# Diskretna matematika 2

Gradiva za vaje iz diskretne matematike 2

Univerza v Ljubljani, Fakulteta za računalnišvo in informatiko

**Marko Boben**

*June 2023*

# Contents

# List of Figures

# List of Tables

# 1. Introduction

## 1.1 What is Sage?

Algorithms in this Notes are implemented in Python programming language using SageMath (https://www.sagemath.org).

> SageMath is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more.

You can download binaries at http://www.sagemath.org/download.html for Mac, and Windows.

Note: Binaries for Windows are avaliable up to version 9.3 (late 2021). For newer versions you will need to install it in WSL. Follow the instructions athttps://doc.sagemath.org/html/en/installation/index.html.

There is also a cloud version available at https://cocalc.com/
Documentation can be found at https://doc.sagemath.org/html/en/index.html.
We will moslty use *graph theory* package https://doc.sagemath.org/html/en/reference/graphs/index.html

## 1.2 Some examples of Sage Graph Theory objects and methods

For representing undirected graphs we use the `Graph` class, while for representing directed graphs we use the `DiGraph` class.

### 1.2.1 Undirected graphs

Undirected graph is represented using `Graph` class.

```
G = Graph({0:[1,2,3], 4:[0,2], 6:[1,2,3,4,5]})
```

There are many methods to access the graph properties. For example, to get a list of vertices use `vertices` method.

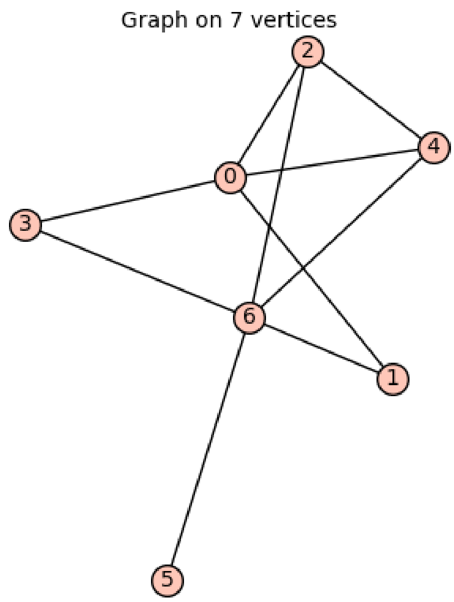
```
G.vertices()
```

```
[0,1,2,3,4,5,6]
```

To display the graph, simply execute a cell with the graph variable name.

```
G
```

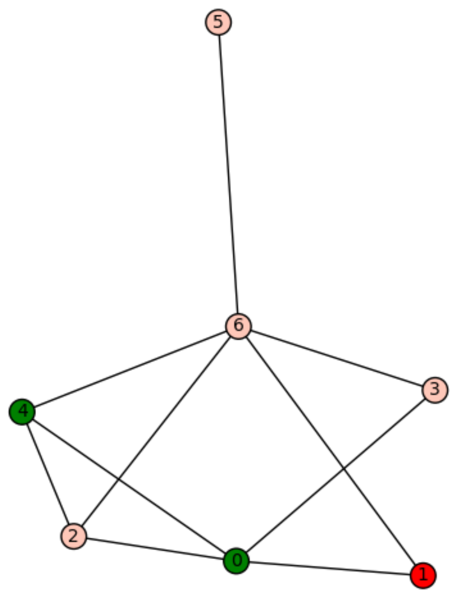
The output is a graphical representation of the graph. If we do not specify vertex coordinates (see below), Sage will use a spring embedder layout algorithm to compute the coordinates.




If a graph is too large, it will not be displayed. In this case, or if you need to specify other display options, you can use the `plot` method. There are many options for the `plot` method, see https://doc.sagemath.org/html/en/reference/plotting/sage/graphs/graph_plot.html for details.

For example. we can specify vertex colors using a dictionary, where keys are colors and values are lists of vertices.

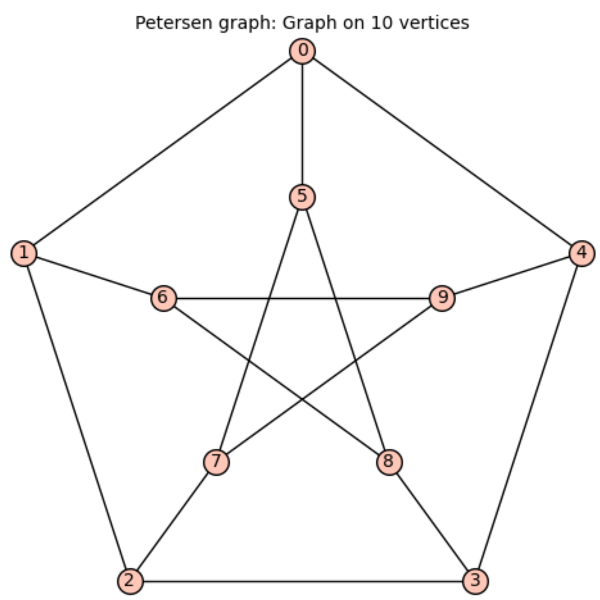
```
G.plot(vertex_colors={'red':[1],'green':[0,4]})
```




#### 1.2.1.1 Some well-known graphs and graph families

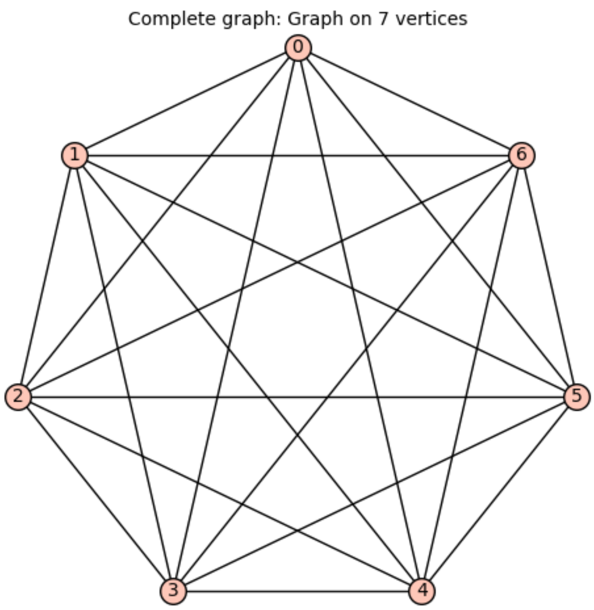
The famous Petersen graph.

```
graphs.PetersenGraph()
```



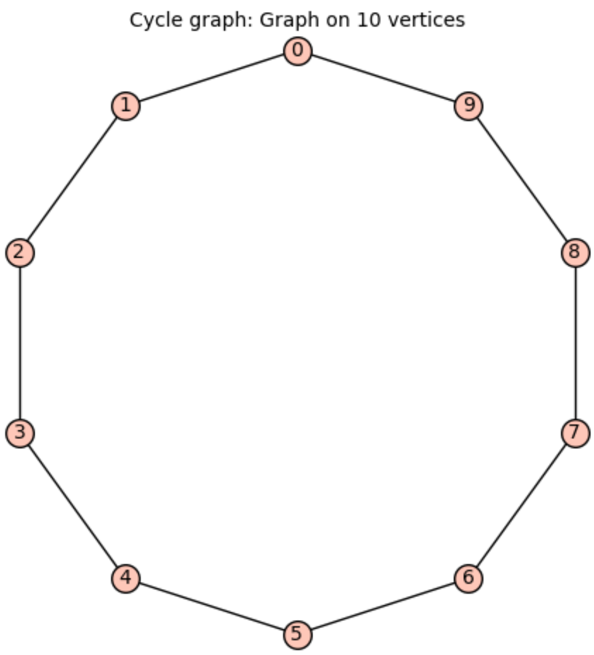

Complete graphs $K_n$.

```
graphs.CompleteGraph(7)
```



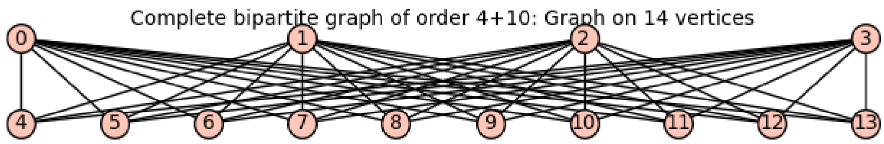

Cycle graphs $C_n$.

```
graphs.CycleGraph(10)
```



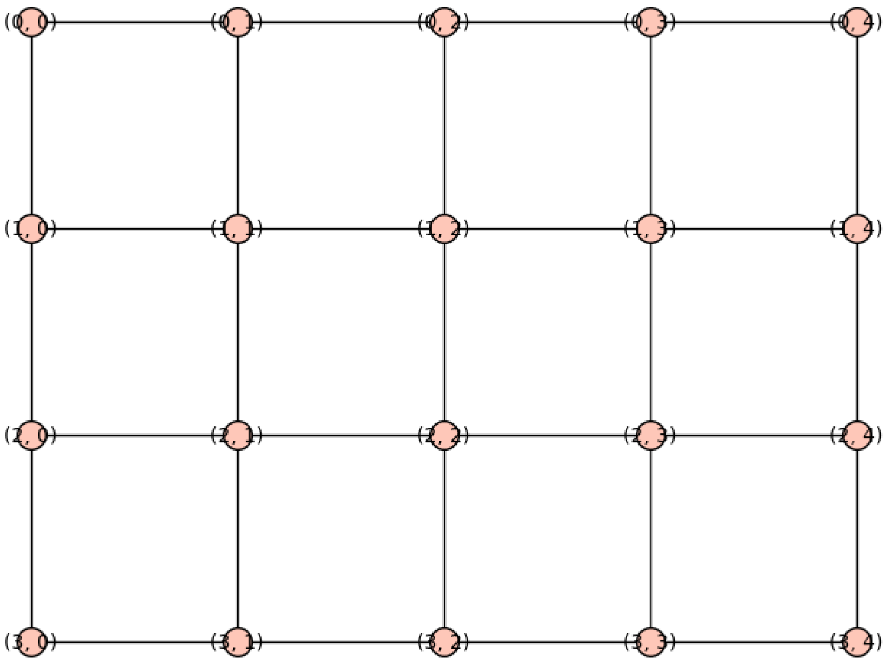

Complete bipartite graphs $K_{n,m}$.

```
graphs.CompleteBipartiteGraph(4, 10)
```




Grid graphs $G_{n,m}$.

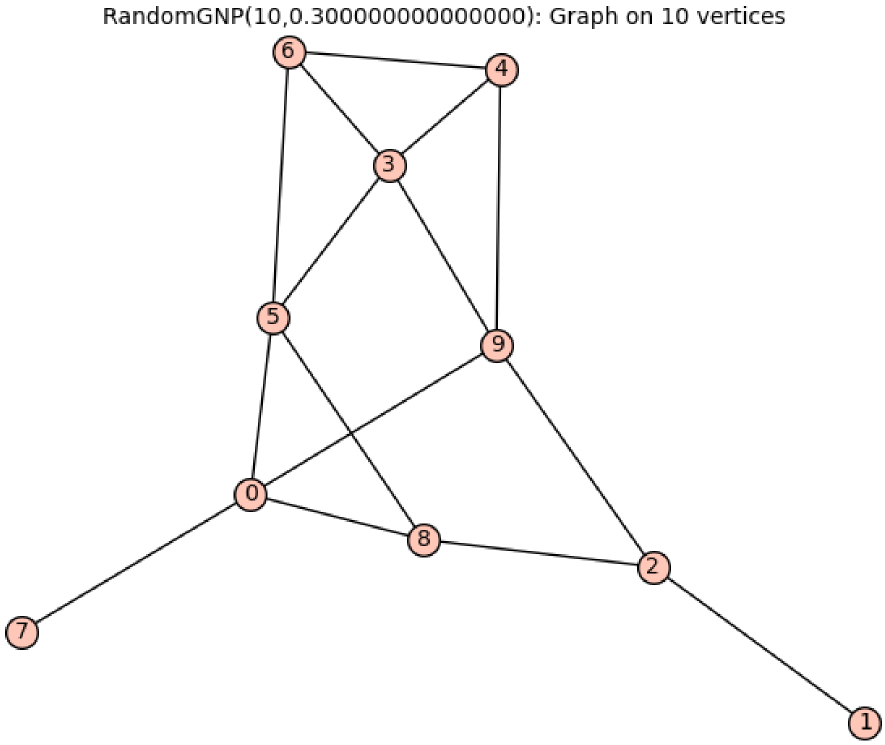
```
GG = graphs.GridGraph([4, 5])
GG.plot()
```




#### 1.2.1.2 Randomly generated graphs

Random graph on 10 nodes. Each edge is inserted independently with probability 0.3.

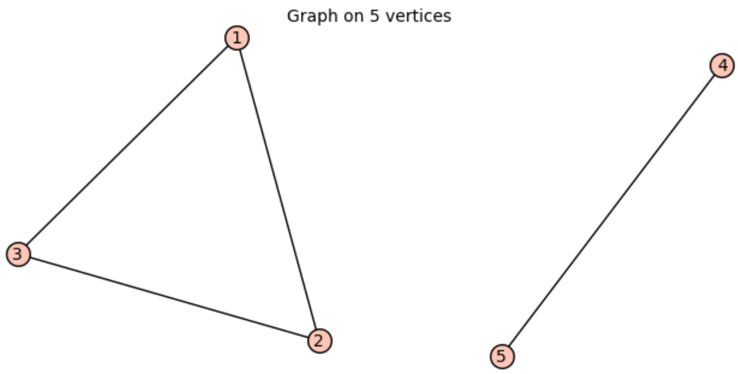
```
graphs.RandomGNP(10, 0.3)
```




### 1.2.1.3 Graph constructors

From a list of edges.

```
Graph([(1,2),(2,3),(3,1),(4,5)])
```

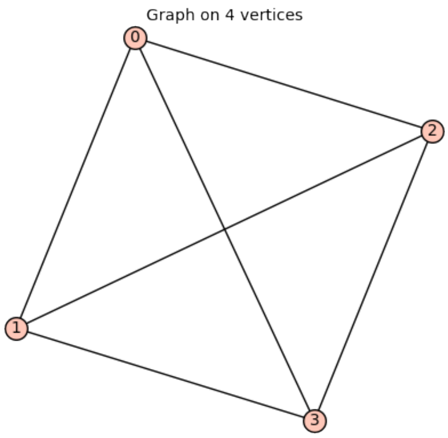



From an adjacency matrix.

```
m = matrix([[int(i != j) for i in range(4)] for j in range(4)])
m
```

```
[0 1 1 1]
[1 0 1 1]
[1 1 0 1]
[1 1 1 0]
```

```
Graph(m)
```



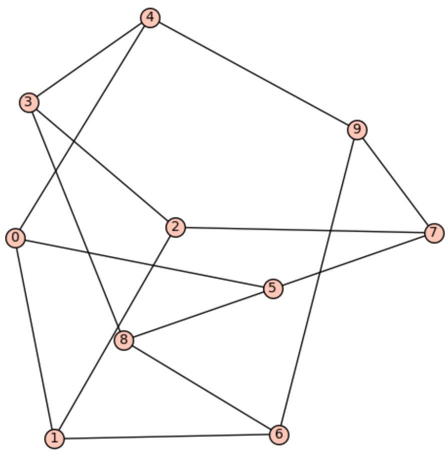

Graph to adjacency matrix.

```
M = G.adjacency_matrix()
m
```

```
[0 1 1 1 1 0 0]
[1 0 0 0 0 0 1]
[1 0 0 0 1 0 1]
[1 0 0 0 0 0 1]
[1 0 1 0 0 0 1]
[0 0 0 0 0 0 1]
[0 1 1 1 1 1 0]
```

From/to graph6 format (compressed string representation of a graph).

```
G = Graph('IheA@GUAo')
G.plot()
```



```
G.graph6_string()
```

```
'IheA@GUAo'
```

Query a graph from local database http://doc.sagemath.org/html/en/reference/graphs/sage/graphs/graph_database.html. For example to get a list of all graphs on 7 vertices with diameter 5.

```
Q = GraphQuery(display_cols=['graph6'], num_vertices=7, diameter=5)
Q.show()
```

```
Graph6
--------------------
F?`po
F?gqg
F@?]O
F@OKg
F@R@o
FA_pW
FEOhW
FGC{o
FIAHo
```

### 1.2.2 Basic graph manipulation

```
G = Graph({0:[1,2,3], 4:[0,2], 6:[1,2,3,4,5]});
```

#### Access edges, verices, neighbors, etc.

Access edges.

```
G.edges(labels=False)
```

```
[(0,1),(0,2),(0,3),(0,4),(1,6),(2,4),(2,6),(3,6),(4,6),(5,6)]
```

Note: Edges can have labels. To get a list of edges without labels, use `labels=False` option. Without this option we get

```
[(0,1,None),(0,2,None),(0,3,None),(0,4,None),(1,6,None),(2,4,None),
(2,6,None),(3,6,None),(4,6,None),(5,6,None)]
```

To check if there is an edge between two vertices use

```
G.has_edge(1,2)
```

```
False
```

Access vertices.

```
G.vertices()
```

```
[0,1,2,3,4,5,6]
```

Access neighbors of a vertex.

```
G.neighbors(0)
```

```
[1,2,3,4]
```

Degree of a vertex is a number of its neighbors

```
G.degree(0)
```

```
4
```

To list degrees of all vertices use

```
G.degree()
```

```
[4,3,3,2,3,2,5]
```

Access number of vertices, edges

```
[G.num_verts(),G.num_edges()]
```

```
[7,10]
```

### Add/remove vertices, edges

Add a vertex. Note that the vertices of a graph can be any *hashable* objects, not just integers.

```
G.add_vertex('a')
```

Method `add_vertex` without arguments adds a single vertex with the smallest available label.

```
newv = G.add_vertex()
newv
```

```
7
```

```
G.vertices(sort=False)
```

```
['a',7,0,1,2,3,4,5,6]
```

Note that in certain versions of Sage sorting of vertices by some methods (e.g. `vertices`) is enabled by default and they may fail if the vertices are not comparable. To disable sorting use `sort=False` option.

To add multiple vertices use `add_vertices` method.

```
H=Graph({0:[1,2,3],4:[0,2],6:[1,2,3,4,5]})
H.add_vertices(range(10,20))
H.vertices()
```

```
[0,1,2,3,4,5,6,10,11,12,13,14,15,16,17,18,19]
```

To add one edge use `add_edge` method, to add multiple edges use `add_edges` method.

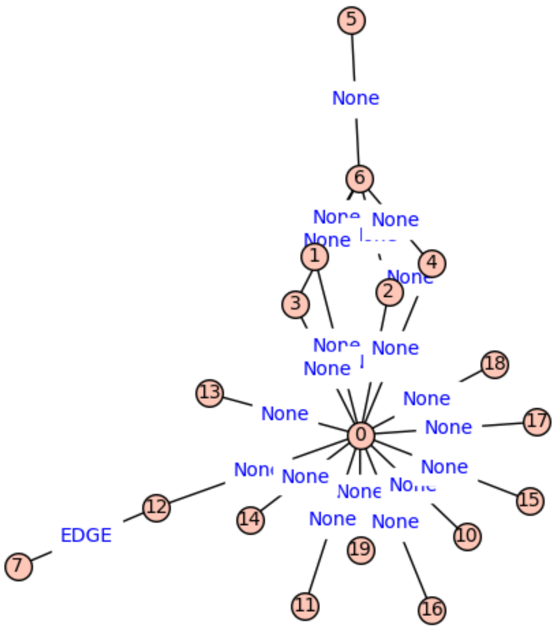
```
H.add_edges([(0, i) for i in range(10, 20)])
```

Note that edges can have labels. To add an edge with a label you need to pass a a triple $u, v, label$ as an argument to `add_edge` method.

```
H.add_edge(7,12,"EDGE")
```

To plot a graph with edge labels use `edge_labels=True` option.

```
H.plot(edge_labels=True)
```




Note that adding an existing vertex (edge) does not result in an error or a warning.

To delete a vertex or and edge use `delete_vertex` and `delete_edge` methods, respectively. For example:

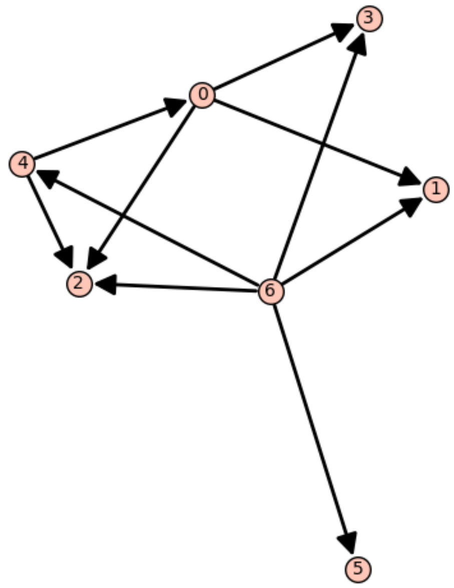
```
H.delete_vertex(7)
H.delete_edge(0,10)
```

Note that deleting a non-existing vertex results in an error while deleting a non-existing edge does not.

### 1.2.3 Directed graphs

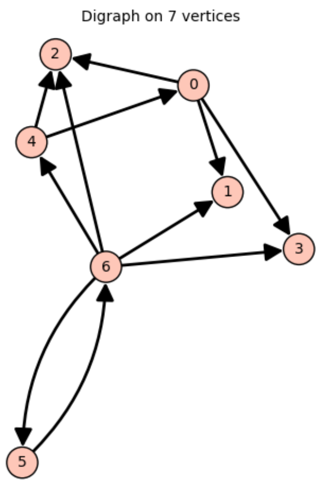
Directed graph is represented using `DiGraph` class.

```
D = DiGraph({0:[1,2,3],4:[0,2],6:[1,2,3,4,5]})
D.plot()
```




Most of the methods for `Graph` class have their counterparts for `DiGraph` class. For example, to add an edge use `add_edge` method.

```
D.add_edge(5,6)
D.plot()
```




Specific methods for `DiGraph` class include `in_degree` and `out_degree` methods to get in-degree and out-degree of a vertex, respectively. Similarly, in addition to `neighbors` there are `in_neighbors` and `out_neighbors` methods.

```
[D.in_degree(0),D.out_degree(0),degree(0)]
```

```
[1,3,4]
```

```
[D.in_neighbors(0),D.out_neighbors(0),D.neighbors(0)]
```

```
[[4],[1,2,3],[1,2,3,4]]
```

To check connectivity of a directed graph use `is_strongly_connected` method.

```
[D.is_connected(),D.is_strongly_connected()]
```

```
[True,False]
```

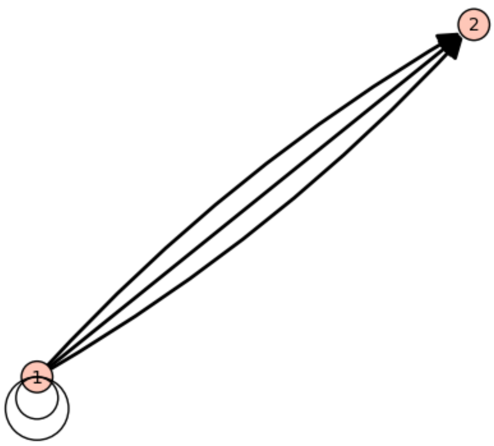
To convert a directed graph to an undirected graph use `to_undirected` (or `to_simple`) method.

### Even more general graphs

To allow multiple edges and/or loops use options `multiedges=True` and `loops=True` to the `DiGraph` constructor. For example, consider the following graph.

```
MG = DiGraph({},multiedges=True,loops=True)
MG.add_vertices([1,2])
MG.add_edges([(1,2),(1,2),(1, 2),(1,1),(1,1)])
MG
```

Looped multi-digraph on 2 vertices



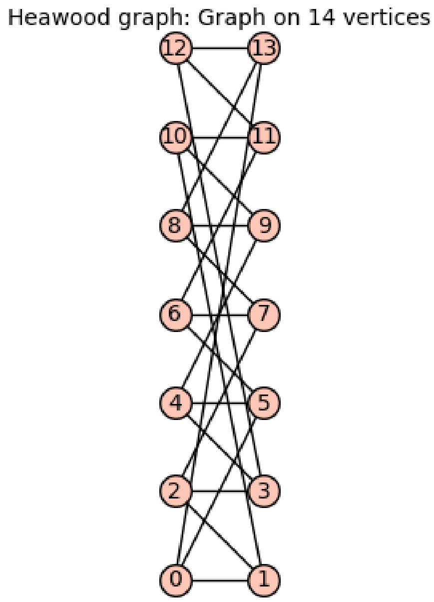

## 1.2.4 Exercises

**Exercise 1.1** Write a function `remove_max_vertex(G)` which removes a vertex with the largest degree from undirected graph $G$ (any of them, if there are more than one with the largest degree). ■

**Exercise 1.2** Write a function `plot_bipartite` which plots a bipartite graph in a way that vertices of each bipartition are arranged on two parallel lines. ■

For example:

```
HGR=graphs.HeawoodGraph()
plot_bipartite(HGR)
```

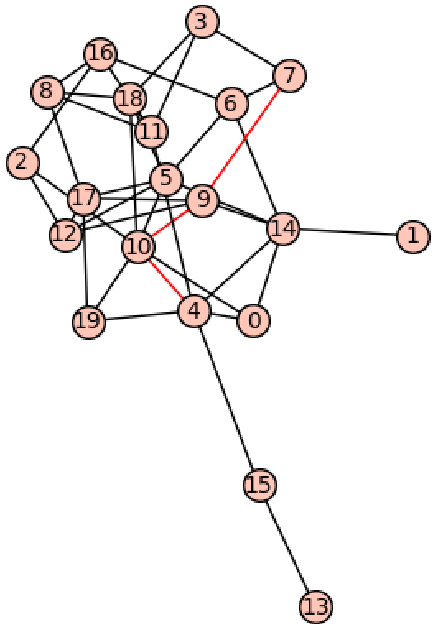



**Exercise 1.3** Write a function `set_random_edge_labels(G,a,b)` which sets edge labels of $G$ to random integers from interval $[a,b]$.

Write a function `mark_shortest_path(G,a,b)` which calculates a shortest path between the vertices $a$ and $b$ in the weighted graph $G$ and colors it with red color. (For calculating shortest paths use built-in function `shortest_path`. ■

Example:

```
X=graphs.RandomGNP(20,0.2)
set_random_edge_labels(X,1,10)
mark_shortest_path(X,4,7)
```


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

# 2. Depth-first search and Breadth-first search

## 2.1 Depth-first search (DFS)

Write a Depth-first search (DFS) implementation using Sage Graph representation

- Write a recursive implementation of the depth-first search.
- Add computation of discovery and finishing times to the implementation.

(See Handouts on Course Homepage for pseudocode)

```
def DFS_recursive(G, r):
    """
    Perform DFS from root r. Result is a dictionary mapping a vertex v to
    its predecessor in DFS tree (root is mapped to None).
    """
    prev = {}
    prev[r] = None
    DFS_recursive_call(G, r, prev)
    return prev

def DFS_recursive_call(G, v, prev):
    for u in G.neighbors(v):
        if u not in prev:
            prev[u] = v
            DFS_recursive_call(G, u, prev)
```

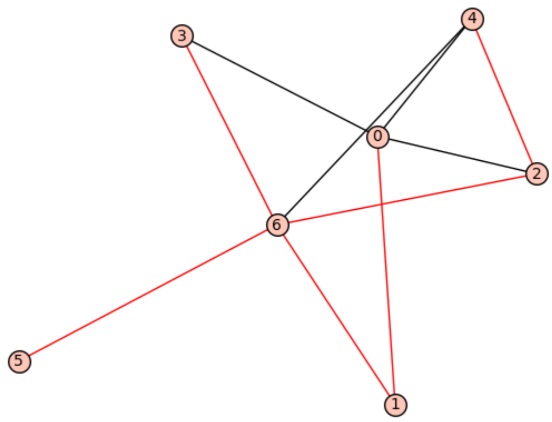
### Examples

```
G = Graph({0:[1,2,3], 4:[0,2], 6:[1,2,3,4,5]})

dfs_dict = DFS_recursive(G, 0)
dfs_dict
```

```
{0: None, 1: 0, 6: 1, 2: 6, 4: 2, 3: 6, 5: 6}
```

```
G.plot(edge_colors={'red': [(u, v) for (u, v) in dfs_dict.items() if v
!= None]})
```




```
H = graphs.Grid2dGraph(3, 3)
DFS_recursive(H, (0, 0))
```

```
{(0, 0): None,
 (0, 1): (0, 0),
 (0, 2): (0, 1),
 (1, 2): (0, 2),
 (1, 1): (1, 2),
 (1, 0): (1, 1),
 (2, 0): (1, 0),
 (2, 1): (2, 0),
 (2, 2): (2, 1)}
```

## 2.2 DFS with start (discovery) time and end (finishing) time

```
def DFS_with_times(G, r):
    """
    Perform DFS from root r. Result is a triple of three dictionaries:
    - dictionary mapping a vertex v to its predecessor in DFS tree
      (root is mapped to None).
    - dictionary mapping a vertex to its start time
    - dictionary mapping a vertex to its end time
    """
    global time
    time = 0
    prev = {}
    start = {}
    end = {}
    prev[r] = None
    DFS_with_times_call(G, r, prev, start, end)
    return (prev, start, end)

def DFS_with_times_call(G, v, prev, start, end):
    global time
    time += 1;
    start[v] = time;
    for u in G.neighbors(v):
        if u not in prev:
            prev[u] = v
```

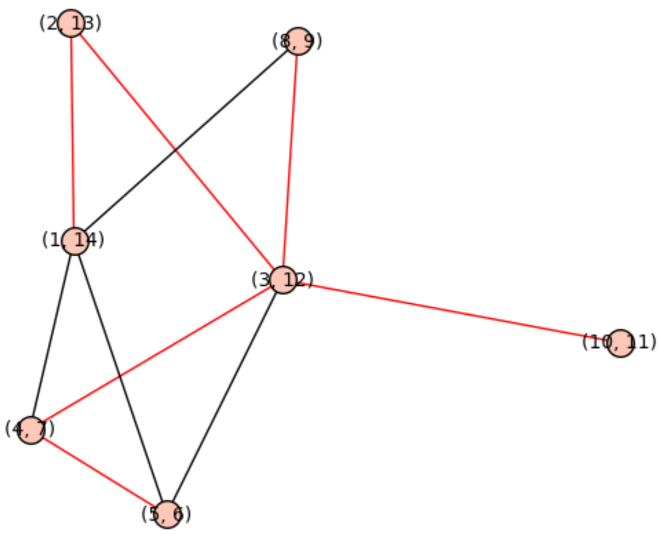
```
            DFS_with_times_call(G, u, prev, start, end)
    time += 1;
    end[v] = time;
```

### Examples

```
G = Graph({0:[1,2,3], 4:[0,2], 6:[1,2,3,4,5]})
(prev, disc, finish) = DFS_with_times(G, 0)
(prev, disc, finish)
```

```
({0: None, 1: 0, 2: 6, 3: 6, 4: 2, 5: 6, 6: 1},
 {0: 1, 1: 2, 2: 4, 3: 8, 4: 5, 5: 10, 6: 3},
 {0: 14, 1: 13, 2: 7, 3: 9, 4: 6, 5: 11, 6: 12})
```

```
G.relabel(dict([(v, (disc[v], finish[v])) for v in G.vertices()]))
G.plot(edge_colors={'red': [((disc[u],finish[u]), (disc[v],finish[v]))
  for (u, v) in prev.items() if v != None]})
```




## 2.3 Breadth-first search (BFS)

Write a Breadth-first search (BFS) implementation using Sage Graph representation.

```
import queue
def BFS(G, r):
    """
    Perform BFS from root r. Result is a dictionary mapping a vertex v
    to its predecessor in BFS tree (root is mapped to None).
    """
    prev = {}
    prev[r] = None
    q = queue.Queue()
    q.put(r)
    while not q.empty():
        v = q.get()
```

```
        for u in G.neighbors(v):
            if u not in prev:
                prev[u] = v
                q.put(u)
    return prev
```

### Example

```
    BFS(H, (0, 0))
```

```
{(0, 0): None,
 (0, 1): (0, 0),
 (1, 0): (0, 0),
 (0, 2): (0, 1),
 (1, 1): (0, 1),
 (2, 0): (1, 0),
 (1, 2): (0, 2),
 (2, 1): (1, 1),
 (2, 2): (1, 2)}
```

## 2.4 Topological sorting

- Use DFS with discovery and finishing times to implement topological sorting of a DAG (directed acyclic) graph
- Help professor Bumstead to dress himself in the correct order. Order of putting his garments is given by the digraph below

```
    T = DiGraph({'undershorts': ['shoes','pants'],'pants':['shoes','belt'],
'belt':['jacket'],'shirt':['belt','tie'],'tie':['jacket'],'socks':['
shoes'],'watch':[]})
```

```
def DFS_DiGraph(G):
    """
    Implement (recursive) DFS on a digraph to create a
    "forest of DFS trees"

    Use G.neighbors_out(v) to get "out" neighbors of vertex v
    """
    global time
    time = 0
    prev = {}
    start = {}
    end = {}
    for v in G.vertices(sort=False):
        if v not in prev:
            prev[v] = None
            DFS_DiGraph_call(G, v, prev, start, end)
    return (prev, start, end)

def DFS_DiGraph_call(G, v, prev, start, end):
    global time
    time += 1;
    start[v] = time;
    for u in G.neighbor_out_iterator(v):
        if u not in prev:
            prev[u] = v
            DFS_DiGraph_call(G, u, prev, start, end)
```

```
    time += 1
    end[v] = time
```

```
DFS_DiGraph(T)
```

```
({'belt': None,
  'jacket': 'belt',
  'tie': None,
  'watch': None,
  'shoes': None,
  'socks': None,
  'pants': None,
  'undershorts': None,
  'shirt': None},
 {'belt': 1,
  'jacket': 2,
  'tie': 5,
  'watch': 7,
  'shoes': 9,
  'socks': 11,
  'pants': 13,
  'undershorts': 15,
  'shirt': 17},
 {'jacket': 3,
  'belt': 4,
  'tie': 6,
  'watch': 8,
  'shoes': 10,
  'socks': 12,
  'pants': 14,
  'undershorts': 16,
  'shirt': 18})
```

```
def topological_sort(G):
    """
    Performs topological sort on a DAG (directed acyclic graph) G
    (calculate finishing times and sort vertices by them in
    descending order)
    """
    (_, _, finish) = DFS_DiGraph(T)
    return sorted(finish.items(), key=lambda x: -x[1])
```

```
topological_sort(T)
```

```
[('shirt', 18),
 ('undershorts', 16),
 ('pants', 14),
 ('socks', 12),
 ('shoes', 10),
 ('watch', 8),
 ('tie', 6),
 ('belt', 4),
 ('jacket', 3)]
```

## Unnumbered Section

### Unnumbered Subsection

#### Unnumbered Subsubsection

# 3. In-text Element Examples

## 3.1 Referencing Publications

This statement requires citation [1]; this one is more specific [2, page 162].

## 3.2 Link Examples

This is a URL link: LaTeX Templates. This is an email link: example@example.com. This is a monospaced URL link: `https://www.LaTeXTemplates.com`.

## 3.3 Lists

Lists are useful to present information in a concise and/or ordered way.

### 3.3.1 Numbered List

1. First numbered item
    a. First indented numbered item
    b. Second indented numbered item
        i. First second-level indented numbered item
2. Second numbered item
3. Third numbered item

### 3.3.2 Bullet Point List

- First bullet point item
    - First indented bullet point item
    - Second indented bullet point item
        - First second-level indented bullet point item
- Second bullet point item
- Third bullet point item

### 3.3.3 Descriptions and Definitions

**Name** Description
**Word** Definition
**Comment** Elaboration

## 3.4 International Support

àáâäãåèéêëìíîïòóôöõøùúûüÿýñçčšž
ÀÁÂÄÃÅÈÉÊËÌÍÎÏÒÓÔÖÕØÙÚÛÜŸÝÑ
ßÇŒÆČŠŽ

## 3.5 Ligatures

fi fj fl ffl ffi Ty Ty

# Part Two Title

# 4. Mathematics

## 4.1 Theorems

### 4.1.1 Several equations

This is a theorem consisting of several equations.

**Theorem 4.1 — Name of the theorem.** In $E = \mathbb{R}^n$ all norms are equivalent. It has the properties:

$$\big| ||\mathbf{x}|| - ||\mathbf{y}|| \big| \leq ||\mathbf{x} - \mathbf{y}|| \tag{4.1}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{4.2}$$

### 4.1.2 Single Line

This is a theorem consisting of just one line.

**Theorem 4.2** A set $\mathscr{D}(G)$ in dense in $L^2(G)$, $|\cdot|_0$.

## 4.2 Definitions

A definition can be mathematical or it could define a concept.

**Definition 4.1 — Definition name.** Given a vector space $E$, a norm on $E$ is an application, denoted $||\cdot||$, $E$ in $\mathbb{R}^+ = [0, +\infty[$ such that:

$$||\mathbf{x}|| = 0 \Rightarrow \mathbf{x} = \mathbf{0} \tag{4.3}$$

$$||\lambda \mathbf{x}|| = |\lambda| \cdot ||\mathbf{x}|| \tag{4.4}$$

$$||\mathbf{x} + \mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}|| \tag{4.5}$$

## 4.3 Notations

■ **Notation 4.1** Given an open subset $G$ of $\mathbb{R}^n$, the set of functions $\varphi$ are:

1. Bounded support $G$;
2. Infinitely differentiable;

a vector space is denoted by $\mathscr{D}(G)$.

## 4.4 Remarks

This is an example of a remark.

R The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K}=\mathbb{R}$, however, established properties are easily extended to $\mathbb{K}=\mathbb{C}$.

## 4.5 Corollaries

**Corollary 4.1 — Corollary name.** The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K}=\mathbb{R}$, however, established properties are easily extended to $\mathbb{K}=\mathbb{C}$.

## 4.6 Propositions

### 4.6.1 Several equations

**Proposition 4.1 — Proposition name.** It has the properties:

$$\left| ||\mathbf{x}|| - ||\mathbf{y}|| \right| \leq ||\mathbf{x}-\mathbf{y}|| \tag{4.6}$$

$$||\sum_{i=1}^{n} \mathbf{x}_i|| \leq \sum_{i=1}^{n} ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \tag{4.7}$$

### 4.6.2 Single Line

**Proposition 4.2** Let $f,g \in L^2(G)$; if $\forall \varphi \in \mathscr{D}(G)$, $(f,\varphi)_0 = (g,\varphi)_0$ then $f = g$.

## 4.7 Examples

### 4.7.1 Equation Example

■ **Example 4.1** Let $G=\{x\in\mathbb{R}^2 : |x|<3\}$ and denoted by: $x^0=(1,1)$; consider the function:

$$f(x)=\begin{cases} \mathrm{e}^{|x|} & \text{si } |x-x^0| \leq 1/2 \\ 0 & \text{si } |x-x^0| > 1/2 \end{cases} \tag{4.8}$$

The function $f$ has bounded support, we can take $A=\{x\in\mathbb{R}^2 : |x-x^0| \leq 1/2+\varepsilon\}$ for all $\varepsilon \in ]0;5/2-\sqrt{2}[$. ■

### 4.7.2 Text Example

■ **Example 4.2 — Example name.** Aliquam arcu turpis, ultrices sed luctus ac, vehicula id metus. Morbi eu feugiat velit, et tempus augue. Proin ac mattis tortor. Donec tincidunt, ante rhoncus luctus semper, arcu lorem lobortis justo, nec convallis ante quam quis lectus. Aenean tincidunt sodales massa, et hendrerit tellus mattis ac. Sed non pretium nibh. Donec cursus maximus luctus. Vivamus lobortis eros et massa porta porttitor. ■

## 4.8 Exercises

**Exercise 4.1** This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ■

## 4.9 Problems

**Problem 4.1** What is the average airspeed velocity of an unladen swallow?

## 4.10 Vocabulary

Define a word to improve a students' vocabulary.

- **Vocabulary 4.1 — Word.** Definition of word.

# 5. Presenting Information and Results with a Long Chapter Title

## 5.1 Table

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor. Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in. Nunc fermentum nisi imperdiet lectus tincidunt vestibulum at ac elit. Nulla mattis nisl eu malesuada suscipit.

| **Treatments** | **Response 1** | **Response 2** |
|---|---:|---:|
| Treatment 1 | 0.0003262 | 0.562 |
| Treatment 2 | 0.0015681 | 0.910 |
| Treatment 3 | 0.0009271 | 0.296 |

Table 5.1: Table caption.

Referencing Table 5.1 in-text using its label.

## 5.2 Figure

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor. Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in. Nunc fermentum nisi imperdiet lectus tincidunt vestibulum at ac elit. Nulla mattis nisl eu malesuada suscipit.

Figure 5.1: Figure caption.

Referencing Figure 5.1 in-text using its label.

| Treatments | Response 1 | Response 2 |
|---|---|---|
| Treatment 1 | 0.0003262 | 0.562 |
| Treatment 2 | 0.0015681 | 0.910 |
| Treatment 3 | 0.0009271 | 0.296 |

Table 5.2: Floating table.

creodocs

Figure 5.2: Floating figure.

# Bibliography

## Articles

[1] A. B. Jones and J. M. Smith. “Article Title”. In: *Journal title* 13.52 (Mar. 2022), pages 123–456. DOI: 10.1038/s41586-021-03616-x (cited on page 29).

## Books

[2] J. M. Smith and A. B. Jones. *Book Title*. 7th. Publisher, 2021 (cited on page 29).

# Index

# A. Appendix Chapter Title

## A.1 Appendix Section Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex. Fusce quis mauris egestas lacus commodo venenatis. Ut at arcu lectus. Donec et urna nunc. Morbi eu nisl cursus sapien eleifend tincidunt quis quis est. Donec ut orci ex. Praesent ligula enim, ullamcorper non lorem a, ultrices volutpat dolor. Nullam at imperdiet urna. Pellentesque nec velit eget est euismod pretium.

# B. Appendix Chapter Title

## B.1 Appendix Section Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex. Fusce quis mauris egestas lacus commodo venenatis. Ut at arcu lectus. Donec et urna nunc. Morbi eu nisl cursus sapien eleifend tincidunt quis quis est. Donec ut orci ex. Praesent ligula enim, ullamcorper non lorem a, ultrices volutpat dolor. Nullam at imperdiet urna. Pellentesque nec velit eget est euismod pretium.