Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

ЛАБОРАТОРНА РОБОТА №4
З ДИСЦИПЛІНИ “ ОРГАНІЗАЦІЯ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ”

Виконав:
Студент ІІІ курсу ФІОТ
групи ІО-82
Шендріков Євгеній

Перевірив:
Сімоненко В. П.

м. Київ – 2021 р.

Завдання

Написати програму, що реалізовує один з алгоритмів динамічного або статичного планування згідно варіанту.

Варіант

$$8227 \% 14 + 1 = 10$$

10. Скласти програму виділення вершин мають ознаку неявної транзитності (число вершин не менше 30).

Теоретичні відомості і опис алгоритму

Вихідною інформацією для структурного аналізу є граф задачі, представлений в ярусно-паралельній формі. Кожна вершина вихідного графа після аналізу маркується і їй присвоюється ознака "транзитності" у відповідності з такими визначеннями:

Резидентна вершина – вершина V^R належить до певного рівня U_K і переміщення її на інший рівень веде до зміни критичного шляху T_{CR}.

Транзитна вершина – вершина V^{tr} має свободу вибору рівня без зміни критичного шляху T_{CR}.

Транзитні вершини поділяються на явно транзитні, неявно транзитні та мультиплікативно транзитні.

«*Неявна транзитність*» – склеювання (кластеризація) вершин цієї категорії на одному рівні не збільшує критичний шлях. Вершина "неявної транзитності" має властивість:

Якщо $\forall\ x_i \in v_i \Leftrightarrow \exists\ e_i = (x_j, x_i) \in E_{i-1}\ /\ x_j \in v_{i-1};\ x_i \in v_i$, де

v_i – множина вершин рівня «i»;

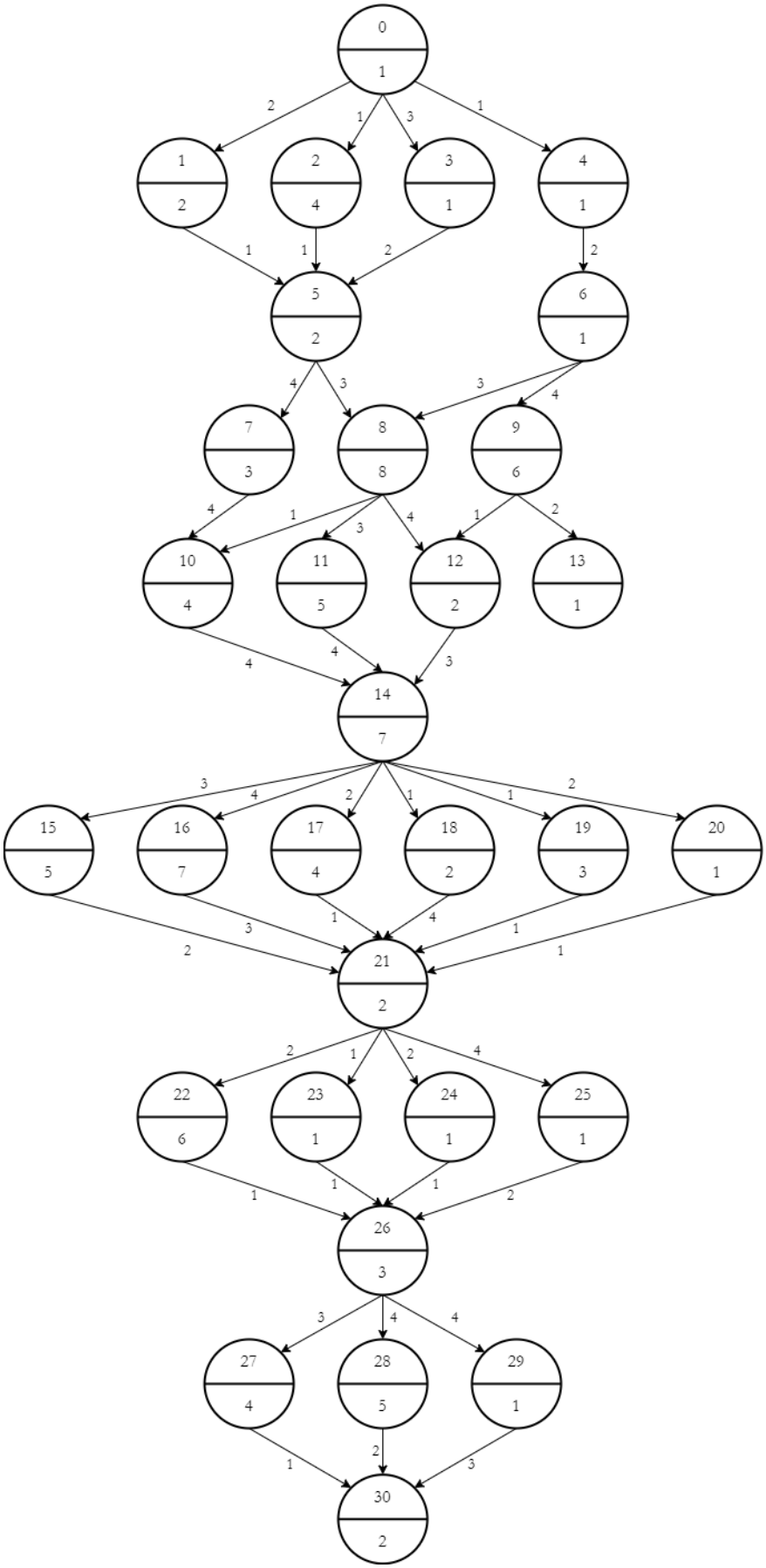
E_{i-1} – множина дуг між (v_{i-1}, v_i) / $\forall\ x_i, x_j \in v_i \Leftrightarrow T_{x_i} \geq T_{x_j}$, де

T – час розв'язання або вага вершини x.

Загальний алгоритм для рішення заданої задачі:

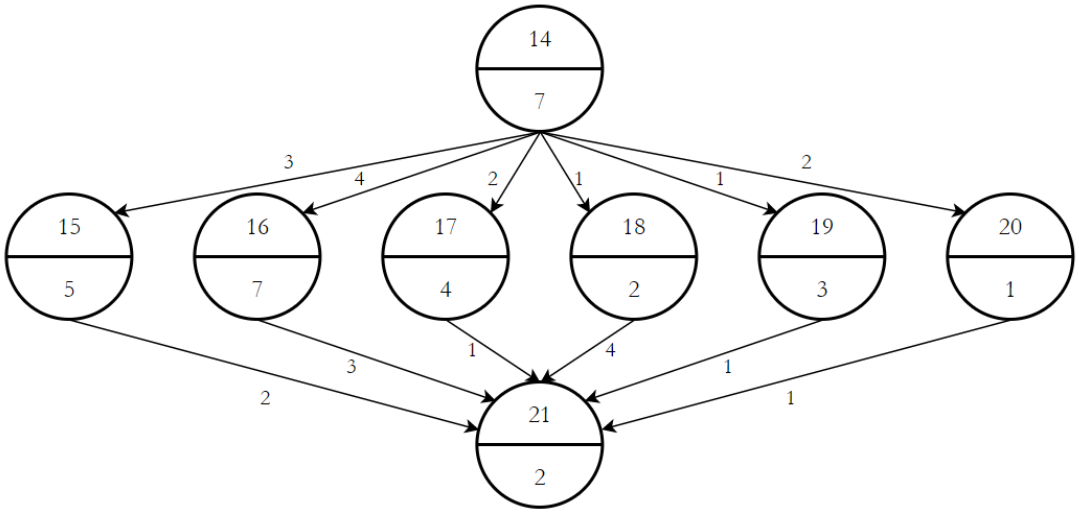
1. Виділити критичний шлях;
2. Кластеризувати неявні транзитні вершини. Це робиться для того, щоб групувати неявні транзитні вершини і таким чином зменшити ширину графа, а отже, кількість процесорів.

Для тестування програмної частини використовувався наступний граф:

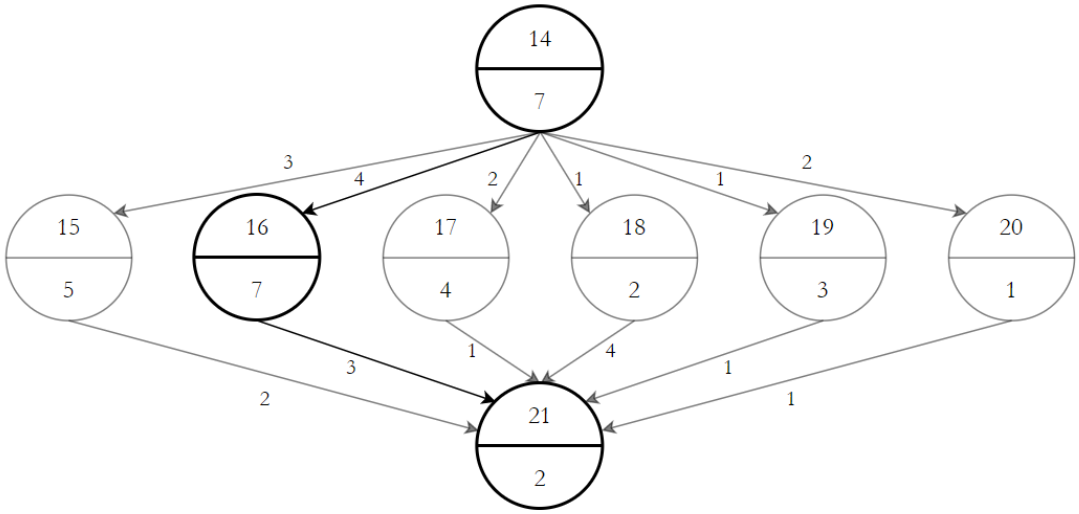


Критичний шлях: 0 → 2 → 5 → 8 → 11 → 14 → 16 → 21 → 22 → 26 → 28 → 30

Перевіримо працездатність на графі меншої розмірності:

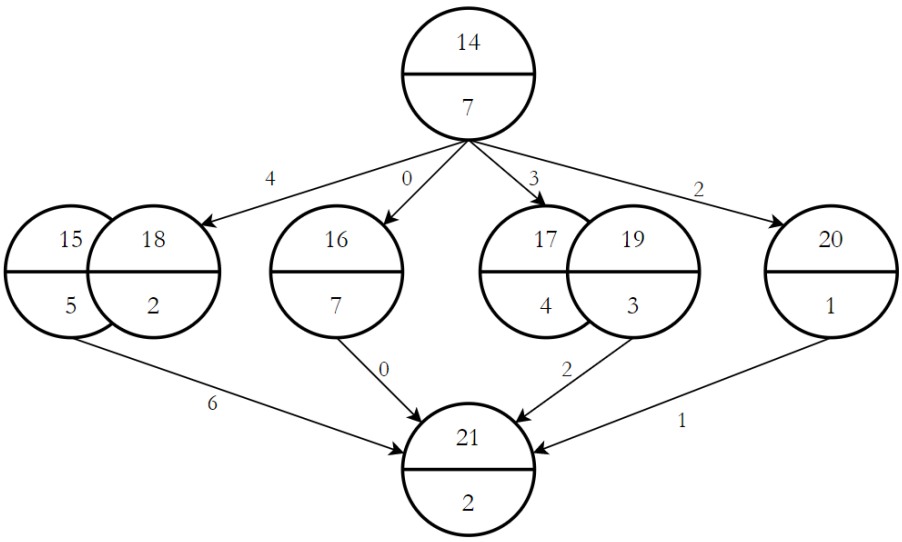


Обираємо критичний шлях:



$$T_{КР} = 7 + 7 + 2 = 16$$

Занулюємо критичний шлях та кластеризуємо вершини:



$$T_{КР} = 7 + 7 + 2 = 16$$

Після кластеризації $T_{КР}$ не збільшився, що підтверджує правильність роботи.

Програма знайшла критичний шлях та кластеризувала вершини так само. Лістинг і скріншот виконання наведено нижче:

Лістинг програми

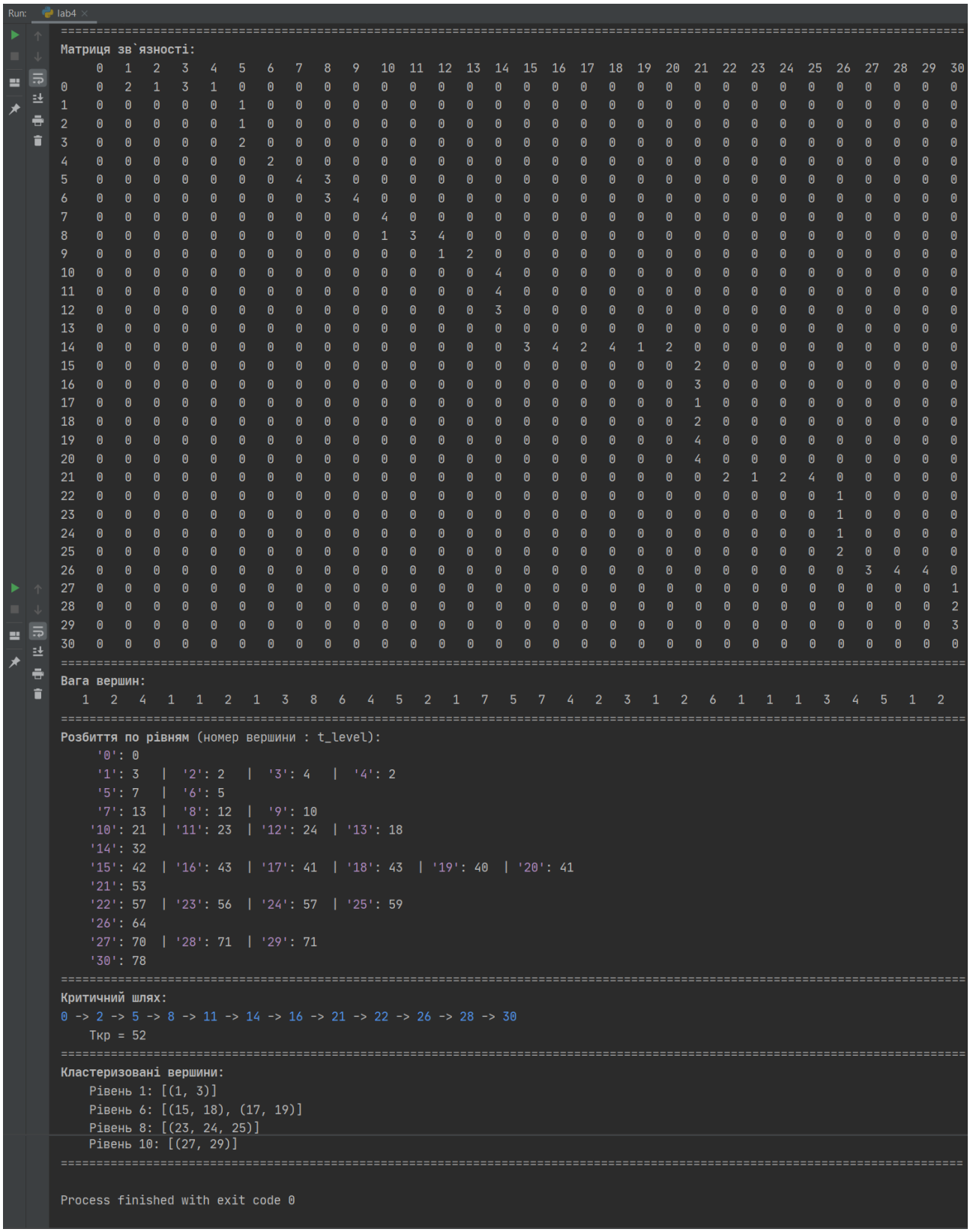
```
import pandas as pd
from random import randint


class Node:
    def __init__(self, idx, weight):
        self.idx = idx
        self.weight = weight
        self.t_level = 0

    def __str__(self):
        fmt = t_level_color + "'{0}'" + reset_color + ": {1}" + reset_color
        return fmt.format(self.idx, self.t_level)


def split_by_levels(nodes):
    levels = [[nodes[0]]]
    free_nodes = nodes[1:]
    current_level = 1

    while len(free_nodes) > 0:
        levels.append([])
        for free_node in free_nodes:
            to_current_level = False

            for prev_node in levels[current_level - 1]:
                edge_weight = matrix[prev_node.idx][free_node.idx]
                if edge_weight > 0:
                    to_current_level = True
                    free_node.t_level = max(free_node.t_level, prev_node.t_level
+ prev_node.weight + edge_weight)

            for node in levels[current_level]:
                if matrix[free_node.idx][node.idx] > 0 or
matrix[node.idx][free_node.idx] > 0:
                    to_current_level = False
                    break

            if to_current_level:
                levels[current_level].append(free_node)

        free_nodes = free_nodes[len(levels[current_level]):]
        current_level += 1
    return levels


def find_critical_way(connection_matrix, weight):
    child, parent, distance = N * [N * []], N * [N * []], N * [0]
    critical_way = [levels[-1][-1].idx]

    for i in range(N):
        for j in range(N):
            if connection_matrix[i][j] != 0:
                child[i] = child[i] + [j]
                parent[j] = parent[j] + [i]

    distance[0] = weight[0]
    for i in range(N):
        for j in range(len(child[i])):
            if distance[i] + weight[child[i][j]] > distance[child[i][j]]:
```

```
                distance[child[i][j]] = distance[i] + weight[child[i][j]]

    while critical_way[0] != 0:
        temp_max = 0
        for i in range(len(parent[critical_way[0]])):
            if distance[parent[critical_way[0]][i]] >= distance[temp_max]:
                temp_max = parent[critical_way[0]][i]
        critical_way.insert(0, temp_max)

    return critical_way, child, distance


def transite(levels):
    res = []
    mins = list(map(lambda level: min([node.t_level for node in level]),
levels))
    for i in range(len(levels) - 1):
        for current_node in levels[i]:
            if current_node.t_level >= mins[i + 1]:
                res.append(current_node)
    return res


def clasterisation(critical_way, child, distance, weights):
    clasterized_edges = N * [[]]

    for i in range(1, len(critical_way) - 1):
        edge_claster = []
        left, right = critical_way[i-1], critical_way[i+1]

        for j in range(len(child[left])):
            if right in child[child[left][j]] and child[left][j] !=
critical_way[i]:
                edge_claster.append(child[left][j])

        for j in range(len(edge_claster)):
            for k in range(j, len(edge_claster)):
                if distance[edge_claster[k]] > distance[edge_claster[j]]:
                    temp = edge_claster[k]
                    edge_claster[k] = edge_claster[j]
                    edge_claster[j] = temp

        temp_edge = 0
        while temp_edge < len(edge_claster):
            temp_weight = weights[edge_claster[temp_edge]]
            level_edges = []

            j = temp_edge + 1
            while j < len(edge_claster):
                if temp_weight + weights[edge_claster[j]] <=
weights[critical_way[i]]:
                    temp_weight += weights[edge_claster[j]]
                    if not level_edges:
                        level_edges = [edge_claster[temp_edge], edge_claster[j]]
                    else:
                        level_edges = level_edges + [edge_claster[j]]
                    del edge_claster[j]
                    j -= 1
                j += 1

            if level_edges:
                del edge_claster[temp_edge]
                clasterized_edges[i] = clasterized_edges[i] +
[tuple(level_edges)]
            else:
                temp_edge += 1
```

```
    return clasterized_edges


matrix = [[0, randint(1, 4), randint(1, 4), randint(1, 4), randint(1, 4), 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, randint(1, 4), randint(1, 4), 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), randint(1, 4), 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), randint(1, 4),
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), randint(1, 4), 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4), 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, randint(1, 4),
randint(1, 4), randint(1, 4), randint(1, 4), randint(1, 4), randint(1, 4), 0, 0,
0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), 0, 0, 0, 0, 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
randint(1, 4), randint(1, 4), randint(1, 4), randint(1, 4), 0, 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, randint(1, 4), 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, randint(1, 4), 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, randint(1, 4), 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, randint(1, 4), 0, 0, 0, 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, randint(1, 4), randint(1, 4), randint(1, 4), 0],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, randint(1, 4)],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, randint(1, 4)],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, randint(1, 4)],
          [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0]]

weights = [1, 2, 4, 1, 1, 2, 1, 3, 8, 6, 4, 5, 2, 1, 7, 5, 7, 4, 2, 3, 1, 2, 6,
1, 1, 1, 3, 4, 5, 1, 2]

N = len(matrix)

bold_color = "\033[1m"
reset_color = "\x1B[0m"
t_level_color = "\x1B[35m"
critical_color = "\x1B[34m"
separator = "\x1B[37m" + '=' * (4 * N + 3) + reset_color

pd.set_option('display.expand_frame_repr', False)
print(separator + bold_color + '\nМатриця зв`язності:\n' + reset_color,
      pd.DataFrame(matrix, columns=[x for x in range(31)]))

print(separator, bold_color + '\nВага вершин:\n' + reset_color +
      ''.join(["{:4}".format(w) for w in weights]))

print(separator, bold_color + '\nРозбиття по рівням ' + reset_color + '(номер
вершини : t_level):')

nodes = [Node(i, weights[i]) for i in range(N)]

levels = split_by_levels(nodes)
for level in levels:
    print('\t' + ' | '.join(map(lambda node: '{:^22}'.format(str(node)),
level)))
print(separator)

print(bold_color + 'Критичний шлях:' + reset_color)
critical, child_arr, distance = find_critical_way(matrix, weights)
print(' -> '.join(map(lambda node: '{:^2}'.format(critical_color + str(node) +
reset_color), critical)))
print('\tTкр =', distance[-1])
print(separator)

print(bold_color + 'Кластеризовані вершини:' + reset_color)
claster_edges = clasterisation(critical, child_arr, distance, weights)
for level in range(len(claster_edges)):
    if claster_edges[level]:
        print('\tРівень ' + str(level) + ': ' + str(claster_edges[level]))
print(separator)

# print(bold_color + 'Вершини з ознакою неявної транзитності ' + reset_color +
'(номер вершини : t_level):')
# result = transite(levels)
# print(' -> '.join(map(lambda node: '{:^7}'.format(str(node)), result)))
```

Результат роботи програми



```
Run:  lab4
================================================================================================================================================
Матриця зв`язності:
     0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30
0    0   2   1   3   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
1    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2    0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
3    0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
4    0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
5    0   0   0   0   0   0   0   0   4   3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
6    0   0   0   0   0   0   0   0   0   3   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
7    0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
8    0   0   0   0   0   0   0   0   0   0   0   1   3   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
9    0   0   0   0   0   0   0   0   0   0   0   0   1   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
10   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
12   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
13   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
14   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   4   2   4   1   2   0   0   0   0   0   0   0   0   0   0
15   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0
16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   0   0   0   0
17   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0
18   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0   0
19   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0
20   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0   0   0   0
21   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   1   2   4   0   0   0   0   0
22   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
23   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
24   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
25   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2   0   0   0   0
26   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3   4   4   0   0
27   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
28   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   2
29   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   3
30   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
================================================================================================================================================
Вага вершин:
   1   2   4   1   1   2   1   3   8   6   4   5   2   1   7   5   7   4   2   3   1   2   6   1   1   1   3   4   5   1   2
================================================================================================================================================
Розбиття по рівням (номер вершини : t_level):
    '0': 0
    '1': 3   |  '2': 2   |  '3': 4   |  '4': 2
    '5': 7   |  '6': 5
    '7': 13  |  '8': 12  |  '9': 10
   '10': 21  | '11': 23  | '12': 24  | '13': 18
   '14': 32
   '15': 42  | '16': 43  | '17': 41  | '18': 43  | '19': 40  | '20': 41
   '21': 53
   '22': 57  | '23': 56  | '24': 57  | '25': 59
   '26': 64
   '27': 70  | '28': 71  | '29': 71
   '30': 78
================================================================================================================================================
Критичний шлях:
0 -> 2 -> 5 -> 8 -> 11 -> 14 -> 16 -> 21 -> 22 -> 26 -> 28 -> 30
    Ткр = 52
================================================================================================================================================
Кластеризовані вершини:
    Рівень 1: [(1, 3)]
    Рівень 6: [(15, 18), (17, 19)]
    Рівень 8: [(23, 24, 25)]
    Рівень 10: [(27, 29)]
================================================================================================================================================

Process finished with exit code 0
```