ВСТУП

В епоху інформаційних технологій кожен з нас розуміє як важливо знайти за певним критерієм необхідну для нас інформацію серед величезного масиву даних за найкоротший проміжок часу, особливо актуальною дана проблема є для тих районів нашої планети де швидкість доступу до мережі інтернет є досить слабкою та одиниця інтернет трафіку коштує величезних грошей (до даних районів відносяться місця проведення воєнних операцій, пустині, різні віддалені від цивілізації місця та місця що спіткали стихійні лиха).

Сучасні додатки вирішуючи різноманітні поставлені перед ними задачі зберігають велику кількість даних, але, враховуючи вище зазначене, крім необхідності просто зберігати дану інформацію існує необхідність в організації ефективного пошуку по заданій вибірці даних за певним критерієм.

Задача організації ефективного пошуку за певним критерієм та задача компактного зберігання цих даних ускладнюється ще більше, якщо даним критерієм пошуку є певний префікс від повного ключа пошуку й існує необхідність знайти всі пари ключ-значення що відповідають даному префіксу.

Розглядаючи стандартну реалізацію найпоширенішої мови програмування для додатків різного типу – ***Java***, постає задача розробити та реалізувати в рамках даної мови програмування структури даних для збереження строкової інформації, що дозволить проводити ефективний пошук за вищезазначеним критерієм та компактне збереження цих даних.

При виборі структури даних необхідно розуміти, що у кожної структури даних є свої обмеження, котрі можуть вплинути на їх застосування.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

* 1. Загальні відомості про структури даних
     1. Що таке структура даних?

**Структури даних** – це способи організації даних в комп'ютерах. Часто разом зі структурою даних пов'язується і специфічний перелік операцій, що можуть бути виконаними над даними, організованими в таку структуру [1].

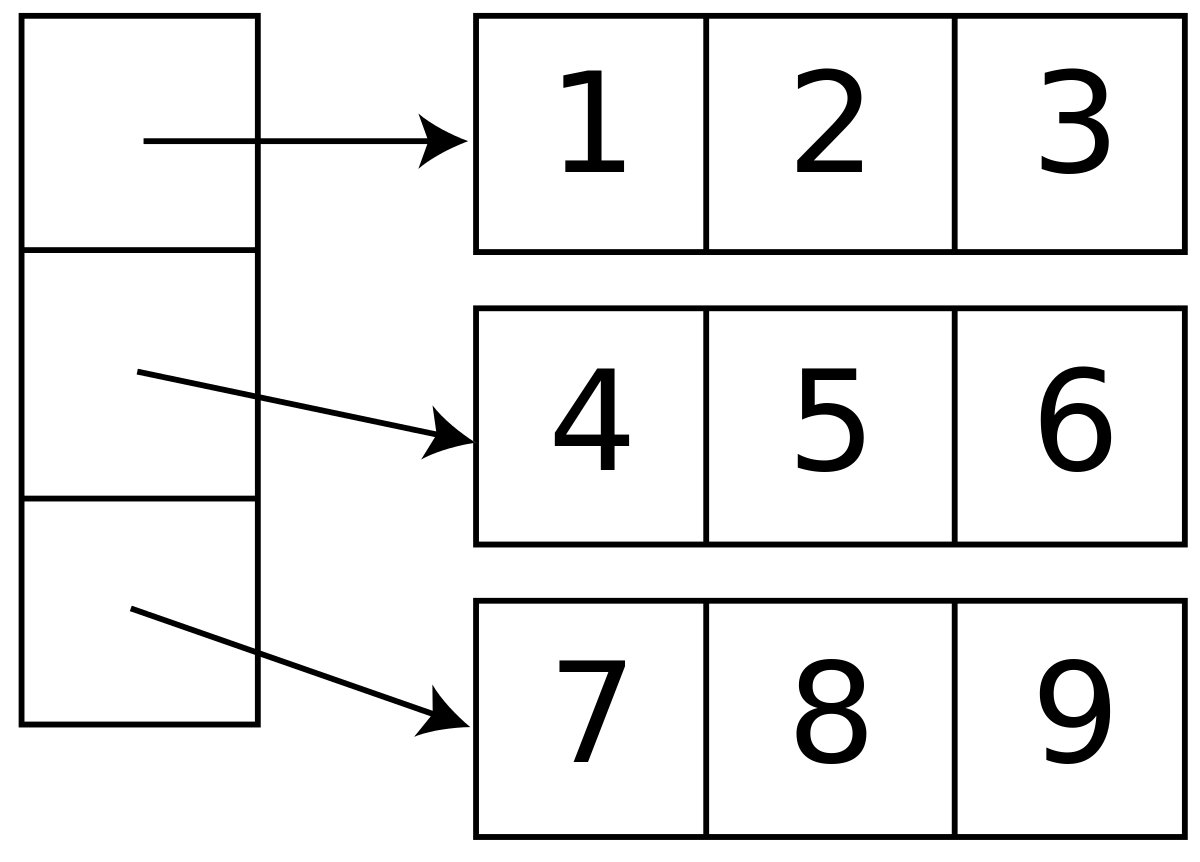


Рис. 1. Структура даних – массив

Правильний підбір структур даних є надзвичайно важливим для ефективного функціонування відповідних алгоритмів їх обробки. Добре побудовані структури даних дозволяють оптимізувати використання машинного часу та пам'яті комп'ютера для виконання найкритичніших операцій.

Відома формула «Програма = Алгоритми + Структури даних» дуже точно виражає необхідність відповідального ставлення до такого підбору.

* 1. Основні структури даних в мові програмування Java
     1. Структура даних ArrayList

Як відомо, в мові програмування Java звичайні масиви мають фіксований розмір, це призводить до того що дана структура не може збільшуватись або зменшуватись в залежності від потреб користувача. Структура даних ArrayList має можливість змінювати свій розмір під час роботи програми.

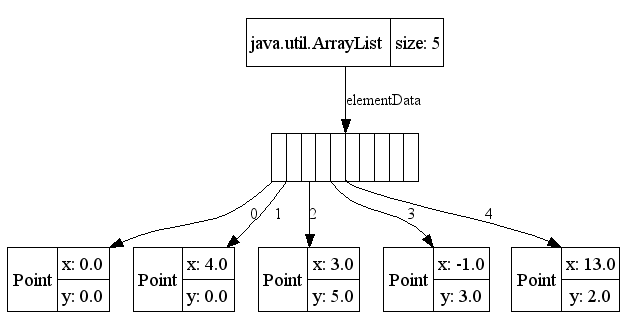


Рис. 2. Структура даних ArrayList

Розглянемо основні операції над даною структурою:

**А) Додавання елементів**

1. Перевіряється, чи достатньо місця в масиві для вставки нового елемента;

2. Якщо місця недостатньо, то створюється новий масив більшого розміру, дані що були в попередньому масиві переносяться в новий масив;

3. Даний елемент додається в кінець масиву;

**Б) Видалення елементу**

В циклі переглядаються всі елементи масиву до тих пір поки не буде знайдено перше співпадіння ключа пошуку з елементом масиву. З усього масиву даних видалений буде лише перший знайдений елемент.

**В) Пошук по індексу**

Пошук по індексу виконується за константний час О(1).

**Г) Пошук по ключу (значенню)**

В циклі переглядаються всі елементи масиву до тих пір поки не буде знайдено перше співпадіння ключа пошуку з елементом масиву.

**Д) Ініціалізація**

Створена структура ArrayList зберігає в собі пустий масив стандартного розміру що має можливість збільшуватися або зменшуватися.

Основні характеристики:

* Доступ до елементів по значенню виконується за лінійний час О(n);
* Доступ до елементів по індексу виконується за О(1);
* Пошук елементів по префіксу виконується за лінійний час О(n);
* Вставка елементів в дану структуру у кінець виконується за константний час О(1);
* Не оптимізований по пам’яті для зберігання строкової інформації що має схожі префікси, наприклад при додаванні ключів «перманент», «пергамент» та «петриківка» будуть додані всі три записи в різні комірки пам’яті;
* Дозволяє додавати значення будь-якого типу.
  + 1. Структура даних LinkedList

Структура даних що представляє собою дво-зв’язаний список елементів, де кожен елемент даної структури містить покажчик на наступний елемент списку та на попередній.

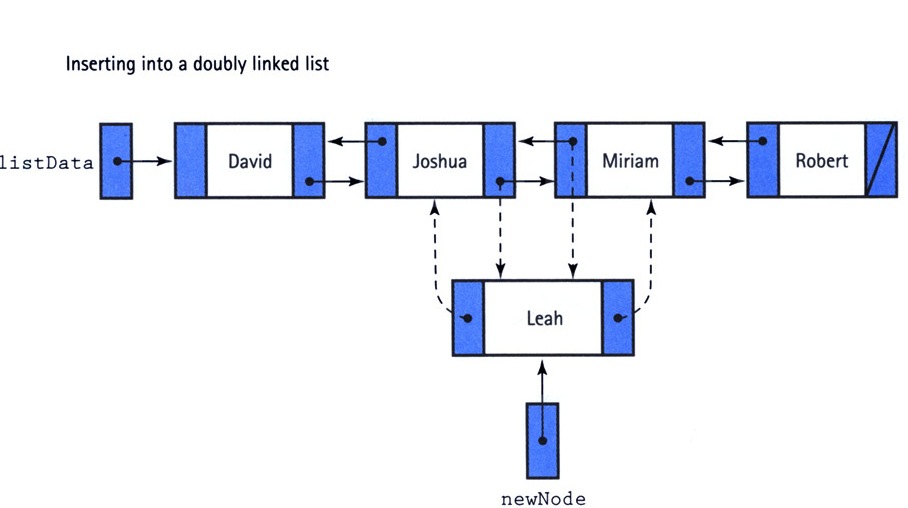


Рис. 3. Структура даних LinkedList, момент вставки елементу

Розглянемо основні операції над даною структурою:

**А) Додавання елементів**

1. Створюється додаткова структура яка зберігає значення нового елементу та покажчики на наступний та попередній елементи;

2. Встановлюються покажчики на наступний та попередній елементи.

**Б) Видалення елементу**

1. Виконується пошук першого елементу з відповідним значенням;

2. Перевизначаються покажчики на попередній та наступний елемент;

3. Зануляється елемент для видалення;

**В) Пошук елементу**

Для пошуку елементів в даній структурі використовується поняття **«ітератор»**

**Ітератор**  — шаблон проектування, належить до класу шаблонів поведінки. Надає спосіб послідовного доступу до всіх елементів складеного об'єкта, не розкриваючи його внутрішнього улаштування.

Отже, пошук елементів в даній структурі здійснюється шляхом послідовного перебору елементів.

**Г) Ініціалізація**

Тільки-но створений об’єкт складається з двох компонентів, це **header** та **size**. **header**  – елемент списку значення якого завжди null, а значення **next** and **prev** завжди вказують на перший та на останній елемент списку відповідно. Після створення даної структури size = 0.

Основні характеристики:

* Доступ до елементів по значенню виконується за лінійний час О(n);
* Доступ до елементів по індексу виконується за О(n);
* Пошук елементів по префіксу виконується за лінійний час О(n);
* Вставка елементів в дану структуру у кінець виконується за константний час О(1);
* Не оптимізований по пам’яті для зберігання строкової інформації що має схожі префікси, наприклад при додаванні ключів «перманент», «пергамент» та «петриківка» будуть додані всі три записи у різні частини списку.
* Дозволяє додавати значення будь-якого типу.
  + 1. Структура даних HashMap

Структура даних що побудована на основі хеш-таблиць та зберігає данні у вигляді ключ/значення. Дана структура може зберігати дані різних типів, вирішення колізій побудовано на основі методу ланцюжків.

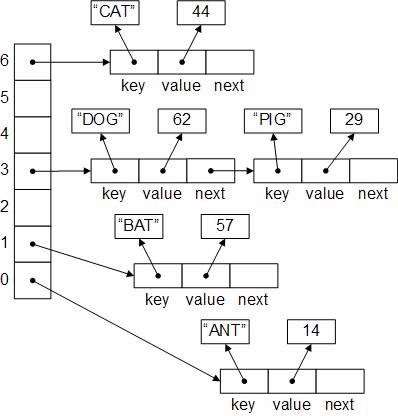


Рис. 4 Структура даних HashMap

**Хеш-таблиця** — структура даних, що реалізує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і здійснювати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення за ключем.

**Метод ланцюжків**

Кожна комірка масиву *H* є вказівником на зв'язаний список (ланцюжок) пар ключ-значення, відповідних одному і тому самому хеш-значенню ключа. Колізії просто призводять до того, що з'являються ланцюжки довжиною більше одного елемента.

Операції пошуку або видалення елемента вимагають перегляду всіх елементів відповідного ланцюжка, щоб знайти в ньому елемент з заданим ключем. Для додавання нового елемента необхідно додати елемент в кінець або початок відповідного списку, і, у випадку якщо коефіцієнт заповнення стане занадто великим, збільшити розмір масиву *H* і перебудувати таблицю.

При припущенні, що кожний елемент може потрапити в будь-яку позицію таблиці *H* з однаковою ймовірністю і незалежно від того, куди потрапив будь-який елемент, пересічний час роботи операції пошуку елемента складає Θ(1 + *α*), де *α* — коефіцієнт заповнення таблиці.

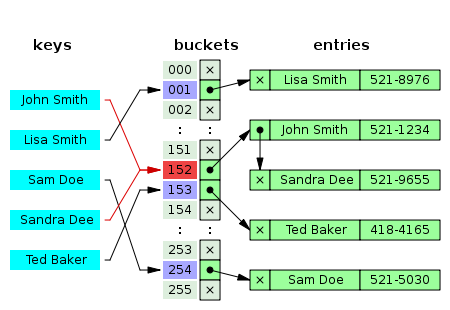


Рис. 5. Вирішення колізій за допомогою методу ланцюжків

Розглянемо основні операції над даною структурою:

**А) Додавання елементів**

1. Перевіряється значення ключа на null, якщо даний ключ є нульовим то буде визваний спеціальний метод для додавання елементів з пустим ключем. Оскільки ключ являється пустим то для даних елементів не може бути визначений хеш-код, тому дані елементи додаються в хеш-таблицю по індексу 0. Всі елементи ланцюжка, що знаходяться в хеш-таблиці по індексу 0 по черзі переглядаються в пошуках елемента з ключем null, якщо такий елемент в ланцюжку існує, його значення перезаписується;

2. Якщо значення ключа не є null, тоді на основі його генерується хеш;

3. На основі даного хеш значення визначається позиція в масиві, куди буде поміщений даний елемент;

4. Знаючи індекс в масиві ми отримуємо ланцюжок елементів, зв’язаних з даною коміркою. Хеш та ключ нового елементу по-черзі порівнюються з хешами й ключами елементів в списку й, при збігу даних параметрів, значення переписується. Якщо збігів не було виявлено, новий елемент буде доданий у кінець ланцюжка у заданій комірці;

5. Якщо дана хеш-таблиця заповнюється до граничного значення, її розмір збільшується вдвічі та викликається перерозподіл елементів по таблиці, внаслідок чого змінюються і ланцюжки.

**Б) Видалення елементу**

1. Якщо видаляється елемент з ключем null, то відбувається пошук даного елементу у ланцюжку що знаходиться у комірці з індексом 0;

2. Якщо значення ключа не пусте, вираховується хеш-значення даного ключа;

3. По даному хеш-значенню вираховується значення комірки в якій зберігається ланцюжок значень;

4. Шляхом порівняння по черзі даного ключа з ключами у ланцюжку знаходиться елемент що підлягає видаленню.

5. Видалення елементу відбувається за рахунок перевизначення покажчика поереднього елементу на наступний.

**В) Пошук елементу**

1. Якщо значення ключа null, то відбувається пошук даного елементу у ланцюжку що знаходиться у комірці з індексом 0;

2. Якщо значення ключа не пусте, вираховується хеш-значення даного ключа;

3. По даному хеш-значенню вираховується значення комірки в якій зберігається ланцюжок значень;

4. Шляхом порівняння по черзі даного ключа з ключами у ланцюжку знаходиться елемент що відповідає даному ключу.

Також дана структура надає можливість використовувати ітератори, дані ітератори дозволяють отримати список всіх ключів та списов всіх значень, або всі пари ключ/значення одразу.

**Г) Ініціалізація**

При створенні HashMap дана структура має наступні властивості:

**table** – сама хеш таблиця, що зберігає в собі ланцюжки значень;

**loadFactor**  - коефіцієнт завантаження, значення за замовченням – 0,75;

**threshold**  – Гранична кількість елементів, при досягненні якого, розмір хеш-таблиці збільшується вдвічі;

**size** – кількість елементів в структурі;

Основні характеристики:

* Доступ до елементів по значенню може виконуватись за час О(1), якщо хеш-функція рівномірно розподіляє елементи й колізії відсутні. Середній час доступу до елемента складає Θ(1 + *α*), де *α* — коефіцієнт заповнення таблиці. В найгіршому випадку час доступу до елемента може бути О(n), якщо всі елементи будуть в одному ланцюжку.
* Пошук елементів по префіксу виконується за лінійний час О(n);
* Вставка елементів в дану структуру виконується за константний час О(1);
* Не оптимізований по пам’яті для зберігання строкової інформації що має схожі префікси, наприклад при додаванні ключів «перманент», «пергамент» та «петриківка» будуть додані всі три записи у різні ланцюжки (або в один ланцюжок).
* Дозволяє додавати значення будь-якого типу (крім примітивних типів, оскільки для них існують «обгортки»).
  + 1. Структура даних TreeMap

Дана структура зберігає пари ключ/значення у вигляді червоно-чорного дерева, в результаті чого дані елементи є відсортованими за значеннями ключа (у випадку строкової інформації – відсортовані дані у лексикографічному порядку).

**Червоно-чорне дерево** — різновид бінарного дерева пошуку, вершини якого мають додаткові властивості (RB-властивості), зокрема «колір» (червоний або чорний). Червоно-чорні дерева — різновид збалансованих дерев, в яких за допомогою спеціальних трансформацій гарантується, що висота дерева *h* не буде перевищувати O(*log n*). Зважаючи на те, що час виконання основних операцій на бінарних деревах (пошук, видалення, додавання елементу) є O(*h*), ці структури даних на практиці є набагато ефективнішими, аніж звичайні бінарні дерева пошуку.

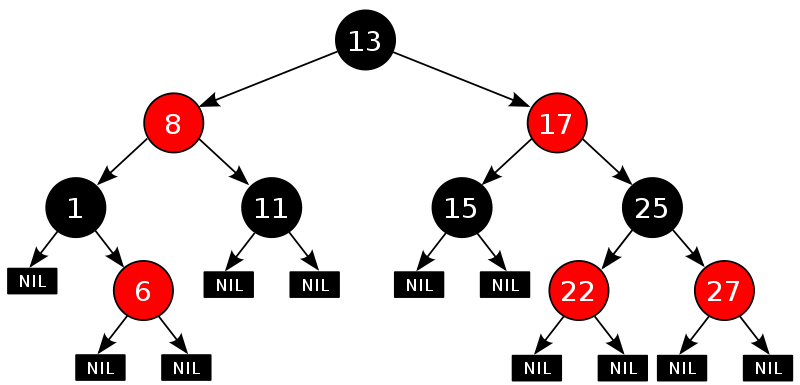


Рис. 6. Червоно-чорне дерево

Бінарне дерево називається червоно-чорним, якщо воно має такі властивості:

1. кожна вершина або червона, або чорна
2. корінь дерева — чорний
3. кожний лист (NIL) — чорний
4. якщо вершина червона, обидві її дочірні вершини чорні (інакше, батько червоної вершини - чорний)
5. усі прості шляхи від будь-якої вершини до листів мають однакову кількість чорних вершин

Такі властивості надають червоно-чорному дереву додаткового обмеження: найдовший шлях з кореня до будь-якого листа перевищує найкоротший шлях не більше ніж вдвічі. В цьому сенсі таке дерево можна назвати збалансованим. Зважаючи на те, що час виконання основних операцій з бінарними деревами пошуку залежить від висоти, таке обмеження гарантує їхню ефективність в найгіршому випадку, чого звичайні бінарні дерева гарантувати не можуть.

Для того, щоби зрозуміти, чому перелічені властивості забезпечують існування такого обмеження, зазначимо, що в червоно-чорному дереві, відповідно до властивості 4 не існує такого шляху, на якому б зустрілись дві червоні вершини підряд. Найкоротший шлях складається з усіх чорних вершин, а в найдовшому червоні та чорні вершини чергуються. З врахуванням властивості 5, отримуємо, що глибина будь-яких двох листів відрізняється не більше ніж в два рази.

**А) Додавання елементів**

Додавання елементу в дану структуру призводить до відповідних операцій балансування на червоно-чорному дереві, в зв’язку з цим повинно бути задано правило, за яким елемент що додається буде порівнюватись з іншими елементами що вже є в дереві (для строкових даних це лексикографічні правила)

**Б) Видалення елементу**

Видалення елементу також призводить до відповідних операцій ре балансування червоно-чорного дерева.

**В) Пошук елементу**

Пошук елементу в даній структурі аналогічний пошуку в звичайному бінарному дереві пошуку.

Також дана структура надає можливість використовувати ітератори, дані ітератори дозволяють отримати список всіх ключів та списов всіх значень, або всі пари ключ/значення одразу.

**Г) Ініціалізація**

Створюється структура яка містить корінь даного дерева **root**, **comparator** – правило сортування записів та **size** – кількість елементів у даній структурі.

Основні характеристики:

* Дана структура даних зберігає відсортований масив пар ключ\значення;
* Дана структура гарантує швидкість доступу до елементу, швидкість додавання та видалення елементу рівною O(log(n));
* Не оптимізований по пам’яті для зберігання строкової інформації що має схожі префікси, наприклад при додаванні ключів «перманент», «пергамент» та «петриківка» будуть додані всі три записи у різні частини дерева;
* Дозволяє додавати значення будь-якого типу (крім примітивних типів, оскільки для них існують «обгортки»).
  1. Постановка задачі розробки структури даних для збереження строкової інформації на основі мови програмування Java

Метою роботи є створення ефективної структури даних для збереження строкової інформації у вигляді пари ключ-значення що повинно відповідати наступним вимогам:

* Простота у використанні;
* Інтуїтивно зрозумілий інтерфейс для роботи з даним типом даних;
* Можливість використовувати дану структуру даних на будь-якій сучасній платформі;
* Структура даних повинна підтримувати пошук по ключам;
* Структура даних повинна підтримувати префіксний пошук;
* Структура даних повинна забезпечувати компактне збереження даних.

Отже задачі, що вирішуються в даній роботі:

1. Створення та опис структури даних, засновану на ідеї навантаженого дерева, що відповідає умовам, зазначеним вище;
2. Порівняння реалізації даної структури з вже існуючими структурами даних в мові програмування Java за наступними параметрами:
   1. Час на вставку елемента;
   2. Час на пошук елемента;
   3. Час на пошук елементів за заданим префіксом;
   4. Час на видалення елемента;
   5. Об’єм пам’яті що використовується при додаванні однакових даних.