Elimination Algorithm for Unwanted Web Pages on the Web to Obtain Optimized Search & Query Results

Akarsh Goyal & Saurabh Thakur
Computer Science and Engineering, VIT University, Vellore

Abstract: In this paper we discuss the issue of unwanted web pages which are returned as a result of a query on the web at length. This problem is addressed by providing three methodologies. The three methods use the concept of adjacency matrix and perform different computations on it because of which they get different time and space complexities. The results of these methods show that the time needed to obtain the query results is reduced significantly. Hence they provide a better search result for a search query.

Keywords- Web Pages, Adjacency Matrix, Time Complexity, Space Complexity

1. Introduction

The current age is the age of information. We are at the peak of Information Revolution, where the need of organized storage, management and Retrieval only rises. This strenuous job of organizational information retrieval is done by almost every search engine across the globe. With the amount of information on the web rising exponentially it is far beyond any human’s capability to understand and no scope to contain all of it. In this era of information a little more organization will only help. The amount and diversity of Web lead researchers to explore faster and better Web searching algorithms, to reduce the time needed by a user to find what he is searching for. Existing Web search engines only provide restricted services for finding related pages. These services are generally based on text corpora of the WWW and return authorities, i.e. pages with high page rank pointed to by many other pages, and especially by many hubs. But most of these search engines return results to queries which are not optimized and as a consequence a lot of time goes in screening these results one-by-one which is often exhaustive.

In this paper we try to work around this problem. We use link structure of the World Wide Web as input [1-3]. If there is a link from page v to page w, then the author of v recommends page w, and the two pages are usually related. In this way for example we have a set of distinct but related pages. We also have two separate webpages who may or may not recommend each other say M and N. The only similar property of M and N is that both these pages are recommended by same pages in the available structure and by same pages in the available structure and both of them also recommend same set of pages in the available graph structure.

2. Graphical Demonstration

As Shown in the figure “A” below we have two nodes M and N which are identically connected. Identically connected only means that all those pages that have a link to M also have a link to N and all links in M are also there in N.

Suppose we have 2 sub graphs of the graph containing both nodes M and N such that sub graph G1 consists of all nodes except N and sub graph G2 consists of all nodes except M. Graphically speaking we argue that sub graphs G1 and G2 are identical (Isomorphic)[4] to each other where node M in G1 acts as Node N in G2.

The properties of M and N are listed as follows:
1. M and N both have same degree.
2. M and N both have same in degree and same out degree.
3. M and N have incoming links from same nodes, also M and N out link to the same nodes.
3. Detailed Analysis

In the experiments we have assumed a certain sample graph which is represented as an adjacency matrix. The adjacency matrix is a Boolean matrix consisting of 0's and 1's.

We are also assuming that a graph has at least two nodes are identically connected for demonstration purposes. We have assumed this to show that at least two rows of the Boolean matrix are going to be identical. So finally we have an adjacency matrix with at least two rows identical which needs to be reduced to remove the duplicity and conserve time and space.

4. Suggested Methods

Finally the problem comes down to reduction of Boolean matrix by removing all the duplicate rows in the matrix to save space and time. There are many ways to approach this problem but here we suggest only three methods which are in good contrast of each other so that the point which we are trying to make is fully covered and exploited.

4.1. The Brute Force Methodology:

In the Brute Force Method each row is processed one at a time. Now, starting from the second row, for each row, compare the row with already processed rows. If the row matches with any of the processed rows then remove it. If the current row doesn’t match with any row move to the next row. This method is very much time consuming but the space consumed is less.

**Time Complexity:**
$O((\text{no\_of\_rows})^2*(\text{no\_of\_columns}))$

**Space Complexity:**
$O(1)$

4.2. Intermediary Method

This methodology incorporates finding the decimal equivalent of each row and inserting it in a Binary Search Tree (BST’s). Each node of the BST will contain two fields, one field for the decimal value, other for row number. We don’t insert a node if it is duplicated. Finally, traverse the BST and store the corresponding rows. The time and space utilized by this method are between the other two methods discussed in this paper.

**Time Complexity:**
$O((\text{no\_of\_rows})^2*(\text{no\_of\_columns})) + ((\text{no\_of\_rows}) * \log(\text{no\_of\_rows}))$

**Space Complexity:**

4.3. Most Optimized methodology

Exploiting the Boolean nature of the matrix we can use a “Trie Data structure”. In computer science, a trie, also called digital tree and sometimes radix tree or prefix tree (as they can be searched by prefixes), is an ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings.

**Algorithm:**

1. Store the row and insert the row in trie.

Suppose every character of input key is inserted as an individual trie node. Note that the child is an array of pointers to next level trie nodes. The key character acts as an index into the array children. If the input key is new or an extension of existing key, we need to construct non-existing nodes of the key, and mark leaf node. If the input key is prefix of existing key in trie, we simply mark the last node of key as leaf. The key length determines trie depth.

2. Finally, if row is already there then do not insert it again.

3. Storing all the rows inserted in a trie gives us the complete set of unique rows in the adjacency matrix of the graph.

**Example:**

**Input:**

```
{0, 1, 0, 0, 1}
{1, 1, 1, 1, 1}
{0, 0, 0, 1, 1}
{0, 1, 0, 0, 1}
{1, 0, 1, 1, 0}
{0, 0, 0, 1, 1}
{1, 1, 1, 1, 1}
{1, 1, 1, 0, 0}
{0, 0, 0, 1, 1}
```

**Output of the Trie:**

```
{0, 1, 0, 0, 1}
{1, 1, 1, 1, 1}
{0, 0, 0, 1, 1}
```
As one can see clearly from above example 4 rows have been removed out of 9 to form a matrix of 5 rows. The time taken to do this computation is very less as compared to the other two methods discussed here.

Time Complexity:
$O((\text{no} \_ \text{of} \_ \text{rows}) \times (\text{no} \_ \text{of} \_ \text{columns}))$

Space Complexity:
$O(\text{no} \_ \text{of} \_ \text{rows})$

5. Conclusion

An improved algorithm to remove the redundancy in the search result is proposed in this paper. This method makes the use of trie data structure. The results show that compared to other methodologies the trie data structure can clearly be used in the adjacency matrix to reduce the time complexity by order of n. i.e. From $O(n^3)$ to $O(n^2)$

6. Future Work

Since we were able to successfully eliminate the possibly same content pages we can also cluster them with other identically connected pages in forming a more robust storage engine.

We can also know the matching percentage of matching the rows in turn predicting whether the pages have same content.

7. References


