

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Лабораторна робота №1**

**з дисципліни «Інженерія якості програмних засобів»**

**на тему:**

**«**Розробка формалізованих специфікацій.»

Виконав:

студент гр. ПЗ2421

Кулик С. В.

Прийняв:

Шинкаренко В.І.

Дніпро, 2025

**Тема.** Розробка формалізованих специфікацій.

**Мета.** Отримати навички формалізації постановки задач з програмування.

**Постановка завдання**

**Задача:** Формування нового графу з різних елементів двох графів, при конфліктів вершин перевагу віддати першому графу, ребер – другому.

**Спосіб представлення:** матриця суміжності  
**Навантаження за вершинами:** так  
**Навантаження по дугам:** ніж

**Хід Роботи**

**1. Опис покрокової розробки специфікації**

Для формалізації задачі використовується мова специфікацій (на зразок RSL). При цьому граф представлено як пара:

* **Множина вершин** – кожна вершина має унікальний ідентифікатор (тип Text) та вагу (тип Int).
* **Матриця суміжності** – функція, яка для кожної пари вершин (ідентифікаторів) повертає булеве значення: *true*, якщо між ними існує дуга, і *false* – інакше.

**Ключові кроки:**

1. **Опис типів даних:**
   * **VertexId:** тип для позначення ідентифікатора вершини (Text).
   * **Weight:** тип для представлення ваги вершини (Int).
   * **Vertex:** структура, що містить ідентифікатор та вагу.
2. **Опис структури графу:**
   * Граф формується як запис з двома полями: множина вершин і матриця суміжності.
   * Матриця визначається як функція від пари ідентифікаторів вершин до булевого значення.
3. **Опис функції об’єднання графів:**
   * **Об’єднання вершин:**  
     Нова множина вершин формується як об’єднання вершин першого графу та вершин другого графу, яких немає у першому (при конфлікті – використовується дані з першого графу).
   * **Об’єднання матриць суміжності:**  
     Для кожної пари вершин з нової множини:
     + Якщо у другому графі відповідна клітинка матриці має значення *true*, то в новому графі вона приймається (пріоритет дуг з другого графу).
     + Інакше, якщо клітинка матриці першого графу має значення *true*, то вона зберігається.
4. **Постумова:**  
   Гарантується, що у новому графі існує хоча б одна дуга (тобто матриця містить хоча б одне значення *true*).

**2. Розроблена специфікація**

### ****Формування множини вершин****

* Формування нової множини вершин із початковим включенням усіх вершин графа **G1** разом із відповідними значеннями навантаження.
* Включення у множину вершин із графа **G2**, за умови їх відсутності у графі **G1**.

### ****Формування множини ребер****

* Формування нової множини дуг із початковим включенням усіх дуг із графа **G2** разом із їх навантаженням.
* Включення у множину дуг із графа **G1** за умови відсутності відповідної дуги між цими вершинами у новому графі.

### ****Фінальний результат****

* Представлення нового графа у вигляді **матриці суміжності**, з урахуванням відповідних навантажень на кожну вершину.

## **Вхідні дані**

* **Два орієнтованих графи** **G1** і **G2**, кожен з яких заданий у вигляді матриці суміжності.
  + **G1(V1, E1):** множина вершин V1 і множина дуг E1.
  + **G2(V2, E2):** множина вершин V2 і множина дуг E2.
  + **Навантаження по вершинах:** кожна вершина має значення навантаження, яке визначає певну міру (наприклад, потужність, пропускну здатність тощо).
  + **Навантаження по дугам:** відсутнє (матриця суміжності лише відображає наявність або відсутність зв’язку).

**Умови**

### **1. Перевага для вершин**

* При конфлікті (наявність однакових вершин у обох графах), у новому графі **пріоритет мають вершини з графа G1**.
* Якщо вершини графів різні, додаємо всі унікальні вершини з обох графів.

### **2. Перевага для ребер**

* Якщо між одними й тими ж вершинами є дуги в обох графах, у новому графі **пріоритет мають ребра з графа G2**.
* Якщо дуги різні, додаються всі унікальні дуги з обох графів.

### **3. Збереження навантаження**

* Навантаження **зберігається лише для вершин** у новоствореному графі.

**Вихідні дані**

* Новий граф **G(V, E)**, представлений у вигляді **матриці суміжності** із зазначенням навантаження на кожну вершину.

**Приклад:**

*G1 (Список ребер):*  
• Вершини: A (10), B (15), C (5)  
• Ребра: A -> B (7), B -> C (8)

*G2 (Список ребер):*  
• Вершини: A (12), B (14), D (9)  
• Ребра: A -> B (5), C -> D (10)

*Новий граф G (Список ребер):*  
• Вершини: A (10), B (15), C (5), D (9)  
• Ребра: A -> B (5), B -> C (8), C -> D (10)

*У цьому прикладі:*  
• Вершина A взята з графа G1 з навантаженням 10, оскільки пріоритет надається першому графу.  
• Ребро A -> B взято з графа G2, оскільки у випадку конфлікту пріоритет мають дуги другого графа.

**Покроковий опис спеціфікації:**

### **1. Модуль GraphMerge**

* Оголошується модуль **GraphMerge**, який містить специфікацію для роботи з орієнтованими графами.
* Графи представлені як множини ребер, де кожне ребро поєднує дві вершини і має навантаження (вагу).

### **2. Типи даних**

* **VertexId = Text** – Тип для позначення ідентифікаторів вершин.
* **Weight = Int** – Тип для представлення навантаження (ваги) на вершинах та ребрах. Використовуються цілі числа.

### **3. Визначення вершини**

Vertex :: id : VertexId

weight : Weight

### **4. Визначення ребра**

Edge :: from : VertexId

to : VertexId

weight : Weight

### **5. Опис графа**

Graph = Edge-set

* Оголошення графа через множину ребер.
* Граф складається з кількох ребер, де кожне ребро з'єднує дві вершини.

### **6. Функція для об'єднання графів**

mergeGraphs: Graph × Graph → Graph

* Оголошується функція, що приймає два графи та повертає новий.
* **Пріоритети:**
  + Вершини беруться з **G1** (якщо є конфлікт).
  + Ребра беруться з **G2** (якщо є конфлікт).

### **7. Проміжні змінні у функції mergeGraphs**

g1\_edges: Graph = { e | e : g1 • not exists e' : g2 • e.from = e'.from ∧ e.to = e'.to }

g2\_edges: Graph = g2

mergedGraph: Graph = g2\_edges ∪ g1\_edges

* **g1\_edges** – множина ребер G1, які не конфліктують з G2.
* **g2\_edges** – множина всіх ребер з G2 (мають пріоритет).
* **mergedGraph** – об'єднання множин ребер g1\_edges і g2\_edges.

### **8. Постумова**

post exists e: mergedGraph • true

* Гарантує, що новий граф **не буде порожнім**.

### **9. Приклад вершин**

vA: Vertex = mkVertex("A", 10)

vB: Vertex = mkVertex("B", 15)

vC: Vertex = mkVertex("C", 5)

vD: Vertex = mkVertex("D", 12)

### **10. Приклад графів**

**G1:**

g1: Graph = {

mkEdge("A", "B", 7),

mkEdge("B", "C", 8)

}

**G2:**

g2: Graph = {

mkEdge("A", "B", 5),

mkEdge("C", "D", 10)

}

### **11. Об'єднання графів**

resultGraph: Graph = mergeGraphs(g1, g2)

* Новий граф містить **об'єднані ребра** з **урахуванням пріоритетів**.

### **12. Опис вершин у новоствореному графі**

vertices: {VertexId | exists e: Edge • e.from ∈ {e.to}}

* Визначається множина ідентифікаторів вершин, що **присутні у новому графі**.
* Всі вершини, які є початковими або кінцевими у ребрах графа, **додаються**.

### **13. Операції для створення вершин та ребер**

mkVertex: VertexId × Weight → Vertex

mkVertex(id, weight) == Vertex(id, weight)

mkEdge: VertexId × VertexId × Weight → Edge

mkEdge(from, to, weight) == Edge(from, to, weight)

Нижче наведено приклад специфікації у стилі RSL для задачі з представленням графу через матрицю суміжності:

**module GraphMergeMatrix**

**types**

**VertexId = Text (\* Ідентифікатор вершини \*)**

**Weight = Int (\* Вага вершини \*)**

**(\* Визначення вершини з ідентифікатором та вагою \*)**

**Vertex :: id : VertexId**

**weight : Weight**

**(\* Опис графу як запис з множиною вершин та матрицею суміжності.**

**Матриця — функція, яка для пари VertexId повертає Bool (належність дуги). \*)**

**Graph :: vertices : set of Vertex**

**matrix : VertexId × VertexId → Bool**

**values**

**(\* Функція для створення вершини \*)**

**mkVertex: VertexId × Weight → Vertex**

**mkVertex(id, w) == Vertex(id, w)**

**(\* Функція об’єднання графів з використанням матриці суміжності \*)**

**mergeGraphs: Graph × Graph → Graph**

**mergeGraphs(g1, g2) as mergedGraph post exists (v, w in mergedGraph.vertices) such that mergedGraph.matrix(v.id, w.id) = true ==**

**let**

**(\* Об’єднання вершин: якщо вершина зустрічається в обох графах, використовується та, що з g1 \*)**

**mergedVertices: set of Vertex =**

**g1.vertices ∪ { v2 in g2.vertices | not exists v1 in g1.vertices • v1.id = v2.id }**

**(\* Об’єднання матриць: для кожної пари (id1, id2) з нової множини вершин \*)**

**mergedMatrix: VertexId × VertexId → Bool =**

**λ (id1, id2).**

**if (id1 ∈ { v.id | v ∈ mergedVertices } ∧ id2 ∈ { v.id | v ∈ mergedVertices })**

**then (if g2.matrix(id1, id2) then true else if g1.matrix(id1, id2) then true else false)**

**else false**

**in**

**Graph(mergedVertices, mergedMatrix)**

**(\* Приклад вершин \*)**

**vA: Vertex = mkVertex("A", 10)**

**vB: Vertex = mkVertex("B", 15)**

**vC: Vertex = mkVertex("C", 5)**

**vD: Vertex = mkVertex("D", 9)**

**(\* Приклад графів з матрицями суміжності \*)**

**(\* Граф g1:**

**Вершини: A, B, C**

**Матриця: A -> B = true, B -> C = true \*)**

**g1: Graph = Graph(**

**{ vA, vB, vC },**

**λ (id1, id2).**

**if (id1 = "A" ∧ id2 = "B") then true**

**else if (id1 = "B" ∧ id2 = "C") then true**

**else false**

**)**

**(\* Граф g2:**

**Вершини: A, B, D (A і B мають інші ваги, але при конфлікті використовуємо дані з g1)**

**Матриця: A -> B = true, C -> D = true \*)**

**g2: Graph = Graph(**

**{ mkVertex("A", 12), mkVertex("B", 14), vD },**

**λ (id1, id2).**

**if (id1 = "A" ∧ id2 = "B") then true**

**else if (id1 = "C" ∧ id2 = "D") then true**

**else false**

**)**

**(\* Результат об’єднання графів \*)**

**resultGraph: Graph = mergeGraphs(g1, g2)**

**end GraphMergeMatrix**

### 3. Аналіз результатів та висновки

**Переваги використання даної специфікації:**

* **Чіткість структури:**  
  Граф представлено як пара (множина вершин і матриця суміжності), що забезпечує однозначне розуміння моделі.
* **Пріоритетність елементів:**
  + При конфлікті вершин дані з першого графу мають пріоритет, що гарантує збереження необхідних характеристик.
  + Дуги з другого графу обираються в разі наявності конфлікту, що забезпечує правильне відображення зв’язків.
* **Формалізованість:**  
  Чітке визначення функції об’єднання дозволяє легко перевірити коректність алгоритму та модифікувати специфікацію при зміні умов задачі.

**Недоліки та можливі ускладнення:**

* **Складність для новачків:**  
  Формалізована специфікація може бути важкою для розуміння тим, хто не знайомий з математичними мовами опису.
* **Обчислювальна складність:**  
  При роботі з великими графами матричне представлення може бути менш ефективним через потребу зберігати дані для кожної пари вершин.

**Висновок**

Розробка специфікації з використанням матриці суміжності дозволяє однозначно описати структуру графу та алгоритм його об’єднання за заданими пріоритетами. Хоча даний підхід може бути громіздким для великих систем, він забезпечує необхідну точність та ясність, що є критично важливим при проектуванні систем зі складною логікою зв’язків.