**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ІВАНА ФРАНКА**

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Кафедра програмування

(повна назва кафедри)

**КУРСОВА РОБОТА**

на тему:

Візуалізація розріджених матриць в середовищі Pharo мовою Smalltalk

Студента 4 курсу, групи ПМІ-41 ,

напряму підготовки інформатика

Кормушина Я.К.

(прізвище та ініціали)

Керівник доц. Ярошко С.А.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Львів – 2018

ЗМІСТ

[Вступ 3](#_Toc513981539)

[1. Формулювання завдання 5](#_Toc513981540)

[2. Аналіз проблеми 8](#_Toc513981541)

[3. Огляд засобів Pharo 9](#_Toc513981542)

[3.1. Основні інструменти середовища 9](#_Toc513981543)

[3.2. Візуалізаційний двигун Roassal 12](#_Toc513981544)

[3.2.1. Стуктурна модель Roassal 12](#_Toc513981545)

[3.2.2. Створення відображення 13](#_Toc513981546)

[3.3. Розріджені матриці 15](#_Toc513981547)

[3.3.1. Матриця на основі списку координат 15](#_Toc513981548)

[3.3.2. Матриця на основі колекції списків 18](#_Toc513981549)

[3.3.3. Матриця на основі мережі списків 20](#_Toc513981550)

[4. Реалізація 23](#_Toc513981551)

[4.1. Модифікація класів розріджених матриць 25](#_Toc513981552)

[4.2. Візуалізатори 27](#_Toc513981553)

[4.2.1. Візуалізатор для матриці на основі списку координат 28](#_Toc513981554)

[4.2.2. Візуалізатор для матриці на основі колекції списків 30](#_Toc513981555)

[4.2.3. Візуалізатор для матриці на основі мережі списків 32](#_Toc513981556)

[4. Аналіз отриманих результатів 33](#_Toc513981557)

[5. Висновки 39](#_Toc513981558)

[Список використаної літератури 40](#_Toc513981559)

# Вступ

**Розріджена матриця** - це матриця, більша частина елементів якої дорівнює нулю. Концептуально, розрідженість відповідає системам, які слабо зв'язані. Це поняття 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, де застосовується метод скінченних елементів.

Вагому роль в таких областях відіграють розріджені матриці, які дозволяють зберігати лише певну частину даних, тим самим зменшуючи час виконання обчислень та обсяг пам’яті, необхідний для їх збереження.

Існує багато способів реалізації розріджених матриць, кожен з яких має свої переваги та недоліки. Деякі з них мають доволі складні внутрішні структури, формальний опис яких зрозуміти важче, ніж їх графічну репрезентацію.

**Об’єкт дослідження:** графічна репрезентація розріджених матриць на основі списку координат, колекції списків, мережі списків.

**Предмет дослідження**: графічна репрезентація розріджених матриць.

**Мета дослідження**: спроектувати, реалізувати, випробувати і описати програмні засоби, які дають змогу відобразити графічно внутрішню структуру розроблених раніше розріджених матриць.

**Гіпотеза дослідження**:

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

1. Розглянути і дослідити доступні графічні програмні засоби мовою Smalltalk в середовищі Pharo 6.1;
2. Спроектувати та реалізувати програмні засоби для графічного відображення внутрішньої структури розріджених матриць;
3. Випробувати і описати ці програмні засоби.

**Наукова новизна дослідження** полягає у тому, що засобів для відображення внутрішньої структури розріджених матриць у бібліотеках Pharo Smalltalk немає, а їх наявність покращить вивчення цих структур і алгоритмів, що над ними діють, зокрема додавання, множення тощо.

# Формулювання завдання

Реалізувати програмні засоби для відображення внутрішньої структури розріджених матриць засобами середовища Pharo Smalltalk 6.1. Внутрішні структури матриць відповідають наступним моделям:

* список координат;
* колекція списків;
* мережа списків.

Випробувати ці засоби, описати їх дію.

Структура «список координат» відповідає такій схемі для загального випадку матриці розмірністю [nx m] елементів:

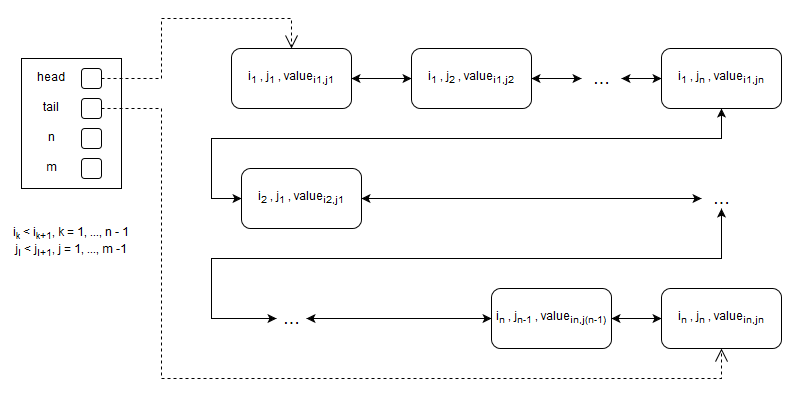


Рисунок 1. Внутрішня структура "список координат"

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:

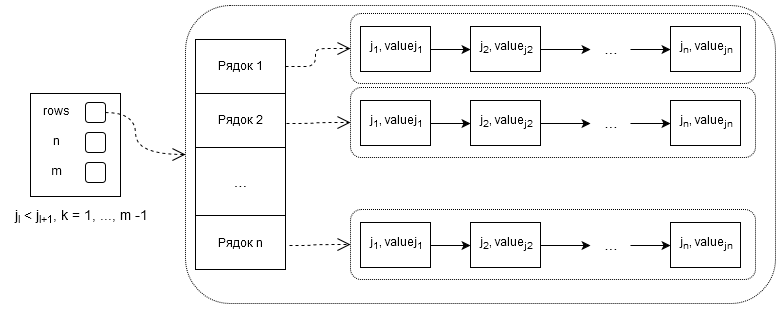


Рисунок 2. Внутрішня структура "колекція списків"

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

Структура «мережа списків» відповідає такій схемі:

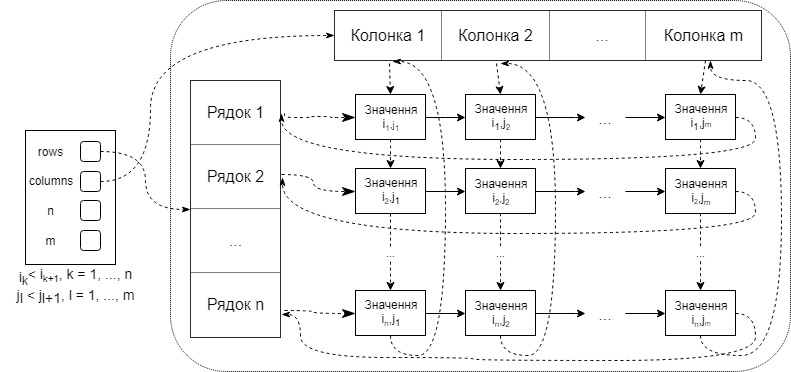


Рисунок 3. Внутрішня структура "мережа списків"

Ця реалізація зберігає по одному елементу (заголовку) на кожен рядок і стовпець, де елементи (значення, рядок, стовпець) розташовані в порядку зростання індексу рядка і стовпця відповідно.

Візуалізація розріджених матриць на основі цих реалізацій передбачає відображення на екрані елементів матриці, включно з допоміжними елементами (заголовками), у форматі сітки. Тобто елементи будуть розташовані так, ніби вони знаходяться в таблиці, але нульові значення не відображатимуться. Кожен елемент буде квадратом, який міститиме значення елемента та позицію всередині матриці, а зв’язки між елементами будуть представлені напрямленими ребрами – суцільними лініями зі стрілками.

# 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.

Розглянемо розріджену матрицю на прикладі (Рис. 4):

Рисунок 4. Приклад розрідженої матриці

Найпростіша програмна реалізація матриці розмірності [n х m] – 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.

Відповідно до цих вимог були розроблені розріджені матриці з різними внутрішніми структурами, які надають їм перевагу перед звичайними двовимірними масивами. Нерідко такі структури є складними для розуміння без графічного представлення, зокрема такими є спискові структури, які містять багато внутрішніх зв’язків між елементами.

# Огляд засобів Pharo

**Pharo** — це сучасна повнофункціональна реалізація середовища мовою Smalltalk з відкритим вихідним кодом.

Для реалізації нам потрібна віртуальна машина Pharo та образ середовища Pharo. Їх можна завантажити з офіційного сайту Pharo за посиланням: <http://pharo.org/download>. Ми будемо використовувати останню версію Pharo 6.1 станом на травень 2018 року.

## 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.

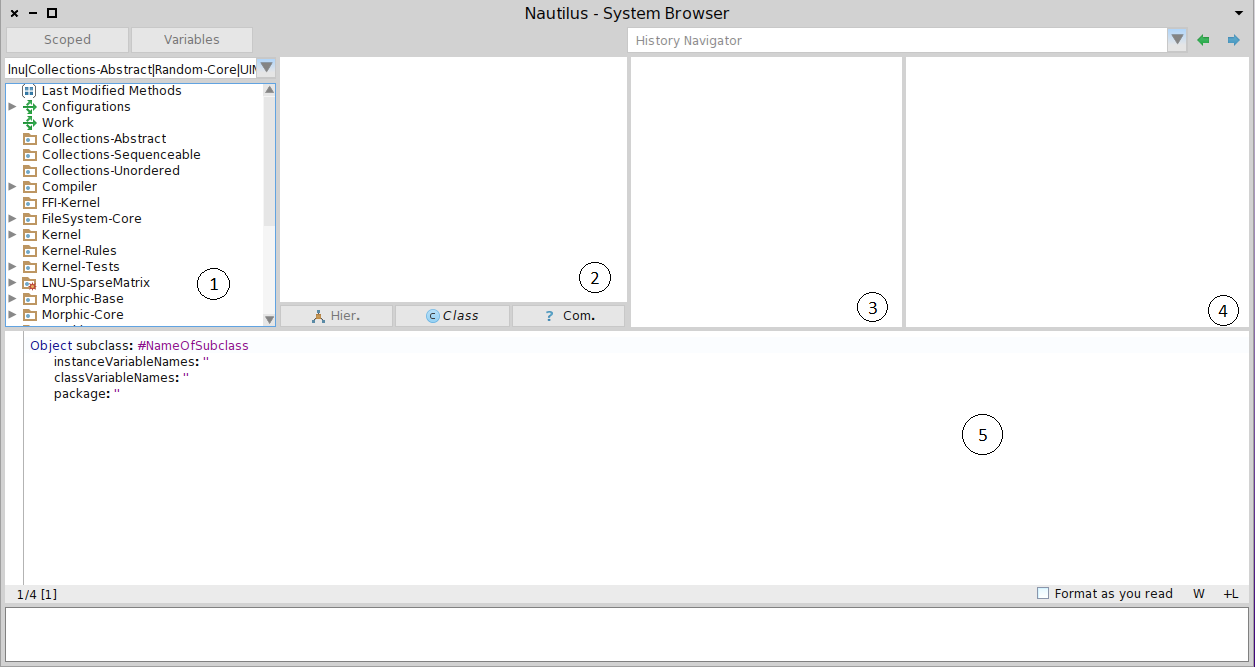


Рисунок 5. Переглядач Nautilus

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.

***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. 6).

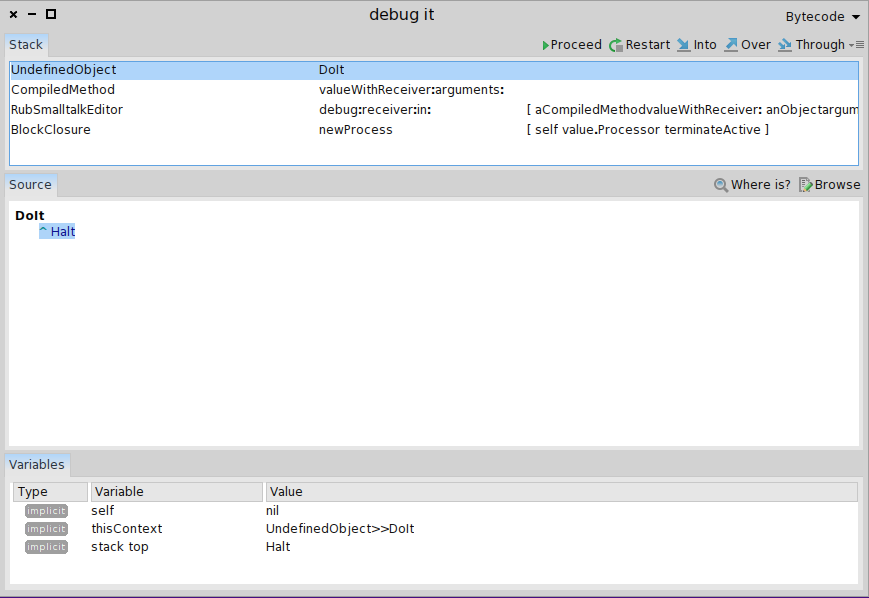


Рисунок 6. Вікно Debugger

**Spotter** використовують для навігації по середовищу. З його допомогою можна швидко знайти необхідний клас чи метод і переглянути його за допомогою Nautilus. Для цього передбачені спеціальні ключові слова, які відфільтровують результати пошуку (Рис. 7).

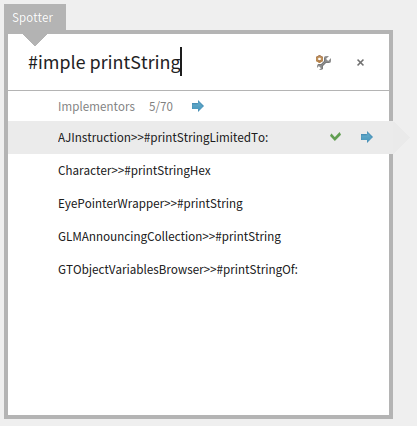


Рисунок 7. Вікно Spotter

**Iceberg** – це набір інструментів, які дозволяють взаємодіяти з репозиторіями git безпосередньо з образу Pharo. Iceberg полегшує контроль версій і має інтуїтивно зрозумілий інтерфейс (Рис. 8).

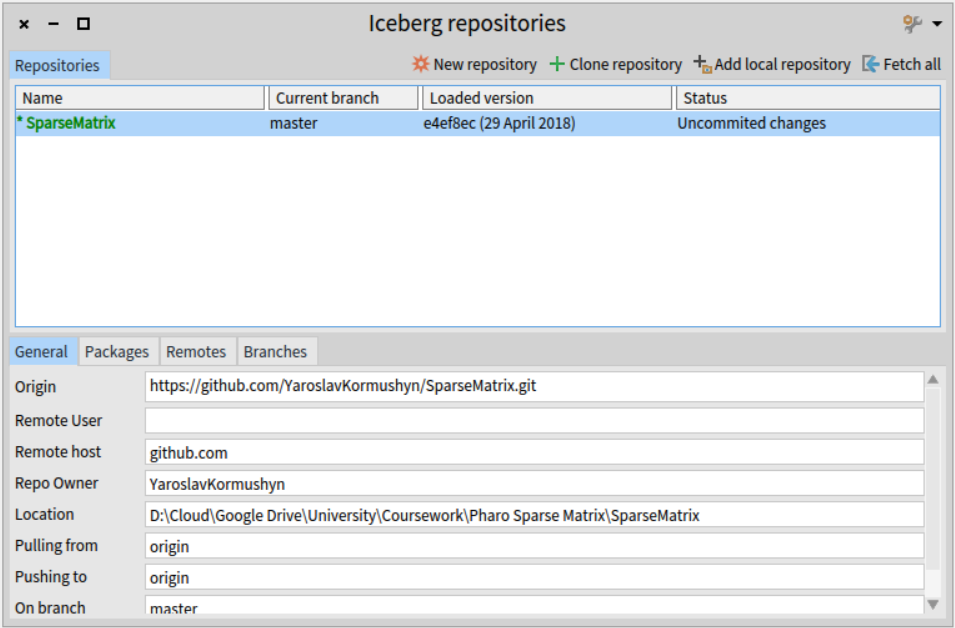


Рисунок 8. Вікно Iceberg

## 3.2. Візуалізаційний засіб Roassal

**Roassal** – це гнучкий програмний засіб, створений для візуалізації та взаємодії з довільними даними, визначеними на основі об'єктів та їх зв'язків. Він написаний за допомогою мов Pharo та VisualWorks. Roassal зазвичай використовується для створення інтерактивної візуалізації, а діапазон його застосування різноманітний. Наприклад, спільнота Moose використовує Roassal для візуалізації програмного забезпечення.

### 3.2.1. Структурна модель Roassal

Roassal відображає об'єкти та зв'язки між ними як графічні елементи та ребра. Значення об’єктів трансформуються у візуальні розміри, такі як ширина, висота чи інтенсивність кольору. Відображення об'єктів у візуальні атрибути – це виразний спосіб створення гнучкої та насиченої візуалізації, а просте задання такого відображення значно зменшує час та зусилля необідні для створення візуалізації.

Roassal структурує візуалізацію з точки зору виглядів, елементів, форм, взаємодій та анімацій. Вигляд – це контейнер графічних елементів. Елементи можна додавати та видаляти з вигляду. Більшість візуалізацій містить один єдиний вигляд, який містить всі елементи Roassal.

Елемент – це графічне зображення об'єкта, яке зазвичай описує довільне явище (наприклад, інформацію про сейсмічну активність, файл, що зберігається на жорсткому диску, тощо). Користувач бачить елементи та взаємодіє з ними за допомогою миші та клавіатури. Елемент описується формою, яка визначає її візуальне представлення. Форма описує примітивне візуальне зображення, таке як прямокутник, коло, рядок або мітка. Форми можна об'єднувати для отримання значно складніших форм. Елемент Roassal – складений об'єкт, що містить двовимірне розташування в просторі, сукупність взаємодій, комбінацію форм та об'єктну модель, яка є довільним об'єктом, що відображається. Щоб бути видимими, елементи повинні мати принаймні одну форму.

Взаємодія – це певна дія, яку може викликати користувач. Діапазон підтримуваних взаємодій широкий: перетягування елементів, відображення спливаючого тексту елемента при наведенні миші, виділення інших елементів, відкривання меню при клацанні правою кнопкою миші на елементі тощо.

Коренем кожної візуалізації в Roassal виступає екземпляр класу RTView, який є контейнером для всіх графічних компонентів, що будуть відображені. Ці компоненти це екземпляри підкласів RTObject, як правило RTElement та RTEdge.

### 3.2.2. Створення відображення

Розглянемо приклад створення деревовидного відображення над числами від 1 до 50. Запишемо наступний код:

1. v := RTView new.

2. shape := RTBox new color: Color blue trans.

3. elements := shape elementsOn: (1 to: 50).

4. v addAll: elements.

5. elements @ RTPopup.

6.

7. RTEdgeBuilder new

8. view: v;

9. objects: (1 to: 50);

10. connectFrom: [ :i | i // 3 ].

11.

12. RTTreeLayout on: elements.

13. v

Результатом його виконання буде відображення на Рис.9.

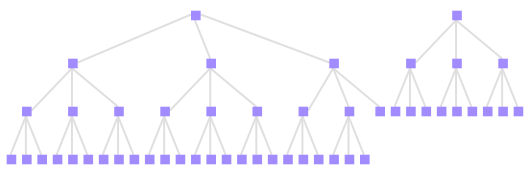


Рисунок 9. Відображення Roassal

Перший рядок коду створює новий вигляд як екземпляр класу RTView. В другому рядку створюється форма напівпрозорого синього квадрата. Тоді ця форма використовується як фабрика елементів за допомогою повідомлення elementsOn:. Це повідомлення приймає список об’єктів моделей і повертає об’єкт типу RTGroup – список з RTElement. Кожен елемент має посилання на об’єкт моделі, тобто натуральне значення від 1 до 50. Тоді ці елементи додаються до вигляду з першого рядка. Вираз @ RTPopup вказує, що при наведенні миші на елемент буде відображено спливаючий рядок, який є рядковим представленням моделі відповідного елемента.

Відношення між елементами задаються за допомогою класу RTEdgeBuilder – будівника ребер. В сьомому рядку ми створюємо екземпляр цього класу і параметризуємо його виглядом, який створили вище. Будівник ребер обирає по два елементи із заданого вигляду за правилом, яке ми йому задаємо, і проводить між ними ребро. В нашому випадку ми задали правило «для кожного числа i від 1 до 50 провести ребро між ‘і’ та ‘і // 3’». Повідомлення // відповідає цілочисловій частці від ділення.

## 3.3. Розріджені матриці

Необхідно візуалізувати розріджені матриці з трьома різними внутрішніми структурами, а саме: список координат, колекція списків і мережа списків. Наведемо стислий опис всіх структурних елементів цих класів. Всі класи належать до пакету LNU-SparseMatrix.

### 3.3.1. Матриця на основі списку координат

Реалізація цієї матриці включає клас елемента матриці і клас, що представляє саму матрицю.

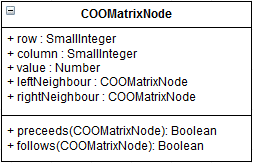


Рисунок 10. Клас COOMatrixNode

COOMatrixNode – клас, що описує елемент для реалізації структури списку координат (Рис. 10). 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);
* методи preceeds: і follows:, які повертають результати типу Boolean, залежно від того, чи екземпляр, що отримав повідомлення, знаходиться в списку матриці лівіше чи правіше від аргументу.

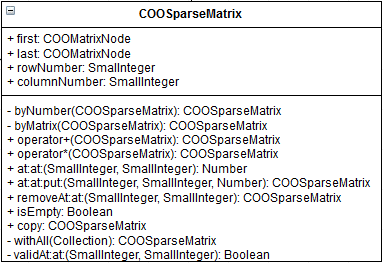


Рисунок 11. Клас COOSparseMatrix

COOSparseMatrix – клас, що описує матрицю для реалізації структури списку координат (Рис. 11). 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).

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;
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;
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.

### 3.3.2. Матриця на основі колекції списків

Реалізація цього типу матриці включає клас елемента матриці і клас, що представляє саму матрицю.

LILSparseMatrixNode – клас, що відображає елемент матриці для реалізації структури колекції списків (Рис. 12). Він містить такі змінні:

* стовпець елемента матриці;
* значення елемента матриці.

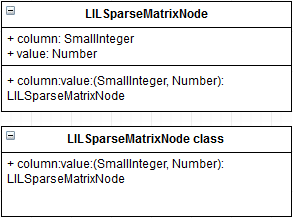


Рисунок 12. Клас LILSparseMatrixNode

Цей клас містить зокрема і метод ініціалізації цих змінних.

LILSparseMatrix – це клас, що описує матрицю для реалізації структури колекції списків (Рис. 13), 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).

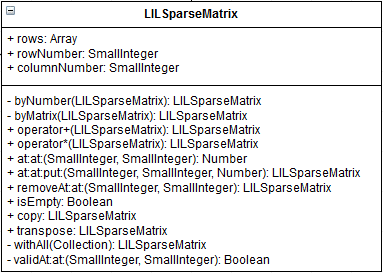


Рисунок 13. Клас LILSparseMatrix

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;
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.

### 3.3.3. Матриця на основі мережі списків

Реалізація цього типу матриці включає клас елемента матриці і клас, що представляє саму матрицю.

TWMatrixNode – клас, що відображає елемент матриці для реалізації структури мережі списків (Рис. 14). Він містить такі змінні:

* рядок елемента матриці;
* стовпець елемента матриці;
* значення елемента матриці.

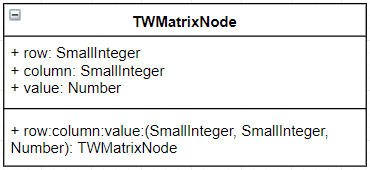


Рисунок 14. Клас TWMatrixNode

TWSparseMatrix – це клас, що описує матрицю для реалізації структури мережі списків (Рис. 15), який містить:

* масив заголовків списків рядків (rows) – екземпляр Array, де кожен елемент це екземпляр класу TWMatrixNode, що виступає заголовком до списку рядка;
* масив заголовків списків стовпців (columns) – екземпляр Array, де кожен елемент це екземпляр класу TWMatrixNode, що виступає заголовком до списку стовпця;
* кількість рядків (rowNumber) – ціле число (екземпляр класу SmallInteger);
* кількість стовпців (columnNumber) – ціле число (екземпляр класу SmallInteger).

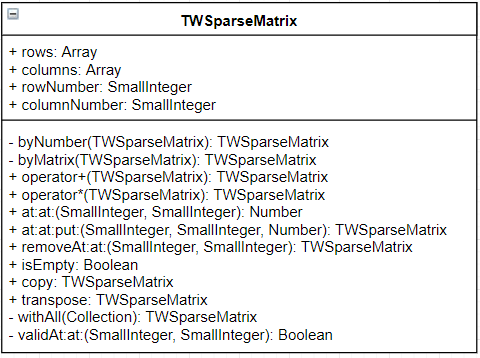


Рисунок 15. Клас TWSparseMatrix

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 заголовки списків елементів матриці по рядках;
   * columns – повертає масив типу Array, що містить заголовки списків елементів матриці по стовпцях.
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;
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.

# 4. Реалізація

Додамо новий тег до пакету LNU–SparseMatrix під назвою Visual (Рис. 16), в якому будемо оголошувати класи для візуалізації.

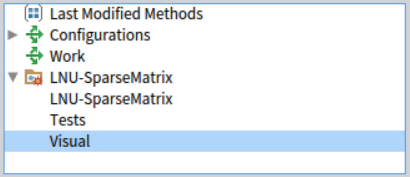


Рисунок 16. Тег Visual в пакеті LNU-SparseMatrix

Візуалізатори міститимуть деяку спільну поведінку, яку варто винести в окремий узагальнений клас. Вони наслідуватимуть клас MatrixVisualizer (Рис. 17), який буде абстракцією візуалізатора для розріджених матриць. MatrixVisualizer, в свою чергу, наслідуватиме клас Object, кореневий клас будь-якого об’єкту системи Pharo.

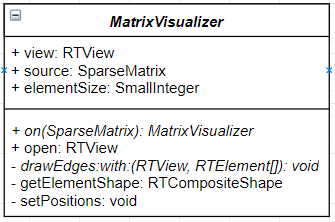


Рисунок 17. Клас MatrixVisualizer

MatrixVisualizer – клас абстракції візуалізаторів розріджених матриць, який містить:

* вигляд (view) на якому буде відображено матрицю – екземпляр RTView;
* посилання на матрицю, яка є прообразом відображення (source);
* розмір відображення елемента (elementSize) – ціле число (екземпляр класу SmallInteger).

Визначені наступні методи:

* on: – абстрактний метод, який отримує розріджену матрицю, яку потрібно відобразити, як аргумент, і повертає екземпляр візуалізатора для цієї матриці;
* open – метод, який повертає вигляд матриці-джерела;
* drawEdgesWith: – абстрактний метод, який отримує елементи, між якими потрібно провести ребра, і додає до змінної вигляду ці ребра;
* getElementShape – метод, який повертає форму для елементів відображення;
* setPositions – метод, який встановлює позиції елементів відображення.

Розглянемо детальніше метод getElementShape (Рис. 18):

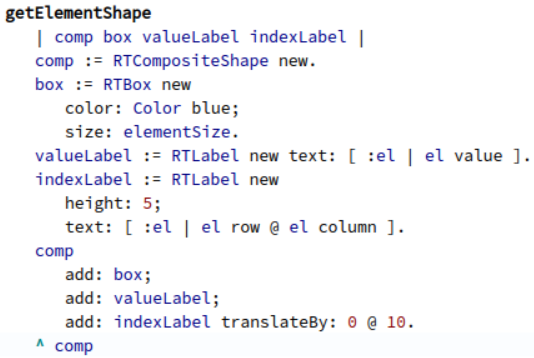


Рисунок 18. Метод getElementShape

Цей метод повертає екземпляр класу RTCompositeShape, який відображає складену форму з прямокутника і двох текстових міток. Прямокутнику задається синій колір і розмір, визначений в змінній elementSize. Мітка valueLabel відображає значення елемента, а мітка indexLabel – позицію елемента в матриці.

Метод setPositions (Рис. 19):

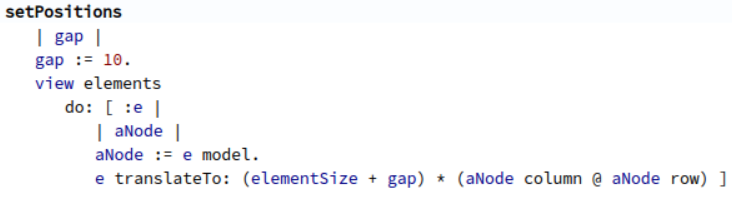


Рисунок 19. Метод setPositions

В цьому методі ми задаємо розташування елементів, які вже були додані до нашого вигляду, за допомогою методу translateTo:, який змінює позицію елемента відображення відповідно до позиції елемента в матриці.

## 4.1. Модифікація класів розріджених матриць

Для побудови відображення розрідженої матриці нам необхідно встановити певний протокол, за яким ми будемо отримувати елементи цієї матриці в класі візуалізатора. Тому для кожної реалізації розрідженої матриці визначимо метод elementsAsCollection, який буде повертати всі елементи матриці (включно з заголовками списків, якщо такі існують) всередині OrderedCollection.

Визначення цього методу для матриці на основі списку координат (Рис. 20):

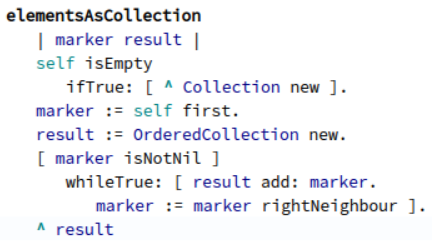


Рисунок 20. elementsAsCollection в COOSparseMatrix

Оскільки елементи цієї матриці містять усю необхідну інформацію для візуалізатора, а саме значення елемента та його позицію в матриці, то достатньо повернути колекцію всіх елементів матриці.

Визначення цього методу для матриці на основі колекції списків (Рис. 21):

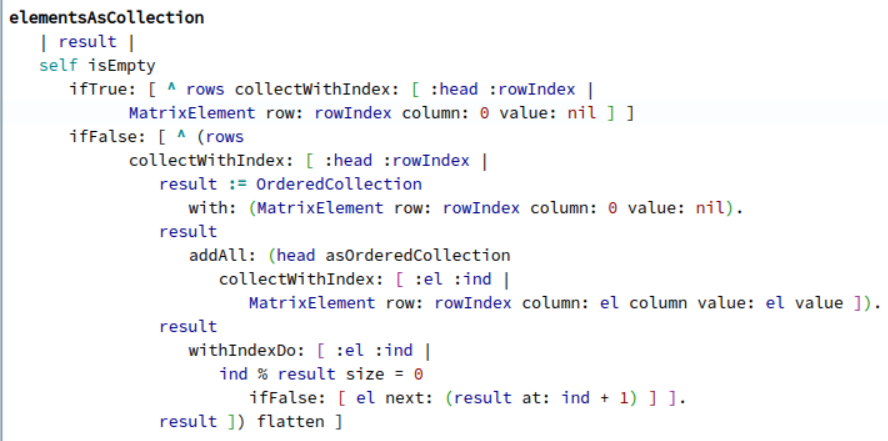


Рисунок 21. elementsAsCollection в LILSparseMatrix

Елементи цього типу матриці не містять усієї інформації, щоб задати їх відображення, тому був створений допоміжний клас MatrixElement, який містить позицію елемента в матриці, його значення і посилання на безпосереднього сусіда правіше в рядку матриці, якщо такий існує.

Визначення цього методу для матриці на основі мережі списків (Рис. 22):

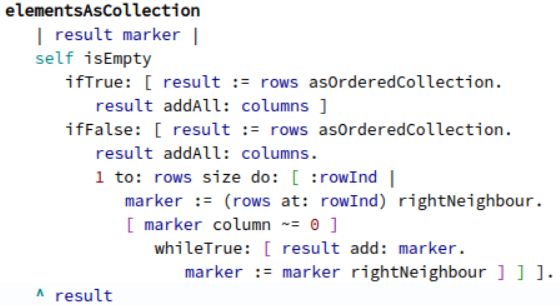


Рисунок 22. elementsAsCollection в TWSparseMatrix

Як і з реалізацією на основі списку координат, елементи цієї матриці містять вичерпну інформацію про себе, необхідну для візуалізатора, тому достатньо повернути колекцію цих елементів, включно з заголовками списків рядків і стовпців.

## 4.2. Візуалізатори

Візуалізатори для кожного типу розрідженої матриці мають спільну поведінку, яка визначена в батьківському класі MatrixVisualizer. Проте деякі речі, зокрема інтерпретація колекції елементів, отриманих з матриці-джерела, та задання відношень між елементами необхідно визначити для кожного візуалізатора окремо, в силу різниці внутрішньої структури матриць.

Ієрархія класів наступна (Рис. 23):

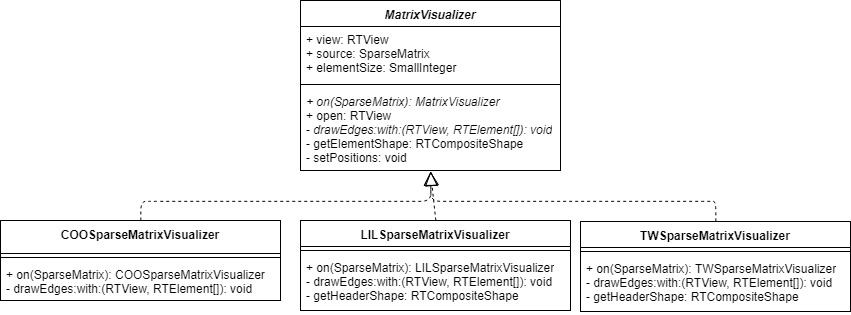


Рисунок 23. Ієрархія класів візуалізаторів

### 4.2.1. Візуалізатор для матриці на основі списку координат

Визначимо метод on: для класу COOSparseMatrixVisualizer так (Рис. 24):

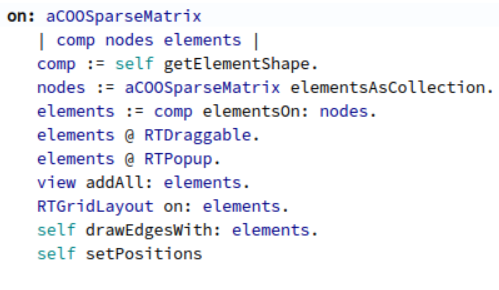


Рисунок 24. Метод on: для класу COOSparseMatrixVisualizer

Хід роботи методу on: такий:

1. Отримуємо форму для елементів матриці;
2. Отримуємо колекцію елементів матриці, яку будемо відображати;
3. Створюємо елементи відображення на основі елементів матриці. Повідомлення elementsOn: повертає об’єкт типу RTGroup, який містить всі елементи відображення;
4. Налаштовуємо елементи відображення так, щоб їх можна було перетягувати (@ RTDraggable) і при клацанні мишки відкривався переглядач елемента матриці для цього елемента відображення (@ RTPopup);
5. Додаємо налаштовані елементи відображення до вигляду;
6. Застосовуємо стиль відображення у вигляді сітки до елементів (RTGridLayout on:);
7. Додаємо зв’язки між елементами на відображення (drawEdgesWith:);
8. Задаємо позиції елементів відображення (setPositions).

Розглянемо метод drawEdgesWith: (Рис. 25), який додає ребра між елементами:

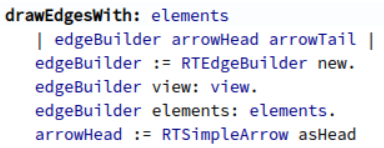




Рисунок 25. Метод drawEdgesWith: для COOSparseMatrixVisualizer

Хід роботи методу drawEdgesWith: такий:

1. Створюється новий екземпляр класу RTEdgeBuilder і параметризується нашим виглядом і елементами відображення;
2. Визначається стиль ребер, які будуть додаватися для елементів, що знаходяться в одному рядку матриці, а саме:
   1. Повідомлення styledLine визначає, що ребро буде суцільною стилізованою лінією типу RTStyledMultiLine з ортогональними перегинами;
   2. orthoHorizontalLine визначає, що це ребро буде виходити з елементів і входити в елементи в горизонтальному положенні;
   3. horizontalAttachPoint визначає, що точки виходу і входу ребра будуть розташовані з правого і лівого боків елемента відповідно;
   4. За допомогою head: і tail: ми визначаємо стиль ребра в точці входу в елемент і виходу з елемента.
3. За допомогою connectTo: визначається, що ребро з елемента el буде проведено до його безпосереднього правого сусіда, якщо він знаходиться в тому ж рядку;
4. Визначається новий стиль ребер, які будуть додаватися для елементів, що знаходяться в різних рядках матриці. Відмінність з пунктом 2 полягає в тому, що ребра будуть мати точки виходу і входу відповідно знизу і згори елементів і будуть розташовані у вертикальному положенні;
5. За допомогою connectTo: визначається, що ребро з елемента ‘el’ буде проведено до його безпосереднього правого сусіда, якщо він знаходиться в наступному рядку.

### 4.2.2. Візуалізатор для матриці на основі колекції списків

Оскільки матриця на основі колекції списків містить заголовки для рядків, то необхідно визначити для них нову форму відображення. Додамо до класу LILSparseMatrixVisualizer метод getHeaderShape (Рис. 26).

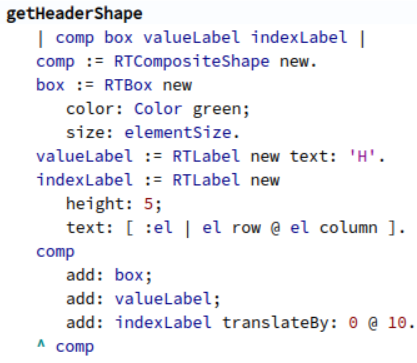


Рисунок 26. Метод getHeaderShape

Основна різниця з формою елемента полягає в тому, що колір прямокутника для заголовка буде зеленим, а мітка значення міститиме підпис ‘H` (Header).

Метод on: визначений аналогічно до класу COOSparseMatrixVisualizer, окрім створення елементів відображення за елементами матриці (Рис. 27).

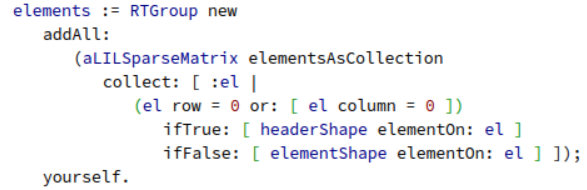


Рисунок 27. Відмінність методу on: для LILSparseMatrixVisualizer

Тут створення відображення для елементів-заголовків і елементів матриці різні.

Метод drawEdgesWith: визначений аналогічно до класу COOSparseMatrixVisualizer, але:

* ребра відображення мають точки виходу і входу відповідно з правого і лівого боку елементів, і розташовані в горизонтальному положенні;
* правило, за яким додаються ребра, визначене відповідно до внутрішньої структури матриці.

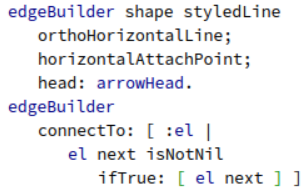


Рисунок 28. Відмінність методу drawEdgesWith: для LILSparseMatrixVisualizer

### 4.2.3. Візуалізатор для матриці на основі мережі списків

Ця реалізація матриці також містить заголовки списків рядків і стовпців, тому був створений метод getHeaderShape, аналогічний до однойменного методу в класі LILSparseMatrixVisualizer.

Метод on: визначений аналогічно до LILSparseMatrixVisualizer, а метод drawEdgesWith: містить правила додавання ребер, які відповідають структурі «мережа списків» (Рис. 29).

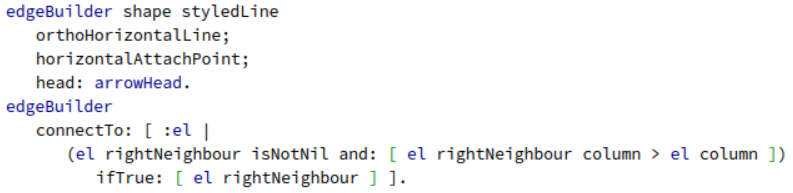


Рисунок 29. Правило для списків рядків

На Рис. 29 визначене правило для списків рядків, без врахування зв’язку замикання останнього елемента рядка матриці і заголовка рядка.

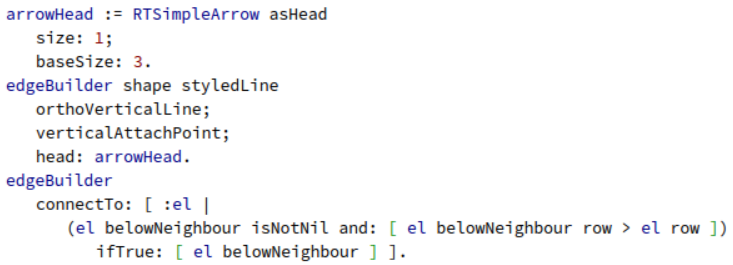


Рисунок 30. Правило для списків стовпців

На Рис. 30 визначене правило для списків стовпців, без врахування зв’язку замикання останнього елемента стовпця і заголовка стовпця.

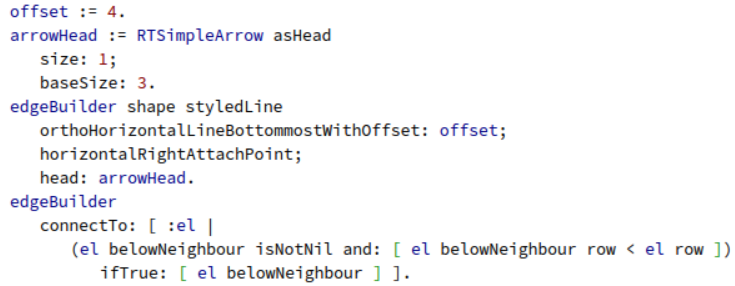


Рисунок 31. Замикання рядків

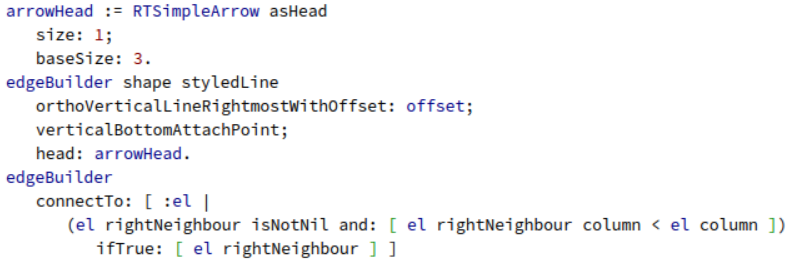


Рисунок 32. Замикання стовпців

На Рис. 31 та Рис. 32 визначені правила для зв’язків замикання списків рядків і стовпців відповідно. Особливу увагу варто звернути на повідомлення, які задають стиль ребер, а саме:

* orthoHorizontalLineBottommostWithOffset: задає горизонтальне ребро, точки перегину якого розташовані під елементом, який знаходиться найнижче у відображенні, з деяким зміщенням offset;
* orthoVerticalLineRightmostWithOffset: задає вертикальне ребро, точки перегину якого розташовані справа від елемента, який знаходиться найправіше у відображенні, з деяким зміщенням offset;
* horizontalRightAttachPoint визначає, що точки виходу і входу будуть розташовані з правого боку елементів відображення;
* verticalBottomAttachPoint визначає, що точки виходу і входу будуть розташовані знизу елементів відображення.

# 4. Аналіз отриманих результатів

Для перегляду роботи класів візуалізації використаємо таку матрицю:

Представимо її в середовищі Pharo так:

col := { { 1 . -1 . 0 . 0 } .

{ 0 . 3 . 0 . 0 } .

{ 1 . 0 . 2 . 1 } .

{ 0 . 0 . 0 . 0 } }.

Створимо на основі цієї колекції три матриці кожного типу: списку координат, колекції списків і мережі списків:

coo := COOSparseMatrix withAll: col.

vis\_coo := COOSparseMatrixVisualizer on: coo.

lil := LILSparseMatrix withAll: col.

vis\_lil := LILSparseMatrixVisualizer on: lil.

tw := TWSparseMatrix withAll: col.

vis\_tw := TWSparseMatrixVisualizer on: tw.

Відкриємо кожне відображення окремо за допомогою методу open.

Для матриці на основі списку координат відображення має наступний вигляд:

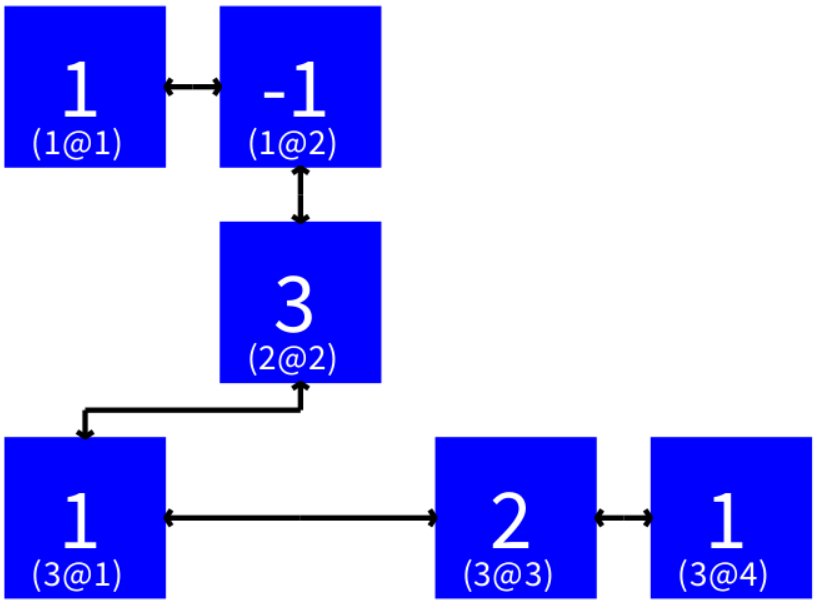


Рисунок 33. Відображення для COOSparseMatrix

Для матриці на основі колекції списків:

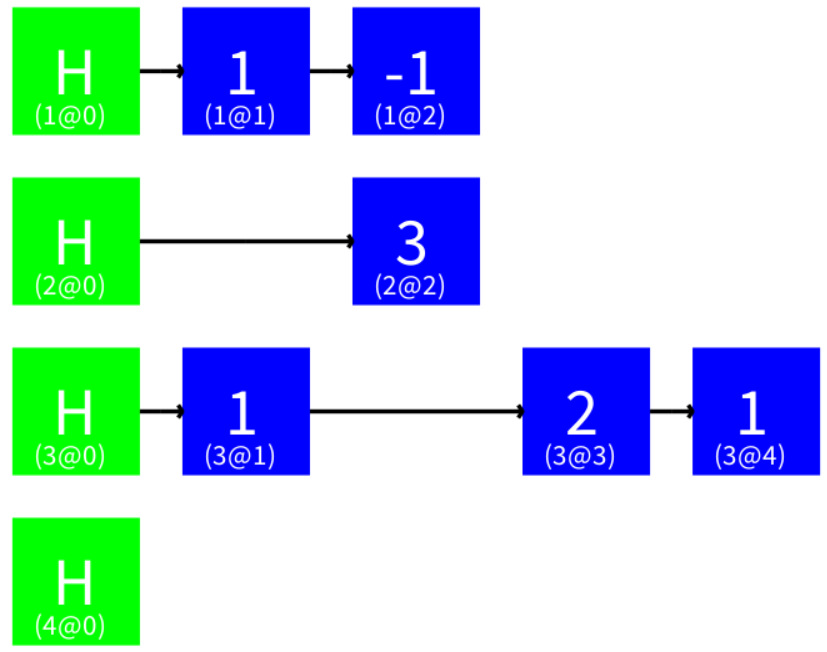


Рисунок 34. Відображення для LILSparseMatrix

Для матриці на основі мережі списків:

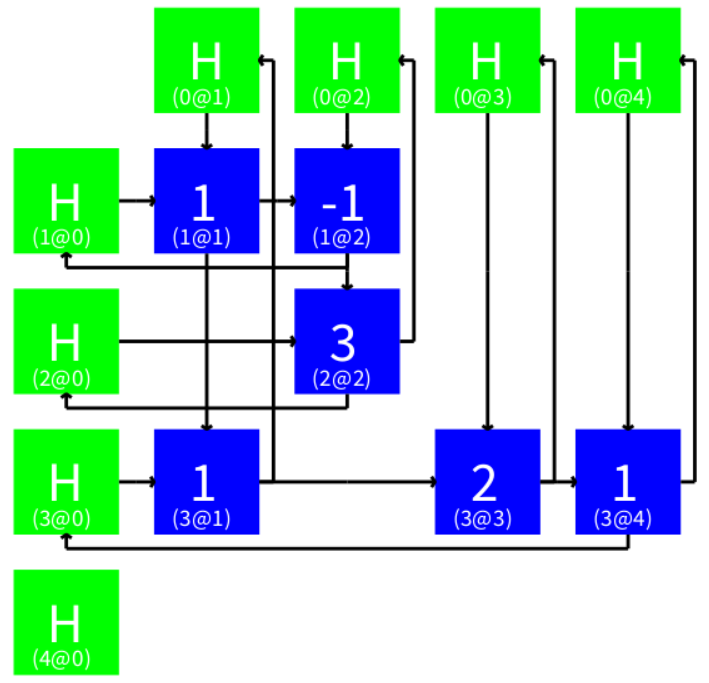


Рисунок 35. Відображення для TWSparseMatrix

Тепер задамо нову колекцію розмірністю [100 x 100], де лише 20 елементів будуть ненульовими, створимо на її основі розріджені матриці і відобразимо їх.

rand := Random new.

col := (1 to: 100) collect: [ :ind | OrderedCollection ofSize: 100 ].

1 to: 20 do: [ :ind |

| first second |

first := rand nextInt: 100.

second := rand nextInt: 100.

(col at: first) at: second put: (rand nextInt: 100)].

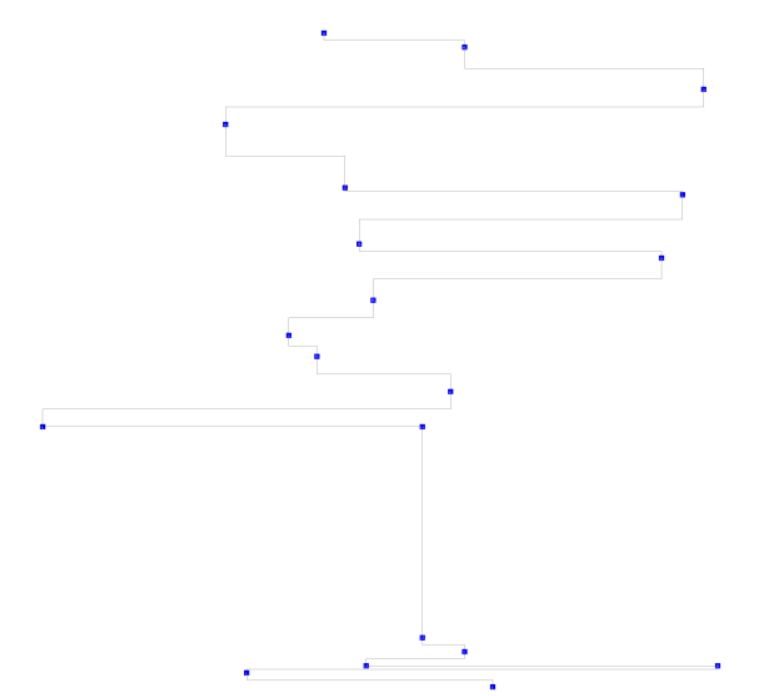


Рисунок 36. Відображення матриці COOSparseMatrix

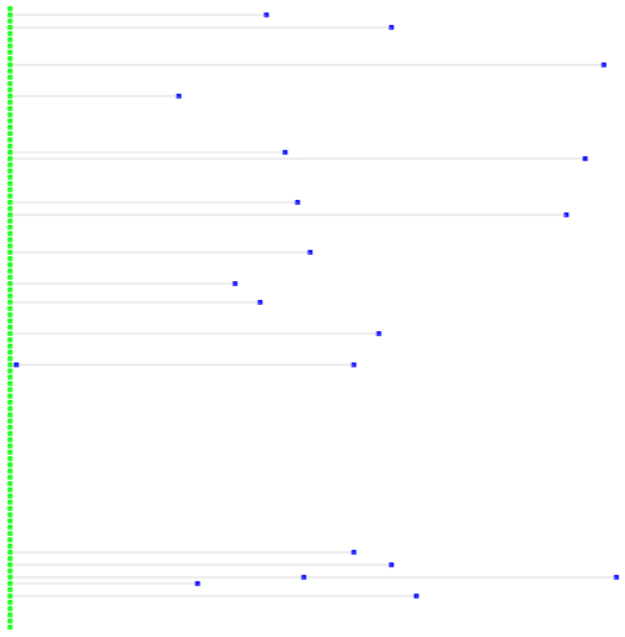


Рисунок 37. Відображення матриці LILSparseMatrix

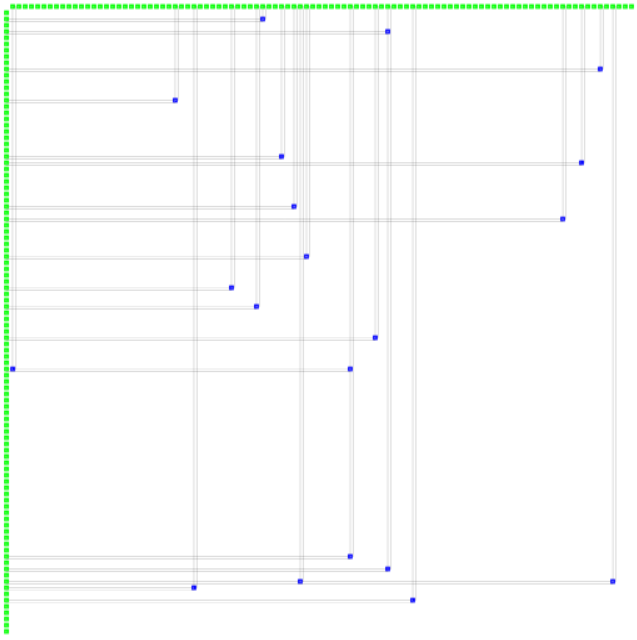


Рисунок 38. Відображення матриці TWSparseMatrix

Як можна бачити, візуалізатори коректно відобразили одну й ту ж матрицю залежно від внутрішньої структури певної реалізації. При потребі можна масштабувати відображення за допомогою кнопок  розташованих на панелі інструментів. Більше того, всі елементи відображення інтерактивні, тобто при натисканні на елемент відкриється його модель – відповідний елемент матриці (Рис. 39).

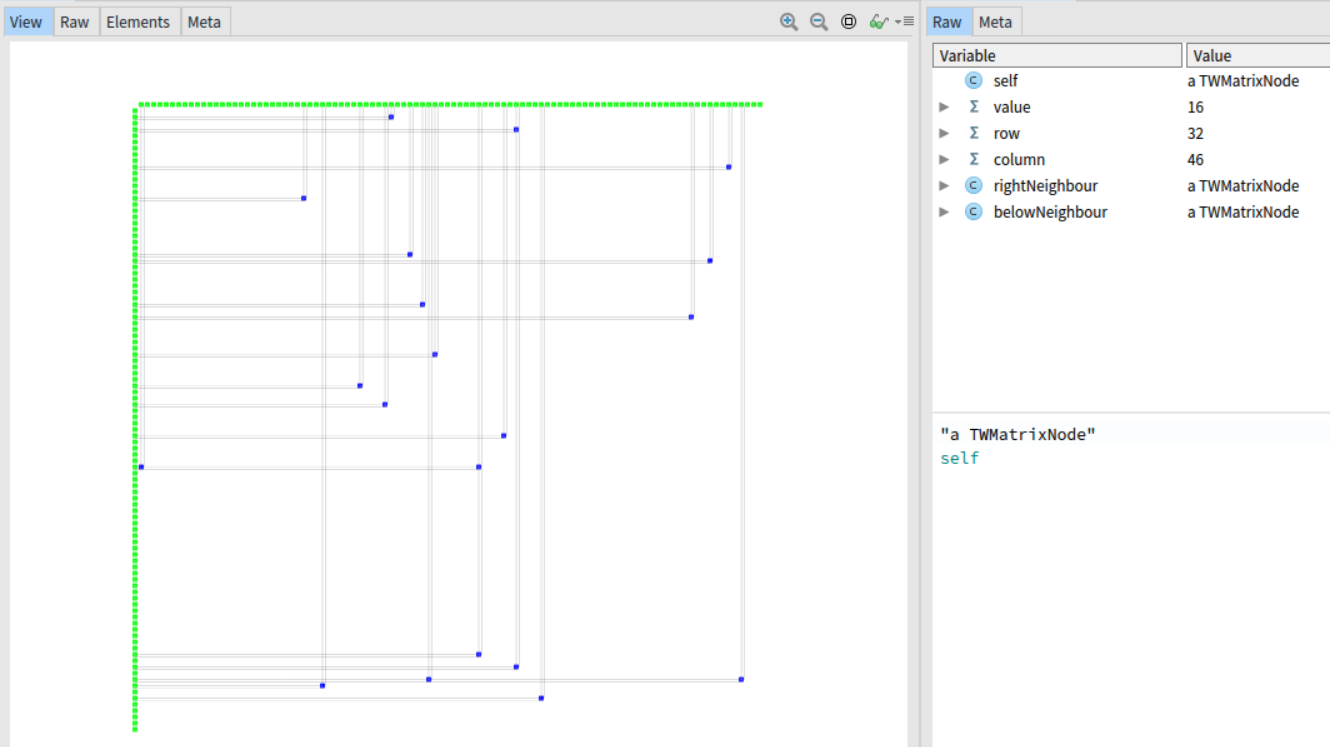


Рисунок 39. Елемент (32;46) матриці TWSparseMatrix

# 5. Висновки

В ході виконання роботи я:

* поглибив знання про середовище розробки Pharo та закріпив навички роботи зі стандартною бібліотекою Pharo Smalltalk 6.1;
* вивчив основні складові візуалізаційного засобу Roassal;
* реалізував програмні засоби для візуалізації розріджених матриць на основі внутрішнього представлення «список координат», «колекція списків» і «мережа списків»;
* перевірив правильність їх роботи.

Внаслідок аналізу виявив, що:

* візуалізація складних спискових структур полегшує їх вивчення і розуміння;
* інтерактивність відображення дозволяє швидко і ефективно проводити паралель між об’єктом відображення (матрицею) і його графічним представленням.

В майбутньому я планую:

* додати відображення основних операцій над матрицями: додавання, множення, транспонування тощо;
* додати зворотній зв’язок між відображенням матриці і об’єктом матриці, тобто можливість редагувати об’єкт матриці через візуалізацію.

# Список використаної літератури

1. Bergel Alexandre. Agile Visualization: Roassal [Онлайновий]. - http://agilevisualization.com/AgileVisualization/Roassal/0104-Roassal.html.
2. **Bergel Alexadre** Agile Visualization First Edition [Книга]. - Alexandre Bergel, 2016.
3. **Bergel Alexandre [та ін.]** Deep into Pharo [Книга]. - Square Bracket Associates, 2013.
4. **Ducasse Stéphane [та ін.]** Pharo By Example 5 [Книга]. - [s.l.] : Lulu.com & Square Bracket Associates, 2017.
5. Pharo [Онлайновий] // Wikipedia. - https://uk.wikipedia.org/wiki/Pharo.
6. Pharo By Example Two [Онлайновий]. - http://pharobooks.gforge.inria.fr/PharoByExampleTwo-Eng/latest/Profiling.pdf.
7. Pharo MOOC [Онлайновий] // Pharo MOOC. - http://rmod-pharo-mooc.lille.inria.fr/.
8. Sparse Matrix [Онлайновий] // Wikipedia. - https://en.wikipedia.org/wiki/Sparse\_matrix.