Міністерство освіти і науки України

Львівський національний університет імені Івана Франка

Факультет прикладної математики та інформатики

Кафедра програмування

Курсова робота на тему:

Розріджені матриці в середовищі Pharo мовою Smalltalk

Виконав студент групи ПМІ-31:

Кормушин Я.К.

Науковий керівник:

доц. Ярошко С.А.

Львів 2017

ЗМІСТ

[Вступ 3](#_Toc483537241)

[1. Формулювання завдання 5](#_Toc483537242)

[2. Аналіз проблеми 7](#_Toc483537243)

[3. Огляд засобів Pharo 8](#_Toc483537244)

[3.1. Основні інструменти середовища 9](#_Toc483537245)

[3.1. Мова Smalltalk 11](#_Toc483537246)

[3.2.1. Типи повідомлень 11](#_Toc483537247)

[3.2.2. Цикли, галуження 11](#_Toc483537248)

[3.2.3. Об’єктна модель 13](#_Toc483537249)

[4. Реалізація 16](#_Toc483537250)

[4.1. Матриця на основі списку координат 17](#_Toc483537251)

[4.2. Матриця на основі колекції списків 23](#_Toc483537252)

[4. Аналіз отриманих результатів 27](#_Toc483537253)

[5. Висновки 32](#_Toc483537254)

[Список використаної літератури 33](#_Toc483537255)

[Додаток А 34](#_Toc483537256)

[Додаток Б 36](#_Toc483537257)

# Вступ

Розрідженою матрицею називають матрицю, більша частина елементів якої дорівнює нулю.

Концептуально, розрідженість відповідає системам, які слабо зв'язані. Розглянемо лінію з м’ячів, з'єднаних пружинами одне з одним: це розріджена система, так як з'єднані тільки сусідні кульки. Навпаки, якщо ж кожен м’яч у лінії з'єднаний пружинами з усіма іншими м’ячами, то система буде відповідати щільній матриці. Поняття розрідженості корисне в комбінаториці і прикладних областях, таких як мережева теорія, які мають низьку щільність значущих даних або з'єднань.

Системи рівнянь з розрідженими матрицями виникають, зокрема, в задачах аналізу міцності конструкцій у цивільному та промисловому будівництві, в авіабудуванні, ракетобудуванні, суднобудуванні тощо. Область застосування методів розв’язування СЛАР з розрідженими матрицями постійно розширюється. Вагому роль в таких операціях відіграють розріджені матриці, які дозволяють зберігати лише певну частину даних, тим самим зменшуючи час виконання обчислень та обсяг пам’яті, необхідний для їх збереження.

**Об’єкт дослідження:** розріджені матриці на основі списку координат, колекції списків.

**Предмет дослідження**: ефективність роботи розріджених матриць.

**Мета дослідження**: вивчити і науково обґрунтувати переваги і недоліки використання певних реалізацій розріджених матриць при виконанні операцій додавання елемента, доступу до елемента, додавання матриць, множення на число, множення на матрицю, транспонування.

**Гіпотеза дослідження**: розріджена матриця з внутрішньою структурою типу «колекція списків» ефективніша в часі при виконанні операцій додавання елемента, доступу до елемента, множення на матрицю. Розріджена матриця з внутрішньою структурою «список координат» ефективніша в часі при виконанні операцій множення на число, додавання матриць, та економніша у витратах пам’яті.

У відповідності з метою і гіпотезою дослідження ставляться такі **завдання**:

1. Вивчити і дослідити особливості мови Smalltalk та середовища Pharo 5.0;
2. Реалізувати структури розрідженої матриці типу «список координат», «колекція списків»;
3. Дослідити ефективність роботи цих реалізацій.

**Наукова новизна дослідження** полягає у тому, що стандартної реалізації структур розрідженої матриці у бібліотеках Pharo Smalltalk немає, а ефективність їх роботи в цьому середовищі не досліджена.

# Формулювання завдання

Реалізувати структуру розріджена матриця засобами середовища Pharo Smalltalk 5.0. Внутрішня структура матриці має відповідати наступним моделям:

* список координат;
* колекція списків.

Оцінити часову ефективність моделей, витрати пам’яті, та порівняти їх.

Структура «список координат» відповідає такій схемі для загального випадку матриці nXm елементів:

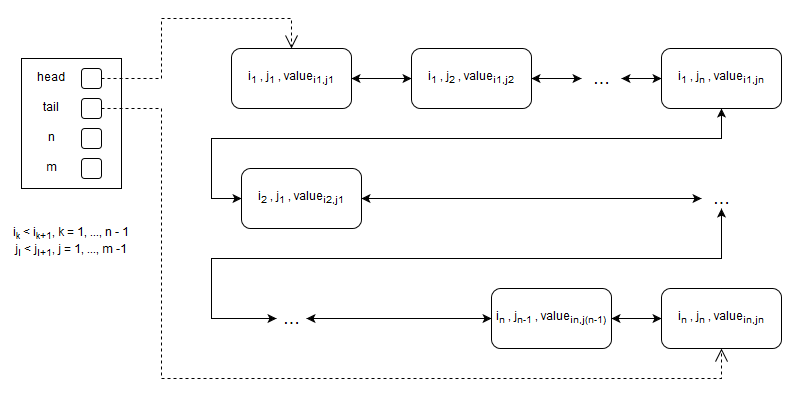


Рисунок 1. Внутрішня структура "список координат"

Ця реалізація зберігає кількість рядків матриці n та кількість стовпців m. Відповідно вона зберігає перший і останній елемент (голову та хвіст) двозв’язного списку елементів (рядок, стовпець, значення), де елементи посортовані у порядку «змійки», тобто за останнім елементом попереднього рядка знаходиться перший елемент наступного.

Структура «колекція списків» відповідає такій схемі:

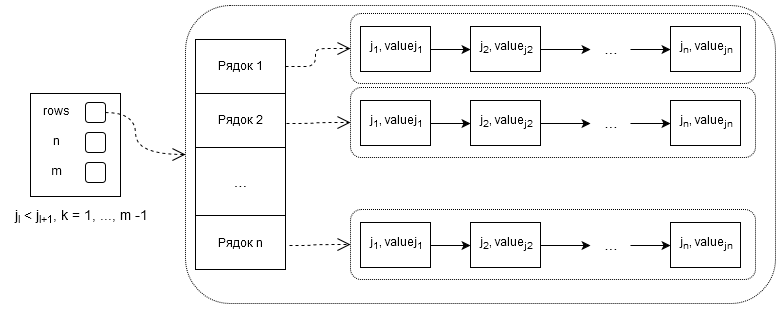


Рисунок 2. Внутрішня структура "колекція списків"

Ця реалізація зберігає по одному списку на кожен рядок, де елементи (стовпець, значення) розташовані в порядку зростання індексу стовпця.

# Аналіз проблеми

**Розріджена матриця** — це матриця, більша частина елементів якої дорівнює нулю. Немає єдиного визначення, яка кількість ненульових елементів має бути в матриці, щоб вона була розрідженою.

**Розрідженістю матриці** називають величину, що дорівнює кількості нульових елементів матриці, поділеній на кількість ненульових елементів.

Розглянемо розріджену матрицю на прикладі:

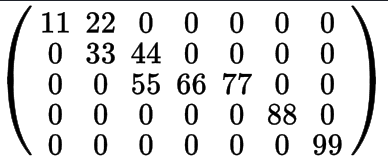


Рисунок 3. Приклад розрідженої матриці

Така розріджена матриця містить лише 9 ненульових елементів і 26 нульових елементів. Відповідно її розрідженість 74%, щільність 26%.

Найпростіша програмна реалізація матриці nXm - це двовимірний масив, тобто масив розміру n, кожен елемент якого є масивом розміру m. Використання такої структури неефективне. По-перше, для вбудованих структур не існує чітких алгоритмів, які б відповідали операціям над матрицями, тому розробник повинен реалізовувати їх власноруч, зважаючи на ефективність їх роботи. По-друге, недоцільно зберігати в пам’яті всі елементи розрідженої матриці, адже більшу їх частину складають нульові елементи.

# Огляд засобів Pharo

**Pharo** — це сучасна повнофункціональна реалізація середовища мовою Smalltalk з відкритим вихідним кодом. Pharo є похідним від Squeak, повторної реалізації класичної Smalltalk-80 системи.

Для реалізації нам потрібна віртуальна машина Pharo та чистий образ середовища Pharo (Рис. 4). Їх можна завантажити з офіційного сайту Pharo за адресою: <http://pharo.org/download>. Ми будемо використовувати останню версію Pharo 5.0 станом на травень 2017 року.

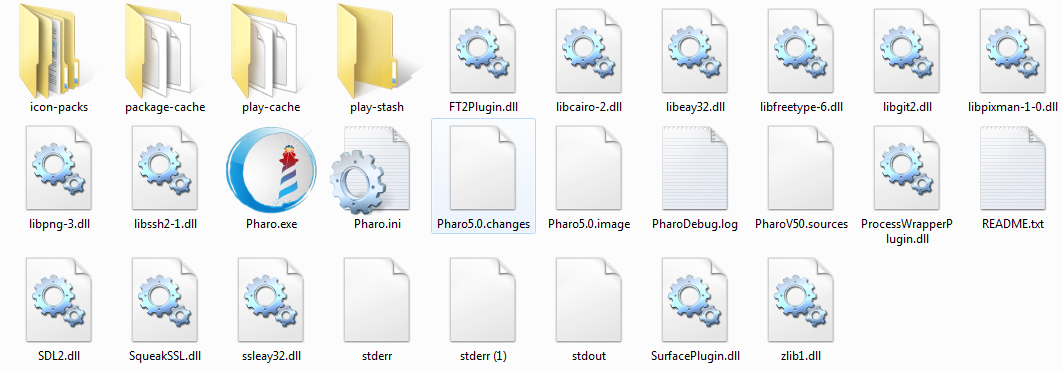


Рисунок 4. Файли, необхідні для роботи в середовищі Pharo

Усі класи та методи в Pharo зберігаються в образі середовища. Образом є файл з розширенням .image.

## 3.1. Основні інструменти середовища

Середовище Pharo надає змогу локалізувати класи у вигляді *пакетів,* які можна переглянути, створити чи редагувати за допомогою ***Nautilus*** (Рис. 5), що є системним переглядачем середовища.Цей інструмент дозволяє переглянути чи змінити будь-який клас чи метод, що знаходиться в образі. Відповідно, можна написати власні структури або переписати вже існуючі.

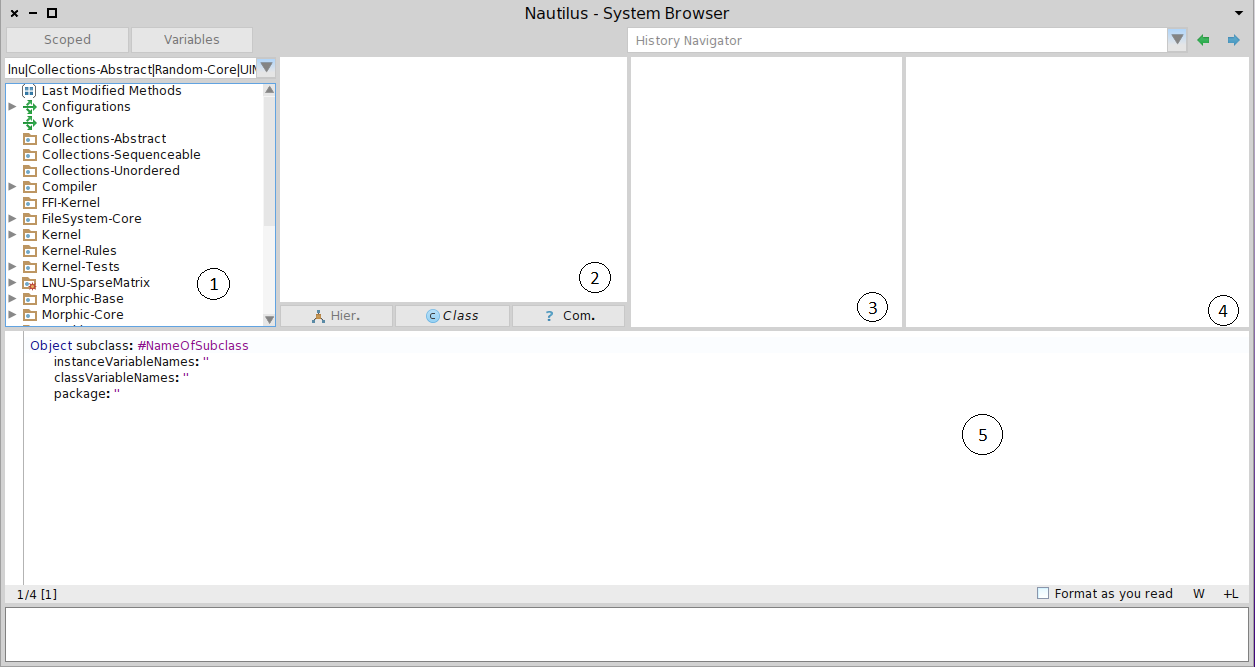


Рисунок 5. Переглядач Nautilus

Вікно Nautilus складається з чотирьох вікон у верхній частині, які відображають ієрархічну модель методів у системі. Панель 1 містить усі пакети класів в образі. На панелі 2 відображені класи, що знаходяться в обраному пакеті. Якщо обрати один з класів пакету, панель 3 міститиме усі протоколи методів для цього класу. Протоколи – це спосіб поділу методів в класі на логічні категорії. Четверта панель відображає імена всіх методів, що належать до певного протоколу. Обравши певний метод, його вихідний код відображатиметься на панелі 5.

***Transcript*** – це окреме текстове вікно, що слугує для виведення повідомлень (Рис. 6). Це своєрідна консоль системи.

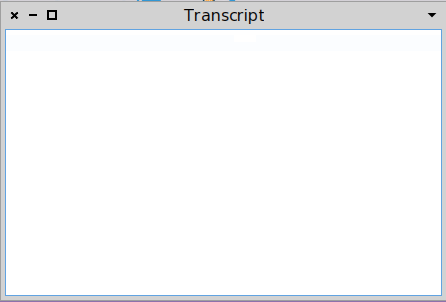


Рисунок 6. Вікно Transcript

***Debugger*** – потужний інструмент, що застосовується не лише для ручного керування ходом програми, а й надає можливість писати код під час виконання цієї ж програми (Рис. 7).

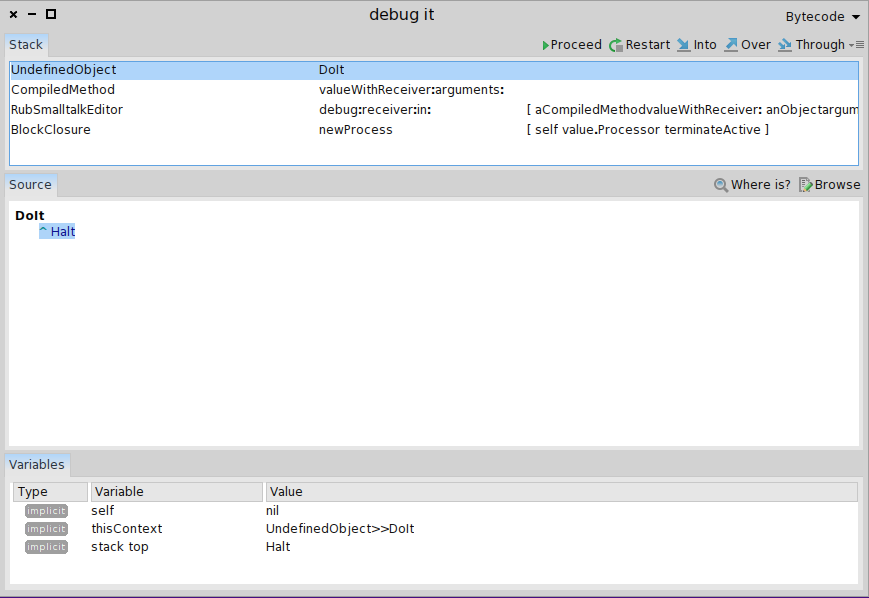


Рисунок 7. Вікно Debugger

***SUnit*** – це фреймворк, який використовується для створення і виконання модульних тестів. З його допомогою тести можна групувати, залежно від контексту частини програми, яку вони перевіряють.

## 3.2. Мова Smalltalk

У мові Smalltalk всі обчислення проводяться шляхом надсилання повідомлень об'єктам. Основний синтаксис в Smalltalk такий: <ОДЕРЖУВАЧ ПОВІДОМЛЕННЯ> <ПРОПУСК> <ПОВІДОМЛЕННЯ> [<АРГУМЕНТИ>]. На відміну від C++, немає крапок після одержувача, і немає дужок навколо аргументів.

Основна різниця з іншими мовами програмування, як С++, С# і т. п. полягає в тому, що об’єкти відповідають за свою поведінку самостійно, а методи, які будуть застосовані, знаходяться динамічно.

### 3.2.1. Типи повідомлень

Існують три типи повідомлень: унарні, бінарні та повідомлення з ключовими словами. *Унарні* повідомлення це повідомлення типу 3 factorial, де одному об’єкту надсилається повідомлення без аргументів. Прикладом *бінарного* повідомлення є , де об’єкту надсилається повідомлення з аргументом . Повідомлення *з ключовими словами* це повідомлення типу , що є аналогом операції піднесення числа до степеня .

Кожен тип повідомлень має чітко визначений пріоритет. Унарні мають найвищий, а повідомлення з ключовими словами – найнижчий. Змінити порядок надсилання повідомлень можна за допомогою круглих дужок. Відповідно виконання дасть результат , а - .

### 3.2.2. Цикли, галуження

Галуження в мові Smalltalk організовані теж за принципом надсилання повідомлень. Розглянемо звичну для Smalltalk конструкцію:

4 factorial > 20

ifTrue: [ 'bigger' ]

ifFalse: [ 'smaller' ]

Цей запис поверне рядок ‘bigger’ якщо факторіал числа 4 буде більшим, ніж 20, і ‘smaller’ в протилежному випадку. Насправді ми відправляємо повідомлення ifTrue:ifFalse: до об’єкта типу Boolean. Абстрактний клас Boolean має дві singleton-реалізації: True і False. Ці класи мають унікальні екземпляри true і false відповідно. Залежно від результату першого рядка, ми відправимо повідомлення одному з цих екземплярів: true виконає блок при ifTrue:, а false при ifFalse:. Boolean розуміє й інші повідомлення, такі як ifTrue:, ifFalse:.

Основною різницею з іншими мовами програмування є те, що об’єкт, який отримує повідомлення, сам вирішує який блок виразів йому виконати. Таким чином усувається фактор імперативного програмування.

Цикли зазвичай виражені надсиланням повідомлень до блоків, цілих чисел чи колекцій. Розглянемо приклад звичайного ітераційного циклу:

n := 1.

[ n < 1000 ] whileTrue: [ n := n \* 2 ].

Тут змінній n присвоєно значення 1, після чого блоку «доки n менше 1000» надсилаємо повідомлення «n присвоїти n \* 2».

Таке повідомлення ідентичне попередньому:

n := 1.

[ n >= 1000 ] whileFalse: [ n := n \* 2 ].

Реалізацією циклу з фіксованою кількістю кроків є таке повідомлення:

n := 1.

10 timesRepeat: [ n := n\*2 ].

Також можна надіслати повідомлення to:do: цілому числу, яке буде виступати початковим значенням кроку циклу. Першим аргументом є верхня межа циклу, а другим – блок виразів, якому передаватиметься значення кроку.

Приклад:

result := String new.

1 to: 10 do: [:n | result := result, n printString, ’ ’].

### 3.2.3. Об’єктна модель

Об’єктна модель Pharo Smalltalk має наступні властивості:

1. Все є об'єктом.
2. Кожен об'єкт є екземпляром класу.
3. Кожен клас має батьківський клас.
4. Все відбувається шляхом надсилання повідомлень.
5. Пошук методу йде по ланцюжку наслідування.
6. Кожен клас є екземпляром метакласу.
7. Ієрархія метакласів паралельна ієрархії класів.
8. Кожен метаклас наслідує Class і Behavior.
9. Кожен метаклас є екземпляром метакласу.
10. Метаклас метакласу є екземпляром метакласу.

***Метаклас*** - це клас класу, який створюється неявно, але в якому можна визначати власні методи. Клас і метаклас – це різні класи, хоч перший є екземпляром другого.

Класи в Pharo Smalltalk оголошують таким чином. Спочатку потрібно обрати до якого пакету належатиме новий клас. Тоді за допомогою контекстного меню переглядача можна додати новий клас (Рис. 8). Оголошення класу обов’язково містить його ім’я, наслідуваний клас, змінні об’єкта, змінні класу і пакет, до якого він належить. Cтворення нового класу це також надсилання повідомлення до наслідуваного класу.

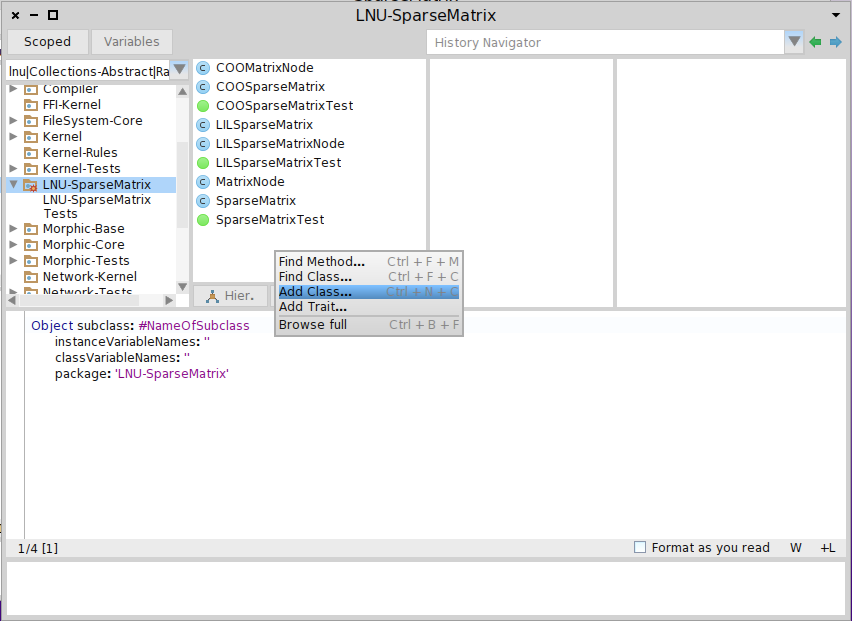


Рисунок 8. Додавання нового класу до пакету LNU-SparseMatrix через меню

Розглянемо оголошення класу LinkedList (Рис. 9), який є частиною стандартної бібліотеки Pharo. Його ми використаємо для реалізації списку елементів у матрицях.

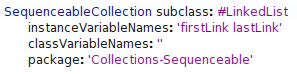


Рисунок 9. Оголошення класу LinkedList

#LinkedList – це символ, що однозначно визначає ім’я створюваного класу. instanceVariableNames – набір змінних, що будуть належати кожному екземпляру класу LinkedList. classVariableNames – набір об’єктів, що належатимуть метакласу цього класу.

Як можна бачити на Рис. 9, створення нового класу зводиться до надсилання повідомлення до батьківського класу з вказаними:

* ім’ям;
* переліком змінних екземпляра класу;
* переліком змінних метакласу;
* іменем пакету.

Методи оголошують наступним чином. У переглядачі потрібно натиснути на будь-який протокол класу. Після цього, на основній панелі відобразиться шаблон для створення нового методу (Рис. 10). Вгорі замість messageSelectorAndArgumentNames вказуємо ім’я повідомлення, на яке відповідатиме об’єкт класу, і імена аргументів цього повідомлення.

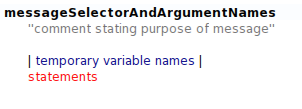


Рисунок 10. Шаблон оголошення методу

Необов’язково, але зручно, в коментарі під оголошенням імені методу вказати суть того, що він робить. Всередині прямих дужок | | оголошуємо локальні змінні. Далі описуємо поведінку методу. На Рис. 11 наведений метод addFirst: класу LinkedList, що додає об’єкт в початок списку.

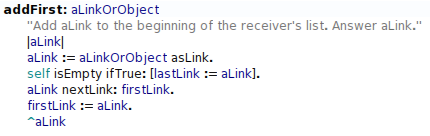


Рисунок 11. Метод addFirst: класу LinkedList

# 4. Реалізація

Для початку створимо новий пакет, який міститиме всі наші класи. Відкриємо вікно переглядача Nautilus і в контекстному меню оберемо «Add package…». Назвемо його LNU-SparseMatrix.

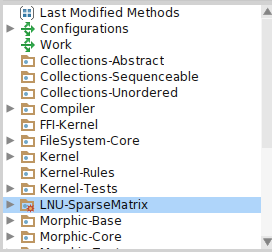


Рисунок 12. Пакет LNU-SparseMatrix

В наших реалізаціях непотрібно виносити певну поведінку в один спільний батьківський клас, тому всі класи наслідуватимуть клас Object, що є кореневим класом будь-якої структури Pharo.

## 4.1. Матриця на основі списку координат

Для реалізації такої структури нам знадобиться клас, що представлятиме елемент матриці, і клас, що представлятиме саму матрицю.

Елемент матриці на основі списку координат матиме схожу поведінку з елементом двозв’язного списку. Він міститиме інформацію про свою позицію в матриці, а також про елемент зліва і справа від нього (Рис. 13).

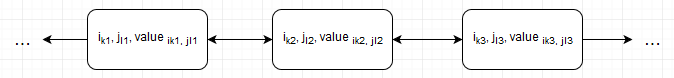


Рисунок 13. Елемент матриці на основі списку координат

Оберемо пакет LNU-SparseMatrix, на другій панелі Nautilus викличемо контекстне меню і виберемо «Add class…». Оголошення класу на Рис. 14.

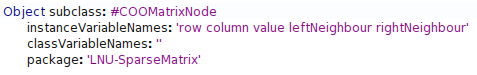


Рисунок 14. Оголошення класу COOMatrixNode

COOMatrixNode – клас, що описує елемент для реалізації структури списку координат (Рис. 15). Кожен екземпляр класу містить такі змінні:

* рядок (row) – ціле число (екземпляр класу SmallInteger);
* стовпець (column) – ціле число (екземпляр класу SmallInteger);
* значення елемента матриці (value) - число (екземпляр класу, що наслідує Number);
* посилання на правого сусіда (rightNeighbour);
* посилання на лівого сусіда (leftNeighbour).

Для зручності визначимо методи preceeds: і follows:, які повертають результат типу Boolean, залежно від того, чи екземпляр, що отримав повідомлення, знаходиться в списку матриці лівіше чи правіше від аргументу.

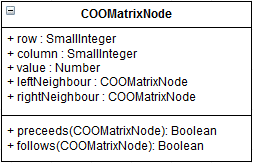


Рисунок 15. Клас COOMatrixNode

Метаклас COOMatrixNode містить метод, що отримує значення рядка, стовпця і значення елемента, і повертає екземпляр класу COOMatrixNode з цими значеннями.

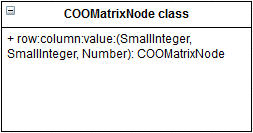


Рисунок 16. Метаклас класу COOMatrixNode

Тепер додамо клас матриці. Його оголошення (Рис. 17):

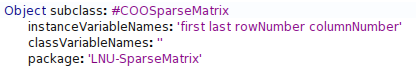


Рисунок 17. Клас COOSparseMatrix

COOSparseMatrix – клас, що описує матрицю для реалізації структури списку координат (Рис. 18). Кожен екземпляр містить:

* посилання на перший елемент списку;
* посилання на останній елемент списку;
* кількість рядків матриці – ціле число (екземпляр класу SmallInteger);
* кількість стовпців матриці – ціле число (екземпляр класу SmallInteger).

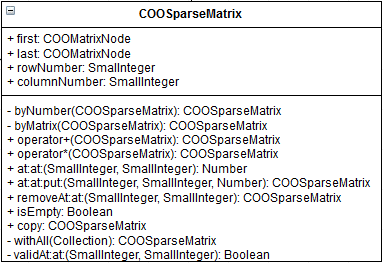


Рисунок 18. Клас COOSparseMatrix

Методи класу розділені на такі протоколи:

1. Accessing – методи доступу до даних:
   * at:at: — метод для доступу до елементів матриці;
   * at:at:put: - метод для запису елементів у матрицю, або зміну вже записаних;
   * rowNumber – повертає кількість рядків матриці;
   * columnNumber – повертає кількість стовпців матриці;
   * first – повертає перший елемент в списку матриці;
   * last – повертає останній елемент в списку матриці.
2. Arithmetic – методи для виконання арифметичних операцій над матрицями:
   * \* - множення матриць або матриці на число (залежно від типу аргументу), що повертає цю матрицю, помножену на матрицю або число-аргумент;
   * + - додавання матриць, що повертає цю матрицю плюс матриця, що передається як аргумент;
3. Copying – методи для копіювання об’єкта:
   * copy – повертає копію цієї матриці;
4. Instance creation – методи для ініціалізації екземплярів класу:
   * rows:columns: - метод для ініціалізації порожньої матриці з відповідною розмірністю;
   * withAll: - метод для зручної ініціалізації матриці за допомогою будь-якої двовимірної колекції (Рис. 19);
5. Private – приватні методи, не призначені для зовнішнього використання:
   * validAt:at: - метод, що дозволяє перевірити чи індекс знаходиться в межах розмірності матриці;
   * byNumber: - метод, що повертає цю матрицю, помножену на число, що передається як аргумент;
   * byMatrix: - повертає цю матрицю, помножену на матрицю-аргумент (див. Додаток А);
6. Removal – методи для видалення елементів:
   * removeAt:at: - видалення існуючого елемента;
7. Testing – методи для перевірки певних властивостей матриці:
   * isEmpty – повертає true, якщо матриця не містить жодного елемента, false в протилежному випадку;
8. Transformation – методи для перетворень матриці:
   * transpose – повертає транспоновану матрицю.



Рисунок 19. Метод withAll: класу COOSparseMatrix

Як можна бачити, метод withAll: застосовує повідомлення collect:, collect:thenReject:, inject:into: та do: для колекцій.

Метаклас COOSparseMatrix (Рис. 20) містить методи для зручної ініціалізації екземплярів матриці, де:

* withAll: - метод, що повертає матрицю, проініціалізовану певною двовимірною колекцією;
* rows:columns: - повертає матрицю з відповідною кількістю порожніх рядків і стовпців.



Рисунок 20. Метаклас COOSparseMatrix

Загальна діаграма (Рис. 21) класів виглядає так:

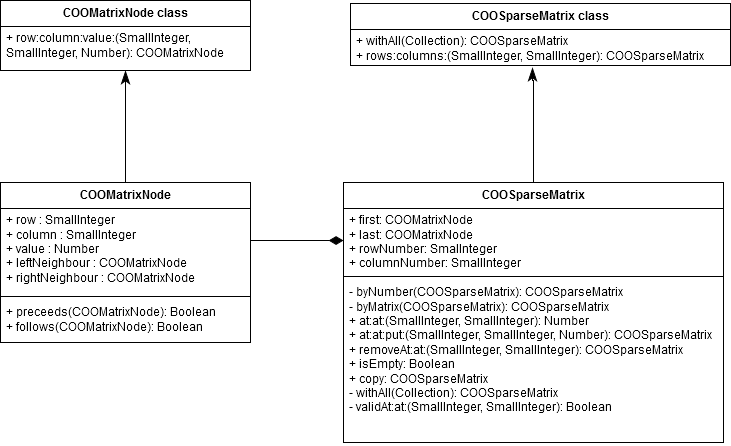


Рисунок 21. Загальна діаграма класів для структури списку координат

Стрілка  позначає відношення «екземпляр …», тобто клас COOMatrixNode екземпляр класу COOMatrixNode class.

## 4.2. Матриця на основі колекції списків

Для цієї реалізації нам також потрібно визначити клас елемента та клас матриці, що міститиме ці елементи.

Елемент матриці на основі колекції списків повинен містити свою позицію стовпця в матриці і значення елемента. Оголосимо цей клас так (Рис. 22):

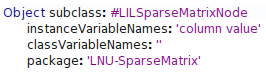


Рисунок 22. Оголошення класу LILSparseMatrixNode

LILSparseMatrixNode – клас, що відображає елемент матриці для реалізації структури колекції списків. Він містить такі змінні:

* стовпець елемента матриці;
* значення елемента матриці.

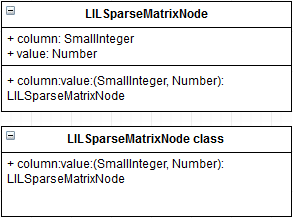


Рисунок 23. Клас LILSparseMatrixNode

Цей клас містить зокрема і метод ініціалізації цих змінних.

Його метаклас має ідентичний метод, який поверне екземпляр з вже проініціалізованими даними:

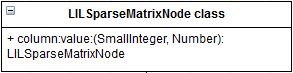


Рисунок 24. Метаклас LILSparseMatrixNode

Додамо клас матриці:

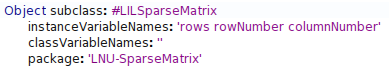


Рисунок 25. Оголошення класу LILSparseMatrix

LILSparseMatrix – це клас, що описує матрицю для реалізації структури колекції списків, який містить:

* масив списків (rows) – екземпляр Array, де кожен елемент це список типу LinkedList;
* кількість рядків (rowNumber) – ціле число (екземпляр класу SmallInteger);
* кількість стовпців (columnNumber) – ціле число (екземпляр класу SmallInteger).

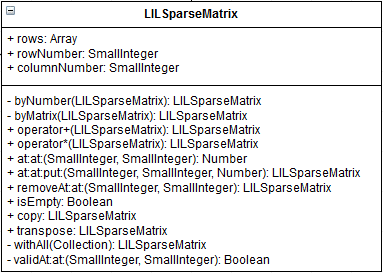


Рисунок 26. Клас LILSparseMatrix

Методи класу розділені на такі протоколи:

1. Accessing – методи доступу до даних:
   * at:at: — метод для доступу до елементів матриці;
   * at:at:put: - метод для запису елементів у матрицю, або зміни вже записаних;
   * rowNumber – повертає кількість рядків матриці;
   * columnNumber – повертає кількість стовпців матриці;
   * rows – повертає масив типу Array, що містить списки елементів матриці;
2. Arithmetic – методи для виконання арифметичних операцій над матрицями:
   * \* - множення матриць або матриці на число (залежно від типу аргументу), що повертає цю матрицю, помножену на матрицю або число-аргумент;
   * + - додавання матриць, що повертає цю матрицю плюс матриця, що передається як аргумент;
3. Copying – методи для копіювання об’єкта:
   * copy – повертає копію цієї матриці;
4. Instance creation – методи для ініціалізації екземплярів класу:
   * rows:columns: - метод для ініціалізації порожньої матриці з відповідною розмірністю;
   * withAll: - метод для зручної ініціалізації матриці за допомогою будь-якої двовимірної колекції (Рис. 19);
5. Private – приватні методи, не призначені для зовнішнього використання:
   * validAt:at: - метод, що дозволяє перевірити чи індекс знаходиться в межах розмірності матриці;
   * byNumber: - метод, що повертає цю матрицю, помножену на число, що передається як аргумент;
   * byMatrix: - повертає цю матрицю, помножену на матрицю-аргумент;
6. Removal – методи для видалення елементів:

* removeAt:at: - видалення існуючого елемента;

1. Testing – методи для перевірки певних властивостей матриці:

* isEmpty – повертає true, якщо матриця не містить жодного елемента, false в протилежному випадку;

1. Transformation – методи для перетворень матриці:

* transpose – повертає транспоновану матрицю.

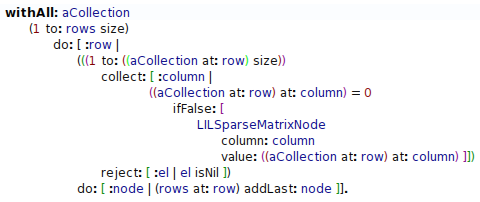


Рисунок 27. Метод withAll: класу LILSparseMatrix

Ініціалізація за допомогою двовимірної колекції тут реалізована значно простіше. Внаслідок використання масиву як колекції рядків-списків типу LinkedList ми змогли застосувати вбудовані методи do: та addLast:.

Зведена діаграма (Рис. 28) класів така:

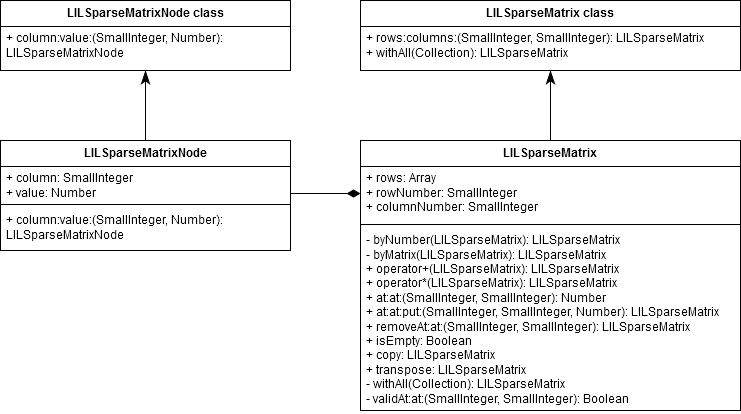


Рисунок 28. Зведена діаграма класів для структури «колекція списків»

Тут, чорна стрілка  позначає відношення «екземпляр …», тобто клас LILSparseMatrix екземпляр класу LILSparseMatrix class.

# 4. Аналіз отриманих результатів

Для модульного тестування класів обох реалізацій використаємо вищезгаданий фреймворк ***SUnit***.

В тому ж пакеті (LNU-SparseMatrix) створимо два класи: COOSparseMatrixTest і LILSparseMatrixTest для тестування матриці на основі списку координат (COOSparseMatrix) і матриці на основі колекції списків (LILSparseMatrix) відповідно. Ці тестові класи повинні наслідувати від TestCase, аби мати доступ до вбудованого функціоналу фреймворку. Їх оголошення подано на Рис. 29 і Рис. 30.

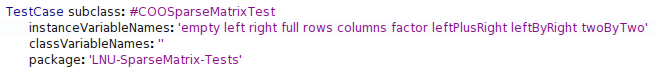


Рисунок 29. Оголошення класу COOSparseMatrixTest

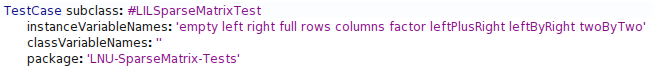


Рисунок 30. Оголошення класу LILSparseMatrixTest

Обидва тестових класи мають набір змінних, ініціалізацію яких винесено в метод setUp, який викликається при запуску кожного модульного тесту:

* rows – кількість рядків матриць (ціле число);
* columns – кількість стовпців матриць (ціле число);
* factor – множник; використовується при перевірці множення матриці на число;
* twoByTwo – довільна матриця розмірності 2x2;
* left – довільна матриця розмірності rows x columns, що використовується при перевірці множення матриць як лівий операнд;
* right – довільна матриця розмірності rows x columns, що використовується при перевірці множення матриць як правий операнд;
* empty – порожня матриця розмірності rows x columns;
* full – матриця розмірності rows x columns, де кожен елемент ненульовий;
* leftPlusRight – матриця розмірності rows x columns, що є результатом додавання матриць left і right;
* leftByRight – матриця розмірності rows x columns, що є результатом множення матриць left і right.

Оскільки функціонал матриць однаковий, можемо навести спільний список тестів для обох реалізацій:

* testAddition – перевірка правильності додавання матриць;
* testAtAt – перевірка читання елементів;
* testAtAtPut – перевірка зміни елементів;
* testByMatrix – перевірка множення матриць;
* testByNumber – перевірка множення матриці на число;
* testColumnNumber – перевірка правильності ініціалізації кількості стовпчиків матриці;
* testRowNumber - перевірка правильності ініціалізації кількості стовпчиків матриці;
* testCopy – перевірка копіювання;
* testRemoveAtAt – перевірка видалення елементів;
* testTranspose – перевірка транспозиції матриці;
* testWithAll – перевірка ініціалізації матриці за допомогою двовимірної колекції.

За допомогою системного переглядача Nautilus запустимо усі тести з класів COOSparseMatrixTest і LILSparseMatrixTest, натиснувши на сірий круг зліва біля кожного з цих класів на другій панелі. В разі успішного проходження усіх тестів круг стане зеленого кольору, а в лівому нижньому куті основного вікна відобразиться сповіщення про успішне виконання (Рис. 31).

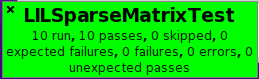


Рисунок 31. Сповіщення про виконання тестів класу LILSparseMatrixTest

Перевірка модульними тестами показала, що всі методи наших реалізацій працюють коректно.

Для аналізу ефективності створимо новий тестовий клас SparseMatrixTest, що наслідуватиме TestCase. Кожен метод цього класу буде здійснювати перевірку ефективності певної реалізації матриці. Для порівняння протестуємо матрицю, створену на основі масиву масивів зі стандартної бібліотеки Pharo. Результати тестів виводитимуться в Transcript. Оголошення класу подано на Рис. 32.

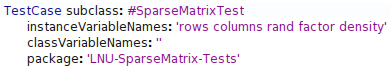


Рисунок 32. Оголошення класу SparseMatrixTest

Клас містить такі змінні:

* rows – кількість рядків матриць;
* columns – кількість стовпців матриць;
* rand – генератор випадкових чисел для заповнення матриць;
* factor – множник, що використовується при аналізі ефективності множення матриці на число;
* density – число між 0.01 та 1.00, яке відображає максимальну відсоткову щільність матриць залежно від їх максимальної кількості елементів.

Наприклад якщо задати величину матриці 10x20, а щільність 0.1, то кількість елементів в матриці не перевищуватиме 10% від 200 елементів, тобто 20 ненульових елементів.

Для перевірки швидкості запису елементів у матриці використовуватимемо вбудований клас Random (Рис. 33) з пакету Random-Core, що дозволяє генерувати випадкові числа у вказаному проміжку. Приклад використання цього об’єкта поданий на Рис. 34.

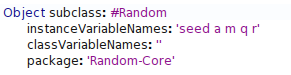


Рисунок 33. Клас Random

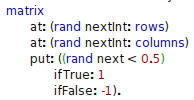


Рисунок 34. Використання генератора випадкових чисел Random

Повідомлення nextInt: rows поверне довільне ціле число на проміжку [1, rows]. Аналогічно діятиме і nextInt: columns. Повідомлення next поверне довільне раціональне число на проміжку [0, 1]. Тобто за допомогою генератора ми випадковим чином обираємо позицію елемента в матриці matrix, куди необхідно додати елемент, а тоді з імовірністю 0,5 обираємо значення для вставки 1 або -1.

Для аналізу тривалості виконання операцій над різними реалізаціями використаємо метод timeToRun. Цей метод визначений в класі BlockClosure (Рис. 35), екземпляри якого зберігають послідовність виразів з довільною кількістю аргументів.

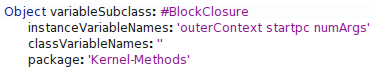


Рисунок 35. Клас BlockClosure

Послідовно запустимо тести з різними розмірностями матриць та різною щільністю для перевірки ефективності в часі. Результати тестів подані в

Додатку Б.

Для аналізу ефективності роботи матриць щодо економії пам’яті використаємо інструмент Test Runner. Він дозволяє виконувати тести з обраних нами класів і отримувати детальний звіт їх роботи. Нас цікавлять саме витрати пам’яті для виконання операцій над матрицями. В Табл. 1 подані наближені результати звітів.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 100 х 100  20% | 200 х 200  25% | 300 х 300  15% | 300 х 300  45% |
| Масив масивів | 518,968 байт | 1,326,672 байт | 1,700,631 байт | 2,310,274 байт |
| Список координат | 43,280 байт | 680,656 байт | 747,248 байт | 2,311,728 байт |
| Колекція списків | 298,080 байт | 756,168 байт | 951,392 байт | 3,179,026 байт |

Таблиця 1. Використання пам'яті різними реалізаціями матриць: рядки – реалізація матриці, стовпці – розмір і щільність

# 5. Висновки

В ході виконання роботи я:

* вивчив середовище розробки Pharo, синтаксис мови Smalltalk і основні складові стандартної бібліотеки Pharo Smalltalk 5.0;
* реалізував структури розріджених матриць на основі внутрішнього представлення «список координат» і «колекція списків»;
* протестував їх правильну роботу за допомогою фреймворку SUnit (unit testing);
* дослідив ефективність роботи цих реалізацій в залежності від їх величини і ступеня заповненості.

Внаслідок аналізу виявив, що:

* матриці з внутрішньою структурою «список координат» мають незначну перевагу в часі над іншими реалізаціями при виконанні операцій додавання матриць та множення на число за низької щільності (менше 40%);
* матриці з внутрішньою структурою «колекція списків» мають незначну перевагу в часі над іншими реалізаціями при виконанні операції множення на матрицю за щільності менше 50%;
* обидві реалізації на базі спискових структур не надають великої переваги в часі над прямою реалізацією масивом масивів при виконанні операцій транспонування, додавання і читання елемента;
* реалізації на базі спискових структур займають менше пам’яті при виконанні операцій, ніж пряма реалізація, але чим більша густина матриці, тим менша перевага.

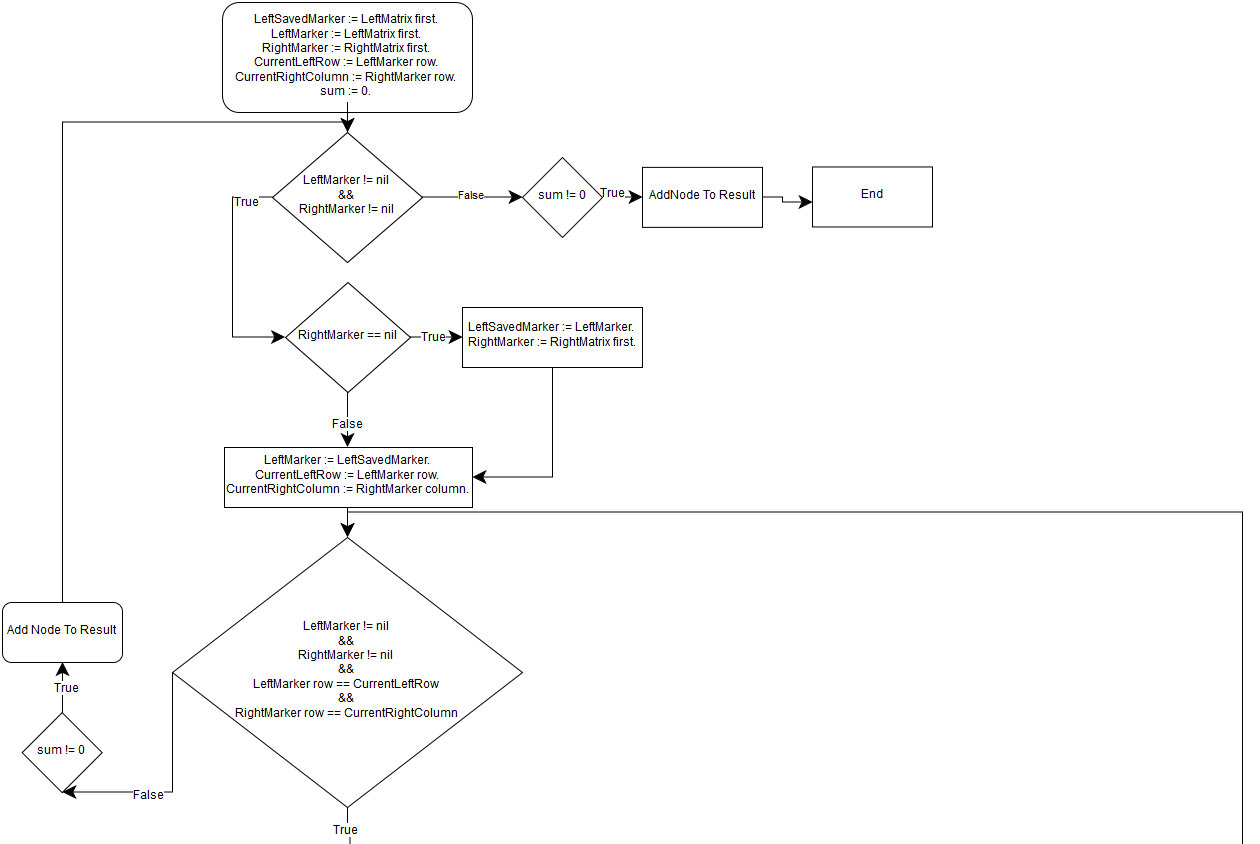
В майбутньому я планую:

* оптимізувати алгоритми множення на матрицю і транспонування;
* зменшити використання пам’яті;
* додати можливість перетворення однієї реалізації в іншу;
* використати ці реалізації при побудові системи з машинним навчанням.

# Список використаної літератури

1. Bergel Alexandre. Deep into Pharo [Книга] / Bergel Alexandre [та ін.]. - Square Bracket Associates, 2013.
2. Pharo [Електронний ресурс] : Web-сайт. - Wikipedia. - Режим доступу: https://uk.wikipedia.org/wiki/Pharo.
3. Ducasse Stéphane. Pharo By Example 5 / Ducasse Stéphane [та ін.]. - Lulu.com & Square Bracket Associates, 2017.
4. Pharo By Example Two [Електронний ресурс]. - Pharo By Example Two. - Режим доступу: http://pharobooks.gforge.inria.fr/PharoByExampleTwo-Eng/latest/Profiling.pdf.
5. Sparse Matrix [Електронний ресурс] : Web-сайт.- Wikipedia. - Режим доступу: https://en.wikipedia.org/wiki/Sparse\_matrix.
6. Pharo MOOC [Електронний ресурс] : Web-сайт. - Pharo MOOC. - Режим доступу: http://rmod-pharo-mooc.lille.inria.fr/.

# Додаток А



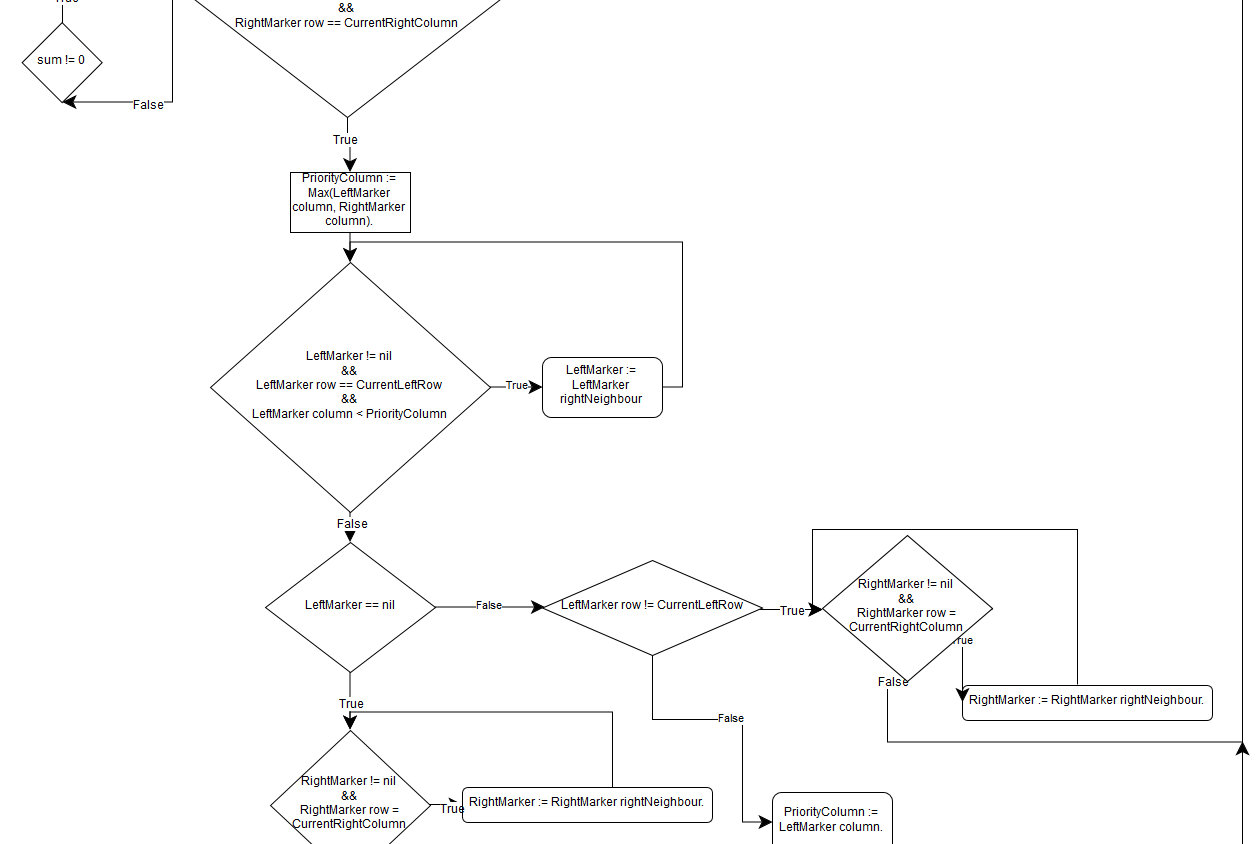


Рис. 1. Перша частина схеми множення матриць на основі списку координат

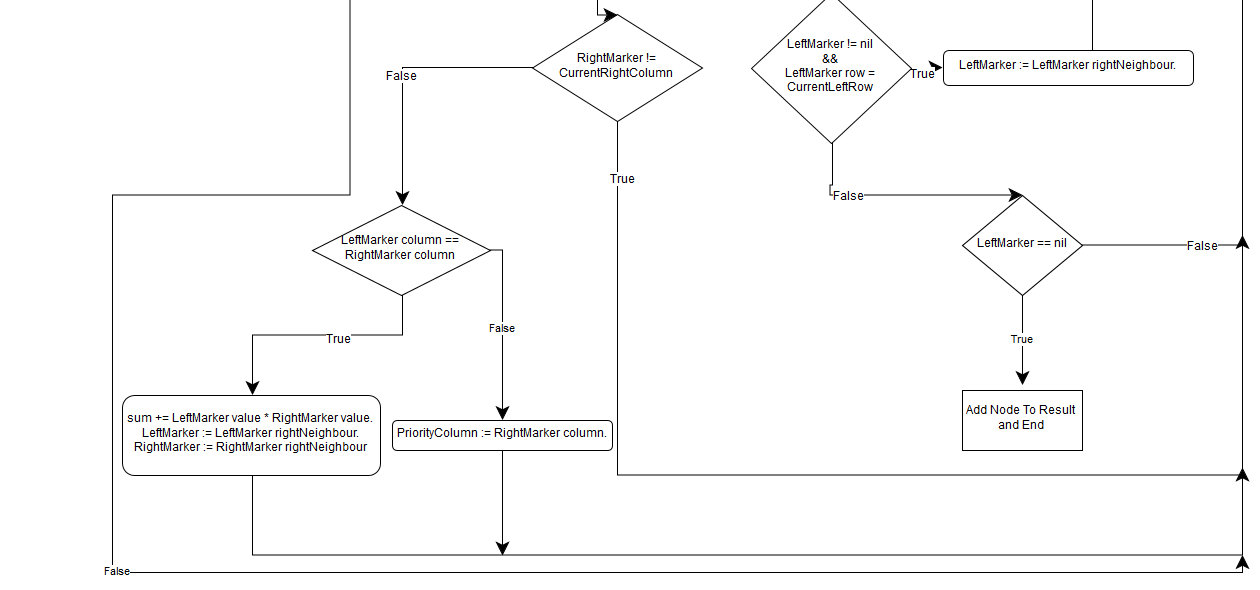
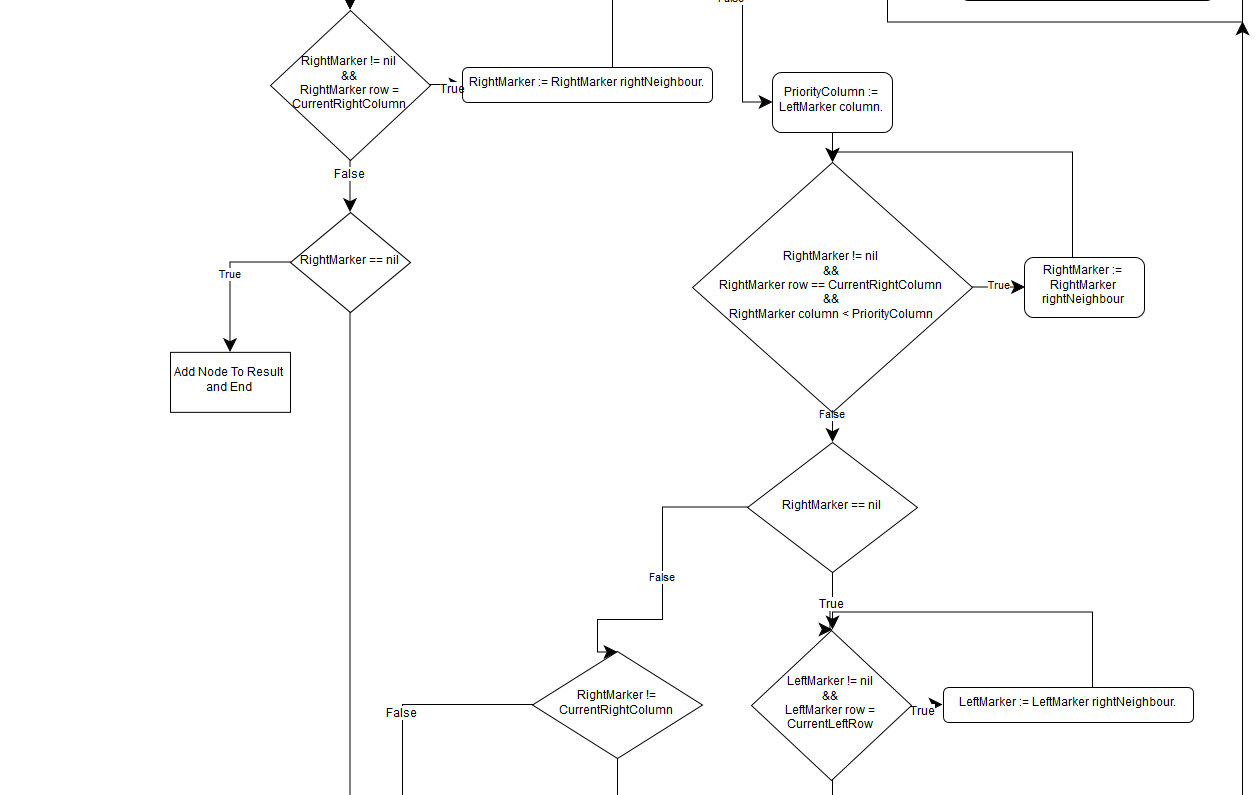


Рис. 2. Друга частина схеми множення матриць на основі списку координат

# Додаток Б

Результати аналізу роботи:

1. Читання / запис елемента

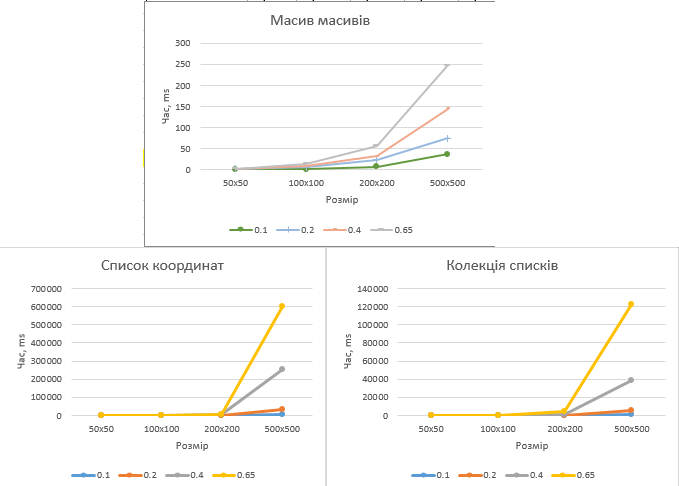


Рис. 1. Результати роботи читання та запису елементів при ініціалізації

1. Додавання матриць

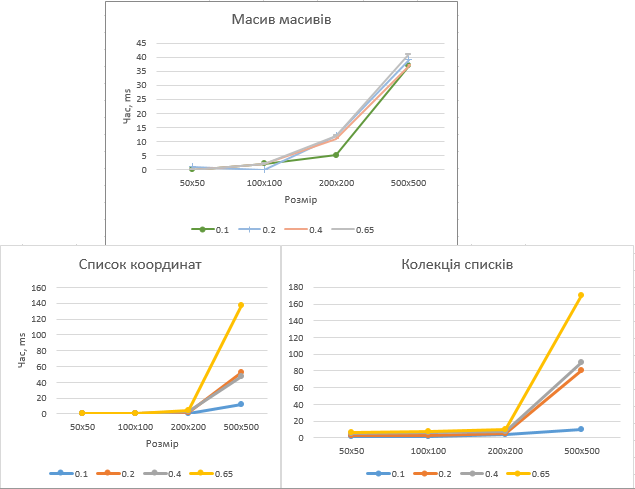


Рис. 2. Результати роботи додавання матриць

1. Множення на число

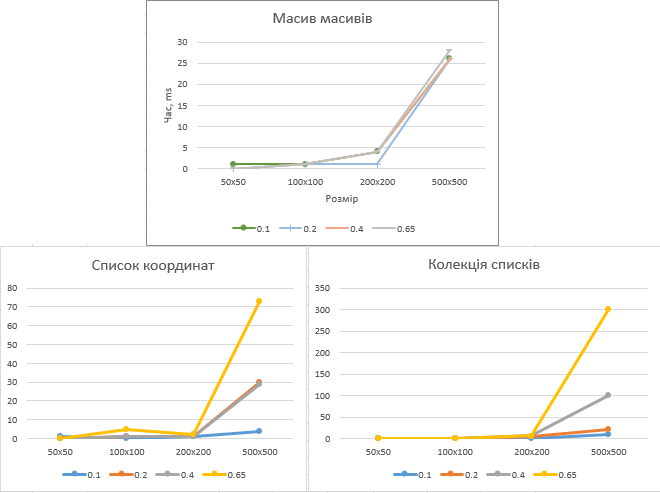


Рис. 3. Результати роботи множення на число

1. Множення на матрицю

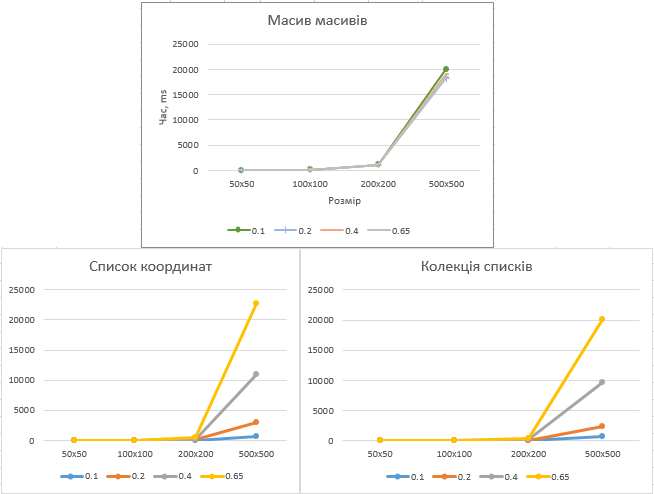


Рис. 4. Результати роботи множення на матрицю

1. Транспонування

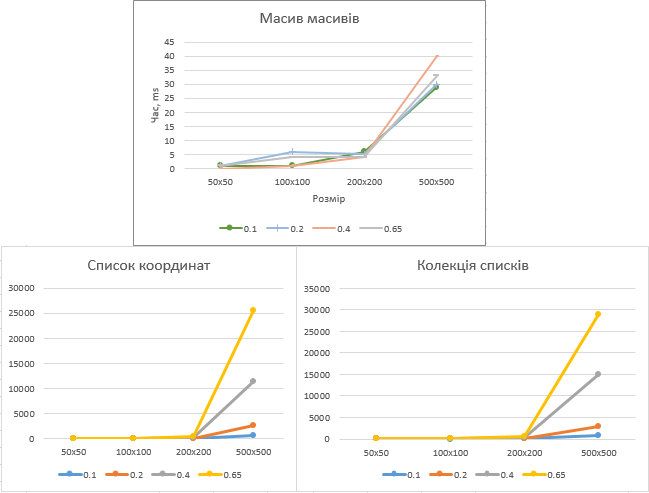


Рис. 5. Результати роботи транспонування