КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

Семестровая работа №2

по алгоритмам и структурам данных

Выполнил:

студент группы 11-906

Мокеева Полина

Казань 2020

**Содержание:**

1. - Название алгоритма, краткую историческую справку.

2. - Основной принцип работы. Основные особенности.

3. - Оценка временной сложности с обоснованием (краткое доказательство).

4. - Таблицы полученных значений количества итераций и времени работы в зависимости от размера и характера данных (в конце презентации как приложение).

5. – Графики.

6. - Выводы. Плюсы и минусы алгоритма, применимость его (т.е., когда его целесообразно применять).

7. - Список использованной литературы (источников).

8. - Код алгоритма.

9. - Код генерации входных файлов для сортировки.

10. - Все входные данные.

1.Название алгоритма, краткую историческую справку.

Radix Sort – Поразрядная илиЦифровая сортировка.

Есть два вида такой сортировки — MSD (most significant digit — сначала старший разряд) и LSD (least significant digit — сначала младший разряд).

В своей работе я буду рассматривать Radix Sort LSD.

Сортировка Radix восходит еще к 1887 году к работе Германа Холлерита над табуляционными машинами. Алгоритмы сортировки радикса вошли в общее употребление в качестве способа сортировки перфокарт еще в 1923 году.

Первый эффективный для памяти компьютерный алгоритм был разработан в 1954 году в Массачусетском технологическом институте Гарольдом Х. Сьюардом. Компьютеризированные сорта радикса ранее были отклонены как нецелесообразные из-за предполагаемой необходимости переменного распределения ведер неизвестного размера. Новшество Сьюарда состояло в использовании линейного сканирования для предварительного определения требуемых размеров ячеек и смещений, что позволяло обеспечить однократное статическое выделение вспомогательной памяти. Линейное сканирование тесно связано с другим алгоритмом Сьюарда — подсчетом сортировки.

2. Основной принцип работы. Основные особенности.

Элементы перебираются по порядку и группируются по самому младшему разряду (сначала все, заканчивающиеся на 0, затем заканчивающиеся на 1, ..., заканчивающиеся на 9). Возникает новая последовательность. Затем группируются по следующему разряду с конца, затем по следующему и т.д. пока не будут перебраны все разряды, от младших к старшим.

Например, сравним числа 264, 85, 363.

Возьмём последний разряд чисел 264, 85, 363 и отсортируем их в порядке возрастания в специальных “корзинках” (номер корзины означает остаток от деления на 10). Тогда получаем следующую картину - 363, 264, 85. Далее выберем следующий разряд, и получим последовательность - 363, 264, 85. В конце, отсортируем старший разряд, но мы видим, что у числа 85 его нет, тогда мы предполагаем, что старший разряд этого числа равен нулю. По итогу мы получаем -85, 264, 363.

Для решения конкретных задач, Radix sort подразделяют на два типа. LSD несколько удобнее для сортировки чисел, т.к. не приходится «приписывать» к числам слева незначащие 0 для выравнивания числа разрядов.  
MSD же удобнее для сортировки строк.

3. Оценка временной сложности с обоснованием (краткое доказательство).

Пусть во входных целых числах будет d цифр. Radix Sort занимает время O (d \* (n + b)), где b является базой для представления чисел, например, для десятичной системы, b равно 10. Каково значение d? Если k — максимально возможное значение, то d будет O (log b (k)). Таким образом, общая сложность времени составляет O ((n + b) \* log b (k)). Который выглядит больше, чем временная сложность алгоритмов сортировки на основе сравнения для большого k. Давайте сначала ограничим k. Пусть k <= n c, где c постоянная. В этом случае сложность становится O (nLog b (n)). Но он все еще не превосходит алгоритмы сортировки, основанные на сравнении.  
Что если мы увеличим значение b? Каким должно быть значение b, чтобы сложность времени была линейной? Если мы установим b как n, мы получим временную сложность как O (n). Другими словами, мы можем отсортировать массив целых чисел с диапазоном от 1 до n c, если числа представлены в базе n (или каждая цифра занимает log 2 (n) битов).

4. Таблицы полученных значений количества итераций и времени работы в зависимости от размера и характера данных (в конце презентации как приложение)



(кликнуть два раза)

5. Графики.

В теории: сложность для обычных и LinkedList массивов сложность составляет O(n+k)

На практике: для обычных массивов О(k),для LinkedList массивов O(n+k).

6. Выводы. Плюсы и минусы алгоритма, применимость его (т.е., когда его целесообразно применять).

В современную эпоху, виды радикса наиболее часто применяются к коллекциям двоичных строк и целых чисел. Было показано, что в некоторых бенчмарках он работает быстрее, чем другие более универсальные алгоритмы сортировки, иногда от 50% до трех раз быстрее.

7. Список использованной литературы (источников).

<https://en.wikipedia.org/wiki/Radix_sort>

<https://ru.wikipedia.org/wiki/Поразрядная_сортировка>

<http://espressocode.top/radix-sort/>

<https://habr.com/en/post/268261/>

<https://youtu.be/ibtN8rY7V5k>

8. Код алгоритма.

import java.util.LinkedList;  
import java.util.Queue;  
  
public class RadixSort {  
  
 public static String radixSortInt(int[] arr) {  
 long iter = 0;  
 Queue<Integer>[] buckets = new Queue[10];  
 for (int i = 0; i < 10; i++)  
 buckets[i] = new LinkedList<Integer>();  
  
 boolean sorted = false;  
 int exponenta = 1;  
  
 while (!sorted) {  
 sorted = true;  
 for (int item : arr) {  
 int bucket = (item / exponenta) % 10;  
 if (bucket > 0) sorted = false;  
 buckets[bucket].add(item);  
 }  
  
 exponenta \*= 10;  
 int index = 0;  
  
 for (Queue<Integer> bucket : buckets)  
 while (!bucket.isEmpty()) {  
 arr[index++] = bucket.remove();  
 }  
  
 iter++;  
 if (!*isSorted*(arr)) sorted = false;  
 }  
  
 return String.*valueOf*(iter);  
 }  
  
  
 public static String radixSortChar(char[] arr) {  
 long iter = 0;  
 Queue<Integer>[] buckets = new Queue[10];  
 for (int i = 0; i < 10; i++)  
 buckets[i] = new LinkedList<Integer>();  
  
 boolean sorted = false;  
 int exponenta = 1;  
  
 while (!sorted) {  
 sorted = true;  
 // 1 10 100 1000  
 for (int item : arr) {  
 int bucket = (item / exponenta) % 10;  
 if (bucket > 0) sorted = false;  
 buckets[bucket].add(item);  
 }  
  
 exponenta \*= 10;  
 int index = 0;  
  
 for (Queue<Integer> bucket : buckets)  
 while (!bucket.isEmpty()) {  
 arr[index++] = (char) (bucket.remove().intValue());  
 }  
 iter++;  
 if (!*isSorted*(arr)) sorted = false;  
 }  
 return String.*valueOf*(iter);  
 }  
  
 public static String radixLinkedSort(LinkedList<Integer> arr) {  
 long iter = 0;  
 Queue<Integer>[] buckets = new Queue[10];  
 for (int i = 0; i < 10; i++)  
 buckets[i] = new LinkedList<Integer>();  
  
 boolean sorted = false;  
 int exponenta = 1;  
  
 while (!sorted) {  
 sorted = true;  
 // 1 10 100 1000  
 for (int item : arr) {  
 int bucket = (item / exponenta) % 10;  
 if (bucket > 0) sorted = false;  
 buckets[bucket].add(item);  
 }  
  
 exponenta \*= 10;  
 int index = 0;  
  
 for (Queue<Integer> bucket : buckets)  
 while (!bucket.isEmpty()) {  
 arr.set(index++, bucket.remove());  
 }  
 iter++;  
 if (!*isSortedLinkedInt*(arr)) sorted = false;  
 }  
 return String.*valueOf*(iter);  
 }  
  
 public static String radixLinkedSortChar(LinkedList<Character> arr) {  
 long iter = 0;  
 Queue<Integer>[] buckets = new Queue[10];  
 for (int i = 0; i < 10; i++)  
 buckets[i] = new LinkedList<Integer>();  
  
 boolean sorted = false;  
 int exponenta = 1;  
  
 while (!sorted) {  
 sorted = true;  
 // 1 10 100 1000  
 for (int item : arr) {  
 int bucket = (item / exponenta) % 10;  
 if (bucket > 0) sorted = false;  
 buckets[bucket].add(item);  
 }  
  
 exponenta \*= 10;  
 int index = 0;  
  
 for (Queue<Integer> bucket : buckets)  
 while (!bucket.isEmpty()) {  
 arr.set(index++, (char) bucket.remove().intValue());  
 }  
 iter++;  
 if (!*isSortedLinkedChar*(arr)) sorted = false;  
 }  
 return String.*valueOf*(iter);  
 }  
  
 private static boolean isSorted(int[] arr) {  
 for (int i = 0; i < arr.length - 1; i++) {  
 if (arr[i] > arr[i + 1]) return false;  
 }  
 return true;  
 }  
  
 private static boolean isSorted(char[] arr) {  
 for (int i = 0; i < arr.length - 1; i++) {  
 if (arr[i] > arr[i + 1]) return false;  
 }  
 return true;  
 }  
  
 private static boolean isSortedLinkedInt(LinkedList<Integer> arr) {  
 for (int i = 0; i < arr.size() - 1; i++) {  
 if (arr.get(i).compareTo(arr.get(i + 1)) > 0) return false;  
 }  
 return true;  
 }  
  
 private static boolean isSortedLinkedChar(LinkedList<Character> arr) {  
 for (int i = 0; i < arr.size() - 1; i++) {  
 if (arr.get(i).compareTo(arr.get(i + 1)) > 0) return false;  
 }  
 return true;  
 }  
}

9. Код генерации входных файлов для сортировки.

import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Paths;  
  
public class WriteData {  
  
 private BufferedWriter writer;  
 private int c1, c2, c3, c4;  
 private int arr1Size, arr2Size, arr3Size, arr4Size;  
  
 public WriteData() throws IOException {  
  
 c1 = (int) ((Math.*random*() \* 50) + 50);  
 c2 = (int) ((Math.*random*() \* 50) + 50);  
 c3 = (int) ((Math.*random*() \* 50) + 50);  
 c4 = (int) ((Math.*random*() \* 50) + 50);  
  
 writer = new BufferedWriter(new FileWriter(Paths.*get*("").toAbsolutePath().resolve("data.txt").toString()));  
 init(writer);  
 writer.close();  
 }  
  
 private void init(BufferedWriter writer) throws IOException {  
 writer.write(c1 + " " + c2 + " " + c3 + " " + c4);  
 writer.newLine();  
 writer.flush();  
  
 for (int i = 0; i < c1; i++) {  
 arr1Size = (int) ((Math.*random*() \* 9900) + 100);  
 writer.write(arr1Size + " ");  
 writer.newLine();  
 for (int j = 0; j < arr1Size; j++) {  
 writer.write((int) ((Math.*random*() \* 16328) + 1) + " ");  
 }  
 writer.newLine();  
 }  
 writer.flush();  
  
 for (int i = 0; i < c2; i++) {  
 arr2Size = (int) ((Math.*random*() \* 9900) + 100);  
 writer.write(arr2Size + " ");  
 writer.newLine();  
 for (int j = 0; j < arr2Size; j++) {  
 writer.write((char) ((Math.*random*() \* 26) + 65) + " ");  
 }  
 writer.newLine();  
 }  
 writer.flush();  
  
 for (int i = 0; i < c3; i++) {  
 arr3Size = (int) ((Math.*random*() \* 9900) + 100);  
 writer.write(arr3Size + " ");  
 writer.newLine();  
 for (int j = 0; j < arr3Size; j++) {  
 writer.write((int) ((Math.*random*() \* 16328) + 1) + " ");  
 }  
 writer.newLine();  
 }  
 writer.flush();  
  
 for (int i = 0; i < c4; i++) {  
 arr4Size = (int) ((Math.*random*() \* 9900) + 100);  
 writer.write(arr4Size + " ");  
 writer.newLine();  
 for (int j = 0; j < arr4Size; j++) {  
 writer.write((char) ((Math.*random*() \* 26) + 65) + " ");  
 }  
 writer.newLine();  
 }  
 writer.flush();  
 }  
}

10. Все входные данные.

(кликнуть два раза)