Абстрактные типы данных, Бинарные деревья, BST

Хайрулин Сергей Сергеевич
s.khayrulin@gmail.com

Overview

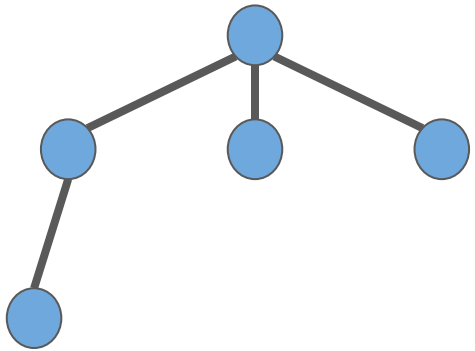
1. Деревья, основные понятия
2. Бинарные деревья
3. Бинарные деревья поиска и основные операции над ними
    - вставка
    - удаление
    - поиска элемента
4. Двоичная куча и основные операции над ними
    - вставка
    - удаление
    - поиска элемента

Литература и др. источники

1. Дональд Эрвин Кнут. Искусство программирования (Том 1, 2, 3) // Вильямс 2015.
2. Альфред В. Ахо, Джон Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы // Вильямс 2000.
3. Емеличев В. А., Мельников О. И., Сарванов В. И., Тышкевич Р. И. Лекции по теории графов // М.: Наука, 1990.
4. Харари Ф. Теория графов // М.: Мир, 1973.
5. Косточка А. В. Дискретная математика. Часть 2 //Новосибирск: НГУ, 2001.
6. Котов В. Е., Сабельфельд В. К. Теория схем программ // Наука 1991.
7. http://algolist.manual.ru
8. ….

Деревья

Связный граф без циклов называется *деревом.*



Деревья

Дерево - это совокупность элементов, называемыми узлами(один из которых определен как корень), и отношений (“**родительских”**), образующих иерархическую структуру узлов. Узлы, так же, как и элементы списков, могут быть элементами любого типа. Мы часто будем изображать узлы буквами, строками или числами. Формально дерево можно определить рекуррентно:

Деревья

1. Один **узел** является деревом. Этот же узел является **корнем** дерева.
2. Пусть **n** - узел, а $T_1,T_1,...,T_k$ - деревья с корнями в узлах n_1, n_2,…, n_k соответственно. Можно построить новое дерево, сделав **n** родительским узлом для n_1, n_2,…, n_k. В этом дереве **n** - будет корнем, а $T_1,T_1,...,T_k$ - поддеревья этого корня. n_2,…, n_k - сыновьями узла **n.**

Деревья

Предок или потомок узла, имеющий предка или потомка, называется **истинным предком** или **истинным потомком** соответственно. В дереве только корень не имеет истинного предка. Узел, не имеющий истинных потомков, называется **листом**.
Высотой узла дерева называется длина самого длинного пути из этого узла до какого-либо листа.
Высота дерева совпадает с высотой корня.
Глубина узла определяется как длина пути (он единственный) от корня до этого узла.

Деревья

- Прямой: Каждый узел посещается до того, как посещены его потомки. Для корня дерева рекурсивно вызывается следующая процедура:
  - Посетить узел
  - Обойти левое поддерево
  - Обойти правое поддерево

Деревья

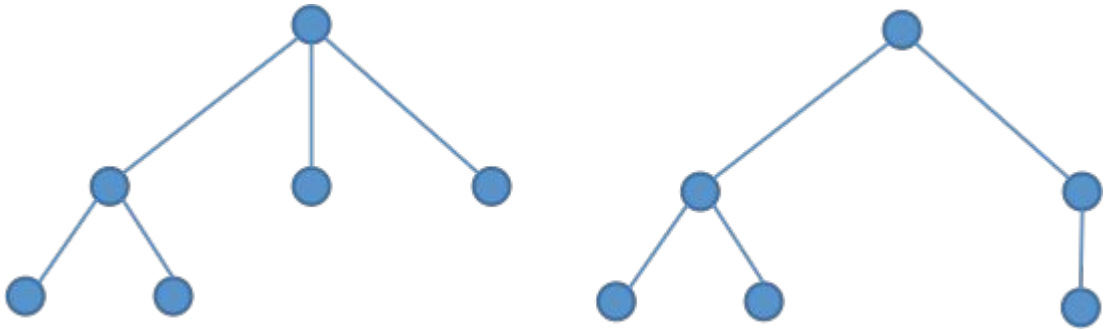
- Обратный: Узлы посещаются 'снизу вверх'. Для корня дерева рекурсивно вызывается следующая процедура:
  - Обойти левое поддерево
  - Обойти правое поддерево
  - Посетить узел

Деревья

- Симметричный: Посещаем сначала левое поддерево, затем узел, затем - правое поддерево. Для корня дерева рекурсивно вызывается следующая процедура:
  - Обойти левое поддерево
  - Посетить узел
  - Обойти правое поддерево

Бинарные деревья

Бинарное дерево (двоичное дерево) – дерево у которого каждый лист имеет не более двух потомков.

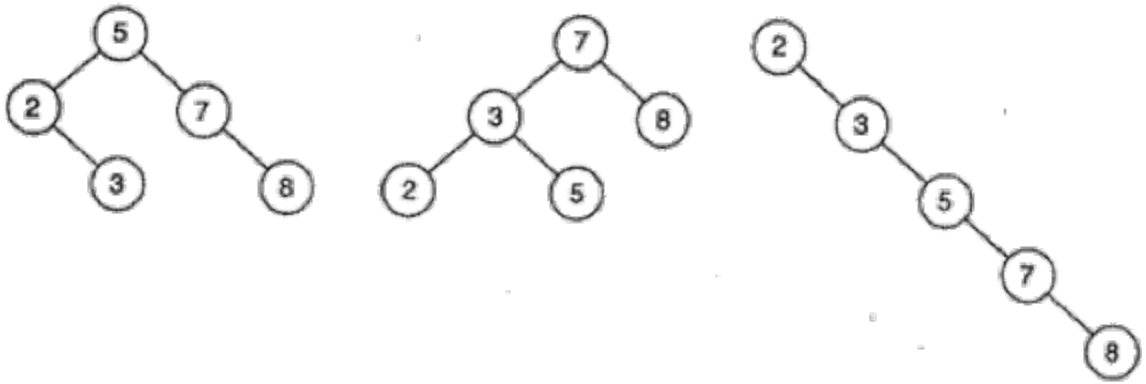


Бинарные деревья поиска

Бинарных дерево поиска (binary search tree, BST) – бинарное дерево, для которого верны следующие утверждения:

- Оба поддерева — левое и правое — являются двоичными деревьями поиска.
- У всех узлов *левого* поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.
- У всех узлов *правого* поддерева произвольного узла X значения ключей данных больше либо равно, нежели значение ключа данных самого узла X.

Бинарные деревья поиска



Поиск

Поиск: Задача в заданном двоичном дереве поиска T проверить есть ли ключ k, и если есть, то вернуть ссылку на этот узел.

Алгоритм:

Если дерево пусто -> сообщить что узел не найден и остановиться.

Иначе сравнить k со значением ключа корня дерева T root_key

- Если k == root_key вернуть ссылку на к корень T
- Если k < root_key рекурсивно продолжить поиск в левом поддереве
- Если k > root_key рекурсивно продолжить поиск в правом поддереве

Вставка элемента

Вставка

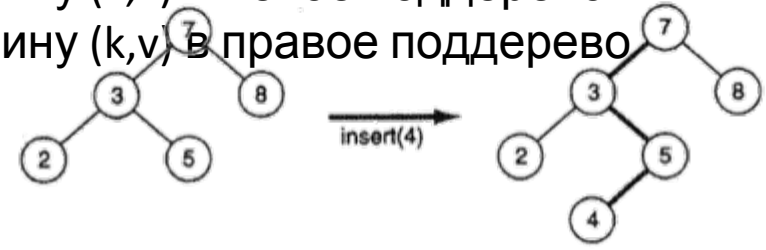
Задача в заданное двоичное дерево поиска T вставить новый узел с ключом k и значение v

Алгоритм:

Если дерево пусто -> создать корневой узел в дереве T с ключом k и значением v.

Иначе сравнить k со значением ключа корня дерева T root_key

- Если k == root_key -> заменить значение v в корне дерева T
- Если k < root_key циклически добавить вершину (k,v) в левое поддерево
- Если k > root_key циклически добавить вершину (k,v) в правое поддерево



Удаление

Удаление

Задача в заданное двоичное дерево поиска T удалить узел со значение ключа k (если такой есть).

Алгоритм:

Если дерево пусто -> вывести предупреждение и выйти из процедуры.

Иначе сравнить k со значением ключа корня дерева T root_key

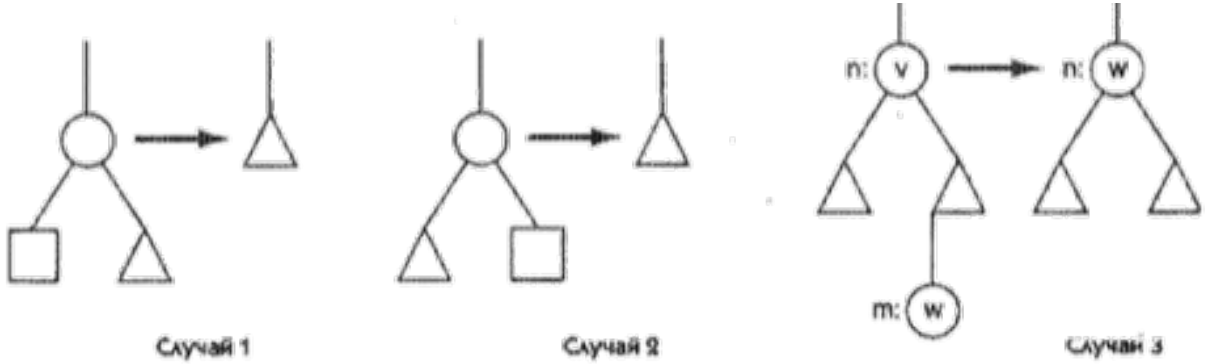
- Если k < root_key циклически удалить вершину с ключем k в левом поддереве
- Если k > root_key циклически удалить вершину с ключем k в правом поддереве

Удаление

Удаление

- Если k == root_key:
  - Узел n имеет пустой левый потомок. В этом случае ссылка на n (записанная в предке n, если он есть) заменяется на ссылку на правого потомка n.
  - У узла n есть непустой левый потомок, но правый потомок пустой. В этом случае ссылка вниз на n заменяется ссылкой на левый потомок узла n.
  - Узел n имеет два непустых потомка. Найдем последователя для n (назовем его m), копируем данные, хранящиеся в m, в узел n и затем рекурсивно удаляем узел m из дерева поиска.

Удаление



Случай 1 Случай 2 Случай 3

Оценка сложности

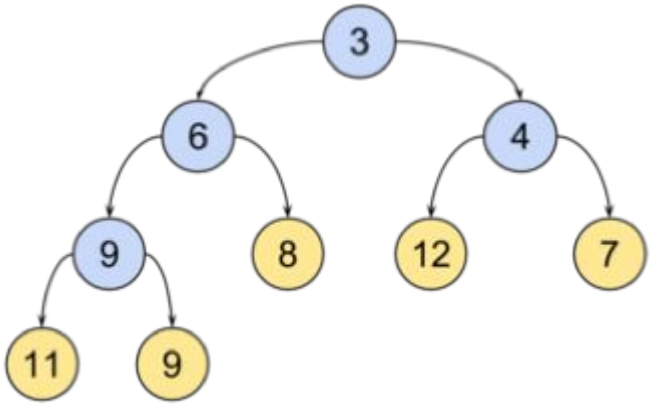
	В среднем	**В худшем случае**
Расход памяти	O(n)	O(n)
Поиск	O(logn)	O(n)
Вставка	O(logn)	O(n)
Удаление	O(logn)	O(n)

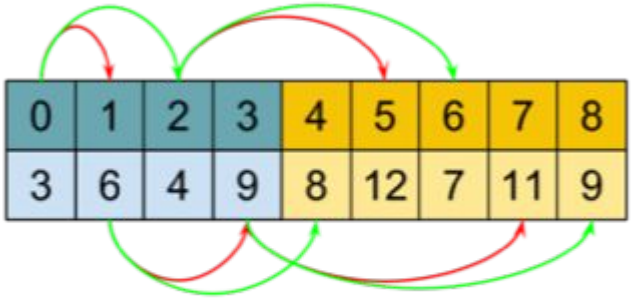
Двоичная куча

Двоичная куча – такое бинарное дерево, для которого выполняются следующие условия:

- Значение в любой вершине не меньше (если куча для максимума), чем значения её потомков.
- Глубина всех листьев (расстояние до корня) отличается не более чем на 1 слой.
- Последний слой заполняется слева направо без «дырок».

Двоичная куча





Двоичная куча

Если в куче изменяется один из элементов, то она может перестать удовлетворять свойству упорядоченности.

```
function siftDown(i : int):
    while 2 * i + 1 < a.heapSize     // heapSize – количество элементов в куче
        left = 2 * i + 1             // left – левый сын
        right = 2 * i + 2            // right – правый сын
        j = left
        if right < a.heapSize and a[right] < a[left]
            j = right
        if a[i] ⩽ a[j]
            break
        swap(a[i], a[j])
        i = j
```

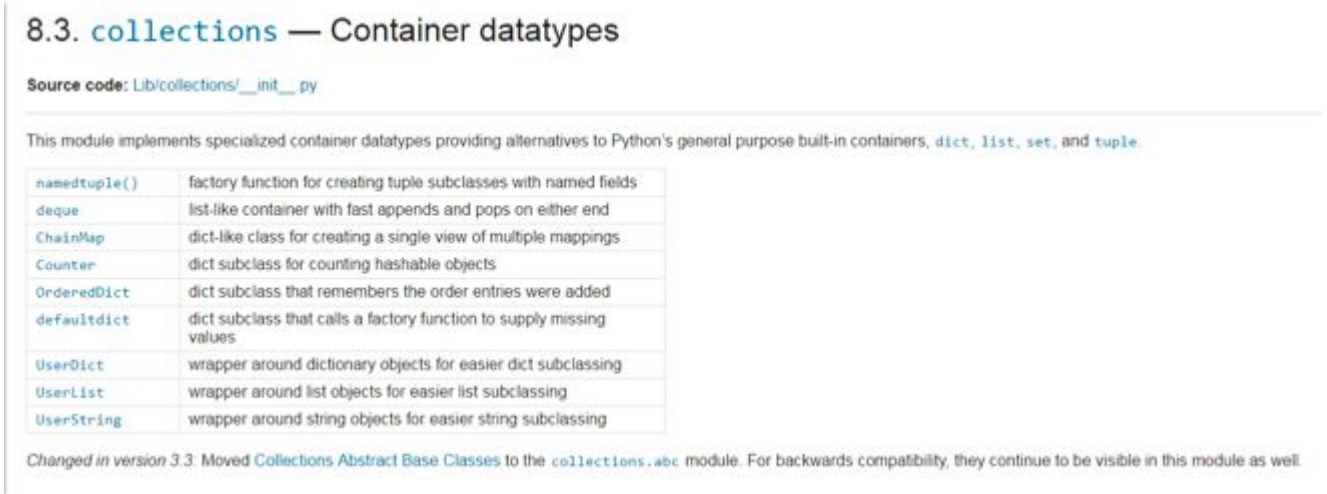
Двоичная куча

```
function siftUp(i : int):
    while a[i] < a[(i - 1) / 2]     // i == 0 – мы в корне
        swap(a[i], a[(i - 1) / 2])
        i = (i - 1) / 2
```

Двоичная куча

	В среднем
Расход памяти	O(n)
Восстановление свойств	O(logn)
Вставка	O(logn)
Извлечение минимального	O(logn)

Python & ADT



8.3. `collections` — Container datatypes

Source code: Lib/collections/__init__.py

This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, `dict`, `list`, `set`, and `tuple`.

`namedtuple()`	factory function for creating tuple subclasses with named fields
`deque`	list-like container with fast appends and pops on either end
`ChainMap`	dict-like class for creating a single view of multiple mappings
`Counter`	dict subclass for counting hashable objects
`OrderedDict`	dict subclass that remembers the order entries were added
`defaultdict`	dict subclass that calls a factory function to supply missing values
`UserDict`	wrapper around dictionary objects for easier dict subclassing
`UserList`	wrapper around list objects for easier list subclassing
`UserString`	wrapper around string objects for easier string subclassing

Changed in version 3.3: Moved Collections Abstract Base Classes to the `collections.abc` module. For backwards compatibility, they continue to be visible in this module as well.

https://docs.python.org/3.6/library/collections.html

Указания для задач

Для замера работы функции нужно использовать метод now() класса datetime модуля datetime

Указания для задач

```
array = [0] * N
array.insert(N,0)
```

```
import datetime


def main():
    t1 = datetime.datetime.now()
    #You'r code here
    ...
    print(datetime.datetime.now() - t1)

if __name__ == '__main__':
    main()
```

Указания для задач

```
import numpy as np


...
# Generate numpy Array with N random numbers
array = np.random.rand(N)
#Sort Array by quick sort
sorted(array)
...
```

Указания для задач

jupyter MyTests Last Checkpoint: a minute ago (unsaved changes)

File Edit View Insert Cell Kernel Help Python 2

Code CellToolbar

In [6]:
```
import datetime
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

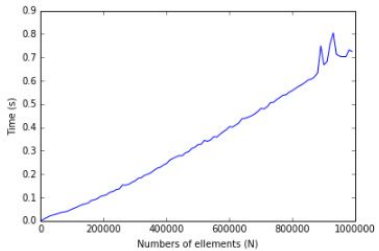
In [7]:
```
times = []
for N in xrange(1000, 1000000, 10000):
    array = np.random.rand(N)
    t1 = datetime.datetime.now()
    sorted(array)
    t_res = datetime.datetime.now() - t1
    times.append([N,t_res])
y = [t[1].total_seconds() for t in times]
x = [t[0] for t in times]
```

In [8]:
```
%matplotlib inline
```

In [9]:
```
plt.plot(x, y)

plt.xlabel('Numbers of ellements (N)')
plt.ylabel('Time (s)')

plt.savefig("test.png")
plt.show()
```



Задачи

- Реализовать алгоритм перемножения квадратных матриц. Матрицы могут задаваться как список списков. Считывать можно из файла потока ввода, или задавать случайным образом (используя функцию np.random.rand(N)). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Найти все пифагоровы тройки ($c^2 = a^2 + b^2$) для заданного интервала. Интервал задается парой чисел через пробел считанных из входного потока (например: 10 100) помните, что верхняя грань отрезка должна быть больше нижней. Если задано одно число, то считаем, что ограничение снизу равно по умолчанию 1. Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм факторизации числа (разложение числа как произведение двух других чисел). Оценить временную и ассимптотическую сложность алгоритма, построить график.
- Реализовать алгоритм рассчитывающий сочетания и размещения.
- Факториал довольно емкостная функция, при расчете которого для больших значений может случится переполнение (т.е. полученное число будет больше чем максимально возможное число в вашей системе). Подумайте как преодолеть эту проблему.

Задачи

Написать оболочку для работы с графами:

- создавать графы
- Выводить граф (в виде таблицы смежности)
- Удалять ребра
- Ищет путь в графе для заданных вершин
  - Флойда-Уоршела
  - Форда-Беллмана
  - Дейкстра

Задачи

1. Скачать файл https://goo.gl/z7H7DU
2. Файл содержит карту препятствия обозначены символом ‘%’ клетки, по которым можно передвигаться обозначены ‘-’, при этом каждая клетка по которой можно двигаться имеет вес 1.
3. Робот начинает движение в клетке обозначенной буквой ‘P’ и движется в клетку обозначенной буквой ‘T’.
4. Нужно рассчитать оптимальную траекторию пути робота с помощью алгоритма A*.
5. Выведите траекторию в отдельный файл.

Задачи

1. Реализуйте функцию DFS
2. С помощью вашей функции реализуйте алгоритм разбиение графа на компоненты связности.
3. Реализуйте алгоритм проверки орграфа на цикличность
4. Реализуйте алгоритм Крускала/Прима для поиска минимального остовного дерева взвешенного графа.

Задачи

1. Как можно объединить два списка. Напишите программу делающую это
2. Реализуйте очередь через два стека.

Спасибо за внимание!