Algorithms

Solve any problem to achieve a rank

View Leaderboard

Topics: Radix Sort

Radix Sort

TUTORIAL PROBLEMS VISUALIZER BETA

Prerequisite: Counting Sort

QuickSort, MergeSort, HeapSort are comparison based sorting algorithms.
CountSort is not comparison based algorithm. It has the complexity of $O(n+k)$, where k is the maximum element of the input array.
So, if k is $O(n)$,CountSort becomes linear sorting, which is better than comparison based sorting algorithms that have $O(nlogn)$ time complexity. The idea is to extend the CountSort algorithm to get a better time complexity when k goes $O(n^2)$. Here comes the idea of Radix Sort.

Algorithm:
For each digit i where i varies from the least significant digit to the most significant digit of a number
Sort input array using countsort algorithm according to ith digit.

We used count sort because it is a stable sort.

Example: Assume the input array is:
10,21,17,34,44,11,654,123
Based on the algorithm, we will sort the input array according to the **one's digit** (least significant digit).
0: 10
1: 21 11
2:

3: 123
4: 34 44 654
5:
6:
7: 17
8:
9:

So, the array becomes 10,21,11,123,24,44,654,17
Now, we'll sort according to the **ten's digit**:
0:
1: 10 11 17
2: 21 123
3: 34
4: 44
5: 654
6:
7:
8:
9:

Now, the array becomes : 10,11,17,21,123,34,44,654
Finally , we sort according to the **hundred's digit** (most significant digit):
0: 010 011 017 021 034 044
1: 123
2:
3:
4:
5:
6: 654
7:
8:
9:

The array becomes : 10,11,17,21,34,44,123,654 which is sorted. This is how our algorithm works.

Implementation:

```
void countsort(int arr[],int n,int place)
{
        int i,freq[range]={0};          //range for integers is 10 as digits
range from 0-9
        int output[n];
        for(i=0;i<n;i++)
```

```
                freq[(arr[i]/place)%range]++;
        for(i=1;i<range;i++)
                freq[i]+=freq[i-1];
        for(i=n-1;i>=0;i--)
        {
                output[freq[(arr[i]/place)%range]-1]=arr[i];
                freq[(arr[i]/place)%range]--;
        }
        for(i=0;i<n;i++)
                arr[i]=output[i];
}
void radixsort(ll arr[],int n,int maxx)                 //maxx is the maximum
element in the array
{
        int mul=1;
        while(maxx)
        {
                countsort(arr,n,mul);
                mul*=10;
                maxx/=10;
        }
}
```

Complexity Analysis:
The complexity is $O((n+b)*log_b(maxx))$ where b is the base for representing numbers and $maxx$ is the maximum element of the input array. This is clearly visible as we make $(n+b)$ iterations $log_b(maxx)$ times (number of digits in the maximum element) . If $maxx \leq n^c$,then the complexity can be written as $O(n*log_b(n))$.

Advantages :
1. Fast when the keys are short i.e. when the range of the array elements is less.
2. Used in suffix array constuction algorithms like Manber's algorithm and DC3 algorithm.

Disadvantages:
1. Since Radix Sort depends on digits or letters, Radix Sort is much less flexible than other sorts. Hence , for every different type of data it needs to be rewritten.
2. The constant for Radix sort is greater compared to other sorting algorithms.
3. It takes more space compared to Quicksort which is inplace sorting.

The Radix Sort algorithm is an important sorting algorithm that is integral to suffix -array construction algorithms. It is also useful on parallel machines.

Contributed by: Shubham Gupta

?