Статьи по Теме

Радиусная сортировка

Уровень сложности: средний ● Последнее обновление: 1 апр 2021 г.

Нижняя граница для алгоритма сортировки на основе сравнения (сортировка слиянием, сортировка кучи, быстрая сортировка и т. Д.) Равна Ω (nLogn), т.е. они не могут работать лучше, чем nLogn.

Подсчетная сортировка - это алгоритм линейной сортировки по времени, который сортирует за O (n + k) время, когда элементы находятся в диапазоне от 1 до k.

Что, если элементы находятся в диапазоне от 1 до n2?

Мы не можем использовать сортировку с подсчетом, потому что сортировка с подсчетом займет O (n2), что хуже, чем алгоритмы сортировки на основе сравнения. Можно ли отсортировать такой массив за линейное время?

Radix Sort - вот ответ. Идея Radix Sort состоит в том, чтобы выполнять сортировку по цифрам, начиная с младшего разряда до самого старшего. Radix sort использует сортировку с подсчетом в качестве подпрограммы для сортировки.

Алгоритм сортировки Radix

1. Выполните следующие действия для каждой цифры i, где i изменяется от наименее значащей цифры до самой старшей цифры.

Сортировка входного массива с помощью счетной сортировки (или любой стабильной сортировки) в соответствии с i-й цифрой.

Исходный, несортированный список:

170, 45, 75, 90, 802, 24, 2, 66

Сортировка по наименьшей значащей цифре (1-е место) дает:

[* Обратите внимание, что мы сохраняем 802 перед 2, потому что 802 предшествует 2 в исходном списке, и аналогично для пар 170 и 90 и 45 и 75.)

170, 90, 802, 2, 24, 45, 75, 66

Сортировка по следующей цифре (10-е место) дает:

[* Обратите внимание, что 802 снова идет перед 2, поскольку 802 идет раньше

2 в предыдущем списке.)

802, 2, 24, 45, 66, 170, 75, 90

Сортировка по старшей цифре (разряды 100) дает: 2, 24, 45, 66, 75, 90, 170, 802

Каково время работы Radix Sort?

Пусть во входных целых числах d цифр. Radix Sort занимает время O (d * (n + b)), где b - основание для представления чисел, например, для десятичной системы b равно 10. Какое значение d? Если k - максимально возможное значение, то d будет O (logb (k)). Таким образом, общая временная сложность составляет O ((n + b) * logb (k)). Что выглядит больше, чем временная сложность алгоритмов сортировки на основе сравнения для большого k. Давайте сначала ограничим k. Пусть k <= nc, где c - постоянная. В этом случае сложность становится O (nLogb (n)). Но он все еще не превосходит алгоритмы сортировки, основанные на сравнении.

Что, если мы увеличим значение b? Каким должно быть значение b, чтобы временная сложность была линейной? Если мы установим b как n, мы получим временную сложность как O (n). Другими словами, мы можем отсортировать массив целых чисел в диапазоне от 1 до nc, если числа представлены в базе n (или каждая цифра занимает log2 (n) бит).

Является ли Radix Sort предпочтительнее алгоритмов сортировки на основе сравнения, таких как Quick-Sort?

Если у нас есть log2n бит для каждой цифры, время работы Radix будет лучше, чем

эффективно. Кроме того, сортировка Radix использует сортировку с подсчетом в качестве подпрограммы, а сортировка с подсчетом занимает дополнительное место для сортировки чисел.

Рекомендуется: сначала попробуйте свой подход в {IDE}, прежде чем переходить к решению.

Реализация Radix Sort

Ниже приводится простая реализация Radix Sort. Для простоты предполагается, что значение d равно 10. Мы рекомендуем вам просмотреть Сортировку подсчета для получения подробной информации о функции countSort () в приведенном ниже коде.

C++

```
// C++ implementation of Radix Sort
#include <iostream>
using namespace std;

// A utility function to get maximum value in arr[]
int getMax(int arr[], int n)

{
    int mx = arr[0];

    for (int i = 1; i < n; i++)

        if (arr[i] > mx)
            mx = arr[i];
    return mx;

}

// A function to do counting sort of arr[] according to
// the digit represented by exp.
void countSort(int arr[], int n, int exp)

{
    int output[n]; // output array
    int i, count[10] = { 0 };

    // Store count of occurrences in count[]
    for (i = 0; i < n; i++)
```

```
    count[(arr[i] / exp) % 10]++;

// Change count[i] so that count[i] now contains actual
//  position of this digit in output[]
```

```
    // Build the output array
    for (i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    // Copy the output array to arr[], so that arr[] now
    // contains sorted numbers according to current digit
    for (i = 0; i < n; i++)
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
void radixsort(int arr[], int n)

{
    // Find the maximum number to know number of digits
    int m = getMax(arr, n);

    // Do counting sort for every digit. Note that instead
    // of passing digit number, exp is passed. exp is 10^i
    // where i is current digit number
    for (int exp = 1; m / exp > 0; exp *= 10)countSort(arr,
        n, exp);

}

// A utility function to print an array
void print(int arr[], int n)

{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
}

// Driver Code
int main()

{
    int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
    int n = sizeof(arr) / sizeof(arr[0]);


      // Function Call
      radixsort(arr, n);
    print(arr, n);
    return 0;

}Java
```

```
import java.util.*;

class Radix {

    // A utility function to get maximum value in arr[]
    static int getMax(int arr[], int n)

    {
        int mx = arr[0];

        for (int i = 1; i < n; i++)

            if (arr[i] > mx)
                mx = arr[i];
        return mx;

    }

    // A function to do counting sort of arr[] according to
    // the digit represented by exp.
    static void countSort(int arr[], int n, int exp)

    {
        int output[] = new int[n]; // output array

        int i;

        int count[] = new int[10];
        Arrays.fill(count, 0);


        // Store count of occurrences in count[]
        for (i = 0; i < n; i++)
            count[(arr[i] / exp) % 10]++;

        // Change count[i] so that count[i] now contains
        // actual position of this digit in output[]
        for (i = 1; i < 10; i++)
            count[i] += count[i - 1];

        // Build the output array
        for (i = n - 1; i >= 0; i--) {
            output[count[(arr[i] / exp) % 10] - 1] = arr[i];
            count[(arr[i] / exp) % 10]--;
        }

        // Copy the output array to arr[], so that arr[] now
        // contains sorted numbers according to curent digit
        for (i = 0; i < n; i++)
```

```
        arr[i] = output[i];
}

// The main function to that sorts arr[] of size n using
// Radix Sort
static void radixsort(int arr[], int n)
```

```
// Do counting sort for every digit. Note that
            // instead of passing digit number, exp is passed.
            // exp is 10^i where i is current digit number
            for (int exp = 1; m / exp >
                0; exp *= 10)
                countSort(arr, n, exp);
        }

        // A utility function to print an array
        static void print(int arr[], int n)

        {
            for (int i = 0; i < n;
                i++)
                System.out.print(ar
                r[i] + " ");
        }

        /*Driver Code*/
        public static void main(String[] args)

        {
            int arr[] = { 170, 45, 75, 90, 802, 24, 2, 66 };
            int n = arr.length;


              //
            Functi
            on
            Call
            radixs
            ort(ar
            r, n);
            print(
            arr,
            n);
        }
    }
    /* This code is contributed by Devesh Agrawal */
```

Output

```
2 24 45 66 75 90 170 802
```

Consider this input array

| 170 | 45 | 75 | 90 | 802 | 24 | 2 | 66 |
|---|---|---|---|---|---|---|---|

First consider the one's place

Consider this input array

| 170 | 45 | 75 | 90 | 802 | 24 | 2 | 66 |
|---|---|---|---|---|---|---|---|
| 170 | 90 | 802 | 2 | 24 | 45 | 75 | 66 |

Consider this input array

| 170 | 45 | 75 | 90 | 802 | 24 | 2 | 66 |
|---|---|---|---|---|---|---|---|
| 170 | 90 | 802 | 2 | 24 | 45 | 75 | 66 |

Observe that 170 has come before 90 this is because it appeared before in the original list.

Consider this input array

| 170 | 45 | 75 | 90 | 802 | 24 | 2 | 66 |
|---|---|---|---|---|---|---|---|

| 170 | 90 | 802 | 2 | 24 | 45 | 75 | 66 |
|---|---|---|---|---|---|---|---|

| 802 | 2 | 24 | 45 | 66 | 170 | 75 | 90 |
|---|---|---|---|---|---|---|---|

Now consider the 100's place.

Array is Now sorted

| 2 | 24 | 45 | 66 | 75 | 90 | 170 | 802 |
|---|---|---|---|---|---|---|---|