Техническое задание

# Введение

Краткое описание задачи: существует иерархически организованная структура хранения статей затрат предприятия. Структура хранения статей заключается в представлении бинарного дерева поиска. Каждый узел такого дерева является статьей затрат предприятия и хранит в себе следующую информацию:

* Номер статьи – натуральное целое число;
* Ссылка на статью родителя (предок);
* Ссылка на статью с меньшим номером (слева);
* Ссылка на статью с большим номером (справа);

Необходимо придумать алгоритм поиска такой статьи, которая будет являться ближайшей к исходной и с большим номером, чем у исходной статьи.

Цель алгоритма: вернуть номер ближайшей статьи с большим номером, чем у исходной статьи.

Для выполнения дополнительного задания, связанного с написанием программного кода, воспользуемся языком программирования Python. Так как в условии задания не уточнено, как именно задается иерархия статей затрат – для лучшего представления и обеспечения возможности тестирования алгоритма определим, что такая структура задается в текстовом файле в виде натуральных чисел, разделенных пробелами.

# Описание алгоритма

Общая концепция работы алгоритма выглядит следующим образом:

1. Пользователь вводит число (далее – номер исходной статьи);
2. Алгоритм считывает заданное бинарное дерево поиска.
3. С помощью инфиксного (симметричного) обхода узлов в (левое поддерево → корень → правое поддерево) ищется минимальная (ближайшая к исходной) статья среди всех бо́льших, чем исходная.
4. Вернуть результат пользователю в виде натурального числа.
5. Если пользователь введет номер, к которому невозможно найти статью по заданным критериям, алгоритм вернет число *-1*. Это может произойти, когда пользователь задаст номер, больше которого в заданной иерархии статей не существует.

Замечание к третьему пункту: так как необходимо вернуть ближайшее бо́льшее – в качестве ответа достаточно вернуть первое найденное число.

# Входные и выходные данные, ограничения

В таблице 1 определим входные и выходные данные для алгоритма, а также их допустимые диапазоны.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Входные данные*** | | | |
| **Наименование** | **Тип** | **Диапазон** | **Примечание** |
| Номер статьи затрат | int |  | *n* – возможный номер статьи |
| Иерархия статей затрат | class Node |  | *s* – количество узлов в дереве |
| ***Выходные данные*** | | | |
| **Наименование** | **Тип** | **Диапазон** | **Примечание** |
| Номер статьи затрат | int |  | - |

Таблица 1. Входные и выходные данные алгоритма.

Ограничения по времени работы алгоритма: 4 секунды[[1]](#footnote-1). Ограничение по памяти: 300 Мб. Версия Python: 3.11.1.

# Логика и структура программы

Заметим специфику задачи в следующем: номера статей затрат должны быть уникальными (не должно существовать двух статей с одинаковым номером), так как одной статье должен соответствовать свой единственный уникальный номер, чтобы ее можно было отличить от остальных и найти в общей базе.

Структура программы должна быть разбита на следующие основные компоненты решения:

* Объект Node (узел), внутри которого должны храниться номер и три ссылки (родитель, левый потомок, правый потомок). Для удобства инициализируем магический метод \_\_str\_\_ для того, чтобы вывести дерево на экран при передаче экземпляра класса Node в функцию print;
* Объект InorderTraversal, предоставляющий возможность обхода бинарного дерева поиска в симметричном отсортированном порядке. В этом объекте будет три метода, один из которых при вызове будет запускать указанный выше обход дерева и возвращать либо найденный номер статьи, либо число *-1*, если статьи не нашлось;
* Функция build\_bst, которая создает бинарное дерево поиска (Binary Search Tree, BST) на основе списка уникальных чисел. Функция build\_bst принимает один аргумент numbers, который может быть либо списком, либо множеством целых чисел. Эти числа будут использованы для построения бинарного дерева.

# Тестирование и ожидаемые результаты

Определим базовый набор тестовых случаев для алгоритма в таблице 2.

|  |  |
| --- | --- |
| **Входные данные** | **Выходные данные** |
| 10 5 7 16 13 2 20  15 | 16 |
| 10 5 7 16 13 2 20  0 | 2 |
| 10 5 7 16 13 2 20  16 | 20 |
| 10 5 7 16 13 2 20  17 | 20 |
| 10 5 7 16 13 2 20  1 | 2 |
| 10 5 7 16 13 2 20  21 | -1 |
| 50 48 30 28 45 215 49 652 104 1 512  215 | 512 |
| 50 48 30 28 45 215 49 652 104 1 512  2 | 28 |
| 50 48 30 28 45 215 49 652 104 1 512  1 | 28 |
| 50 48 30 28 45 215 49 652 104 1 512  48 | 49 |
| 50 48 30 28 45 215 49 652 104 1 512  38 | 45 |
| 50 48 30 28 45 215 49 652 104 1 512  974 | -1 |

Таблица 2. Базовый набор тестовых случаев.

Кроме указанных тестов необходимо написать случайные тесты, а также тесты, затрагивающие краевые случаи работы программы исходя из диапазонов входных данных: 2000000 узлов в дереве со значениями от 1 до 10000000. Все тесты должны успешно проходиться и подчиняться установленным ограничениям.

# Дополнительные требования

Алгоритм должен работать одинаково на Windows, Linux и macOS (не зависимо от операционной системы, разрядности и остальных характеристик). Программа может использовать дополнительные зависимости. Программа может использовать модульную систему – в таком случае в репозитории должен быть файл «interface.py», из которого можно запустить программу.

# Ожидаемый результат

Должна быть разработана консольная программа, выполняющая поставленные задачи в рамках указанных ограничений по диапазону, времени работы и используемой памяти. Алгоритм должен удовлетворять дополнительным требованиям и работать с любыми входными данными в рамках поставленных ограничений.

Подготовил: Подстречный А. В.

Участник Case Lab SAP BI Express

10.07.2023

1. По условию задачи заявлено, что иерархическая структура статей затрат уже была представлена и загружена в программу. Поэтому считаем, что за указанное время должен совершаться только инфиксный обход бинарного дерева поиска, без учета времени чтения из файла, формирования структуры самого дерева и других подготовительных операций. [↑](#footnote-ref-1)