ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

Кафедра ВС

Отчет по лабораторной работе №3

Выполнил:

Студент группы МГ-171

Бочкарев Б.В.

Проверил:

д.т.н., профессор

Родионов А.С.

Новосибирск 2018

Постановка задачи

Необходимо реализовать генерацию бинарного дерева 2 способами:

1) На вход подается N - количество элементов на одном ветке дерева и M - количество элементов, которое нужно сохранить в дереве. Необходимо распределять элементы равномерно по дереву, чтобы на каждом узле было N потомков. Глубина дерева не ограничена.

2) На вход подается D - глубина дерева и M - количество элементов в дереве. Необходимо разместить элементы так, чтобы сначала дерево заполнилось в глубину, затем элементы распределялись равномерно по дереву. Количество на одной ветке не ограниченно.

Результат

Результат работы программы для дерева с ограниченным количеством уровней до 4 представлен на рисунке 1

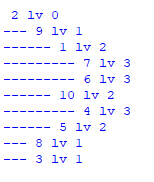


Рисунок 1 – дерево с ограниченным количеством уровней

Результат работы программы для дерева с ограничение на количества предков числом два представлен на рисунке 2.

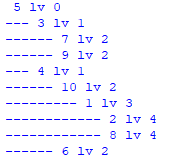


Рисунок 2 – дерево с ограниченным количеством предков

Листинг программы

import random as r

class Node(object):

"""Узел дерева

root - номер узла

level - уровень в дереве

children - спиоск дочерних узлов"""

def \_\_init\_\_(self, root, level):

self.root = root

self.level = level

self.children = []

def \_\_str\_\_(self):

tmp = str(self.root)

return tmp

def countCh(self):

return len(self.children)

class Tree(object):

"""Описание дерева

node - корень

children - дочернии узлы

"""

def \_\_init\_\_(self, node, root=None):

if root is None:

self.node = Node(node,0)

else:

self.node = Node(node,root)

def \_\_repr\_\_(self):

return str(self.node.root)

def \_\_str\_\_(self):

return str(self.node.root)

def addChild(self, root, node):

tmp = self.find(root)

if tmp is not None:

print("addChild", tmp, type(tmp), "add", node)

new = Tree(node,tmp.level+1)

tmp.children.append(new)

def printTree(self, root, lv = 0):

tmp = self.node

print("---"\*lv, tmp.root, "lv", tmp.level)

if tmp.children is not []:

for i in tmp.children:

i.printTree(i, lv+1)

def find(self, root):

tmp = self.node

print("find", root, type(root), "->", tmp)

if int(str(root),10) == int(str(tmp.root),10):

return tmp

else:

print("find rek in", tmp.children)

for i in tmp.children:

r = i.find(root)

if r is not None:

return r

else:

continue

def countChild(self, root=None):

tmp = self.node

if root is None:

print("Vertex",str(tmp.root),"have",tmp.countCh(),"child")

if tmp.children is not []:

for i in tmp.children:

i.countChild()

if root is not None:

if str(tmp.root) == str(root):

return tmp.countCh()

else:

if tmp.children is not []:

res = 0

for i in tmp.children:

res += i.countChild(root)

return res

def randTree(n, countNode = None, countLevel = None):

"""Случайное дерево

Аргументы:

n - количество элементов дерева

countNode - ограничение на количество дочерних

countLevel - ограничение на количество уровней

Возвращает:

head - голова дерева

tr - случайное дерево

"""

left = list(range(1,n+1))

right = []

head = r.choice(left)

tr = Tree(head)

right.append(left.pop(left.index(head)))

#print('left =',left, ' right =',right,'\n')

while left:

print("Подготовка")

tmp = r.choice(left)

ind = r.choice(right)

if (countNode is None or tr.countChild(ind) < countNode) \

and (countLevel is None or tr.find(ind).level < countLevel):

print("lv",tr.find(ind).level)

right.append(left.pop(left.index(tmp)))

else:

print("right",right)

right.pop(right.index(ind))

continue

print("Добавление")

print("add vertex", ind, tmp)

tr.addChild(ind, tmp)

print()

tr.printTree(head)

print()

#print('left =',left, ' right =',right,'ind =',ind,'\n')

#tr.printTree (head)

return head,tr

head, tr = randTree(10, countLevel = 3)

tr.printTree (head)

print()