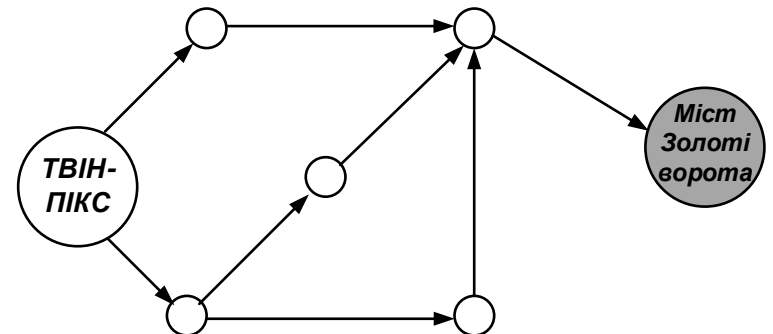
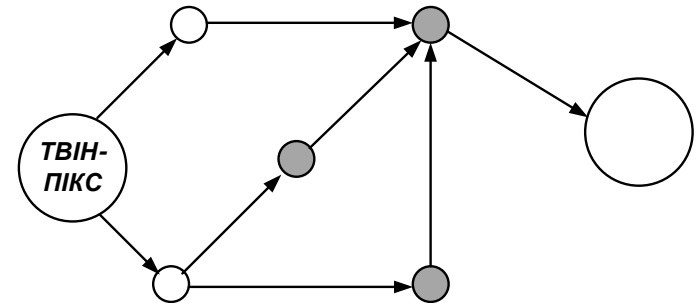
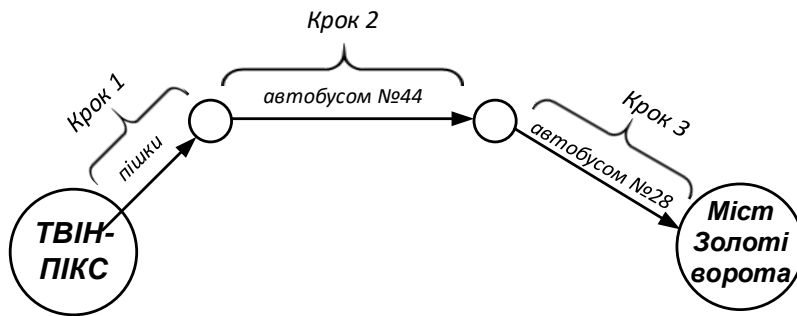
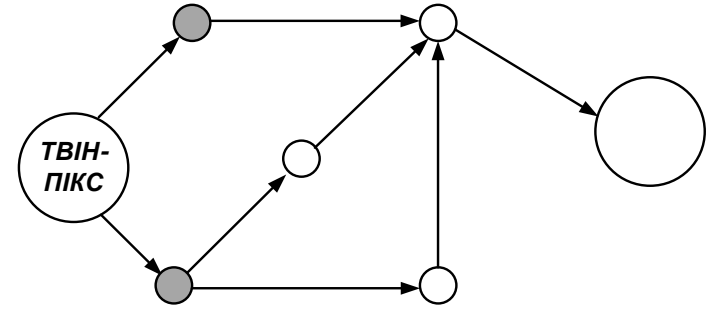
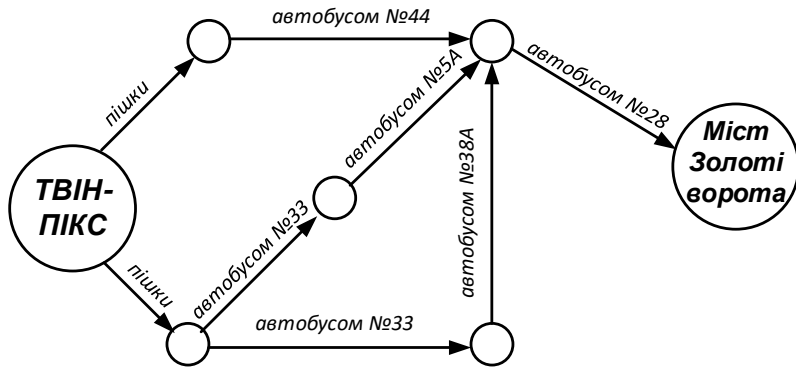


Графи



```
graph LR; A((АЛЕКС)) --> M((РАМА)); T((ТОМ)) --> M; T --> D((АДИТ)); M --> D;
```

Вузел

Ребро

Вузел

АЛЕКС

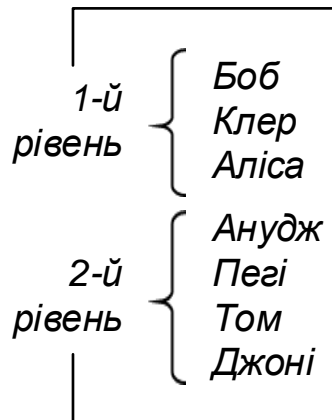
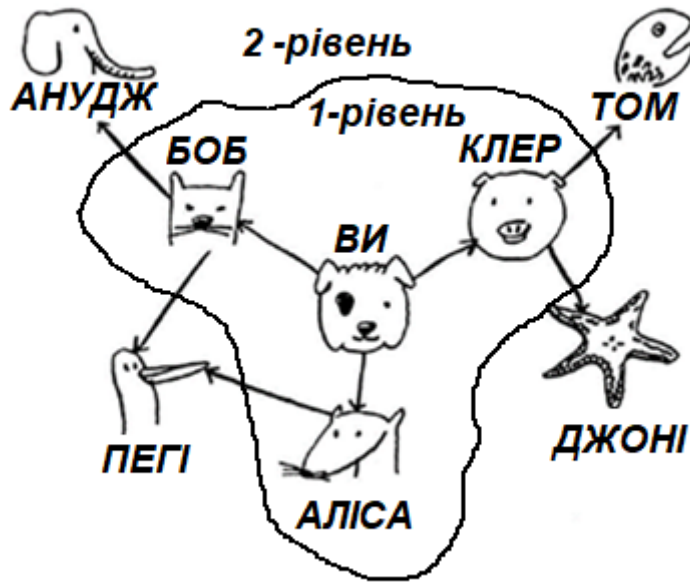
РАМА

Він допомагає дати відповідь на запитання двох типів:

-
- ```

graph LR
 A["АЛИСА
□ БОБ
□ КЛЭР"] --> B["АЛИСА ПРОДАЕТ
МАНГО?"]
 B -- "ДА: ЗАВЕРШИТЬ" --> C["ПЕГГИ ДОБАВЛЕНА
В СПИСОК"]
 B -- "НЕТ: ДОБАВИТЬ
ВСЕХ ДРУЗЕЙ
АЛИСЫ В СПИСОК
ПОИСКА" --> D["БОБ
КЛЭР
ПЕГГИ"]
 D --> C

```



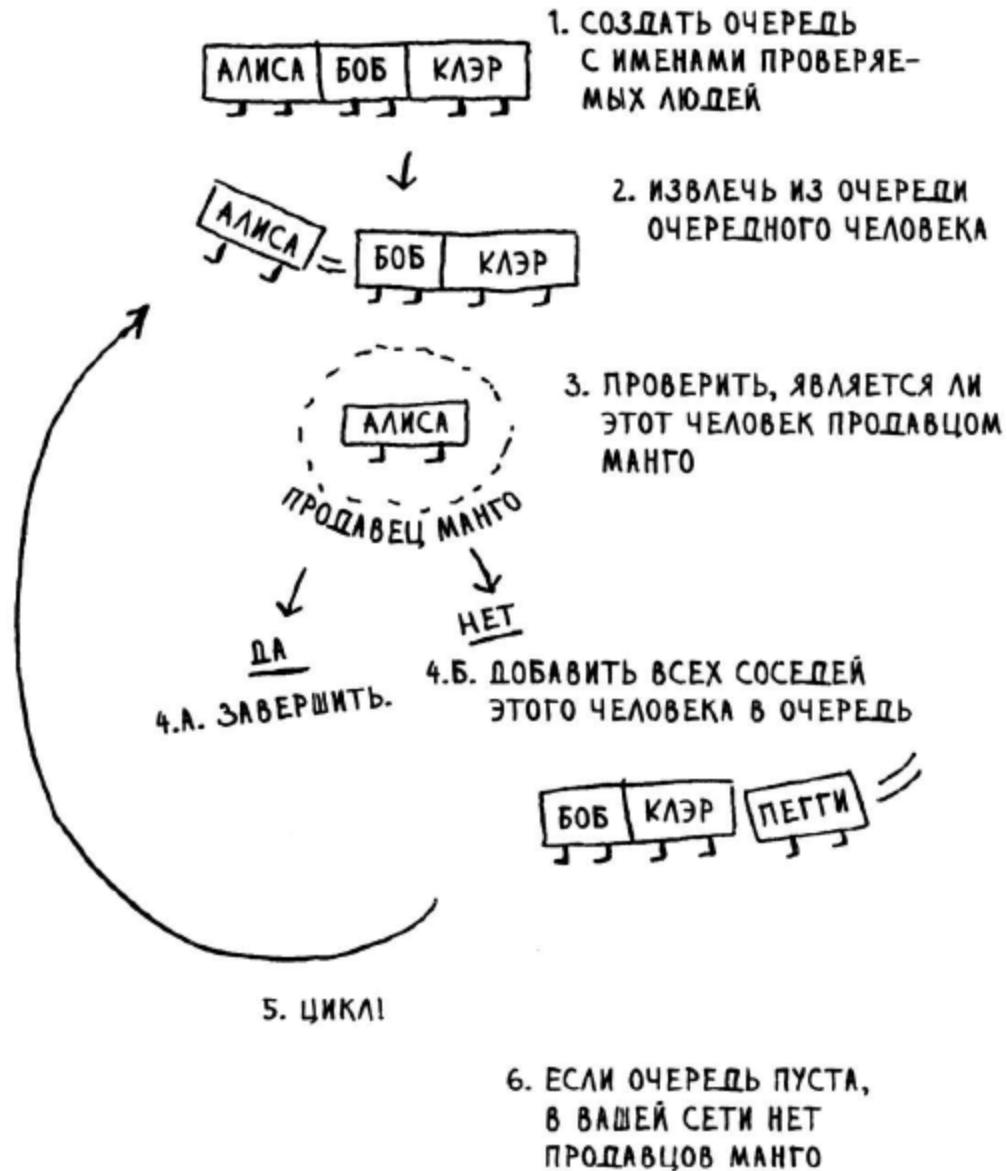
`graph["claire"] = ["thom", "jonny"]`  
`graph["anuj"] = []`

```
graph = {}
graph["you"] = ["alice", "bob", "claire"]

graph = {}
graph["you"] = ["alice", " bob", "claire"]
graph["bob"] = ["anuj", "peggy"]
graph["alice"] = ["peggy"]
graph["claire"] = ["thom", "jonny"]
graph["anuj"] = []
graph["peggy"] = []
graph["thom"] = []
graph["jonny"] = []
```

`graph["anuj"] = []`  
`graph["claire"] = ["thom", "jonny"]`

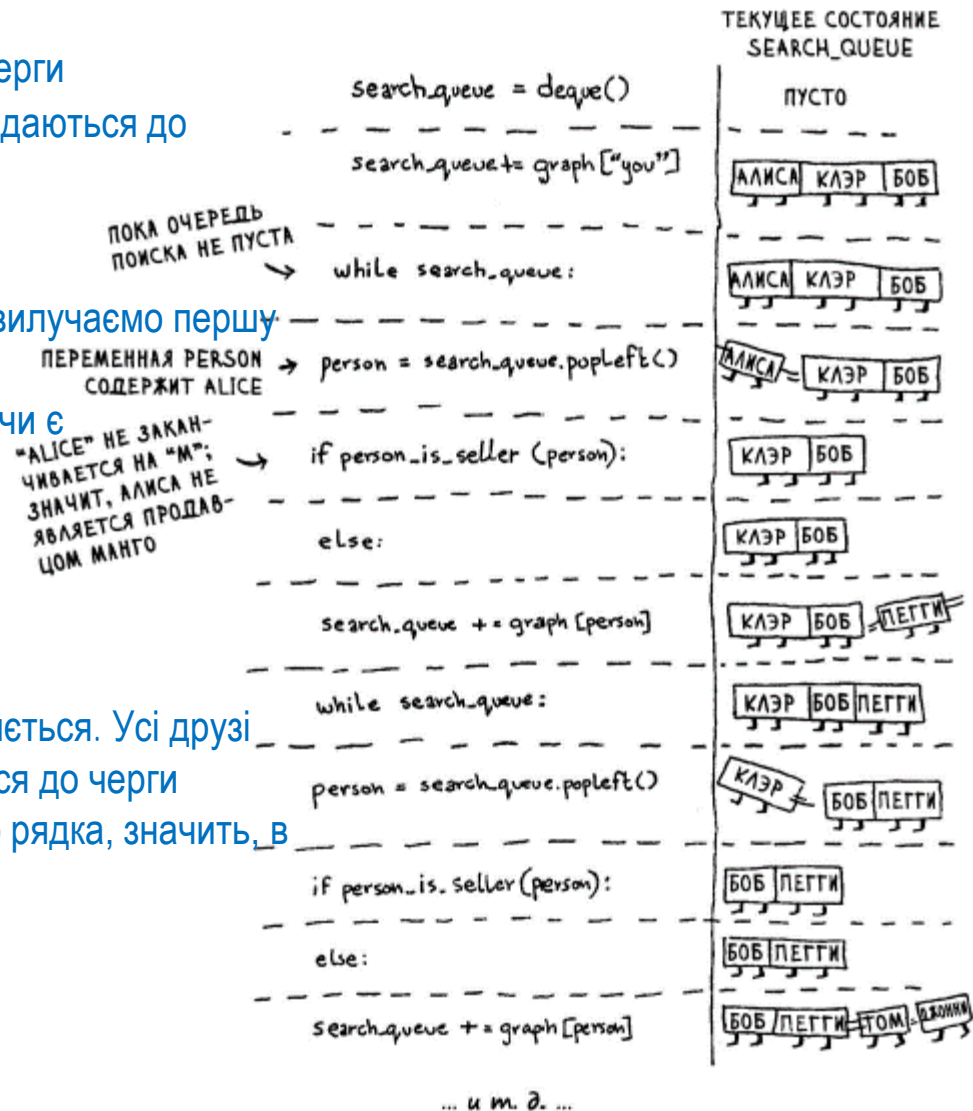
## Реалізація алгоритму



```
from collections import deque
search_queue = deque() //Створення нової черги
search_queue += graph["you"] //Усі сусіди додаються до
черги пошуку
```

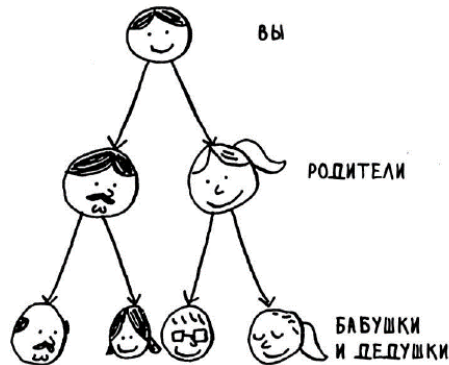
```
while search_queue: // Пока черга не порожня
 person = search_queue.popleft() // з черги вилучаємо першу
 людину
 if person_is_seller(person): //Перевіряємо, чи є
 // людина продавцем манго
 print person + " is a mango seller !"
 //Да, це продавець манго
 return True
 else:
 search_queue += graph[person] //Ні, не являється. Усі друзі
 //даної людини додаються до черги
return False //Якщо виконання дійшло до цього рядка, значить, в
черзі немає продавця манго
```

```
def person_is_seller(name):
 return name[-1] == 'm'
```



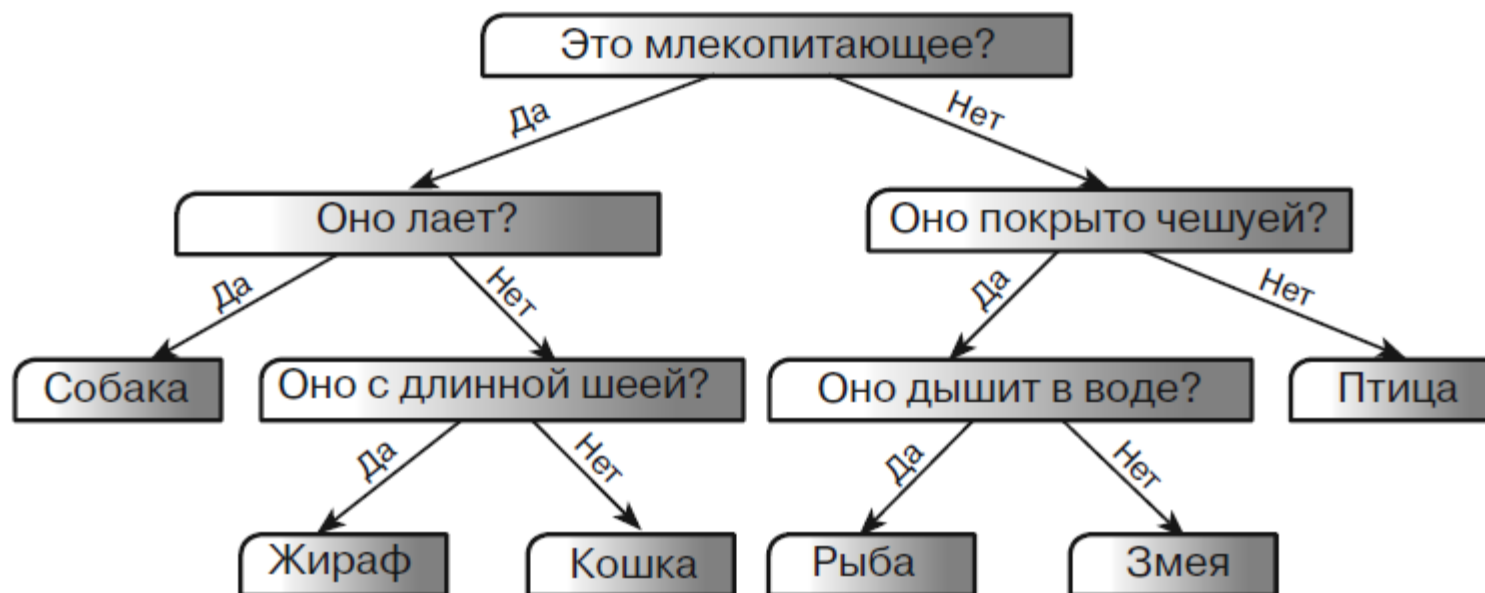
```
def search(name):
 search_queue = deque()
 search_queue += graph[name]
 searched = [] <----- Этот массив используется для отслеживания
 уже проверенных людей
 while search_queue:
 person = search_queue.popleft()
 if not person in searched: <----- Человек проверяется только в том случае,
 если он не проверялся ранее
 if person_isSeller(person):
 print person + " is a mango seller!"
 return True
 else:
 search_queue += graph[person]
 searched.append(person) <----- Человек помечается как
 уже проверенный
 return False
```

search("you")



Такий особий різновид графу, в якому немає ребер, що вказуються в зворотньому напрямку, називається *деревом*.

1. Пошук в ширину дозволяє визначити, чи існує шлях з А в В .
2. Якщо шлях існує, то пошук в ширину знаходить найкоротший шлях.
3. У наведеному графі є стрілки, а відношення діють в напрямку стрілки.
4. У ненаправлених графах стрілок немає, а відношення йде в обидві сторони
5. Черги відносяться до категорії FIFO.
6. Людей слід перевіряти в порядку їх додавання до списку пошуку, тому список пошуку повинен бути оформлений у вигляді черги, інакше знайдений шлях не буде найкоротшим.
7. Подбайте про те, щоб уже перевірений людина не перевірявся заново, інакше може виникнути нескінченний цикл.



**Таблица 10.2.** Игра «Животные» пытается угадать змею

| Вопрос программы    | Ответ пользователя |
|---------------------|--------------------|
| Это млекопитающее?  | Нет                |
| Оно покрыто чешуей? | Да                 |
| Оно дышит в воде?   | Нет                |
| Это змея?           | Да                 |

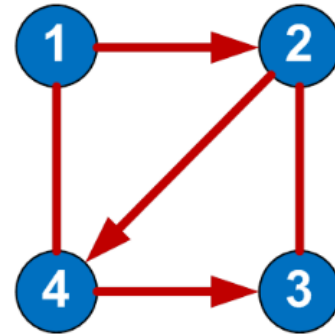
**Таблица 10.3.** Игра «Животные» пытается угадать жирафа

| Вопрос программы                               | Ответ пользователя  |
|------------------------------------------------|---------------------|
| Это млекопитающее?                             | Да                  |
| Оно лает?                                      | Нет                 |
| Это кошка?                                     | Нет                 |
| Какое животное вы загадали?                    | Жираф               |
| Какой вопрос поможет отличить кошку от жирафа? | У него длинная шея? |
| Для жирафа ответ утвердительный?               | Да                  |

## Способи представлення графа

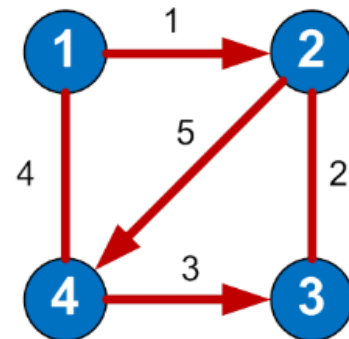
Матриця суміжності

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



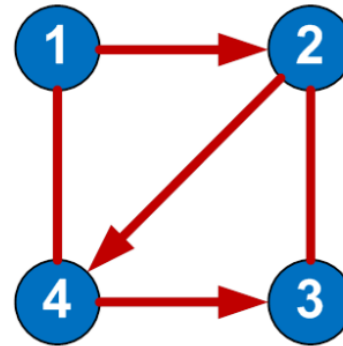
Матриця інцидентності

|   | 1  | 2 | 3  | 4 | 5  |
|---|----|---|----|---|----|
| 1 | 1  | 0 | 0  | 1 | 0  |
| 2 | -1 | 1 | 0  | 0 | 1  |
| 3 | 0  | 1 | -1 | 0 | 0  |
| 4 | 0  | 0 | 1  | 1 | -1 |



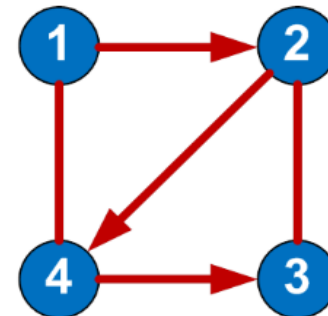
### Список суміжності

| 1 | 2, 4 |
|---|------|
| 2 | 3, 4 |
| 3 | 2    |
| 4 | 1, 3 |



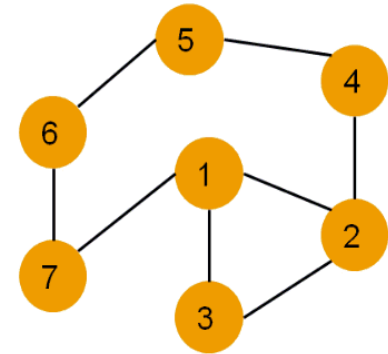
### Список ребер.

|   | Начало | Конец | Вес |
|---|--------|-------|-----|
| 1 | 1      | 2     |     |
| 2 | 1      | 4     |     |
| 3 | 2      | 3     |     |
| 4 | 2      | 4     |     |
| 5 | 3      | 2     |     |
| 6 | 4      | 1     |     |
| 7 | 4      | 3     |     |



**Пошук в ширину** передбачає дослідження графа:

- спочатку відвідуємо корень – довільно вибраний вузол,
- далі – усі нащатки даного вузла,
- після цього відвідуються нащадки нащадків.



Вершини переглядаються в порядку зростання їх відстані від корня.

Алгоритм припиняє роботу після обходу усіх вершин графа, або у випадку виконання необхідної умови (наприклад, знайти найкоротчайший шлях з вершини 1 до вершини 6).

Кожна вершина може знаходитися в одному з 3 станів:

- 1 помаранчевий – невиявлена вершина;
- 2 зелений – виявлена але не відвідана вершина;
- 3 сірий – оброблена вершина.

Фіолетовий – вершина, що розглядається.

### **Застосування алгоритму пошуку в ширину**

Пошук найкоротшого шляху у незваженому графі (орієнтований або неорієнтований).

- Пошук компоненту зв'язності.
- Знаходження рішення будь-якої задачі (гри) з найменшим числом ходів.
- Знайти всі ребра, що лежать на якомусь найкоротшому шляху між заданою парою вершин.
- Знайти всі вершини, що лежать на якомусь найкоротшому шляху між заданою парою вершин

## Задача пошуку найкоротшого шляху

<https://prog-cpp.ru/data-graph/>

```
int mas[7][7] = { { 0, 1, 1, 0, 0, 0, 1 },
 { 1, 0, 1, 1, 0, 0, 0 },
 { 1, 1, 0, 0, 0, 0, 0 },
 { 0, 1, 0, 0, 1, 0, 0 },
 { 0, 0, 0, 1, 0, 1, 0 },
 { 0, 0, 0, 0, 1, 0, 1 },
 { 1, 0, 0, 0, 0, 1, 0 } };
```

```
> clang++-7 -pthread -o main main.cpp
> ./main
sh: 1: chcp: not found
sh: 1: cls: not found
N = 5
1
2
3
7
4
6
5
Путь до вершины 5
5 <- 4 <- 2 <- 1
```

