

Laboratory Session 05

Graphs and Network Analysis

1 User Group Discovery

An enterprise wants to analyze user interactions on its platform to identify separate user groups within the network.

The dataset is represented by an **adjacency matrix** M : each row and column corresponds to a user. An entry $M[i][j] = 1$ indicates that user i and user j have interacted at least once, while 0 indicates no interaction. Interactions are mutual, so the matrix is symmetric ($M[i][j] = M[j][i]$).

Two users belong to the same group if they interacted directly or are connected through a chain of intermediate users. An isolated user with no interactions constitutes a group by itself.

Write an algorithm `user_group_discovery(adj_matrix, user_labels)` that identifies all these groups and returns a list of tuples in the following format:

[(set of users in group, number of users), ...]

The order of the groups in the returned list and the order of users within each set are not important.

State the corresponding asymptotic time complexity at the end of the procedure in terms of the number of users n .

Examples

Below are three example scenarios showing different network topologies side-by-side:

Example 1 (M_1)	Example 2 (M_2)	Example 3 (M_3)
labels: ['A', 'B', 'C', 'D', 'E', 'F']	labels: ['A', 'B', 'C', 'D', 'E', 'F']	labels: ['A', 'B', 'C']
$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

Input	Output
<code>user_group_discovery(M_1, labels)</code>	<code>[({'A', 'B', 'C'}, 3), ({'D', 'E'}, 2), ({'F'}, 1)]</code>
<code>user_group_discovery(M_2, labels)</code>	<code>[({'A', 'B', 'C'}, 3), ({'D', 'E', 'F'}, 3)]</code>
<code>user_group_discovery(M_3, labels)</code>	<code>[({'A'}, 1), ({'B'}, 1), ({'C'}, 1)]</code>
<code>user_group_discovery([], [])</code>	<code>[]</code>

In Example 1, users 'A' and 'C' belong to the same group because they are connected through 'B', even though they did not interact directly. 'D' and 'E' form a second group, while 'F' is isolated. In Example 2, two separate triadic cliques form two distinct groups of 3 users each. In Example 3, no interactions exist, so each user forms a single-element group.

2 Minimizing Distribution Costs

An enterprise needs to optimize product distribution from a central warehouse to various destination locations (intermediate hubs and retail outlets) across its transportation network.

Each available direct route between two locations has a fixed transportation cost per shipment. The overall cost to deliver along a sequence of routes is the sum of their individual costs. All costs are strictly positive.

The transportation network is provided as a **cost matrix** C , where each row and column corresponds to a specific location:

- $C[i][j] = w > 0$ represents the fixed transport cost of a direct route from location i to location j ;
- $C[i][i] = 0$ for all locations;
- An entry of ∞ (`float('inf')`) indicates that no direct route exists from location i to location j .

Given a starting central warehouse s , write an algorithm `min_distribution_costs(cost_matrix, node_labels, start_node)` that computes the minimum transportation cost to deliver a shipment from the central warehouse to every location in the network. The function must return a list of tuples:

[(location, minimum cost), ...]

If a location cannot be reached from s , its minimum cost must be reported as ∞ (`float('inf')`).

State the corresponding asymptotic time complexity at the end of the procedure in terms of the number of locations n .

Examples

Below are three example scenarios showing different network configurations side-by-side:

Example 1 (C_1)	Example 2 (C_2)	Example 3 (C_3)
labels: ['A', 'B', 'C', 'D', 'E']	labels: ['A', 'B', 'C']	labels: ['A', 'B', 'C']
start: 'A'	start: 'A'	start: 'B'
$\begin{pmatrix} 0 & 8 & 3 & \infty & \infty \\ \infty & 0 & \infty & 4 & \infty \\ \infty & 2 & 0 & 9 & \infty \\ \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 4 & \infty \\ \infty & 0 & 1 \\ \infty & \infty & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 2 & 5 \\ \infty & 0 & \infty \\ \infty & 3 & 0 \end{pmatrix}$

Input	Output
<code>min_distribution_costs(C_1, labels, 'A')</code>	<code>[('A', 0), ('B', 5), ('C', 3), ('D', 9), ('E', inf)]</code>
<code>min_distribution_costs(C_2, labels, 'A')</code>	<code>[('A', 0), ('B', 4), ('C', 5)]</code>
<code>min_distribution_costs(C_3, labels, 'B')</code>	<code>[('A', inf), ('B', 0), ('C', inf)]</code>

In Example 1, starting from warehouse 'A': reaching hub 'B' directly costs 8, whereas routing through 'C' costs $3 + 2 = 5$. Delivering to outlet 'D' via $'A' \rightarrow 'C' \rightarrow 'B' \rightarrow 'D'$ costs $3 + 2 + 4 = 9$. Retail outlet 'E' cannot be reached, so its cost is ∞ . In Example 3, starting from 'B', no outgoing routes exist, so 'A' and 'C' are unreachable.

3 Instructions

How to Annotate Your Code

To report the asymptotic computational complexity, place a Python comment containing a $\text{\LaTeX}$ formula at the end of each function, after its final return statement. Keep the comment indented inside the function. For example, if your analysis establishes a quadratic upper bound, write:

```
def example(adj_matrix):
    # ... your implementation ...
    return result
    # Time complexity (worst case):  $O(n^2)$ 
```

The dollar signs and backslash commands are literal text in the Python comment. For other bounds, use valid $\text{\LaTeX}$ such as $O(n^2)$ or $O(n + m)$. Include the tight bound and your derivation in the comments immediately before the final complexity comment. Annotate any helper functions too, specifying their input-size parameter.

How to Complete This Lab

In this lab, you are provided with this PDF containing the exercise instructions, necessary data structures, and expected Input/Output examples.

Along with this document, you have received two Python files:

- **exercises.py**: This file contains the function signatures with empty bodies for all exercises. For example:

```
def user_group_discovery(adj_matrix: list, user_labels: list) -> list:
    pass
```

Your task is to replace the `pass` keyword with your actual implementation. Do not change the function names or their signatures. You can add other functions to support your code.

- **main.py**: This file contains a suite of automated tests. Once you have written your code in **exercises.py**, run **main.py**. It will automatically import your functions, test them against various inputs, and report if your outputs are correct (True for correct result, False otherwise).