Семинар 12

Введение в программирование на Python

Папулин С.Ю.
papulin_hse@mail.ru

2015

Семинар 11

1. Структуры данных
   - Стек
   - Очередь
   - Дек
2. Принципы объектно-ориентированного программирования (ООП)
3. Класса в Python

План семинара 12

Структуры данных

- Дерево
- Граф

Структуры данных

Граф

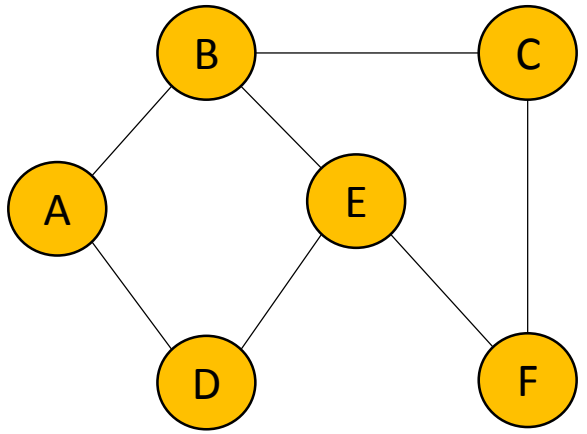
Граф – структура данных, состоящая из множеств **вершин** (vertex) и **ребер** (edge). Ребро – связь между двумя вершинами.

$$G = (V, E)$$

V – множество вершин, E – множество ребер

$$V = \{A, B, C, D, E, F\}$$

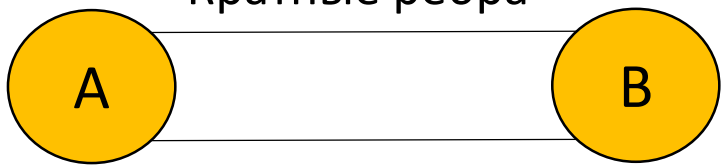
$$E = \{(A, B), (A, D), (B, C), (B, E), (D, E), (E, F), (C, F)\}$$



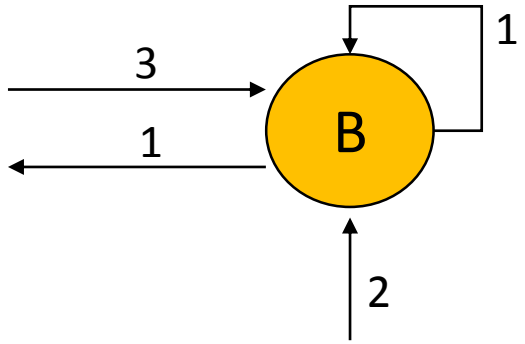
Виды графов

Простой граф – граф без петель и кратных ребер

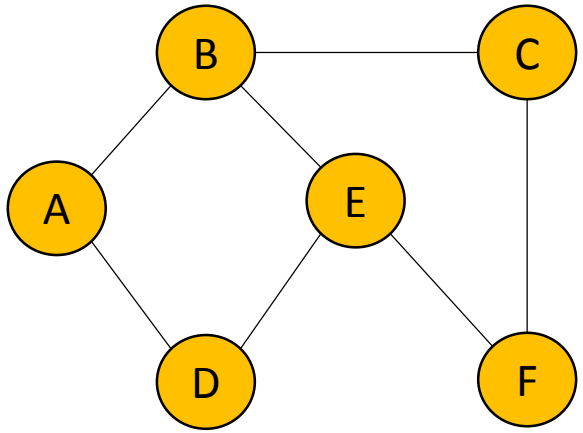
Кратные ребра



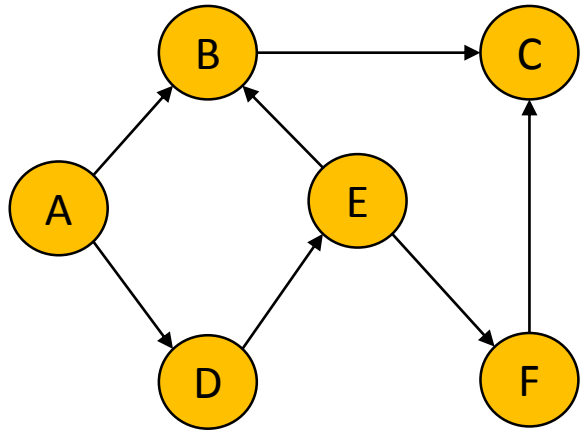
Петля



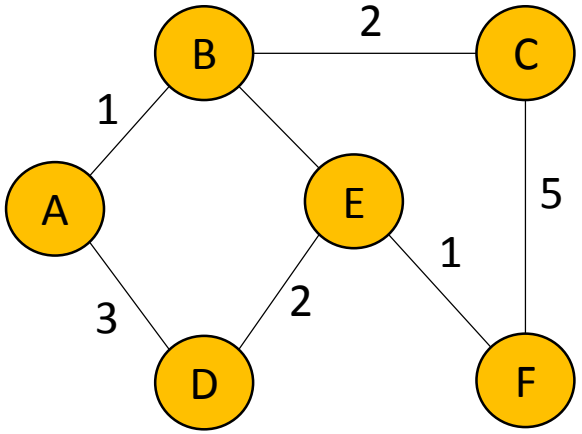
Неориентированный граф



Ориентированный граф

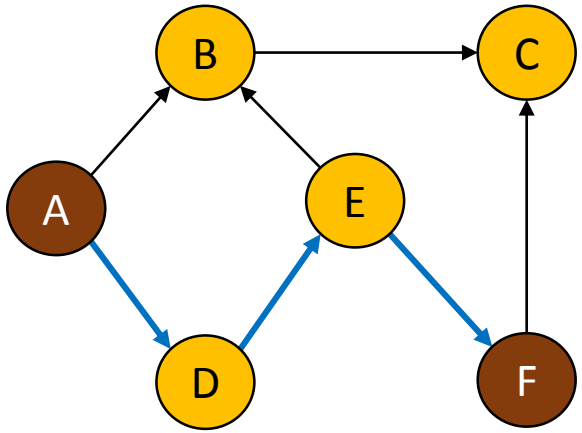


Взвешенный граф

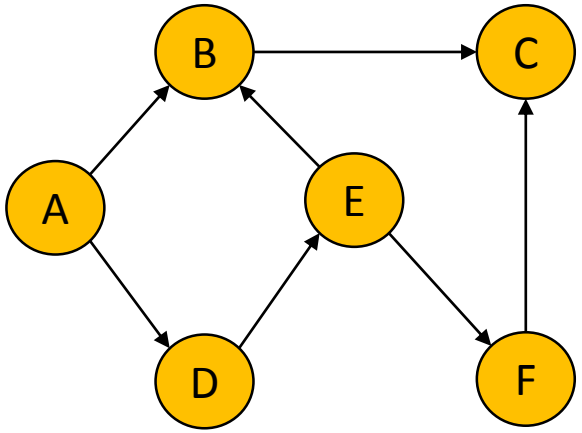


Граф

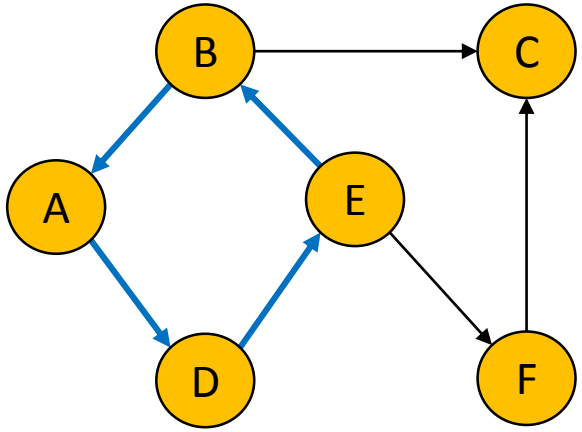
Путь



Без цикла (DAG)

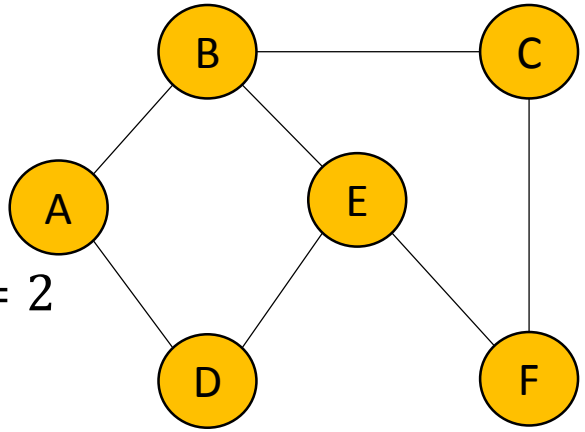


С циклом



Степень вершин

$deg(B) = 3$

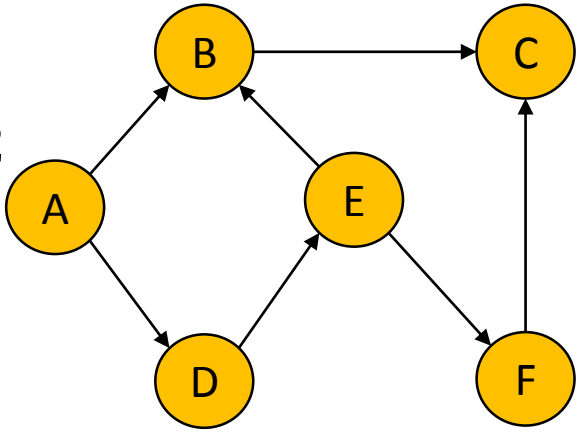


$deg(A) = 2$

$deg(F) = 2$

$outdeg(C) = 1$
$indeg(C) = 2$

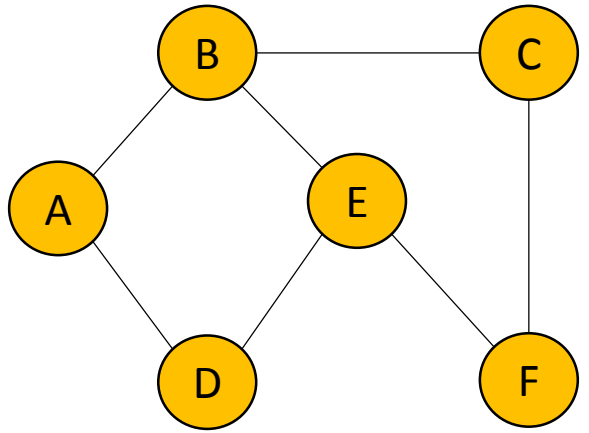
$outdeg(C) = 2$
$indeg(C) = 0$



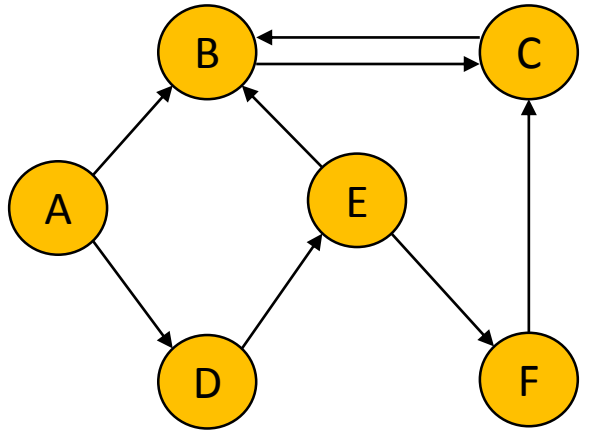
$outdeg(C) = 1$
$indeg(C) = 1$

Представление графа. Матрица смежности

	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	1	0
C	0	1	0	0	0	1
D	1	0	0	0	1	0
E	0	1	0	1	0	1
F	0	0	1	0	1	0



	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	1	0	0	0	0
D	0	0	0	0	1	0
E	0	1	0	0	0	1
F	0	0	1	0	0	0



Представление графа. Матрица смежности. Python

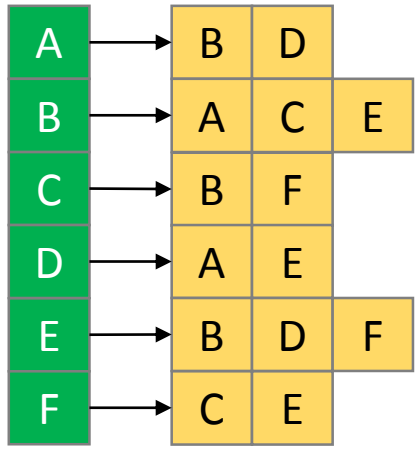
	A	B	C	D	E	F
A	0	1	0	1	0	0
B	1	0	1	0	1	0
C	0	1	0	0	0	1
D	1	0	0	0	1	0
E	0	1	0	1	0	1
F	0	0	1	0	1	0

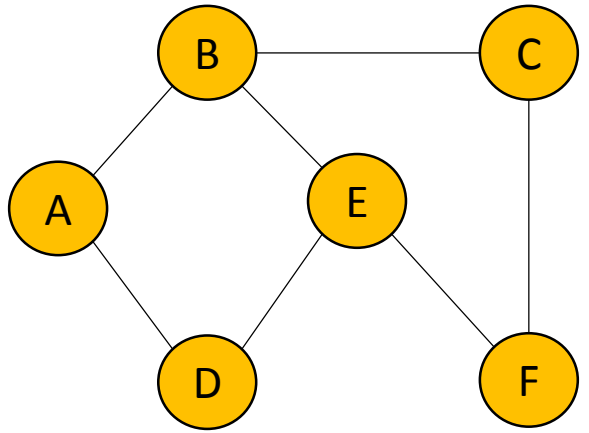
```
mUDAM = [[0,1,0,1,0,0],
         [1,0,1,0,1,0],
         [0,1,0,0,0,1],
         [1,0,0,0,1,0],
         [0,1,0,1,0,1],
         [0,0,1,0,1,0]]
```

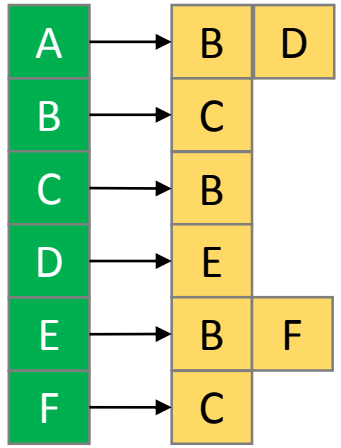
	A	B	C	D	E	F
A	0	1	0	1	0	0
B	0	0	1	0	0	0
C	0	1	0	0	0	0
D	0	0	0	0	1	0
E	0	1	0	0	0	1
F	0	0	1	0	0	0

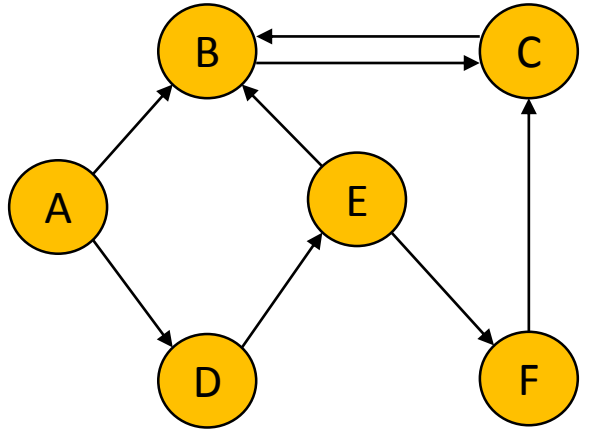
```
mDAM = [[0,1,0,1,0,0],
        [0,0,1,0,0,0],
        [0,1,0,0,0,0],
        [0,0,0,0,1,0],
        [0,1,0,0,0,1],
        [0,0,1,0,0,0]]
```

Представление графа. Список смежности

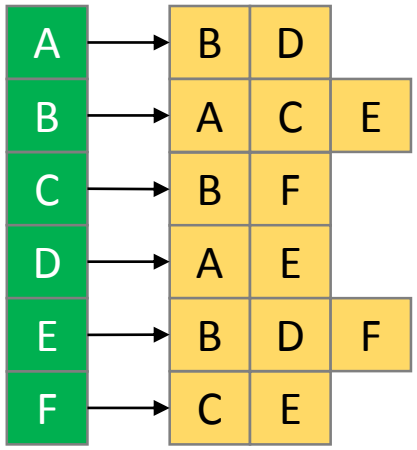




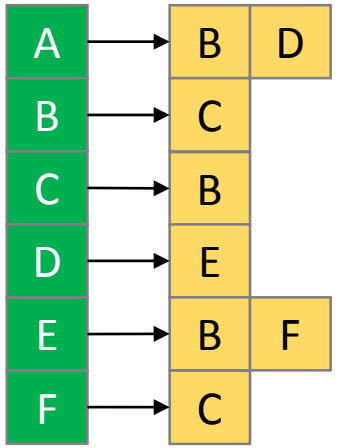




Представление графа. Список смежности. Python




```
lUDAL = {"A": set(["B", "D"]),
         "B": set(["A", "C", "E"]),
         "C": set(["B", "F"]),
         "D": set(["A", "E"]),
         "E": set(["B", "D", "F"]),
         "F": set(["C", "E"])}
```




```
lDAL = {"A": set(["B", "D"]),
        "B": set(["C"]),
        "C": set(["B"]),
        "D": set(["E"]),
        "E": set(["B", "F"]),
        "F": set(["C"])}
```

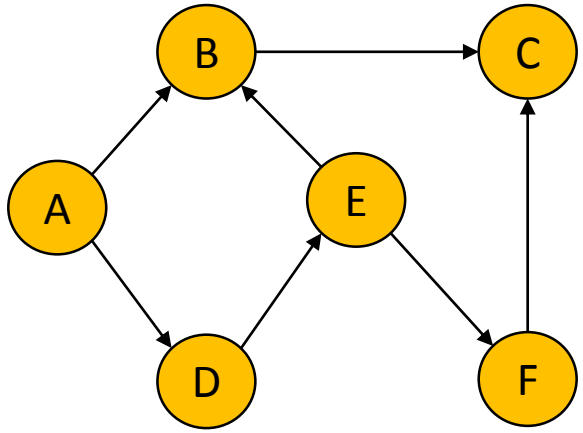
Представление графа

Матрица или список смежности?

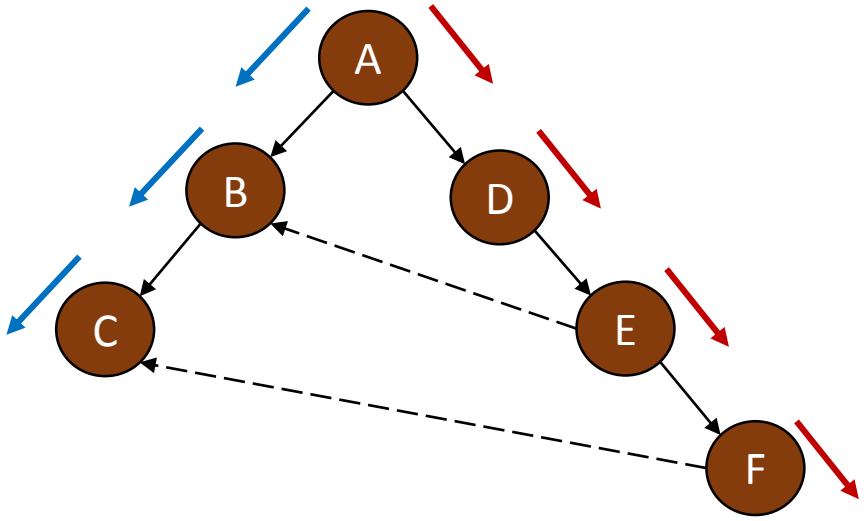
Поиск в графе

- Поиск в глубину
- Поиск в ширину

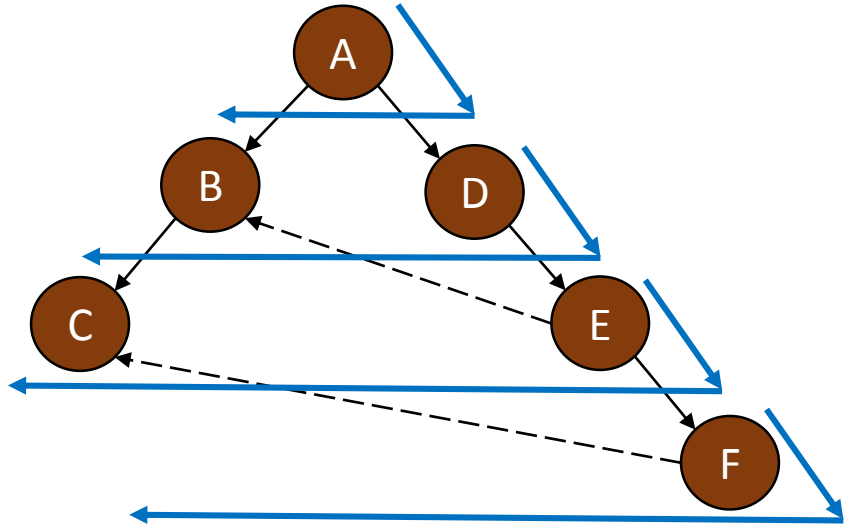
Глубина vs ширина



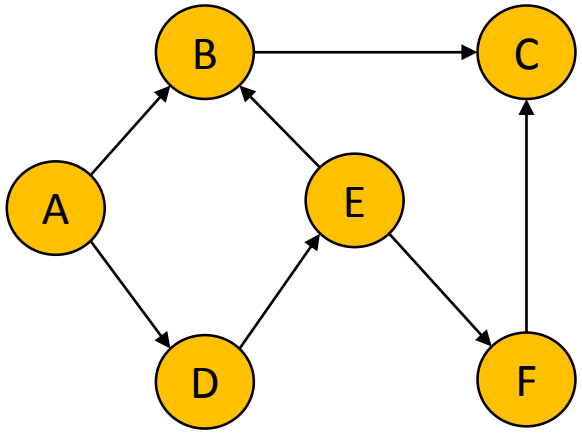
Поиск в глубину

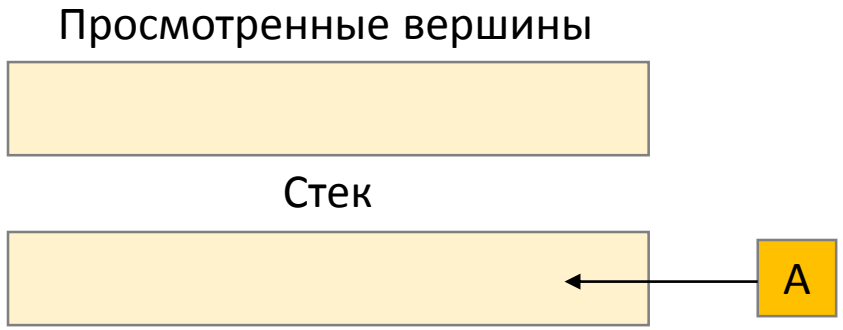


Поиск в ширину

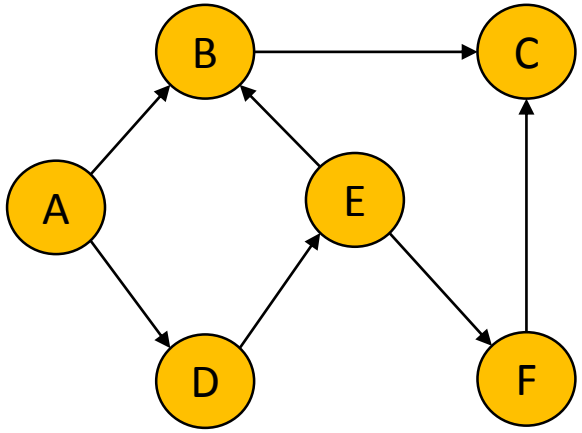


Обход в глубину



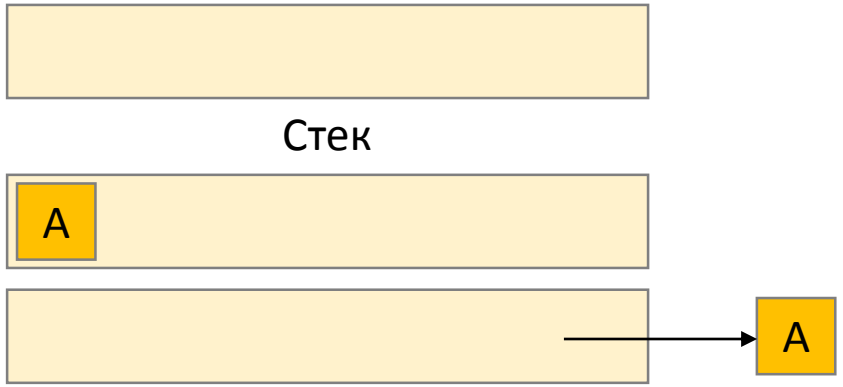


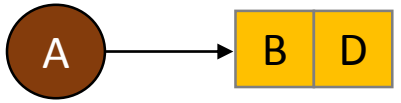
Обход в глубину

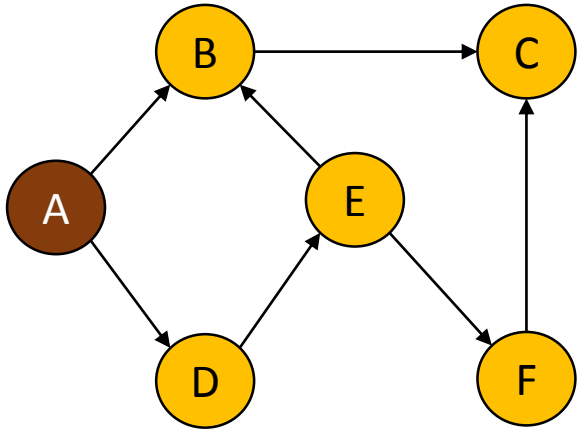


Просмотренные вершины

Стек



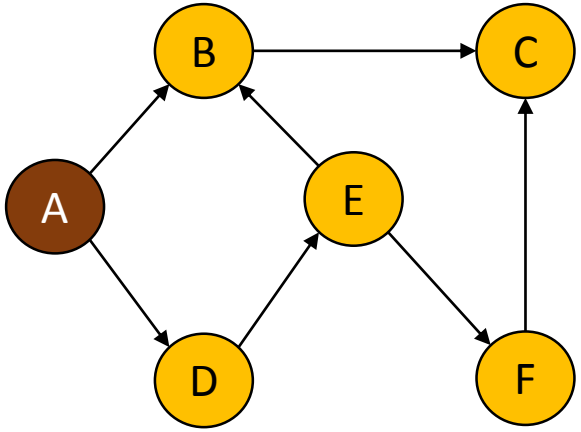




Просмотренные вершины

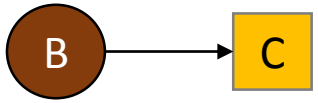
Стек

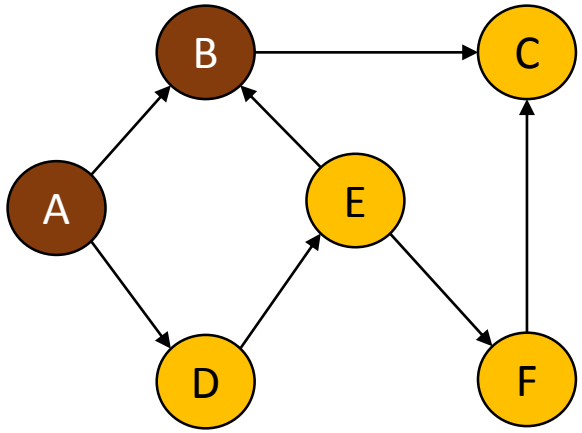
Обход в глубину

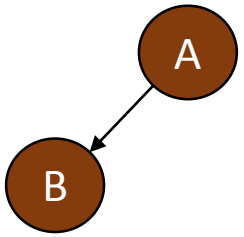


Просмотренные вершины

Стек



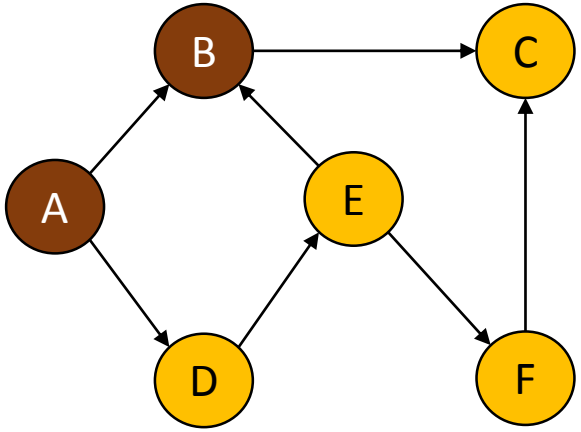




Просмотренные вершины

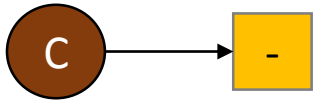
Стек

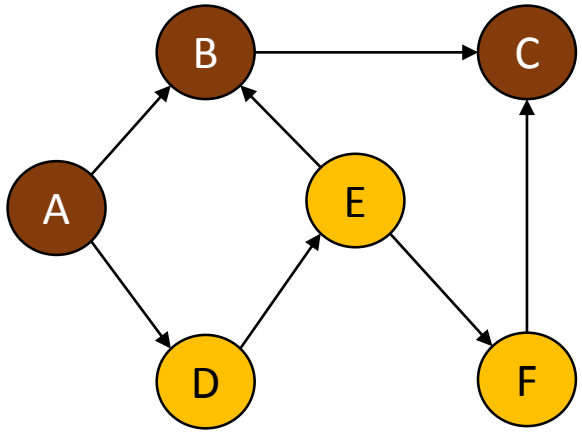
Обход в глубину

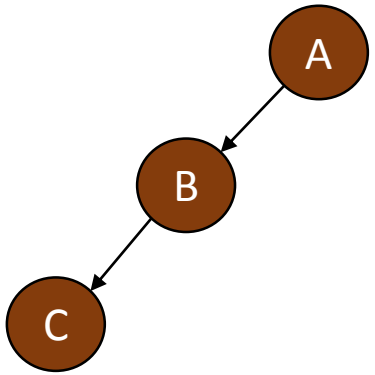


Просмотренные вершины

Стек



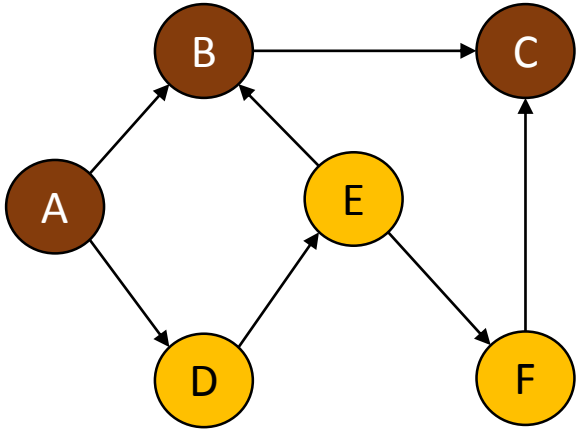




Просмотренные вершины

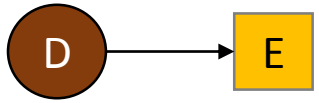
Стек

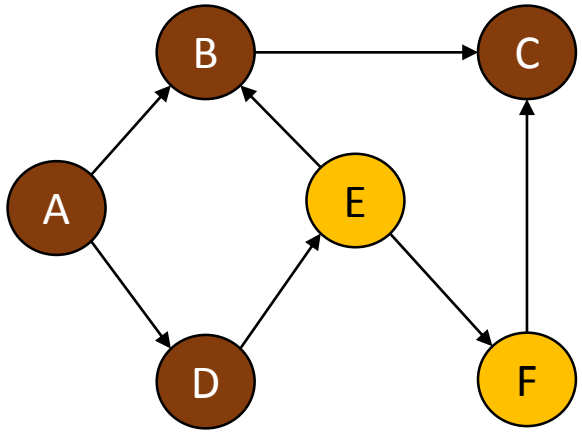
Обход в глубину

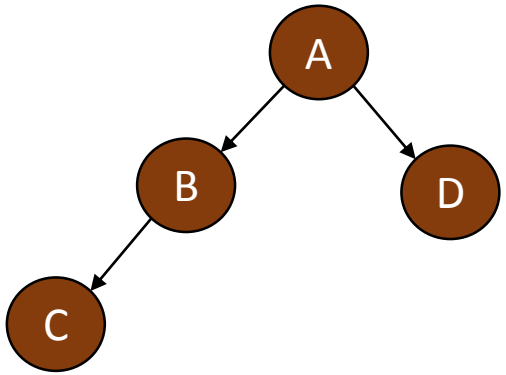


Просмотренные вершины

Стек



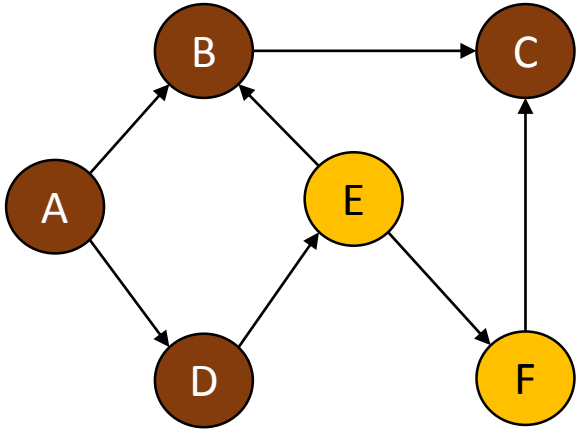




Просмотренные вершины

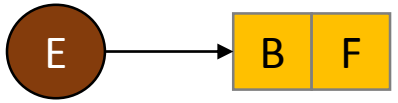
Стек

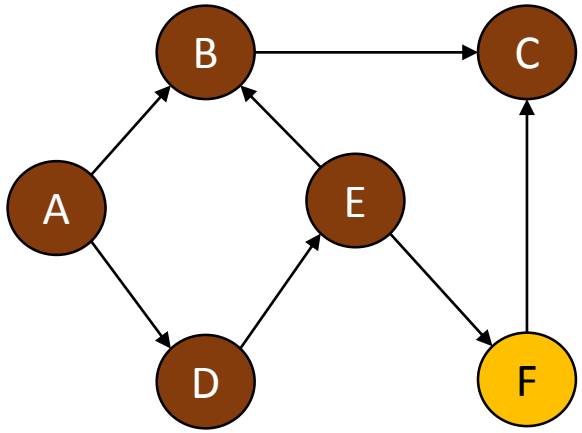
Обход в глубину

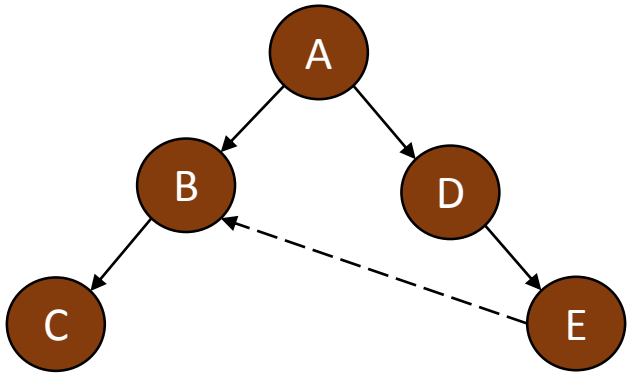


Просмотренные вершины

Стек



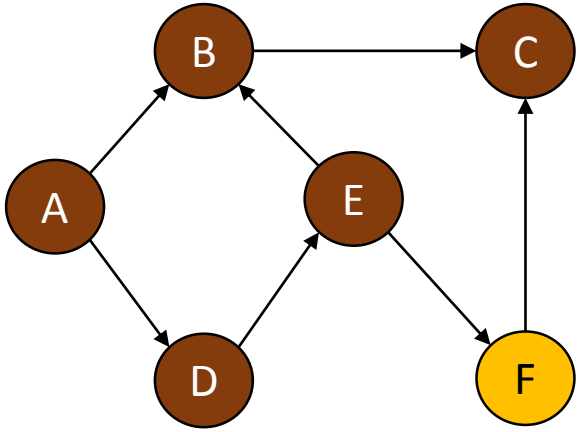




Просмотренные вершины

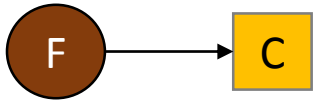
Стек

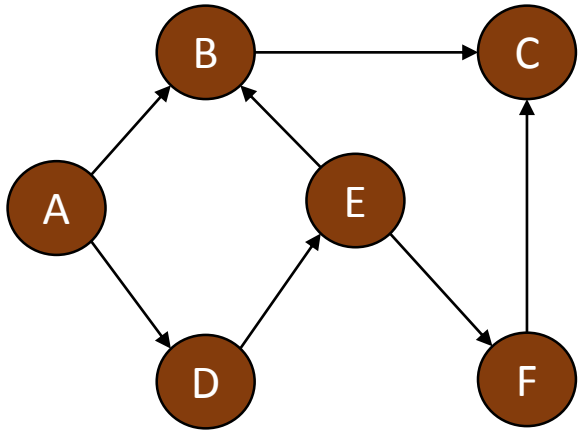
Обход в глубину

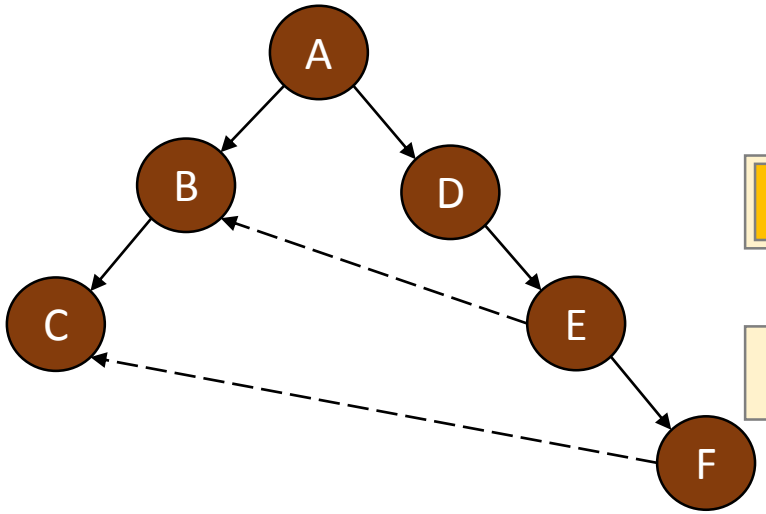


Просмотренные вершины

Стек







Просмотренные вершины

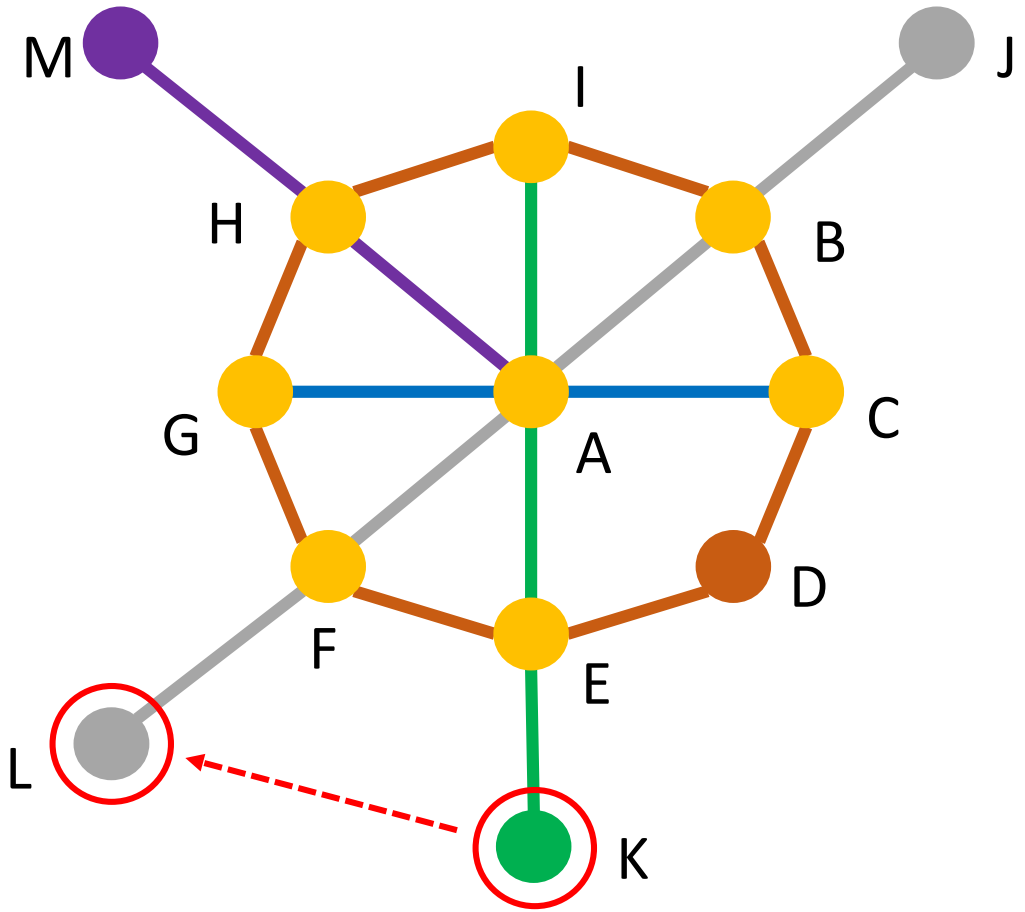
Стек

Вариант реализации обхода в глубину

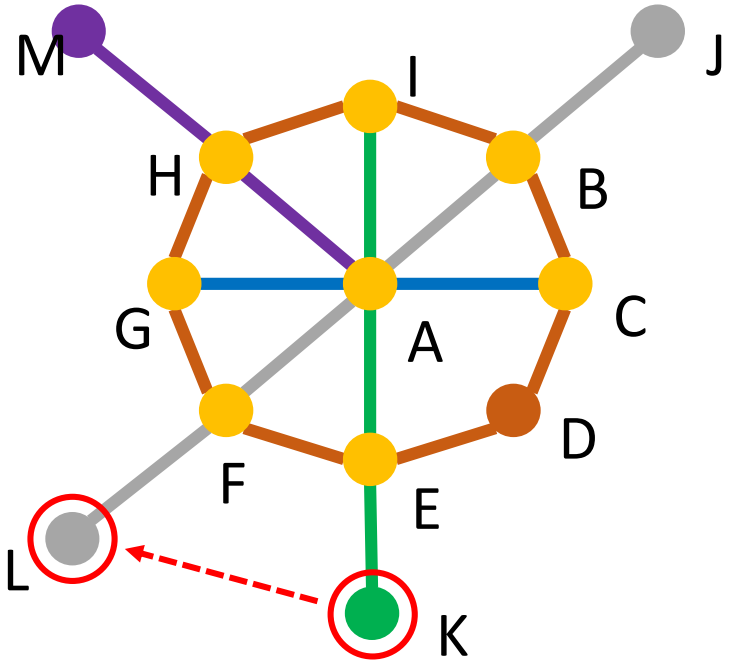
```
graph = {"A": set(["B", "D"]),
         "B": set(["C"]),
         "C": set(["B"]),
         "D": set(["E"]),
         "E": set(["B", "F"]),
         "F": set(["C"])}

def dfs(graph, start):
    visited, stack = set(), [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            stack.extend(graph[vertex] - visited)
    return visited

dfs(graph, 'A')
```

http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

Поиск маршрута

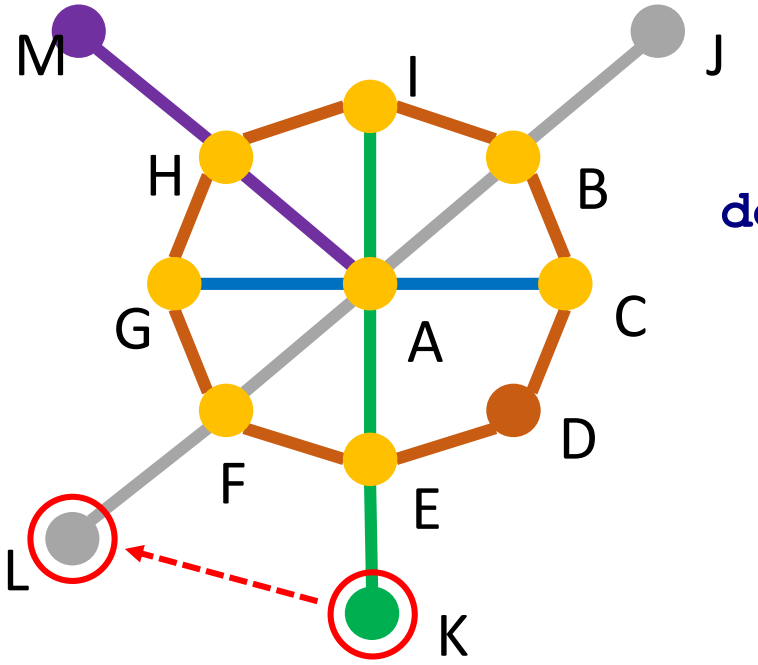


Поиск маршрута. Поиск в глубину




```
graph = {"A": set(["B", "C", "E","F", "G", "H", "I"]),
         "B": set(["A", "J", "I", "C"]),
         "C": set(["A", "B", "D"]),
         "D": set(["C", "E"]),
         "E": set(["A", "D", "F", "K"]),
         "F": set(["A", "E", "G", "L"]),
         "G": set(["A", "F", "H"]),
         "H": set(["A", "G", "I", "M"]),
         "I": set(["A", "H", "B"]),
         "J": set(["B"]),
         "K": set(["E"]),
         "L": set(["F"]),
         "M": set(["H"])}
```

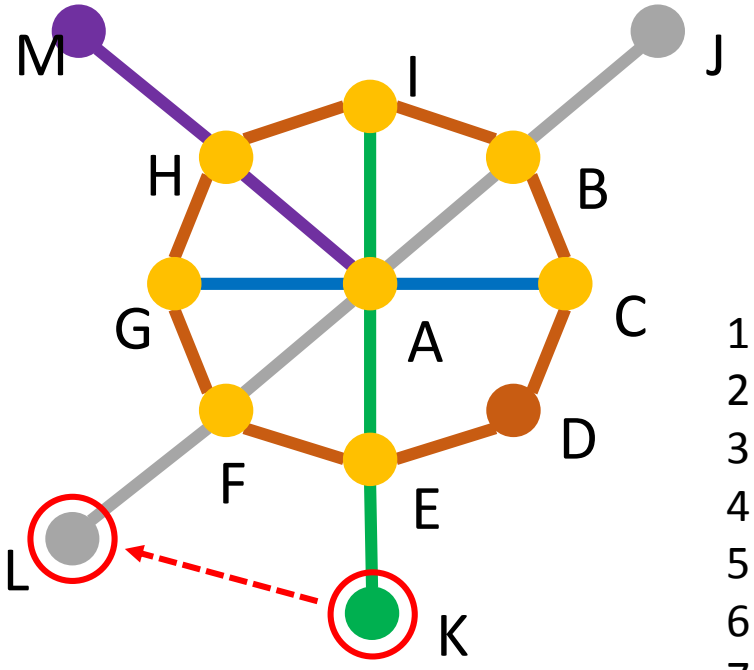
Поиск маршрута. Поиск в глубину




```
def dfs_paths(graph, start, goal):
    stack = [(start, [start])]
    while stack:
        (vertex, path) = stack.pop()
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                stack.append((next, path + [next]))

for i, el in enumerate(dfs_paths(graph, 'K', 'L')):
    print(i+1, el)
```

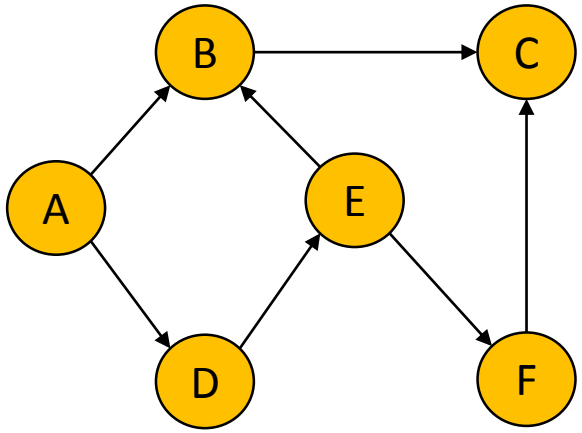
http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

Поиск маршрута. Поиск в глубину

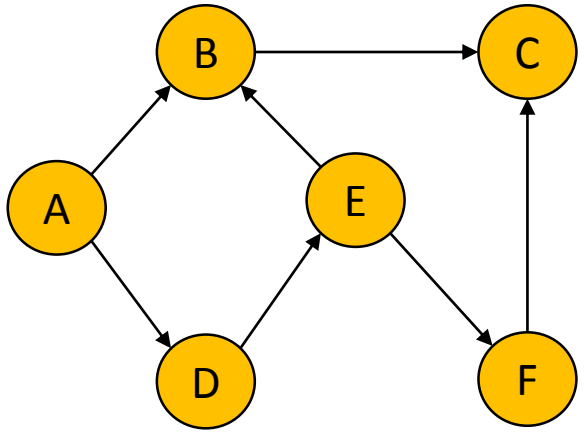


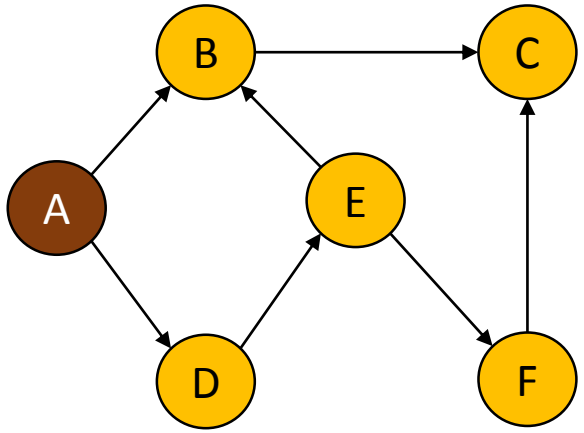
1 ['K', 'E', 'A', 'C', 'B', 'I', 'H', 'G', 'F', 'L']
2 ['K', 'E', 'A', 'F', 'L']
3 ['K', 'E', 'A', 'B', 'I', 'H', 'G', 'F', 'L']
4 ['K', 'E', 'A', 'H', 'G', 'F', 'L']
5 ['K', 'E', 'A', 'I', 'H', 'G', 'F', 'L']
6 ['K', 'E', 'A', 'G', 'F', 'L']
7 ['K', 'E', 'D', 'C', 'A', 'B', 'I', 'H', 'G', 'F', 'L']
8 ['K', 'E', 'D', 'C', 'A', 'H', 'G', 'F', 'L']
9 ['K', 'E', 'D', 'C', 'A', 'I', 'H', 'G', 'F', 'L']
10 ['K', 'E', 'D', 'C', 'A', 'G', 'F', 'L']
11 ['K', 'E', 'D', 'C', 'A', 'F', 'L']
12 ['K', 'E', 'D', 'C', 'B', 'A', 'H', 'G', 'F', 'L']
13 ['K', 'E', 'D', 'C', 'B', 'A', 'I', 'H', 'G', 'F', 'L']
14 ['K', 'E', 'D', 'C', 'B', 'A', 'G', 'F', 'L']
15 ['K', 'E', 'D', 'C', 'B', 'A', 'F', 'L']
16 ['K', 'E', 'D', 'C', 'B', 'I', 'A', 'H', 'G', 'F', 'L']
17 ['K', 'E', 'D', 'C', 'B', 'I', 'A', 'G', 'F', 'L']
18 ['K', 'E', 'D', 'C', 'B', 'I', 'A', 'F', 'L']
19 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'A', 'G', 'F', 'L']
20 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'A', 'F', 'L']
21 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'G', 'A', 'F', 'L']
22 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'G', 'F', 'L']
23 ['K', 'E', 'F', 'L']

Обход в ширину



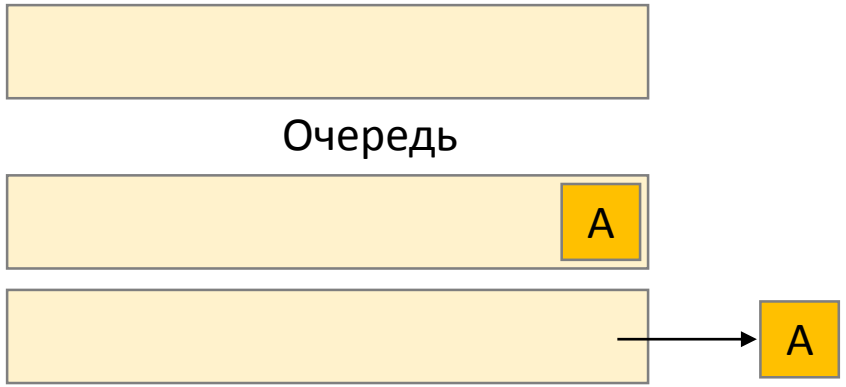
Обход в ширину

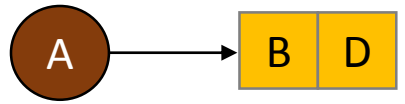




Просмотренные вершины

Очередь

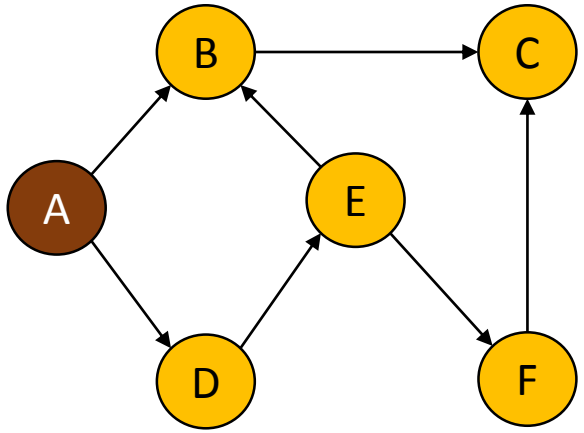




Просмотренные вершины

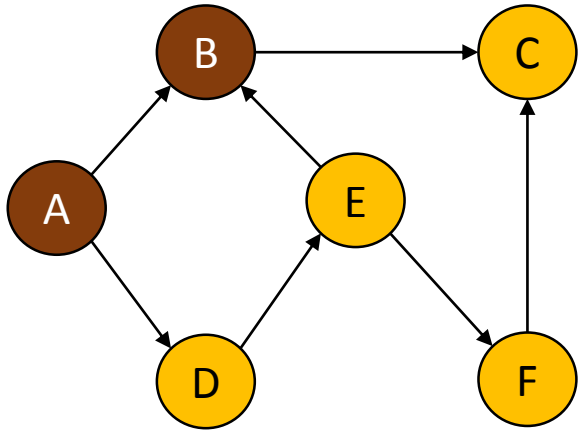
Очередь

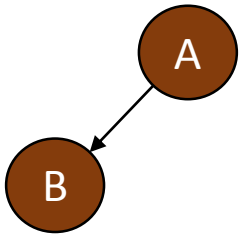
Обход в ширину

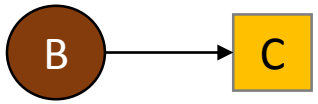


Просмотренные вершины

Очередь



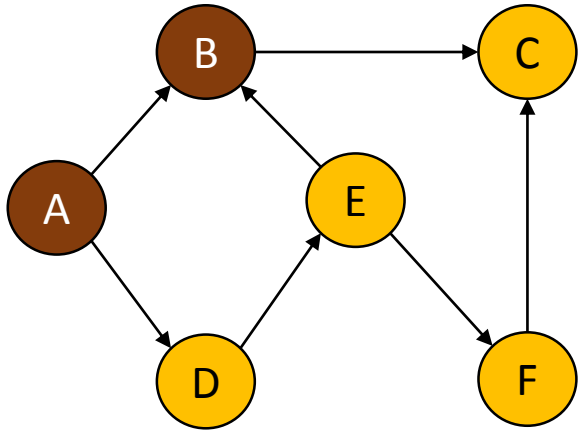




Просмотренные вершины

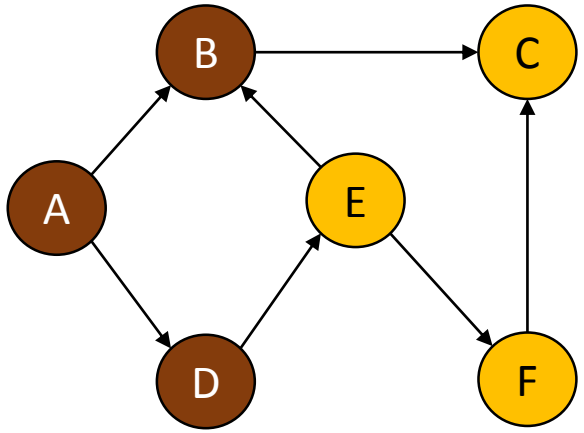
Очередь

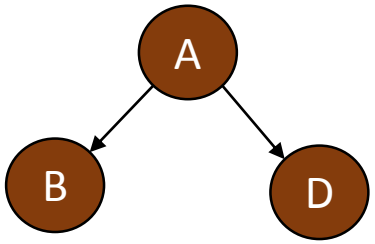
Обход в ширину

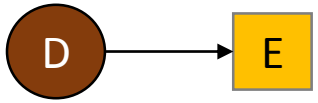


Просмотренные вершины

Очередь



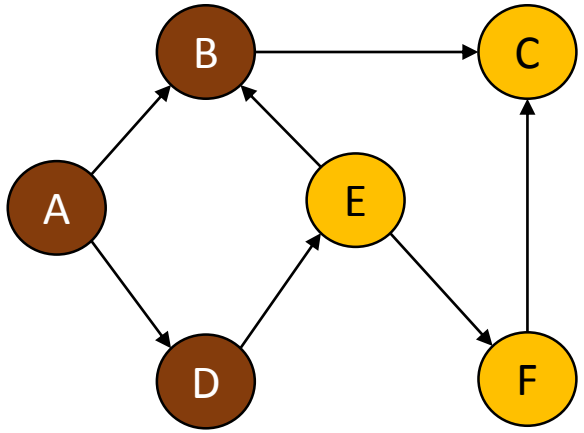




Просмотренные вершины

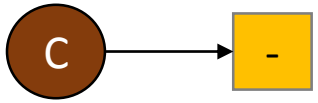
Очередь

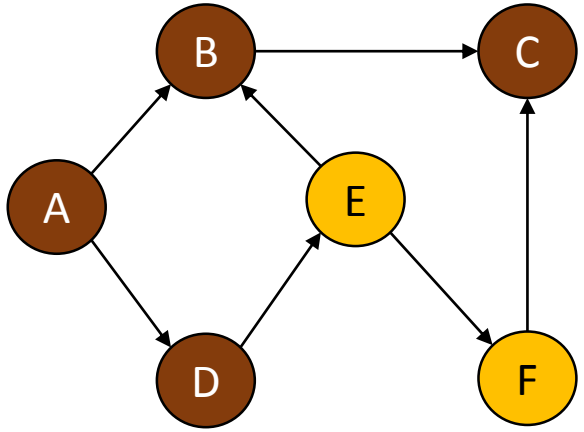
Обход в ширину

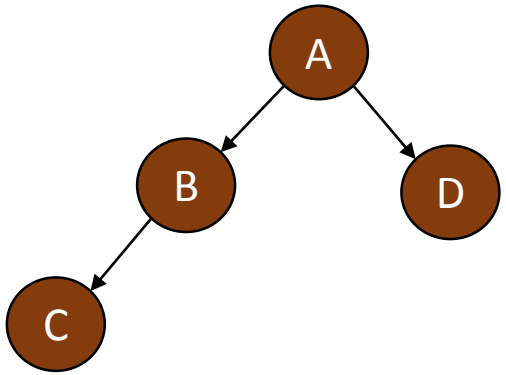


Просмотренные вершины

Очередь



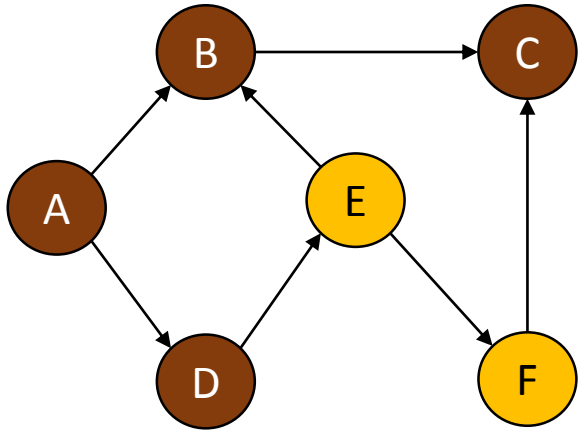




Просмотренные вершины

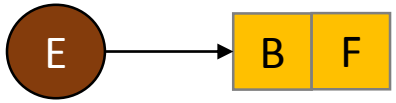
Очередь

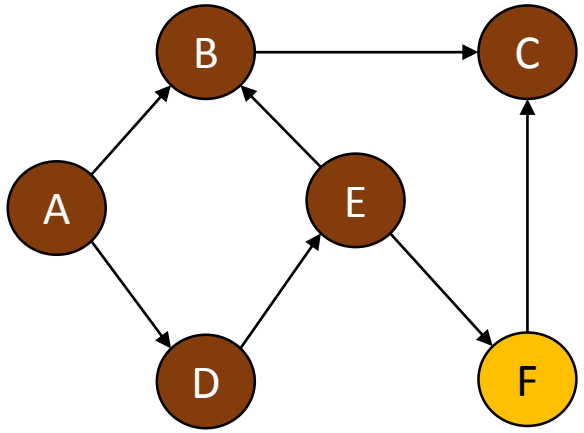
Обход в ширину

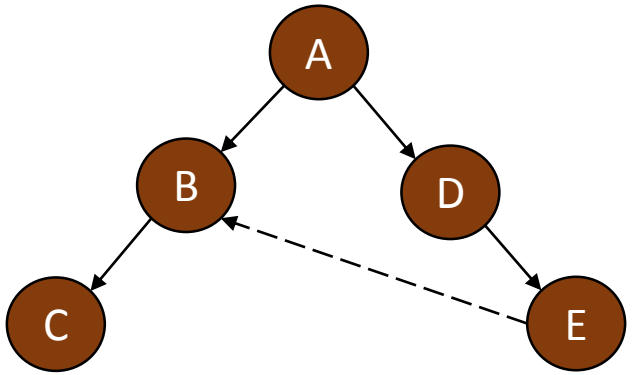


Просмотренные вершины

Очередь



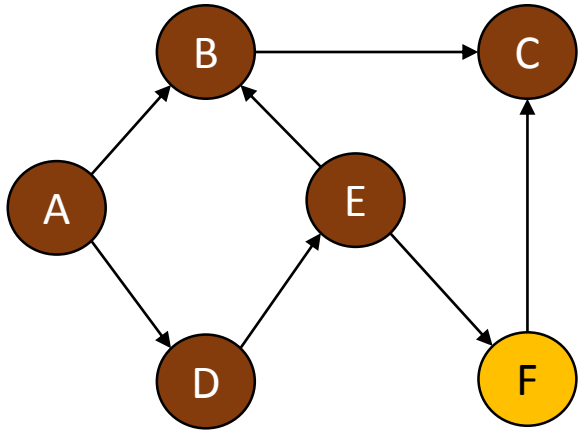




Просмотренные вершины

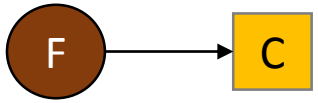
Очередь

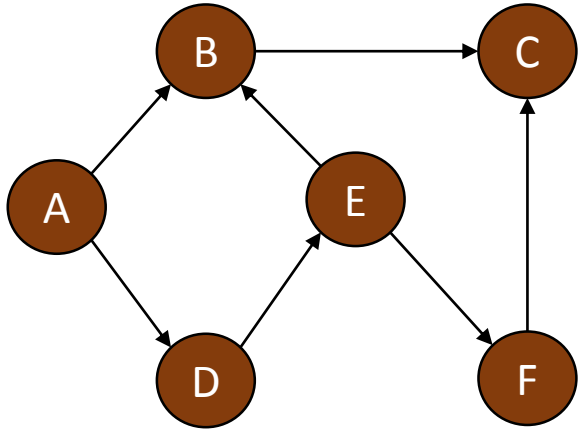
Обход в ширину

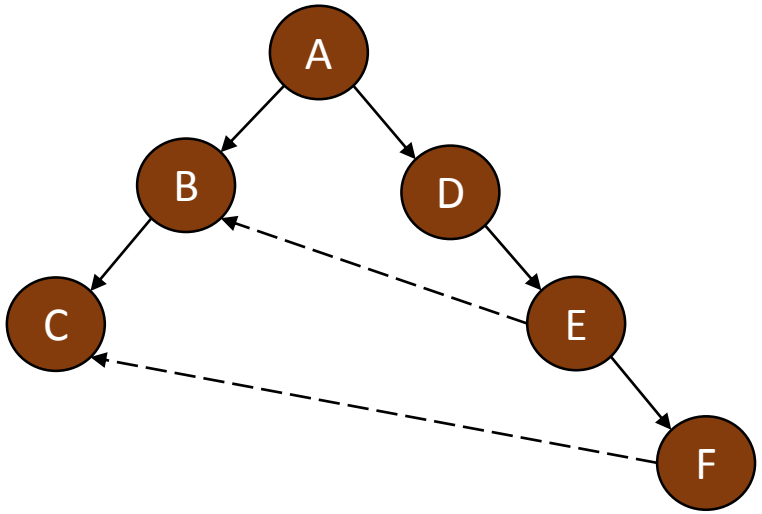


Просмотренные вершины

Очередь







Просмотренные вершины

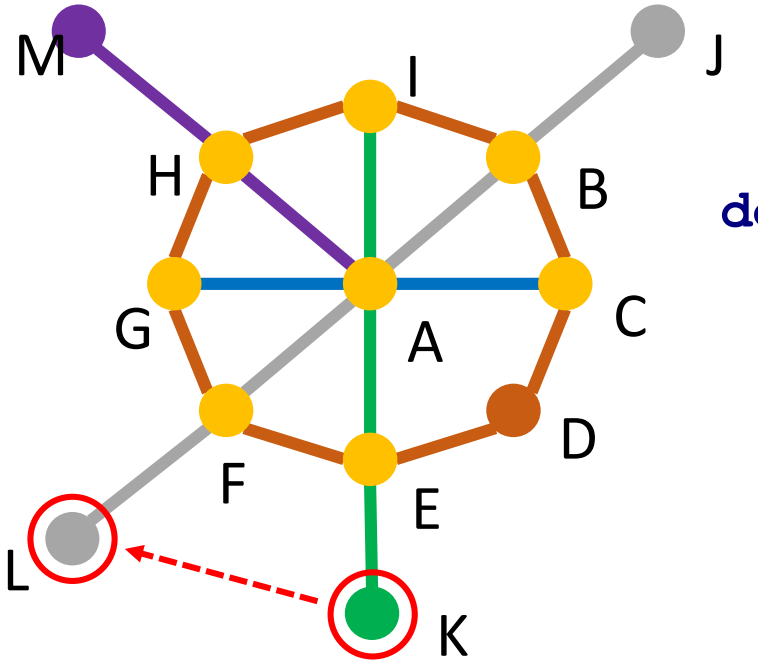
Очередь

Вариант реализации обхода в ширину

```
graph = {"A": set(["B", "D"]),
         "B": set(["C"]),
         "C": set(["B"]),
         "D": set(["E"]),
         "E": set(["B", "F"]),
         "F": set(["C"])}

def bfs(graph, start):
    visited, queue = set(), [start]
    while queue:
        vertex = queue.pop(0)
        if vertex not in visited:
            visited.add(vertex)
            queue.extend(graph[vertex] - visited)
    return visited

bfs(graph, 'A')
```

http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

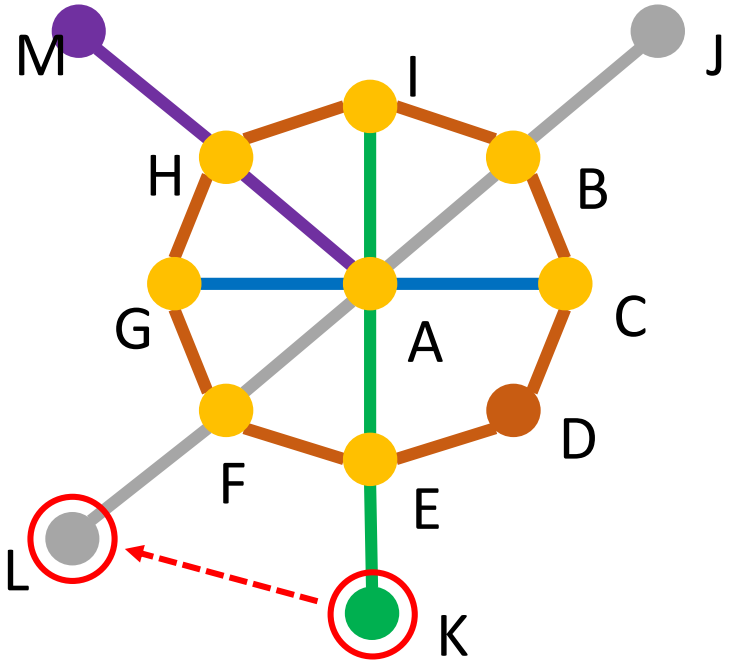
Поиск маршрута. Поиск в глубину




```
def bfs_paths(graph, start, goal):
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                queue.append((next, path + [next]))

for i, el in enumerate(bfs_paths(graph, 'K', 'L')):
    print(i+1, el)
```

http://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

Поиск маршрута. Поиск в ширину



1 ['K', 'E', 'F', 'L']
2 ['K', 'E', 'A', 'F', 'L']
3 ['K', 'E', 'A', 'G', 'F', 'L']
4 ['K', 'E', 'D', 'C', 'A', 'F', 'L']
5 ['K', 'E', 'A', 'H', 'G', 'F', 'L']
6 ['K', 'E', 'D', 'C', 'B', 'A', 'F', 'L']
7 ['K', 'E', 'D', 'C', 'A', 'G', 'F', 'L']
8 ['K', 'E', 'A', 'I', 'H', 'G', 'F', 'L']
9 ['K', 'E', 'D', 'C', 'B', 'A', 'G', 'F', 'L']
10 ['K', 'E', 'D', 'C', 'B', 'I', 'A', 'F', 'L']
11 ['K', 'E', 'D', 'C', 'A', 'H', 'G', 'F', 'L']
12 ['K', 'E', 'A', 'B', 'I', 'H', 'G', 'F', 'L']
13 ['K', 'E', 'D', 'C', 'B', 'A', 'H', 'G', 'F', 'L']
14 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'G', 'F', 'L']
15 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'A', 'F', 'L']
16 ['K', 'E', 'D', 'C', 'B', 'I', 'A', 'G', 'F', 'L']
17 ['K', 'E', 'D', 'C', 'A', 'I', 'H', 'G', 'F', 'L']
18 ['K', 'E', 'A', 'C', 'B', 'I', 'H', 'G', 'F', 'L']
19 ['K', 'E', 'D', 'C', 'B', 'A', 'I', 'H', 'G', 'F', 'L']
20 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'G', 'A', 'F', 'L']
21 ['K', 'E', 'D', 'C', 'B', 'I', 'H', 'A', 'G', 'F', 'L']
22 ['K', 'E', 'D', 'C', 'B', 'I', 'A', 'H', 'G', 'F', 'L']
23 ['K', 'E', 'D', 'C', 'A', 'B', 'I', 'H', 'G', 'F', 'L']

Ссылка на задачи 12-го семинара

https://official.contest.yandex.ru/contest/1891/enter

Вопрос

1
```
def func(ml):
    ml.append(5)
    ml = []
    print(ml)

myList = [1,2,3]

func(myList)

print(myList)
```

2
```
def func(ml):
    ml.append(5)
    ml[:] = []
    print(ml)

myList = [1,2,3]

func(myList)

print(myList)
```

3
```
def func(ml):
    ml.append(5)
    ml = []
    print(ml)
    return ml

myList = [1,2,3]

myList = func(myList)

print(myList)
```

Источники

Основные определения теории графов

Depth-First Search and Breadth-First Search in Python

Graphs and Graph Algorithms