Смусь Елена Сергеевна
smus.elena.s@gmail.ru
https://github.com/Smusss
https://t.me/lilly_dallas

Техническое задание
на разработку программы для автоматизированного поиска статей затрат предприятия

1. Общие сведения

Предмет, наименование: **Программа для автоматизированного поиска статей затрат предприятия.**

Цель: реализация функции автоматизированного поиска статей с заданными характеристиками в общем перечне статей затрат, в рамках разработки нового блока отчетов отдела бухгалтерского и налогового учета.

Заказчик: Отдел бухгалтерского и налогового учета, ответственный сотрудник: аналитик Смусь Е.С., моб. тел., email.

Родительский проект, задача: Разработка нового блока отчетов отдела бухгалтерского и налогового учета.

2. Сроки реализации

Желаемый срок реализации – 30.08.2023

Приоритет работы – средний.

3. Требования

Название программы: «Поиск большей статьи расходов»

Назначение: реализация функции автоматизированного поиска статей с бóльшим номером в общем перечне статей затрат, в рамках разработки нового блока отчетов отдела бухгалтерского и налогового учета.

Пользователи: отдел бухгалтерского и налогового учета (бухгалтер - просмотр).

Программа будет использоваться отделом бухгалтерского и налогового учета в рамках формирования отчетов / будет использована в составе комплексного ПО для формирования отчетов.

«Я, как бухгалтер, хочу найти и посмотреть статью, с целью сформировать отчет.»

Доступ к программе имеют все пользователи, имеющие доступ к перечню статей расходов.

Источники данных: SAP (справочники статей 20, 25,26 …. счетов) / внутренние базы – в части списка статей затрат.

Описание задачи:

Статьи затрат пронумерованы (числовые значения) и организованы в иерархию.

У каждой из статей затрат есть:

- ✓ ее номер;
- ✓ ссылка на статью-родителя;
- ✓ две ссылки на статей-потомков. При этом в левой ссылке находятся статьи с меньшим номером, а в правом - с большим.

Алгоритм, при вводе произвольной статьи затрат будет находить ближайшую к ней статью с бóльшим номером.

Логика работы программы:

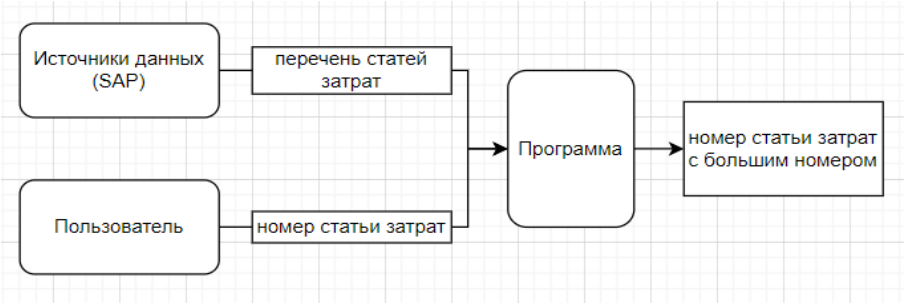

рис.1

Основной алгоритм:

Начало работы

1. Создание бинарного дерева, наполнение дерева значениями из перечня статей затрат в соответствии с иерархией. Загрузка из файла / БД / ручной ввод.

Методика загрузки данных зависит от их исходного вида.

2. Запрос на ввод пользователем произвольной статьи затрат для поиска – заданного значения (article).

3. Поиск элемента с заданным значением среди элементов дерева (отдельный метод).

По результатам ввода:

3.1 Если элемент с заданным значением не найден:

3.1.1 Вывод сообщения «Указанная статья не найдена»;

3.1.2 Предложение повторного ввода статьи: да – переход к пункту 2, нет - завершение работы.

3.2 Если элемент с заданным значением найден (node.data = article), начинаем поиск элемента, содержащего статью с бóльшим номером:

3.2.1 Проверка 1 условия поиска - левый потомок элемента дерева, содержащего заданное значение, существует? (node.left is not None)

А) Да – бóльшая статья равна значению в левом потомке элемента, содержащего заданное значение. Завершаем работу. (result = node.left, result.data)

Б) Нет – переходим к следующему пункту.

3.2.2 Проверка 2 условия поиска - правый потомок элемента дерева, содержащего заданное значение существует? (node.right is not None)

А) Да - бóльшая статья равна значению в правом потомке элемента, содержащего заданное значение. Завершаем работу. (result = node.right, result.data)

Б) Нет – переходим к следующему пункту.

3.3.3 Проверка 3 условия поиска - у элемента, содержащего заданное значение, нет элемента-родителя? (None? Элемент — это корень?) (node.parent is None)

А) Да – вывод сообщения «Статьи с бóльшим номером не существует». Завершаем работу;

Б) Нет – присваиваем заданному значению значение родителя элемента, содержащего заданное значение. Переход к пункту 3.2.2. (node = node.parent)

Т.о. в следующем цикле проверки условий рассматриваем в качестве элемента с заданным значением элемент, который является элементом-родителем для элемента с заданным значением прошлого цикла.

Завершение работы.

Блок-схема алгоритма:

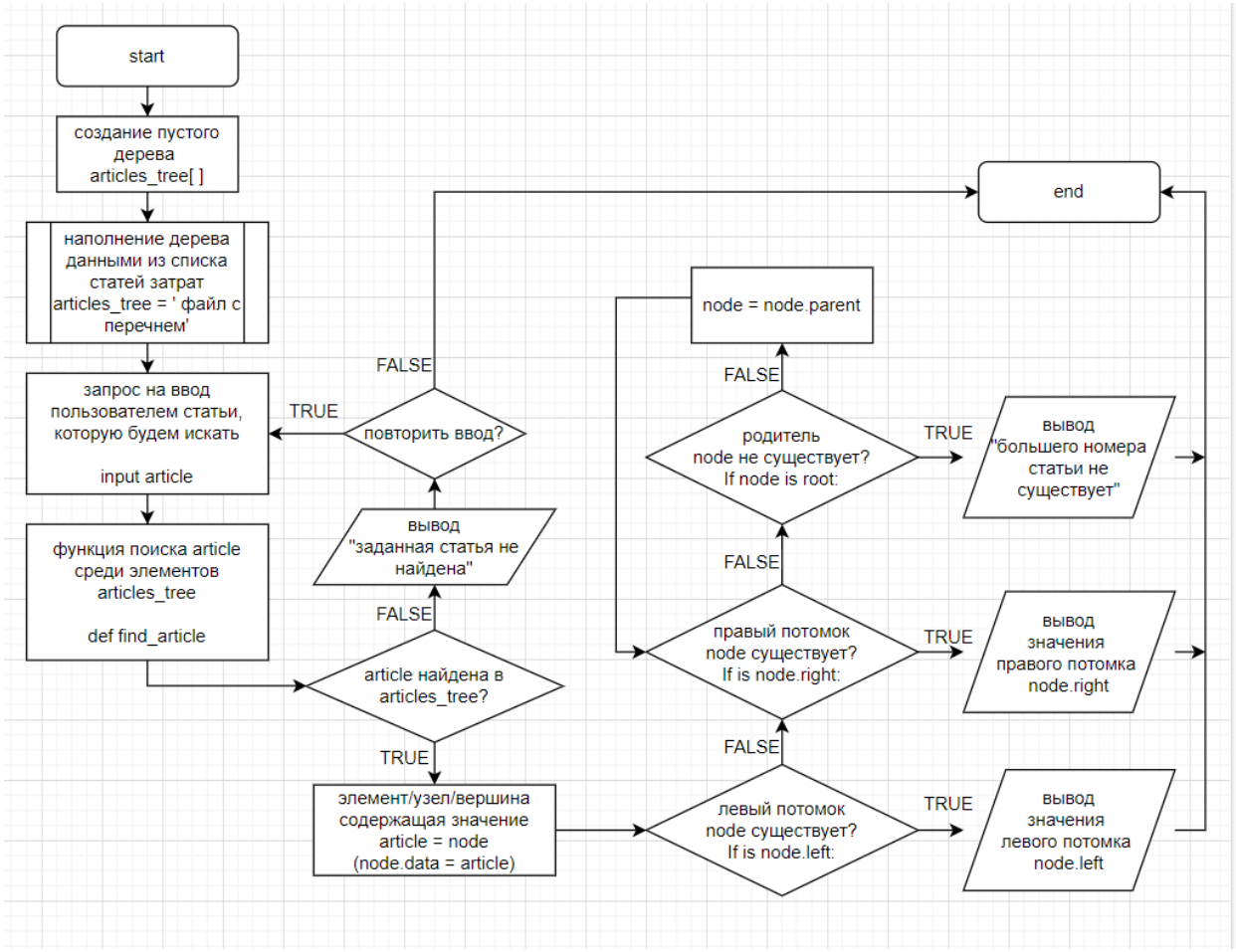

рис.2

Псевдо код:

```
Class Tree (root)                                   #класс Tree с параметром root
Class Node (data, left, right, parent)              #класс Node (вершины/узлы) с параметрами left, right(потомки), parent (родитель), data - значение

                                                    #поиск параметра parent можно реализовать как метод
article_tree = Tree()                               #создаем дерево
sap_list = read_excel('article_list.xls')           #условно наполняем из файла - зависит от формата ИД
for x in sap_list:
    atrical_tree.append(Node(x))

while True:                                         #цикл проверки повторного ввода значения от пользователя (не учитыв. некорректный ввод)
    answer = 1
    article = int(input(print('Введите номер статьи: ')))     #запрос ввода статьи от пользователя
    result, node = None, None                       #result - результат работы (значение большей статьи), node - элемент(вершина) с которой работаем

    for item in article_tree:                       #поиск узла с нужным номером статьи
        if item.data == article:
            node = item

    if node is not None:                            #если узел найден:
        if node.left is not None:                  #проверка наличия левого потомка
            result = node.left                     #если существует - результат = левому потомку
            print(f'Большая статья: {result.data}.')  #вывод значения в вершине-результате
        while result is None:
            if node.right is not None:             #если левый не существует - проверка наличия правого потомка
                result = node.right                #если существует - результат = правому потомку
                print(f'Большая статья: {result.data}.')  #вывод значения в вершине-результате
            elif node.parent is None:              #если левый и правый не существуют - проверка наличия элемента-родителя
                print(f'Большей статьи не существует.') #если родителя нет - элемент является корнем, большей статьи нет
                break                              #выход из цикла проверки
            else:                                  #элемент найден, родитель есть, потомков нет
                node = node.parent                 #меняем элемент(вершину) с которой работаем на текущего родителя
    else:
        print('Указанная статья не найдена')       #номер статьи не найден в значениях вершин дерева
        answer = int(input('Для повторного ввода 0, для выхода 1: ')) #предложение повторного ввода / выхода
    if answer == 1:
        break
print(f'END')
```

рис.3

Тестовый пример (для случая если формат статей задан в виде пунктов и подпунктов):

Дан перечень статей (только коды):

1.
1.1
1.1.1
1.1.1.1
1.1.1.2
1.1.2
1.2
1.2.1
1.2.1.2
1.2.1.2.1
1.2.2
…

При загрузке в дерево с учетом иерархии получим:

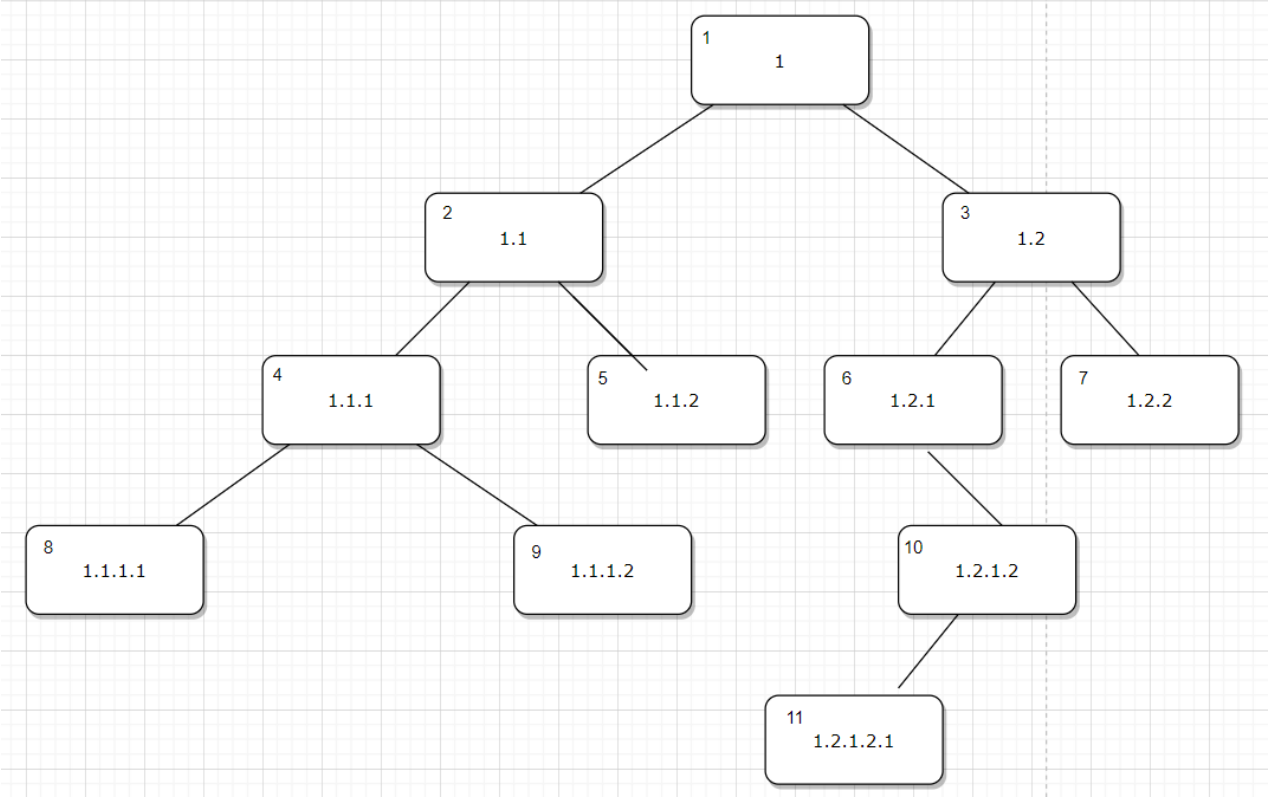

рис.4

Для удобства вершины пронумерованы.

⇨ Пользователь ввел номер 1.2.1.2.1.

Значение соответствует вершине № 11 – это рассматриваемая вершина.

Проверяем:

Есть левый потомок? Нет.

Есть правый потомок? Нет.

Есть родитель? Да, родитель – вершина №10, значит рассматриваемая вершина меняется на №10, проверки начиная с правого потомка запускаются повторно.

Есть правый потомок? Нет.

Есть родитель? Да, родитель – вершина №6, значит рассматриваемая вершина меняется на №6, проверки начиная с правого потомка запускаются повторно.

Есть правый потомок? Нет.

Есть родитель? Да, родитель – вершина №3, значит рассматриваемая вершина меняется на №3, проверки начиная с правого потомка запускаются повторно.

Есть правый потомок? Да – вершина № 7. Result = вершина №7, result.data = 1.2.2

<= Ответ: для статьи 1.2.1.2.1 большей статьей является статья номер 1.2.2

⇨ Пользователь ввел номер 1

Значение соответствует вершине № 1 – это рассматриваемая вершина.

Проверяем:

Есть левый потомок? Да. – вершина № 2. Result = вершина №2, result.data = 1.1

<= Ответ: для статьи 1 большей статьей является статья номер 1.1