1 БІБЛІОТЕКА ФУНКЦІЙ ТА ТИПІВ ДЛЯ РОБОТИ З ДЕРЕВАМИ МЕРКЛА

1.1 Початок роботи

Було представлено три види дерев Меркла: бінарне, розріджене та індексоване. Для роботи з кожним деревом вам необхідно додати файли – MTree.java, SparseMTree.java, IndexMTree.java відповідно, в одну директорію з файлом вашої програми. Також доступний файл для тестів – Main.java. Для початку роботи необхідно створити об'єкт дерево. У випадку бінарного дерева, вам необхідно подати підготовлений масив даних, у інших двох випадках – генерується порожнє дерево. Бібліотека містить наступні функції (в дужках зазначено для яких дерев): додавання нового елемента (розріджене дерево, індексоване дерево), доказ включення (бінарне, розріджене, індексоване дерево), доказ невключення (розріджене дерево, індексоване дерево).

Далі детально опишемо об'єкти і функції бібліотеки:

1.2 Типи бібліотеки

1) **MTree tree = new MTree(Arraylist<String> value)** — тип даних бінарне дерево, де value – масив значень, що будуть поміщені.

2) **SparseMTree tree = new SparseMTree(int elements)** — тип даних розріджене дерево, де elements – кількість елементів, що будуть в дереві. Дерево створюється пустим.

3) **IndexedMTree tree = new IndexedMTree(int elements)** — тип даних індексоване дерево, де elements – кількість елементів, що будуть в дереві. Дерево створюється пустим.

Також, кожен з цих класів містить дочірний клас **Tree.Node A = new Tree.Node(String value)** — де value величина, що буде гешуватися у ноді,

a Tree – дерево, для якого створюється нода.

1.3 Функції бібліотеки для бінарного дерева Меркла

1) **Node getRoot()** — функія повертає значення кореня дерева.

2) **String SHA-512 (String input)** — функція обчислення гешу значення input.

3) **void MP (Node x, int index)** — функція обчислення доказу включення точки (листка) x на місці index у дереві.

4) **Arraylist<String> keyGen (int elements)** — функція повертає масив ненульових значень, кількість яких дорівнює elements.

1.4 Функції бібліотеки для розрідженого дерева Меркла

1) **Node getRoot()** — функія повертає значення кореня дерева.

2) **String SHA-512 (String input)** — функція обчислення гешу значення input.

3) **SparseMTree AddNode (Node x, int index)** — функція додає ноду x у дерево на місце index. Поверетається нове, переобчислене дерево.

4) **void MP (Node x, int index)** — функція обчислення доказу включення точки (листка) x на місці index у дереві.

5) **void EP (int index)** — функція обчислення доказу невключення точки (листка) на місці index у дереві.

1.5 Функції бібліотеки для індексованого дерева Меркла

1) **Node getRoot()** — функія повертає значення кореня дерева.

2) **String SHA-512 (String input)** — функція обчислення гешу значення input.

3) **IndexedMTree AddNode (Node x, int index)** — функція додає

ноду x у дерево на місце index. Поверетається нове, переобчислене дерево.

4) **void MP (Node x)** — функція обчислення доказу включення точки (листка) x у дереві.

5) **void EP (Node x)** — функція обчислення доказу невключення точки (листка) x у дереві.

1.6 Алгоритм доказу включення у дерево

Зрозуміло, що для бінарного дерева ми повинні знати місце точки (листка), що хочемо перевірити, оскільки можемо працювати з даними лише відносно кореня нашого дерева. Отже алгоритм виглядає наступним чином:

1) Формуємо доказову базу, тобто поступово опускаючись до нашої точки від кореня, ми відмічаємо ноди, що також є дитиною ноди з попереднього рівня, поки не дійдемо до нашої точки. У результаті отримаємо масив нод, що дозволить не обчислювати усе дерево і не потребує знання усіх листків.

2) Переобчислюємо корінь починаючи з точки, що хочемо перевірити. Якщо значення оригінального кореня та переобчисленого збіглися – точка належить дереву, інакше – ні.

У розрідженому дереві все виконується аналогічно бінарному.

У індекованому дереві ми маємо масив листів і кожен лист містить індекс наступного гешу за зростанням. З однієї сторони, якщо геш буде найбільшим, то ми будемо перебирати усі значення. А з іншого боку ми можемо знайти геш точно не перебираючи значення вищі за його значення. Це доволі зручно, оскільки ми можемо однозначно шукати по порядку. Алгоритм буде будуватися на пошуку співпадінь значення поступово перебираючи значення від найменшого.

Псевдокод для перевірки доказу включень бінарного та розрідженого дерева void MP(Node x, int index)

Вхід: Node x, int index, int allelem, int hight, Node root

1. $ListProofBase = \varnothing$, $int[]binarytrip = newint[hight - 1]$, Node temp = root

2. для і від 0 до hight-2:

Якщо $index > allelem/2$:

$ProofBase.add(0, temp.left)$

$temp = temp.right$

$binarytrip[i] = 1$

$index = index - allelem/2$

інакше:

$ProofBase.add(0, temp.right)$

$temp = temp.left$

$binarytrip[i] = 0$

$allelem/ = 2$

3. Якщо index — парне :

$ProofBase.add(0, temp.left), binarytrip[height - 2] = 1$

3.1 Якщо index — непарне :

$ProofBase.add(0, temp.left), binarytrip[height - 2] = 0$

4. Xroot = x.value

5. для і від 0 до hight-1:

Якщо $binarytrip[hight-2-i] == 0 : SHA(Xroot+ProofBase.get(i))$

Інакше : $SHA(ProofBase.get(i) + Xroot)$

6. Якщо Xroot == root.value : вивести "Лист належить дереву"

6.1 Інакше : "..."

Псевдокод для перевірки доказу включень індексованого дерева void MP(Node x)

Вхід: Node x, Arraylist<Node> leaves

1. Node temp = leaves.get(0)

2. Поки (temp.ind != 0 і x.value > temp.value)

Якщо x.value == temp.value, вивести і вийти з функції : "Лист належить дереву"

інакше temp = leaves.get(temp.ind)

3. Якщо x.value == temp.value, вивести і вийти з функції : "Лист належить дереву"

3.1 Інакше : "..."

1.7 Доказ невключення у дерево

Даний алгоритм застосовний до розрідженого та індексованого дерева. Для індексованого дерева алгоритм полягає у пошуку значень, між якими імовірно міг бути наш лист з гешом, і відповідно, якщо такого у дереві немає, то відповідно цей лист не міститься у дереві. Для розрідженого дерева процедура по суті включає перевірку того, що на місці де повинно міститися наше значення є нульовим і відповідно наш лист не міститься у дереві.

Псевдокод для перевірки доказу невключень у розріджене дерева void EP(int index)

Вхід: int index, int allelem, int hight, Node root

1. $ListProofBase = \varnothing$, $int[]binarytrip = newint[hight - 1]$, Node temp = root

2. для і від 0 до hight-2:

Якщо $index > allelem/2$:

$ProofBase.add(0, temp.left)$

$temp = temp.right$

$binarytrip[i] = 1$

$index = index - allelem/2$

інакше:

$ProofBase.add(0, temp.right)$

$temp = temp.left$

$binarytrip[i] = 0$

$allelem/ = 2$

3. Якщо index — парне :

$ProofBase.add(0, temp.left), binarytrip[height - 2] = 1$

3.1 Якщо index — непарне :

$ProofBase.add(0, temp.left), binarytrip[height - 2] = 0$

4. Xroot = null

5. для i від 0 до hight-1:

Якщо $binarytrip[hight-2-i] == 0 : SHA(Xroot+ProofBase.get(i))$

Інакше : $SHA(ProofBase.get(i) + Xroot)$

6. Якщо Xroot == root : вивести "Лист не належить дереву"

6.1 Інакше : "Дерево містить цей елемент"

Псевдокод для перевірки доказу невключень індексованого дерева void EP(Node x)

Вхід: Node x, Arraylist<Node> leaves

1. Node temp = leaves.get(0)

2. Поки (temp.ind != 0 і x.value > temp.value)

Якщо x.value > temp.value : temp = leaves.get(temp.ind)

інакше вивести і вийти з функції : "Лист належить дереву"

3. Якщо x.value == temp.value, вивести : "Лист належить дереву"

3.1 Інакше : "Лист не належить дереву"

1.8 Функція додавання листа

Оскільки за стандартом бінарне дерево будується цілісно, то дана функція була реалізована для розрідженого та ідексованого дерев.

По суті, алгоритм додавання нової точки є перебудовою дерева,

оскільки в нас змінюється внутрішня структура і корінь відповідно. По суті алгоритм додання нового листа для розрідженого дерева буде являти собою алгоритм перевірки включення у дерево за виключенням того, що новий обрахований корінь і буде нашим новим деревом. У індексованому дереві ми просто додаємо до масиву листів нашу нову точку і, упорядкувавши індекси, будуємо нове дерево.

Псевдокод для додавання нового лситка у розріджене дерево SparseMTree AddNode(Node x, int index)

Вхід: Node x, int index, int allelem, int hight, Node root

1. $ListProofBase = \varnothing$, $int[]binarytrip = newint[hight - 1]$, Node temp = root

2. для і від 0 до hight-2:

Якщо $index > allelem/2$:

$ProofBase.add(0, temp.left)$

$temp = temp.right$

$binarytrip[i] = 1$

$index = index - allelem/2$

інакше:

$ProofBase.add(0, temp.right)$

$temp = temp.left$

$binarytrip[i] = 0$

$allelem/ = 2$

3. Якщо index — парне :

$ProofBase.add(0, temp.left), binarytrip[height - 2] = 1$

3.1 Якщо index — непарне :

$ProofBase.add(0, temp.left), binarytrip[height - 2] = 0$

4. Xroot = x

5. для і від 0 до hight-1:

Якщо $binarytrip[hight - 2 - i] == 0$:

$newNode(Xroot, ProofBase.get(i))$

Інакше :

$newNode(ProofBase.get(i), Xroot)$

6. Повернути new SparseMTree(Xroot, hight)

Псевдокод для додавання нового листка у індексоване дерево IndexedMTree AddNode(Node x, int index)

Вхід: Node x, int index, Arraylist<Node> leaves

1. leaves.set(index, x)

2. leaves = sortNodes(leaves)

3. Повернути new IndexedMTree(leaves, hight)

Псевдокод функції Arraylist<Node> sortNodes (Arraylist<Node> leaves)

Вхід: Arraylist<Node> leaves

1. $ArrayListvalues = \varnothing$

2. Для кожної Node з leaves : values.add(Node.value)

3. $ArrayListsortvalues = values.sort()$

4. Для і від 0 до leaves.size()

Якщо Node не останній елемент у sortvalues : $Node_i.ind$ = values.index(sortvalues.index(Node.value) + 1)

Інакше $Node_i.ind$ = values.index(sortvalues.get(0))

5. Повернути leaves

Алгоритм	Додавання елемента	Доказ включення	Доказ невключення	Розмір MP	Розмір EP	Верифікація MP	Верифікація EP
Бінарне дерево	-	~83 операцій O(log N)	-	16L	-	~33 операцій O(logN)	-
Розріджене дерево	~116 операцій O(logN)	~83 операцій O(log N)	~83 операцій O(log N)	16L	16L	~33 операцій O(logN)	~33 операцій O(logN)
Індексоване дерево	3N+5 операцій O(N^2)	1 операція O(1)	1 операція O(1)	L	L	131070 операцій (у найгіршому випадку) O(N)	131070 операцій (у найгіршому випадку) O(N)

Рисунок 1.1 – Результати. L – кількість байтів у ноді, N – кількість листів у дереві

Підбиваючи підсумки, можна вказати, що кожне дерево має свої переваги. Бінарне дерево досить зручне, оскільки ми можемо заповнювати усі листки і зберігати багато даних, але це ж і не дає нам застосовувати доказ невключення. Розріджене дерево має таку властивість, але ми не зможемо записати в нього багато елементів, або прийдеться розширювати дерево, що ускладнить роботу з ним. Індексоване дерево має дійсно гарні методи доказів, не потребує зберігання бази доказу, але при повному переборі (у найгіршому випадку) буде давати погані оцінки.