**ДНІПРОВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ОЛЕСЯ ГОНЧАРА**

Факультет прикладної математики

Кафедра обчислювальної математики та математичної кібернетики

ЗВІТ

про виконання лабораторних робіт №1, 2

з дисципліни «Спеціальні задачі дискретної оптимізації»

Виконав:

студент групи ПА-18-1

Щербак Роман

Викладач:

канд. фіз.-мат. наук, доцент

В. А. Турчина

Дніпро

2022

1. **Постановка задачі**
2. Створити програму побудови впорядкувань та для орієнтованого графу;
3. Створити програму побудови впорядкування ширини для дерева.
4. **Опис розв’язку**

Створено програму мовою Python, що дозволяє вибирати бажану операцію та генерує відповідні вхідні дані. У випадку роботи з графом просить ввести кількість вершин графу, виводить утворений граф на екран, показує утворені зв’язки та результати побудови впорядкувань та . Якщо робота відбувається з деревом, просить ввести кількість рівнів дерева, кількість максимальних дочірніх вершин та ширину впорядкування .

1. **Код програми мовою Python**

* Модуль main.py

import choosing\_ordering as co

operation\_chosen = 1

while operation\_chosen == 1 or operation\_chosen == 2:

operation\_chosen = int(input("Оберіть задачу (1 – побудова упорядкувань для графу, 2 – побудова впорядкування для дерева): "))

if operation\_chosen == 1:

print()

quantity = int(input("Введіть кількість вершин графу: "))

print()

s\_upper, s\_lower = co.graph\_ordering(quantity)

print("Упорядкування S верхнє:")

print(s\_upper)

print()

print("Упорядкування S нижнє:")

print(s\_lower)

print()

if operation\_chosen == 2:

levels = int(input("Введіть кількість рівнів дерева: "))

max\_child\_vert = int(input("Введіть максимальну кількість дочірніх вершин: "))

h = int(input("Введіть ширину впорядкування: "))

print()

tree\_order = co.tree\_ordering(levels, h, max\_child\_vert)

print(f"Упорядкування дерева ширини {h}:")

print(tree\_order)

print()

print("Дякуємо за роботу")

* Модуль choosing\_ordering

import graph as g

import S\_algorithms as s

import tree as t

import tree\_order\_algorithm as tro

def graph\_ordering(num\_vert):

vert\_arr, connections = g.build\_graph(num\_vert)

g.show\_graph(connections)

s\_upper\_res = s.s\_upper(vert\_arr)

s\_lower\_res = s.s\_lower(vert\_arr)

return s\_upper\_res, s\_lower\_res

def tree\_ordering(num\_levels, width, num\_child\_vert):

tree\_created = t.build\_tree(num\_levels, num\_child\_vert)

tree\_order\_res = tro.tree\_order(tree\_created, width)

return tree\_order\_res

* Модуль vertex\_class

class Vertex:

index = 0

def \_\_init\_\_(self, i, arr\_out, arr\_in):

self.index = i

self.connected\_out = arr\_out

self.connected\_in = arr\_in

self.connected\_out\_copy = arr\_out

self.connected\_in\_copy = arr\_in

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

return f"{self.index}"

def connect\_vertex(self, vert):

self.connected\_out.append(vert)

vert.connected\_in.append(self)

def get\_connections\_out(self):

return self.connected\_out

def get\_connections\_in(self):

return self.connected\_in

def count\_connections\_in\_arr(self, vert\_arr):

connections\_arr = self.get\_connections\_in()

return self.count\_connections(connections\_arr, vert\_arr)

def count\_connections\_out\_arr(self, vert\_arr):

connections\_arr = self.get\_connections\_out()

return self.count\_connections(connections\_arr, vert\_arr)

def count\_connections(self, conn\_arr, vert\_arr):

count = 0

for v in conn\_arr:

if type(vert\_arr[0]) == Vertex:

if v in vert\_arr:

count += 1

else:

i = 0

while i < len(vert\_arr):

if v in vert\_arr[i]:

count += 1

i += 1

return count

def get\_index(self):

return self.index

def check\_already\_connected(self, vert):

if vert in self.get\_connections\_out():

return True

else:

return False

def check\_backward\_connection(self, vert):

if vert in self.get\_connections\_in():

return True

else:

return False

def add\_mark(self, num):

self.mark = num

def check\_mark\_exists(self):

try:

return True

except AttributeError:

return False

def \_\_lt\_\_(self, other):

return (self.index < other.index)

* Модуль graph

import random as r

import vertex\_class as vc

import networkx as nx

import matplotlib.pyplot as plt

def build\_graph(quant):

vert\_arr = []

i = 1

while i <= quant:

vert\_arr.append(vc.Vertex(i, [], []))

i += 1

i = 0

while i < quant:

connected\_vertices = r.sample(vert\_arr, 2)

if not(connected\_vertices[0].check\_already\_connected(connected\_vertices[1])) and not(connected\_vertices[0].check\_backward\_connection(connected\_vertices[1])):

connected\_vertices[0].connect\_vertex(connected\_vertices[1])

i += 1

for v in vert\_arr:

if len(v.get\_connections\_in()) == 0 and len(v.get\_connections\_out()) == 0:

num = r.randint(0, 1)

v\_1 = r.sample(vert\_arr, 1)[0]

while v\_1.get\_index() == v.get\_index():

v\_1 = r.sample(vert\_arr, 1)[0]

if num == 0:

v.connect\_vertex(v\_1)

else:

v\_1.connect\_vertex(v)

connections = []

for v in vert\_arr:

connected\_arr = v.get\_connections\_out()

if len(connected\_arr) > 0:

for t in connected\_arr:

connections.append((v, t))

print("Зв'язки між вершинами: ")

print(connections)

print()

return vert\_arr, connections

def show\_graph(connections):

G = nx.DiGraph()

G.add\_edges\_from(connections)

pos = nx.spring\_layout(G)

nx.draw\_networkx\_nodes(G, pos, cmap=plt.get\_cmap('jet'), node\_size = 250)

nx.draw\_networkx\_labels(G, pos)

nx.draw\_networkx\_edges(G, pos, arrows = True)

plt.show()

* Модуль S\_algorithms

def s\_upper(vert\_arr):

res\_arr = []

vert\_arr\_copy = vert\_arr.copy()

res\_arr = build\_s\_upper(vert\_arr\_copy, res\_arr)

return res\_arr

def build\_s\_upper(vert\_arr\_copy, res\_arr):

temp\_arr = []

for v in vert\_arr\_copy:

if v.count\_connections\_out\_arr(vert\_arr\_copy) == 0:

temp\_arr.append(v)

for v in temp\_arr:

vert\_arr\_copy.remove(v)

res\_arr.insert(0, temp\_arr)

if len(vert\_arr\_copy) == 0:

return res\_arr

else:

res\_arr = build\_s\_upper(vert\_arr\_copy, res\_arr)

return res\_arr

def s\_lower(vert\_arr):

res\_arr = []

vert\_arr\_copy = vert\_arr.copy()

res\_arr = build\_s\_lower(vert\_arr\_copy, res\_arr)

return res\_arr

def build\_s\_lower(vert\_arr\_copy, res\_arr):

temp\_arr = []

for v in vert\_arr\_copy:

if v.count\_connections\_in\_arr(vert\_arr\_copy) == 0:

temp\_arr.append(v)

for v in temp\_arr:

vert\_arr\_copy.remove(v)

res\_arr.append(temp\_arr)

if len(vert\_arr\_copy) == 0:

return res\_arr

else:

res\_arr = build\_s\_lower(vert\_arr\_copy, res\_arr)

return res\_arr

* Модуль tree

import random as r

import vertex\_class as vc

import treelib as tl

def build\_tree(levels, num\_child\_vert):

vert\_arr = []

root\_vertex = vc.Vertex(1, [], [])

vert\_arr.append([root\_vertex])

tree = tl.Tree()

tree.create\_node(root\_vertex, root\_vertex)

created\_vertices = 1

i = 0

while i < levels-1:

vert\_arr\_temp = []

while len(vert\_arr\_temp) == 0:

vert\_arr\_temp, created\_vertices, tree = create\_level(created\_vertices, vert\_arr[i], tree, num\_child\_vert)

vert\_arr.append(vert\_arr\_temp)

i += 1

tree.show()

return vert\_arr

def create\_level(vert\_count, vertices\_prev\_level, tree, num\_child\_vert):

i = 0

vert\_arr\_temp = []

while i < len(vertices\_prev\_level):

new\_vertices = r.randint(0, num\_child\_vert)

j = 0

while j < new\_vertices:

new\_vert = vc.Vertex(vert\_count+j+1, [], [])

vert\_arr\_temp.append(new\_vert)

vertices\_prev\_level[i].connect\_vertex(new\_vert)

tree.create\_node(new\_vert, new\_vert, parent = vertices\_prev\_level[i])

j += 1

vert\_count += new\_vertices

i += 1

return vert\_arr\_temp, vert\_count, tree

* Модуль tree\_order\_algorithm

import copy as c

def tree\_order(tree, h):

tree\_for\_ordering = c.deepcopy(tree)

res\_arr = build\_ordering(tree\_for\_ordering, h, [])

return res\_arr

def build\_ordering(tree, h, res\_arr):

temp\_arr = []

tree\_len = 0

for t in tree:

for v in t:

if v.count\_connections\_in\_arr(tree) == 0:

temp\_arr.append(v)

tree\_len += 1

while len(temp\_arr) > h:

temp\_arr.remove(temp\_arr[0])

for v in temp\_arr:

for t in tree:

if v in t:

t.remove(v)

if len(temp\_arr) != 0:

res\_arr.append(temp\_arr[::-1])

if tree\_len == 0:

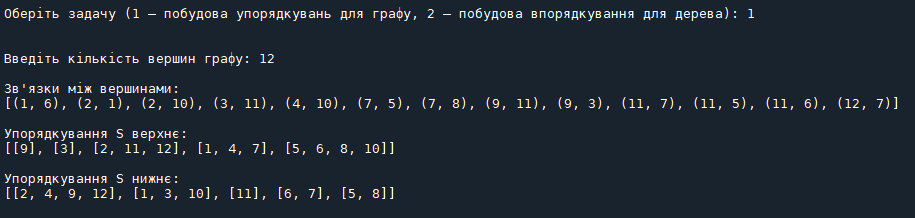
return res\_arr

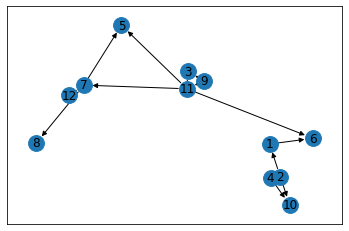
else:

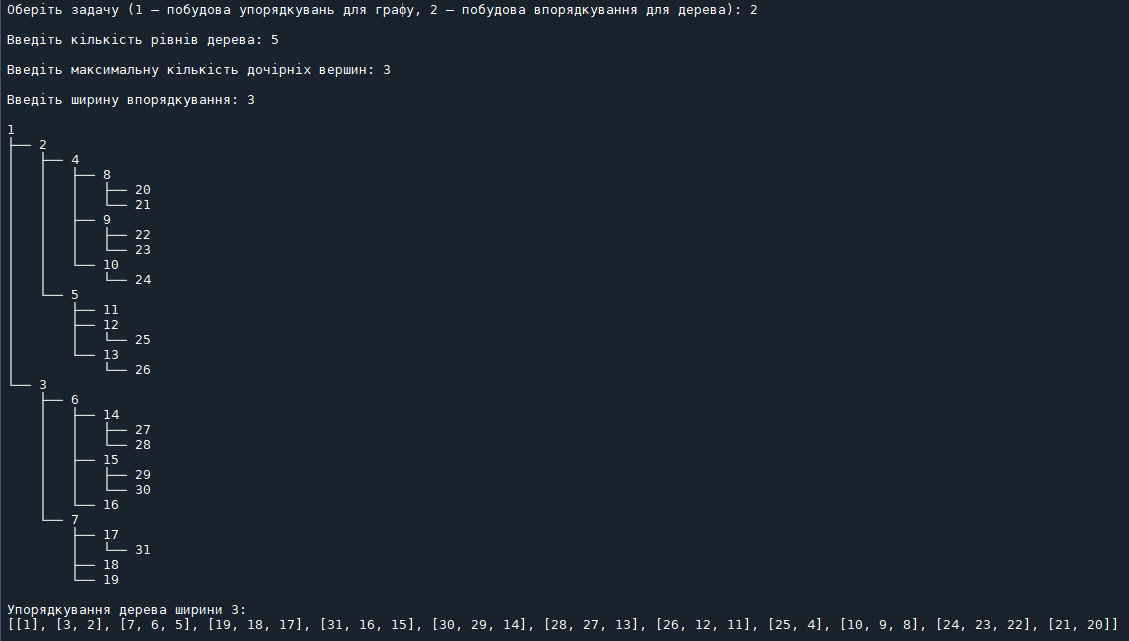
res\_arr = build\_ordering(tree, h, res\_arr)

return res\_arr

1. **Скриншоти роботи програми**

****





1. **Висновок**

Було створено та протестовано програмою мовою Python, що будує впорядкування графу та та впорядкування дерева ширини h.