Usman Institute Of Technology
(A Project Of Usman Memorial Foundation)
Affiliated with NED University Of Engineering & Technology

BURSTSORT

DATA STRUCTURES AND ALGORITHM

Group Members

M. Ahsan Khan (17B - 050 - CS)
Syed Saim Jaffri (17B - 036 - CS)
Areeba Asif Baig (17B - 014 - CS)

Introduction

Authors: RANJAN SINHA and JUSTIN ZOBEL (RMIT University).

Sorting is a basic algorithmic job. Formerly many different algorithms like traditional radix sort, MSD radix sort etc were used but these algorithms outperform previous approaches, but do not make decent usage of cache due to the necessity to repetitively re-reference each string. To overcome this a new algorithm was designed named "Burst Sort" in which each string is accessed once only while trie nodes are accessed randomly.

Burst Sort

* In burst sort a trie is used to store the prefixes of strings, with growable arrays of pointers as end nodes containing sorted, unique, suffixes (referred to as buckets).
* The suffixes are added into the buckets.
* As the buckets grow beyond a particular threshold, the buckets are "burst", and the size is increased.
* A more recent variant uses a bucket index with smaller sub-buckets to reduce memory usage.
* Most implementations delegate to multikey quicksort, an extension of three-way radix quicksort, to sort the contents of the buckets.
* By dividing the input into buckets with common prefixes, the sorting can be done in a cache-efficient manner.
* A burst trie contains three distinct components, set of buckets, set of strings and access trie.

Pseudo Code

1.Insertion

```
Create an instance of a trie node
    for i is less than array length
        Start at root each time
        Find the prefix of element from array of strings
        Locate strings prefix in trie node
        Connect bucket to the particular prefix in the trie node
        Add suffix to the bucket according to these conditions
        if the respective prefix bucket size do not reaches limit and is
        less than THRESHOLD
            Insert string suffix in the bucket
        end
        if current bucket size reaches limit and is less than THRESHOLD
            Increase size of the bucket by BUCKET_GROWTH_FACTOR
            Create new array
            Copy the elements from the previous array to the newly
            created array
        end
```

2. Bursting

```
    if bucket size is greater than or equal to THRESHOLD
        Create new trie node
        Connect the newly created trie node to the prefix of the
        bucket
        Search the prefix in the trie node of previous bucket
        elements
        Add their respective suffix to the newly created bucket
    end
end
```

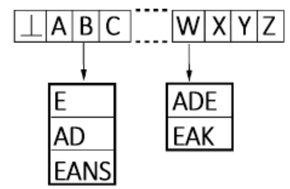
3. Traversal

```
The burst trie is traversed using depth first search from left to right
According to these conditions
if the size of the bucket is one or empty
    The string can be output immediately
end
if the size of the bucket is greater than one
    Apply multikey quicksort for sorting the elements of the strings
    at the time of creation
        Select a pivot randomly from the bucket
        Divide bucket into three partitions
            1* Less than pivot
            2* Greater than pivot
            3* Equal to pivot
        By character-wise comparison between portions to
        arrange the elements
end
```

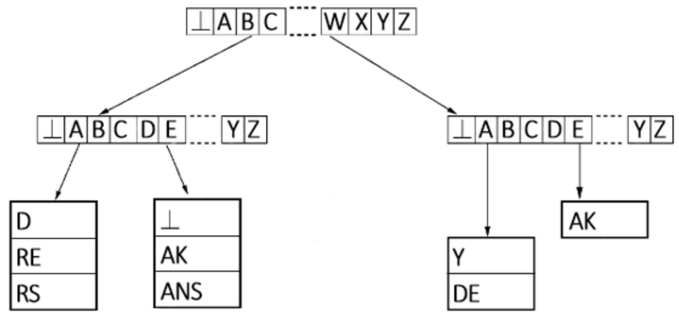
Burst Trie Structure

Array of strings used in the burst trie are:
be, bad, weak, beans, wade.

Before burst:



After bursts:
be, bad, way, bars, beak, weak, bare, beans, wade.



Array of strings generated in sorted order:
bad, bare, bars, be, beak, beans, way, wade, weak.

Multi-Key Quicksort

* It's a hybrid of quicksort and MSD radix sort.
* Working of multi-key quicksort:
  - Randomly select a pivot from the bucket.
  - Distribute bucket into three portions by using three conditions
    1. Less than pivot
    2. Greater than pivot
    3. Equal to pivot
  - Compare between the portions for arranging elements character by character.

Pros & Cons

Pros:

* Fast, efficient data structure for strings.
* Can handle large amount of data.
* Works very fast on large amount of data.
* Has a large number of cache-hits and very little cache-misses.

Cons:

* The efficiency of this algorithm varies with the size of data.

Details

Class: Sorting algorithm

Data structure: Burst Trie

Application of Burstsort

* Telephone directories
* English Dictionaries.

Time Complexity

Worst-case performance: O(wn)

Worst-case space complexity:O(wn)

where 'w' is word length and 'n' is the number of strings to be sorted.