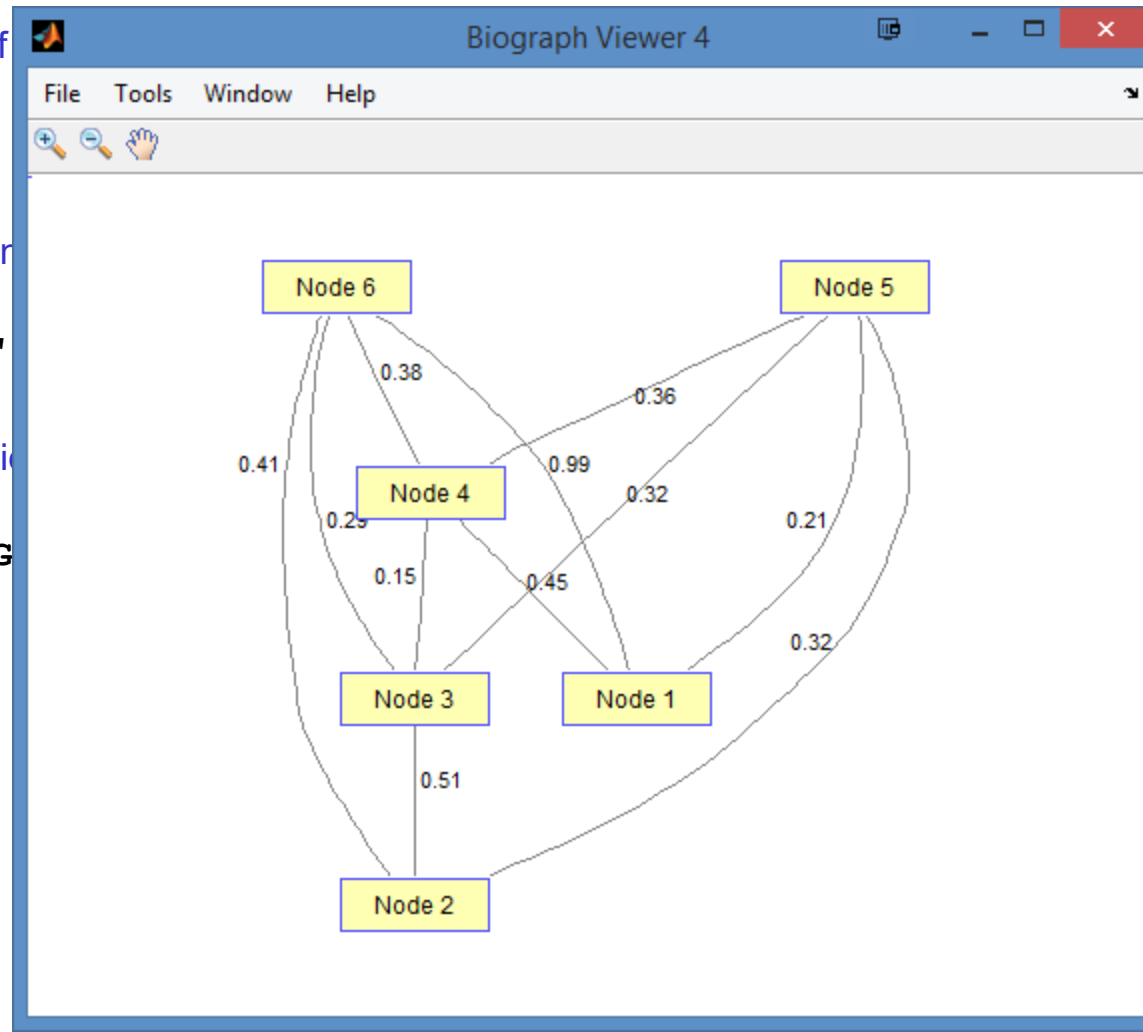
```
>> UG
```

- dla nadgorliwych – utwórz graf

```
>> UG = tril ( DG + DG'
```

- dla wszystkich – obejrzyj graf

```
>> view ( biograph ( UG
```



# Wszystkie najkrótsze ścieżki – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

- znajdź długości najkrótszych połączeń między wszystkimi parami węzłów grafu nieskierowanego:

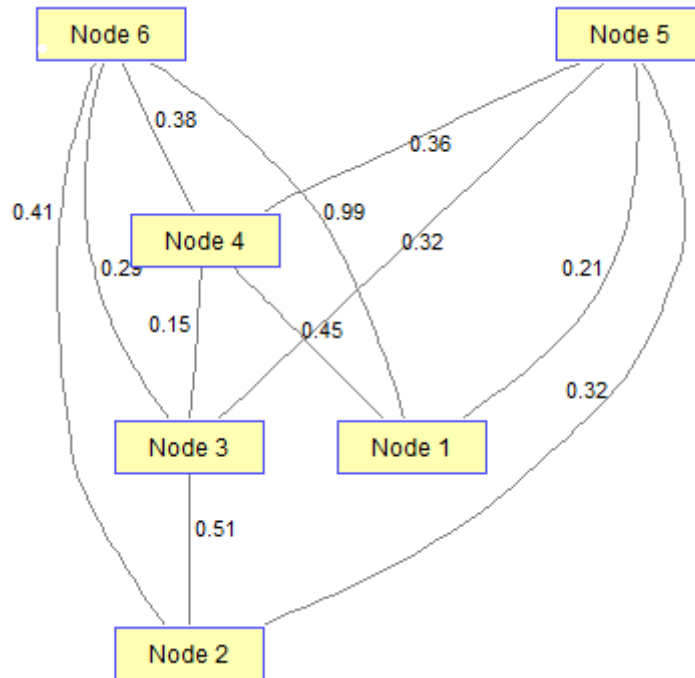
```
>> distUG = graphallshortestpaths ( UG , 'directed' , false )  
      % graph all shortest paths
```

```
distUG =  
          0      0.5300      0.5300      0.4500      0.2100      0.8200  
      0.5300          0      0.5100      0.6600      0.3200      0.4100  
      0.5300      0.5100          0      0.1500      0.3200      0.2900  
      0.4500      0.6600      0.1500          0      0.3600      0.3800  
      0.2100      0.3200      0.3200      0.3600          0      0.6100  
      0.8200      0.4100      0.2900      0.3800      0.6100          0
```



# Wszystkie najkrótsze ścieżki – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne



|          |               |        |        |        |        |               |
|----------|---------------|--------|--------|--------|--------|---------------|
| distUG = | 0             | 0.5300 | 0.5300 | 0.4500 | 0.2100 | <b>0.8200</b> |
|          | 0.5300        | 0      | 0.5100 | 0.6600 | 0.3200 | 0.4100        |
|          | 0.5300        | 0.5100 | 0      | 0.1500 | 0.3200 | 0.2900        |
|          | 0.4500        | 0.6600 | 0.1500 | 0      | 0.3600 | 0.3800        |
|          | 0.2100        | 0.3200 | 0.3200 | 0.3600 | 0      | 0.6100        |
|          | <b>0.8200</b> | 0.4100 | 0.2900 | 0.3800 | 0.6100 | 0             |



Wyższa Szkoła Bankowa  
we Wrocławiu

[www.wsb.pl](http://www.wsb.pl)

# Wszystkie najkrótsze ścieżki – graf skierowany

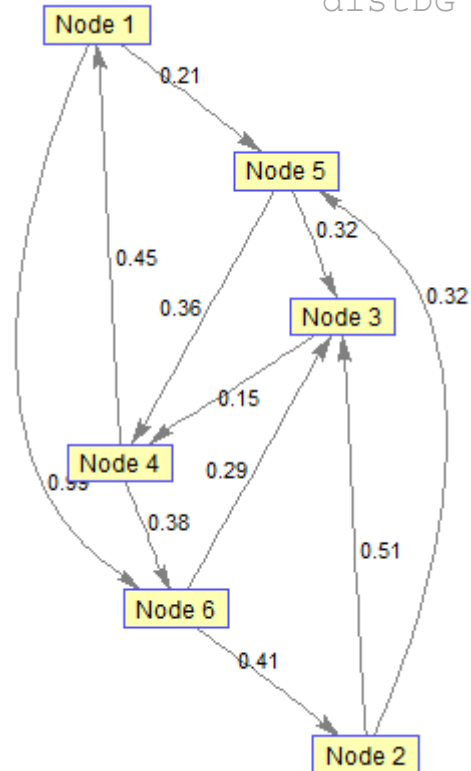
źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

- porównaj wynik z macierzą odległości dla grafu skierowanego:

```
>> distDG
```

```
distDG =
```

|               |        |        |        |        |               |
|---------------|--------|--------|--------|--------|---------------|
| 0             | 1.3600 | 0.5300 | 0.5700 | 0.2100 | <b>0.9500</b> |
| 1.1100        | 0      | 0.5100 | 0.6600 | 0.3200 | 1.0400        |
| 0.6000        | 0.9400 | 0      | 0.1500 | 0.8100 | 0.5300        |
| 0.4500        | 0.7900 | 0.6700 | 0      | 0.6600 | 0.3800        |
| 0.8100        | 1.1500 | 0.3200 | 0.3600 | 0      | 0.7400        |
| <b>0.8900</b> | 0.4100 | 0.2900 | 0.4400 | 0.7300 | 0             |



nkowa

# Wszystkie najkrótsze ścieżki – graf nieskierowany

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphallshortestpaths.html> + własne

---

## References

---

- [1] Johnson, D.B. (1977). Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM* 24(1), 1-13.



---

## przykład 7

Trawers



Wyższa Szkoła Bankowa  
we Wrocławiu

[www.wsb.pl](http://www.wsb.pl)



Bioinformatics Toolbox

High-Throughput Sequencing

Network Analysis and Visualization

## graphtraverse

R2014a

Traverse graph by following adjacent nodes

[expand all in page](#)

### Syntax

```
[disc, pred, closed] = graphtraverse(G, S)
```

```
[...] = graphtraverse(G, S, ...'Depth', DepthValue, ...)
```

```
[...] = graphtraverse(G, S, ...'Directed', DirectedValue, ...)
```

```
[...] = graphtraverse(G, S, ...'Method', MethodValue, ...)
```

### Arguments

|                 |                                                                                                                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $G$             | N-by-N sparse matrix that represents a directed graph. Nonzero entries in matrix $G$ indicate the presence of an edge.                                                                                                       |
| $S$             | Integer that indicates the source node in graph $G$ .                                                                                                                                                                        |
| $DepthValue$    | Integer that indicates a node in graph $G$ that specifies the depth of the search. Default is Inf (infinity).                                                                                                                |
| $DirectedValue$ | Property that indicates whether graph $G$ is directed or undirected. Enter <code>false</code> for an undirected graph. This results in the upper triangle of the sparse matrix being ignored. Default is <code>true</code> . |



- funkcja `graphtraverse`:

```
[disc, pred, closed] = graphtraverse(G, S)
```

```
[...] = graphtraverse(G, S, ...'Depth', DepthValue, ...)
```

```
[...] = graphtraverse(G, S, ...'Directed', DirectedValue, ...)
```

```
[...] = graphtraverse(G, S, ...'Method', MethodValue, ...)
```

- wybór metody: `[...] = graphtraverse(G, S, ...'Method', MethodValue, ...)`
  - 'BFS'—Breadth-first search
    - Time complexity is  $O(N+E)$
  - 'DFS'—Default algorithm. Depth-first search
    - Time complexity is  $O(N+E)$

# Trawers

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphtraverse.html> + własne

---

- **zrób porządek:**

```
>> clear all
```

```
>> close all
```

```
>> clc
```



- utwórz graf z 10 węzłami i 12 krawędziami:

```
>> DG = sparse ( [ 1 2 3 4 5 5 5 6 7 8 8 9 ] , ...  
                 [ 2 4 1 5 3 6 7 9 8 1 10 2] , true , 10 , 10 )
```

```
DG =  
      (3,1)      1  
      (8,1)      1  
      (1,2)      1  
      (9,2)      1  
      (5,3)      1  
      (2,4)      1  
      (4,5)      1  
      (5,6)      1  
      (5,7)      1  
      (7,8)      1  
      (6,9)      1  
      (8,10)     1
```

- utwórz graf z 10 węzłami i 12 krawędziami:

```
>> DG = sparse ( [ 1 2 3 4 5 5 5 6 7 8 8 9 ] , ...  
                 [ 2 4 1 5 3 6 7 9 8 1 10 2] , true , 10 , 10 )
```

- obejrzyj graf – jako macierz pełną:

```
>> DGfull = full ( DG )
```

```
DGfull =
```

|          |          |   |          |          |          |          |          |          |          |
|----------|----------|---|----------|----------|----------|----------|----------|----------|----------|
| 0        | <b>1</b> | 0 | 0        | 0        | 0        | 0        | 0        | 0        | 0        |
| 0        | 0        | 0 | <b>1</b> | 0        | 0        | 0        | 0        | 0        | 0        |
| <b>1</b> | 0        | 0 | 0        | 0        | 0        | 0        | 0        | 0        | 0        |
| 0        | 0        | 0 | 0        | <b>1</b> | 0        | 0        | 0        | 0        | 0        |
| 0        | 0        | 1 | 0        | 0        | <b>1</b> | <b>1</b> | 0        | 0        | 0        |
| 0        | 0        | 0 | 0        | 0        | 0        | 0        | 0        | <b>1</b> | 0        |
| 0        | 0        | 0 | 0        | 0        | 0        | 0        | <b>1</b> | 0        | 0        |
| <b>1</b> | 0        | 0 | 0        | 0        | 0        | 0        | 0        | 0        | <b>1</b> |
| 0        | <b>1</b> | 0 | 0        | 0        | 0        | 0        | 0        | 0        | 0        |
| 0        | 0        | 0 | 0        | 0        | 0        | 0        | 0        | 0        | 0        |

- utwórz graf z 10 węzłami i 12 krawędziami:

```
>> DG = sparse ( [ 1 2 3 4 5 5 5 6 7 8 8 9 ] , ...  
                 [ 2 4 1 5 3 6 7 9 8 1 10 2] , true , 10 , 10 )
```

- obejrzyj graf – jako macierz pełną:

```
>> DGfull = full ( DG )
```

- obejrzyj graf – na rysunku:

```
>> h = view ( biograph ( DG ) )
```

- utwórz graf z 10 węzłami i 12 krawędziami

```
>> DG = sparse ( [ 1 2  
                  [ 2 4
```

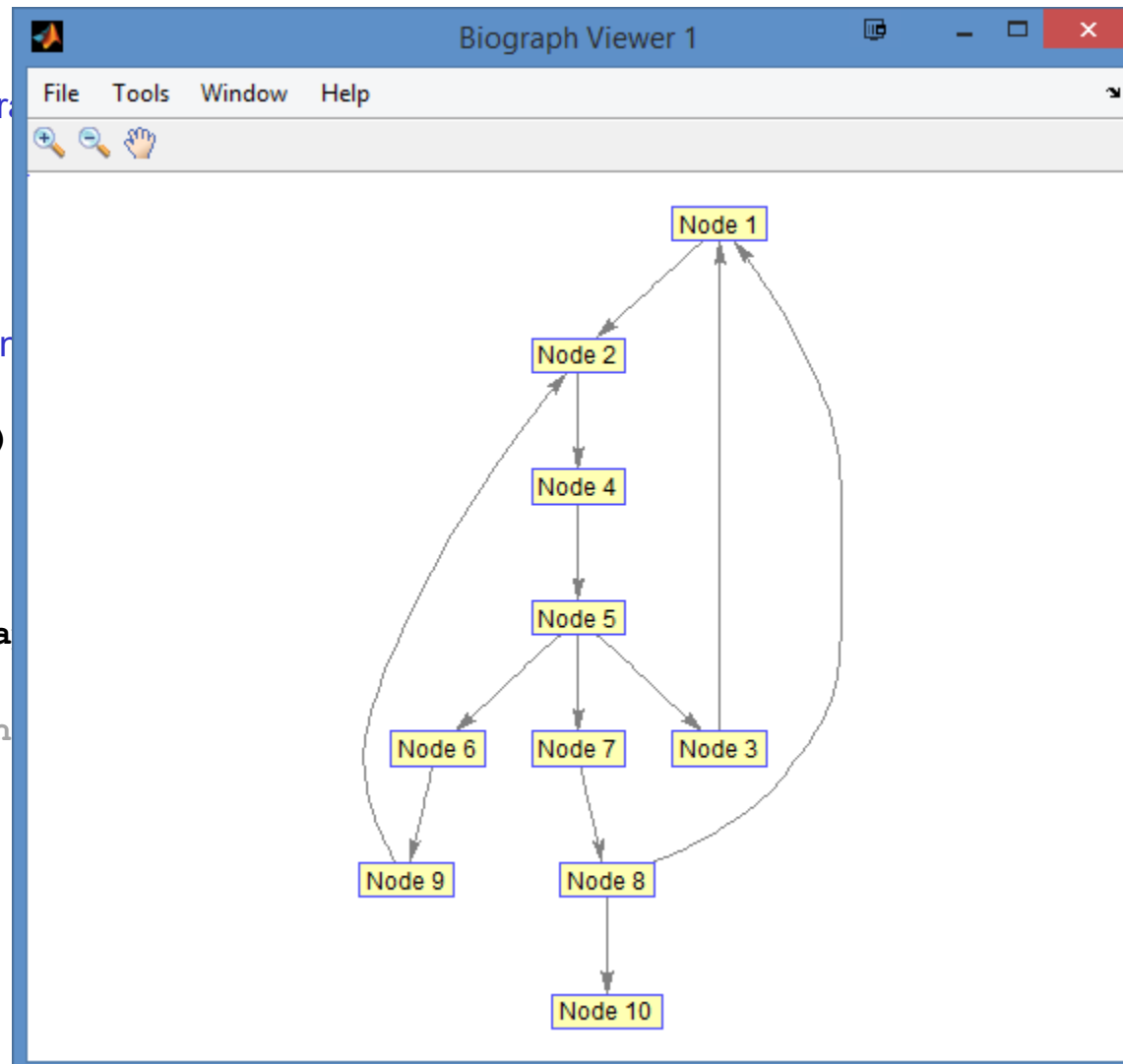
- obejrzyj graf – jako macierz pełna

```
>> DGfull = full ( DG )
```

- obejrzyj graf – na rysunku:

```
>> hDFS = view ( biograph
```

Biograph object with



- stawersuj graf - zaczynając od węzła 4
  - przy użyciu metody DFS – przeszukiwania włąb

```
>> orderDFS4 = graphtraverse ( DG , 4 )
```

```
orderDFS4 =
```

```
         4         5         3         1         2         6         9         7         8        10
```

- ponumeruj węzły – w kolejności ich przechodzenia

```
>> for i = 1:10
    hDFS.Nodes( orderDFS4(i) ).Label = ...
        sprintf ( '%s:%d' , hDFS.Nodes( orderDFS4(i) ).ID , i ) ;
end

hDFS.ShowTextInNodes = 'label'

dolayout ( hDFS )
```

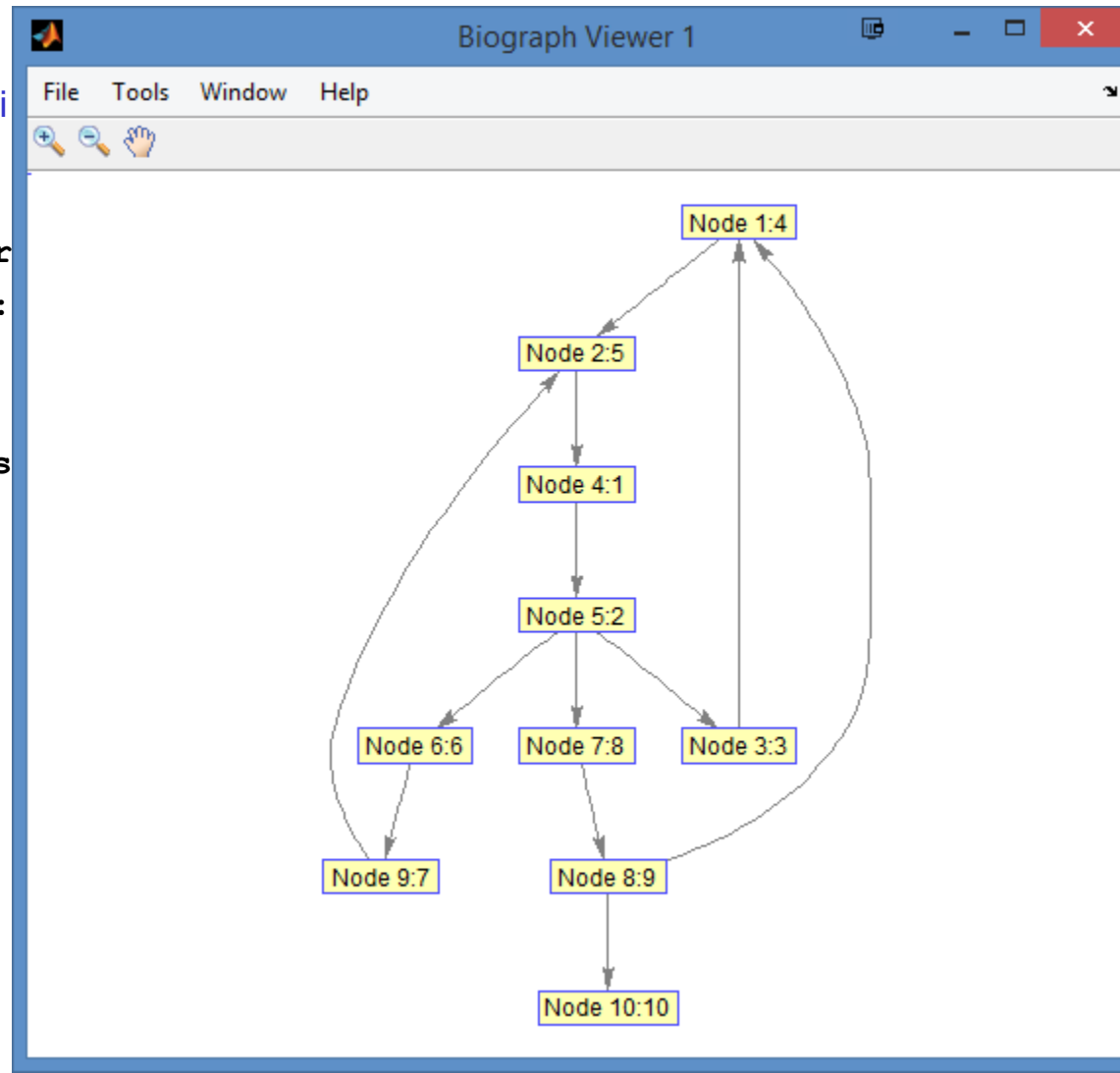


- ponumeruj węzły – w kolejności

```
>> for i = 1:10
    hDFS.Nodes( order
        sprintf ( '%s:
end

hDFS.ShowTextInNodes

dolayout ( hDFS )
```



- stawersuj graf - zaczynając od węzła 4
  - przy użyciu metody BFS – przeszukiwania wszerz

```
>> orderBFS4 = graphtraverse ( DG , 4 , 'Method' , 'BFS' )
```

```
orderBFS4 =
```

```
         4         5         3         6         7         1         9         8         2        10
```

- narysuj graf w nowym okienku

```
>> hBFS = view ( biograph ( DG ) )
```

- ponumeruj węzły – w kolejności ich przechodzenia

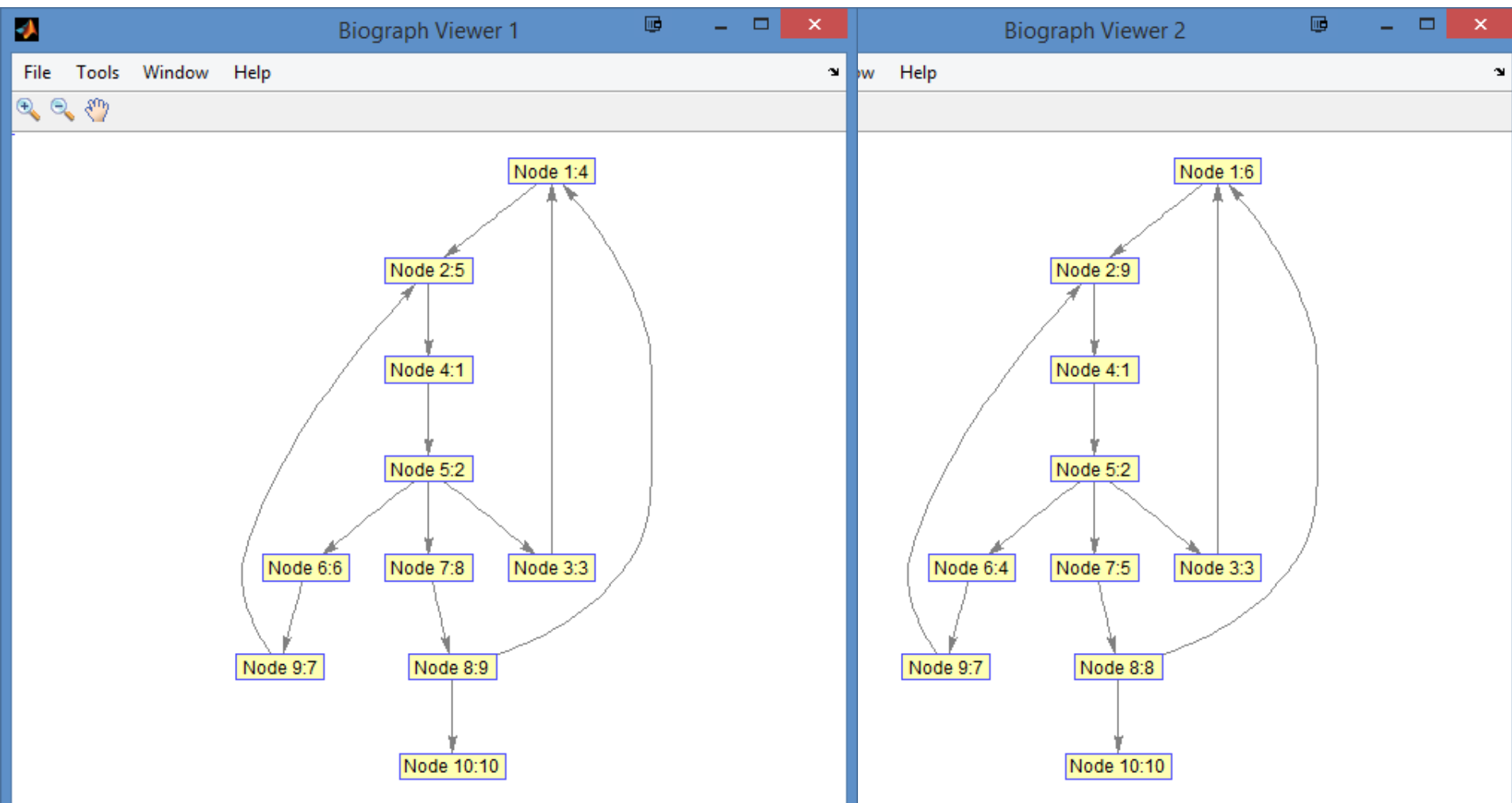
```
>> for i = 1:10
    hBFS.Nodes( orderDFS4(i) ).Label = ...
        sprintf ( '%s:%d' , hBFS.Nodes( orderBFS4(i) ).ID , i ) ;
end

hBFS.ShowTextInNodes = 'label'

dolayout ( hBFS )
```

# Trawers

źródło: <http://www.mathworks.com/help/bioinfo/ref/graphtraverse.html> + własne



- znajdź węzły w odległości 2 od węzła 4:

```
>> node_idx = graphtraverse ( DG , 4 , 'depth' , 2 )
```

```
node_idx =      4      5      3      6      7
```

- zaznacz je na rysunku:

```
>> set ( hDFS.nodes(node_idx) , 'Color' , [ 1 0 0 ] )
```

- znajdź węzły w odległości 2 od v

```
>> node_idx = graphtra
```

```
node_idx = 4
```

- zaznacz je na rysunku:

```
>> set ( hDFS.nodes (nod
```

