

Improved Double-Skip String Matching Algorithm (IDSA)

Dr. Ghassan Issa & Dr. Hussein Al-Omari
Computer Science Department
Faculty of Computer Science and Information
Technology
Applied Science University, Amman, Jordan
issa@asu.edu.jo ; al_omari@asu.edu.jo

Abdullah Aref & Farah Kandah
Computer Science Department
King Abdullah II School for IT
The University of Jordan, Amman, Jordan

ABSTRACT

The string-matching problem is defined as a given large text string and a pattern string to find all its occurrences in the given text. A modification of one of the recent and fast string-matching algorithms is presented here. The modification was tested with English text. The results suggest a reduction in the average number of comparisons performed up to 36% comparing to the original Double-Skip Algorithm (DSA).

Key words: String search, string matching, double skip, algorithms.

1. Introduction

String matching algorithms try to find an occurrence of a pattern P in a large text T using the smallest number of character comparisons. There are several famous string matching algorithms such as: The Straight Forward Simple Scan (Brute-Force) Algorithm, The Knuth-Morris-Pratt (KMP) Algorithm, The Boyer-Moore Algorithm, and The Double-Skip algorithm.

2. The Double Skip Algorithm

As any string matching algorithm, **DSA** finds all the occurrences of a pattern $P_1...P_m$ in the text $T_1 ... T_n$. It preprocesses the pattern to produce three different arrays: the sign array, the skip array, and the double-skip array. The sign array tells whether the character in T corresponds to the last character in P exists in the given pattern or not. The skip array expresses how much a pattern is to be shifted forward in relation to the text when a match or mismatch is found and the character in text that corresponds to P_m exists in the pattern. The double-skip array expresses how much the pattern to be shifted forward in relation to

the text when the referenced character doesn't exist in the pattern.

The algorithm works as follows: The character T_j will be tested first to see whether it occurs in P or not. This can be done using the sign array. If it doesn't the text is shifted forward according to double-skip array. Otherwise any string-searching algorithm can be applied.

3. The Improved Double Skip Algorithm

As any string matching algorithm, **IDSA** finds all the occurrences of a pattern $P_1...P_m$ in the text $T_1 ... T_n$. It preprocesses the pattern to produce three different arrays: the sign array, the position array, and the double-skip array. The sign array tells whether the character in T corresponds to the last character in P exists in the given pattern or not. The double-skip array expresses how much the pattern to be shifted forward in relation to the text when the referenced character doesn't exist in the pattern. The position array express how much to reduce the double-skip of a pattern in relation to the text when a match or mismatch

is found and the character in text that corresponds to P_m exists in the pattern.

The algorithm works as follows: The character T_j will be tested first to see whether it occurs in P or not. This can be done using the sign array. If it doesn't exist in the P , then the text is shifted forward according to the double-skip array. Otherwise, the algorithm performs a regular matching until the whole pattern is matched or until a mismatch occurs. The position array is used to jump to the proper character to the right of the last one and to the left of that the double-skip array will determine.

The algorithm:

```
int new_double_skip_algorithm(char * T,
    int n, char* P, int m)
{
    unsigned long int i,i0,j,last;
    int position[256];
    int sign[256];
    int double_skip[256];

    //prepare the position table
    for(i=0; i<256; i++)
        position[i]=0;

    for(i=0;i<256;i++)
    {
        for(j=0; j<m-1; j++)
            if(P[j] == i)
                position[i]=j+1;
        position[P[m-1]]=0;
    }

    //prepare the double_skip table
    for(j=0; j<256; j++)
        double_skip[j]=2*m-1;

    for(j=0;j<m;j++)
        double_skip[P[j]]=2*m-j-1;

    //prepare the sign table
    for (j=0; j<256; j++)
        sign[j]=0;

    for (j=0; j<m; j++)
```

```
        sign[P[j]]=1;

    i0=m;
    j=0;
    i=0;
    long int pos;
    int count=0;

    //start the searching process
    while(i0 <= n)
    {
        last=i0-1;
        if(++NCC && !sign[getChar(last)])
            i0 += double_skip[getChar(last+m)];
        else
        {
            i=i0-m;
            j=0;
            count=0;
            while(P[j] == T(i))
            {
                count++;
                if (count==m)
                    break;
                i++;
                j++;
            }

            //skipping step
            pos= position[T [last]];
            i0 +=
                double_skip[rext[last+m-pos]]-pos;
        }
    }
    return 1;
}
```

4. Experimental Results

In this experiment two algorithms were implemented using C++ language. An English text of 1.5 million characters is used. The modification was tested and compared to the original algorithm against the number of comparisons performed, the logical end comparisons, and the number of shifts. The program designed to select randomly 1000 patterns divided into 10 groups. Each group consists of 100 patterns. Table 1 lists the groups and their ranges.

The modification enhanced the number of shifts, the number of comparisons and the number of logical-end comparisons. The following charts summarize the results.

| Group Number | Range of Pattern Length in characters |
|--------------|---------------------------------------|
| 1 | 3-20 |
| 2 | 8-26 |
| 3 | 14-35 |
| 4 | 17-41 |
| 5 | 19-57 |
| 6 | 27-72 |
| 7 | 27-78 |
| 8 | 36-74 |
| 9 | 35-87 |
| 10 | 44-96 |

The figures at the end summarize the obtained results. We got saving in the number of comparisons up to 36% and in the number of shifts up to 35%.

5. Conclusion

This paper introduced a modification to the double-skip string matching algorithm that enhanced its performance. The modification enhanced the number of shifts and the number of comparisons by around 36%, and it improved the number of logical end comparisons.

References:

[1] Mahmoud M. Mhashi “A Faster String Matching Algorithm Using Double-Length Skip Distances”, Mu’tah University, Jordan.

[2] S. Baase and A. Van Gelder, *Computer Algorithms: Introduction to Design and*

Analysis, Third Edition, Addison Wesley, 2000.

[1] K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: Fast access to linkage information on the web. In Proceedings of the 7th International World Wide Web Conference, pages 469–477, Brisbane, Australia, April 1998. Elsevier Science.

[2] S. Brin and L. Page. The anatomy of a large scale hyper textual web search engine. In Proceedings of the 7th International World Wide Web Conference, pp 107– 117, Brisbane, Australia, April 1998. Elsevier Science.

[3] M. Burner. Crawling towards eternity: Building an archive of the world wide web. *Web Techniques Magazine*, 2(5): 37– 40, May 1997.

[4] J. Cho, H. Garcia Molina, and L. Page. Efficient crawling through URL ordering. In Proceedings of the 7th International World Wide Web Conference, pp 161–172, Brisbane, Australia, April 1998. Elsevier Science.

[6] M. Goodrich and R. Tamassia, *Algorithm Design*, John Wiley & Sons, 2002.

[7] V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[8] V. Aho, J. E. Hopcroft, and J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, 1983.

[9] V. Aho and J. D. Ullman, *Foundation of Computer Science*, Compute Science Press, 1992.

[10] Ian Parberry, *Lecture Notes on Algorithm Analysis and Computational Complexity*, Fourth Edition, Computer Science Department, University of Texas, 2001.

[11] R. Sedgewick, *Algorithms*, Addison-Wesley, 1983.

[12] N. Alon, M. Blum, A. Fiat, S. Kannan, M. Naor, and R. Ostrovsky, Matching Nuts and Bolts, In Proc. 5th, Annual Symposium on Discrete Algorithms, Pages 690-696, 1994.

