МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КУБГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**Отчет**

**по индивидуальному заданию №1**

**«ДЕРЕВЬЯ»**

**по курсу**

**«ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ»**

Работу выполнил

Студент 36 группы

Корнилов К.А.

Преподаватель:

Лапина О.Н.

**Рекурсивная процедура, подсчитывающая количество вершин в дереве на заданном уровне**

Метод возвращает искомое число.

Оценка сложности алгоритма:

Пространственная сложность алгоритма совпадает с количество узлов в дереве – это V.

Оценим временную сложность алгоритма. Пусть – временная сложность алгоритма.

На вход поступает корень. Мы проверяем на удовлетворения условия по номеру уровня. Если его номер уровня совпадает с заданным, то возвращаем значение 1. В противном случае проверяем не являются ли пустыми его дети. Если это так, то для каждого ребенка вызываем метод от текущий номер уровня + 1.

Точную оценку сложности алгоритма для произвольного бинарного дерева дать сложно, поэтому будут даны точные оценка для двух граничных случаев: идеально сбалансированное дерево и полностью вырожденное дерево.

Код методы имеет следующий вид:

def countOfNodesOnLevel(self,currentLevel,K):  
 if currentLevel==K:  
 return 1  
 else:  
 count = 0  
 if self.leftNode is not None:  
 count = count + self.leftNode.countOfNodesOnLevel(currentLevel+1,K)  
 if self.rightNode is not None:  
 count = count + self.rightNode.countOfNodesOnLevel(currentLevel+1,K)  
 return count

def countOnLevelCall(self,K):  
 return self.countOfNodesOnLevel(1,K)

Тогда точная оценка рекурсивной процедуры для вырожденного дерева имеет следующий вид:

,

где – оценка для случая, когда требуемый уровень не существует в дереве в принципе; *–* оценка для случая, когда требуемый уровень был достигнут.

В виде не рекурсивной формулы данная оценка имеет следующий вид:

Поскольку процедура начинает всегда работу С = 1, то данные формулу не рекурсивную можно записать в следующем виде:

Сделаем аналогичный расчет для идеально сбалансированного дерева.

В виде рекурсивной формулы оценка алгоритма имеет следующий вид:

,

где – оценка для случая, когда требуемый уровень не существует в дереве в принципе; *–* оценка для случая, когда требуемый уровень был достигнут.

В виде не рекурсивной формулы данная оценка имеет следующий вид:

Поскольку процедура начинает всегда работу С = 1, то данные формулу не рекурсивную можно записать в следующем виде:

Тогда итоговая оценка находится в промежутке:

**Не Рекурсивная процедура, подсчитывающая количество вершин в дереве на заданном уровне**

Метод возвращает искомое число.

Оценка сложности алгоритма:

Пространственная сложность алгоритма совпадает с количество узлов в дереве – это V.

Оценим временную сложность алгоритма. Пусть – временная сложность алгоритма.

Первая операция – внесения корня в очередь: 1.

Следующим идет цикл while для каждой из вершин проверяет ее значение уровня, если оно совпадает, то значение счетчика увеличивается на 1. В противном случае, добавляются ее потомки в конец список. Сама обработанная вершина при этом извлекается из списка.

Точную оценку сложности алгоритма для произвольного бинарного дерева дать сложно, поэтому будут даны точные оценка для двух граничных случаев: идеально сбалансированное дерево и полностью вырожденное дерево.

Код метода имеет следующий вид:

def countOnLevelNonRec(self,K):  
 count=0  
 nodes=[self]  
 while nodes:  
 if nodes[0].level==K:  
 count = count + 1  
 else:  
 if nodes[0].leftNode is not None:  
 nodes.append(nodes[0].leftNode)  
 if nodes[0].rightNode is not None:  
 nodes.append(nodes[0].rightNode)  
 nodes.pop(0)  
 return count

Тогда для вырожденного дерева оценка имеет следующий вид:

Сделаем аналогичный расчет для идеально сбалансированного дерева.

Тогда для сбалансированного дерева оценка имеет следующий вид:

Тогда итоговая оценка находится в промежутке:

**Результаты работы программы:**

На рисунке 1 представлен результат работы программы – подсчет количество узлов на определенном уровне.

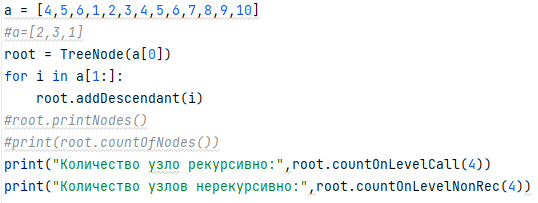




Рисунок 1 – Работа программы

Результаты изменений количества операций представлены в таблице 1.

Таблица №1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество | Рекурсия сбалансированная | Рекурсия вырожденная | Итерация сбалансированная | Итерация вырожденная |
| 1 | 2 | 2 | 10 | 10 |
| 2 | 15 | 10 | 28 | 20 |
| 3 | 41 | 18 | 64 | 30 |
| 4 | 93 | 26 | 136 | 40 |
| 5 | 197 | 34 | 280 | 50 |

**Вывод:** Была проведена оценка рекурсивного и не рекурсивного алгоритма подсчета количества узлов на заданном уровне. Были выведены формулы, ограничивающие оценку алгоритма для произвольного дерева в случае существования заданного уровня и в случае его отсутствия.

**ПРИЛОЖЕНИЕ**

**Приложение 1. Код программы**

class TreeNode:  
 def \_\_init\_\_(self, value,level=1):  
 self.leftNode = None  
 self.rightNode = None  
 self.value = value  
 self.level= level  
 def addDescendant(self,value):  
 if value <= self.value:  
 self.addleftNode(value)  
 else:  
 self.addRightNode(value)  
  
 def addleftNode(self, value):  
 if self.leftNode is None:  
 self.leftNode = TreeNode(value,level=self.level+1)  
 else:  
 self.leftNode.addDescendant(value)  
  
 def addRightNode(self, value):  
 if self.rightNode is None:  
 self.rightNode = TreeNode(value,level=self.level+1)  
 else:  
 self.rightNode.addDescendant(value)  
 def printNodes(self):  
 print(self.value)  
 if self.leftNode is not None:  
 print('left of ',self.value,self.leftNode.level)  
 self.leftNode.printNodes()  
 if self.rightNode is not None:  
 print('right of ',self.value,self.rightNode.level)  
 self.rightNode.printNodes()  
  
 def countOfNodes(self):  
 count = 1  
 if self.leftNode is not None:  
 count = count + self.leftNode.countOfNodes()  
 if self.rightNode is not None:  
 count = count + self.rightNode.countOfNodes()  
 return count  
 def countOfNodesOnLevel(self,currentLevel,K):  
 if currentLevel==K:  
 return 1  
 else:  
 count = 0  
 if self.leftNode is not None:  
 count = count + self.leftNode.countOfNodesOnLevel(currentLevel+1,K)  
 if self.rightNode is not None:  
 count = count + self.rightNode.countOfNodesOnLevel(currentLevel+1,K)  
 return count  
 def countOnLevelCall(self,K):  
 return self.countOfNodesOnLevel(1,K)  
  
 def countOnLevelNonRec(self,K):  
 count=0  
 nodes=[self]  
 while nodes:  
 if nodes[0].level==K:  
 count = count + 1  
 else:  
 if nodes[0].leftNode is not None:  
 nodes.append(nodes[0].leftNode)  
 if nodes[0].rightNode is not None:  
 nodes.append(nodes[0].rightNode)  
 nodes.pop(0)  
 return count  
  
a = [4,5,6,1,2,3,4,5,6,7,8,9,10]  
*#a=[2,3,1]*root = TreeNode(a[0])  
for i in a[1:]:  
 root.addDescendant(i)  
*#root.printNodes()  
#print(root.countOfNodes())*print("Количество узло рекурсивно:",root.countOnLevelCall(4))  
print("Количество узлов нерекурсивно:",root.countOnLevelNonRec(4))