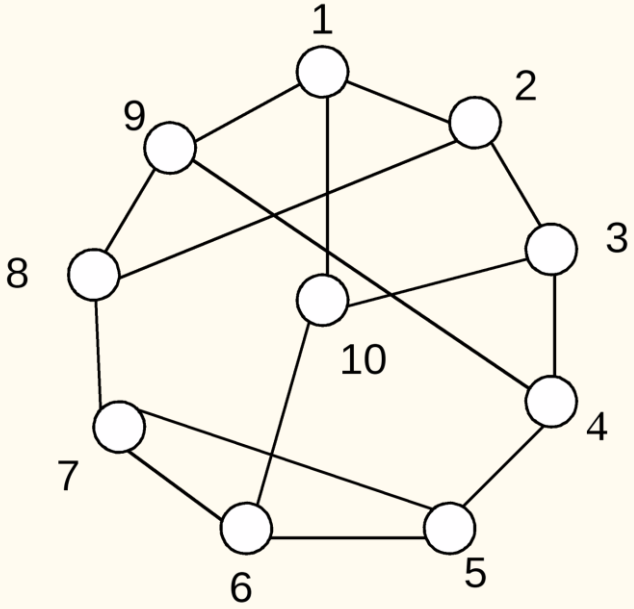
Работа с графами в языке Python

Презентация подготовлена учеником 11А класса
МБОУ СОШ №6 под руководством Баурова А.Ю

Немного о графах в общем



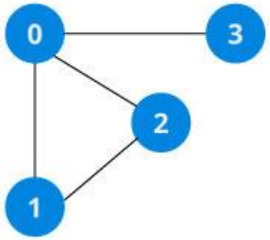
Хеширование

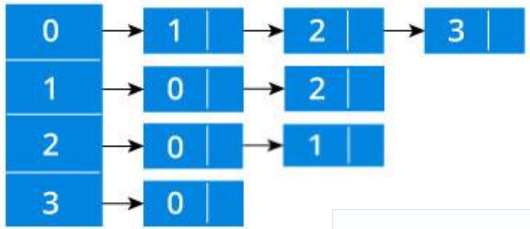
Существует техника, подробно описываемая в большинстве книг об алгоритмах, и неявно используемая программистами на Python, это — хэширование. Хэширование подразумевает вычисление целого числа (часто выглядящего как случайное), основанного на произвольном объекте. Потом это значение можно будет использовать, например, как индекс в массиве (после некоторых преобразований, чтобы не выйти за границы массива).

```
Стандартный механизм хэширования в Python представлен функцией hash:
>>> hash(42)
42
>>> hash("Hello, world!")
-1886531940
```

Списки смежности

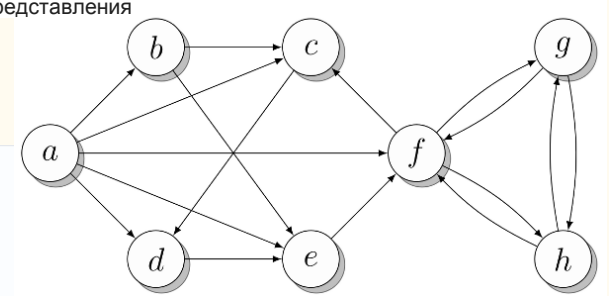
Один из наиболее очевидных способов представления графов — это списки смежности. Смысл их в том, что для каждой вершины создается список (или множество, или другой контейнер или итератор) смежных с ней вершин. Разберем простейший способ реализации такого списка, предположив, что имеется *n* вершин, пронумерованных *0… n-1*.






```
a, b, c, d, e, f, g, h = range(8)
N = [
        {b, c, d, e, f}, # a
        {c, e}, # b
        {d}, # c
        {e}, # d
        {f}, # e
        {c, g, h}, # f
        {f, h}, # g
        {f, g} # h
]
```

Примерный граф для демонстрации разных видов представления



Матрицы смежности

Другая распространенная форма представления графов — это матрицы смежности. Основное отличие их в следующем: вместо перечисления всех смежных с каждой из вершин, мы записываем один ряд значений (массив), каждое из которых соответствует возможной смежной вершине (есть хотя бы одна такая для каждой вершины графа), и сохраняем значение (в виде *True* или *False*), показывающее, действительно ли вершина является смежной. И вновь простейшую реализацию можно получить используя вложенные списки, как видно из листинга ниже. Обратите внимание, что это также требует, чтобы вершины были пронумерованы от *0* до *V-1*. В качестве значений истинности используются 1 и 0 (вместо *True* и *False*), чтобы сделать матрицу читабельной.

```
a, b, c, d, e, f, g, h = range(8)
# a b c d e f g h
N =  [[0,1,1,1,1,1,0,0], # a
      [0,0,1,0,1,0,0,0], # b
      [0,0,0,1,0,0,0,0], # c
      [0,0,0,0,1,0,0,0], # d
      [0,0,0,0,0,1,0,0], # e
      [0,0,1,0,0,0,1,1], # f
      [0,0,0,0,0,1,0,1], # g
      [0,0,0,0,0,1,1,0]] # h
```

```
>>> N[a][b]
1
>>> sum(N[f])
3
```

Матрицы смежности

```
a, b, c, d, e, f, g, h = range(8)
_ = float('inf')
                # a b c d e f g h
W = [[0,2,1,3,9,4,_,_], # a
         [_,0,4,_,3,_,_,_], # b
         [_,_,0,8,_,_,_,_], # c
         [_,_,_,0,7,_,_,_], # d
         [_,_,_,_,0,5,_,_], # e
         [_,_,2,_,_,0,2,2], # f
         [_,_,_,_,_,1,0,6], # g
         [_,_,_,_,_,9,8,0]] # h
```

```
>>> W[a][b] < inf # смежность
True
>>> W[c][e] < inf # смежность
False
>>> sum(1 for w in W[a] if w < inf) - 1 # степень
5
```

Массивы специального назначения из NumPy

Библиотека NumPy содержит много функциональности, связанной с многомерными массивами. Для представления графов большая ее часть не нужна, но массивы из NumPy весьма полезны, например, для реализации матриц весов или смежности.

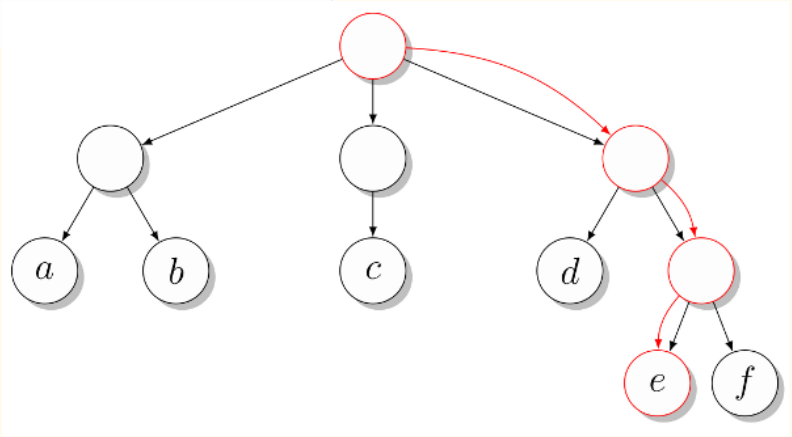
```
Вместо создания пустой матрицы весов или смежности из списков для n вершин, вроде такого:
>>> N = [[0]*10 for i in range(10)]

в NumPy можно использовать функцию zeros:
>>> import numpy as np
>>> N = np.zeros([10,10])
```

Реализация деревьев

```
>>> T = [["a", "b"], ["c"], ["d", ["e", "f"]]]
>>> T[0][1]
'b'
>>> T[2][1][0]
'e'
```

Пример дерева с отмеченным путем от корня к листу



Реализация деревьев

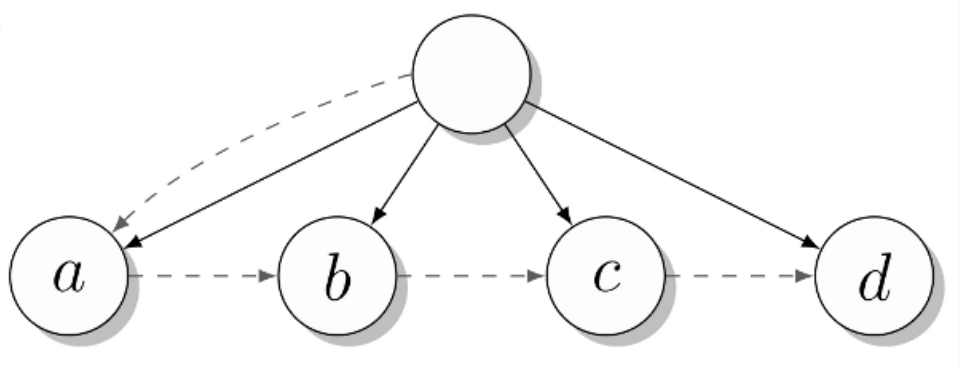
```
class Tree:
        def __init__(self, left, right):
                self.left = left
                self.right = right
>>> t = Tree(Tree("a", "b"), Tree("c", "d"))
>>> t.right.left
'c'
```

```
class Tree:
        def __init__(self, kids, next=None):
                self.kids = self.val = kids
                self.next = next
```

Реализация деревьев

```
>>> t = Tree(Tree("a", Tree("b", Tree("c", Tree("d")))))
>>> t.kids.next.next.val
'c'
```

А вот как выглядит это дерево:



Шаблон проектирования “Набор”

При проектировании (да и реализации) таких структур данных как деревья может оказаться полезным гибкий класс, позволяющий задавать набор атрибутов через конструктор. Здесь нас может выручить шаблон проектирования «Набор» (названный так Алексом Мартелли в «Python Cookbook»). Есть много способов его реализации, но суть видна из следующего кода:

```
class Bunch(dict):
        def __init__(self, *args, **kwds):
                super(Bunch, self).__init__(*args, **kwds)
                self.__dict__ = self
```

Есть несколько полезных способов его применения. Во-первых, он позволяет создать и задать значения атрибутов, передав их как аргументы при создании объекта:

```
>>> x = Bunch(name="Jayne Cobb", position="PR")
>>> x.name
'Jayne Cobb'
```

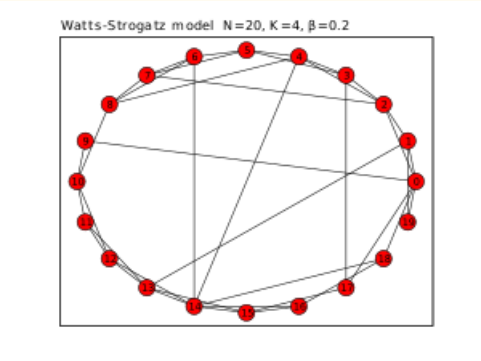
Во-вторых, наследование от dict дает много дополнительного функционала, такого как получение всех ключей (атрибутов) или простая проверка наличия атрибута. Вот пример:

```
>>> T = Bunch
>>> t = T(left=T(left="a", right="b"), right=T(left="c"))
>>> t.left
{'right': 'b', 'left': 'a'}
>>> t.left.right
'b'
>>> t['left']['right']
'b'
>>> "left" in t.right
True
>>> "right" in t.right
False
```

Конечно же этот шаблон можно использовать не только для создания деревьев. Он пригодится в любой ситуации, где необходим гибкий объект, умеющий задавать свои атрибуты при создании.

Библиотеки для работы с графами в Python

Граф, созданный с помощью NetworkX



Gato

Источники информации

https://habr.com/ru/post/112421/
https://pythonworld.ru/tipy-dannyx-v-python/slovari-dict-funkcii-i-metody-slovarej.html
http://gato.sourceforge.net/
https://pyprog.pro/array_creation/zeros.html
https://evileg.com/ru/post/502/