Shuffle-Selective Search

Abdul Khadar A, Asst Professor, Dept of ISE, SJCIT, Chikkaballapur, Karnataka, India H
Dr. Shrishail Math, Prof & HOD, Dept of CSE, SVCE, Bengaluru, Karnataka, India
Srinivasamurthy, Asso Professor, Dept of CSE, SJCIT, Chikkaballapur, Karnataka, India

Abstract: There are scenarios when one need to find the key element in a set of data, where the key element's presence can be felt but the literal position is the hindrance. In deactivating the intrusion effect on the system where a bulk of nodes is connected, finding the exact node within the network will be the solid challenge. The Shuffle-Selective Search could ease the job.

Introduction: The Shuffle-Selective Search is an approach towards identifying the key element's position within a set of data. Though there a few searching algorithms [1] like linear search, binary search, hash search etc., which are efficient enough in their own context [2], each of these approaches would need comparisons to be done at point of searching the key element. And the main concern of all the searching approaches are time and space. Since comparison is directly proportional to the time of search, it takes the efficiency of the search to a questionable platform. The approach presented in this paper will not perform any comparisons and still will be able to find out the key element but with selective search method. The user needs to input the bucket which the key belongs to three times. The elements are sorted and placed into two different buckets. Each time the user has to confirm the becket where the key present.

The approach: The Shuffle-Selective search will contain 16 elements sorted in ascending order. These elements are divided into two buckets say A and B. The searcher has to confirm the bucket that the key element belongs to. Then the elements are shuffled. Second time the bucket name will provided and the elements are shuffled for the second time. For the third time when the bucket name of the key is provided the last shuffling will happen and the key element will be available after the bucket name is given. The C code for the same is as follows.

The C code:

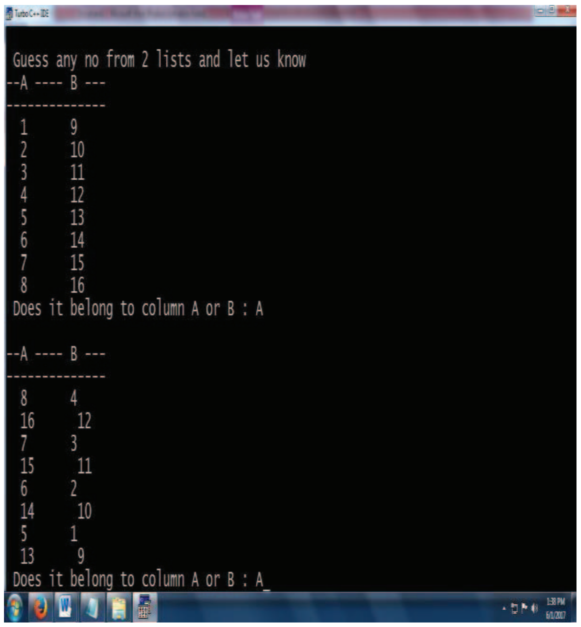
```
#include<stdlib.h>
int  a[16]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,
15,16};
int b[16],i,j,n,A,B;
char ch;
main()
{
        clrscr();
        printf("\n Guess any no from 2
                  buckets and let us know");
        printf("\n--A ---- B ---");
        printf("\n--------------");
        for(i=0;i<=7;i++)
                printf("\n%d%d",a[i],a[i+8]);

        shuffle();
        for(i=0;i<=15;i++)
                a[i]=b[i];
        shuffle();
        for(i=0;i<=15;i++)
        a[i]=b[i];
        shuffle();
        for(i=0;i<=15;i++)
                a[i]=b[i];

        printf("\nIs key in bucket A or B : ");
        ch=getchar();
        if(ch=='A')
                printf("\nKey is %d",a[2]);
        else
                printf("\nKey is %d",a[10]);
        getch();
}
shuffle()
{
        int j;
        printf("\nIs key in bucket A or B : ");
        ch=getchar();
        if(ch=='A')
        {
                for(i=0,j=7;i<=15; i++,j--)
                {
                        b[i]=a[j];
                        b[++i]=a[j+8];
                }
        }
        else if(ch=='B')
        {
                for(i=0,j=7;i<=15; i++,j--)
                {
                        b[i]=a[j+8];
                        b[++i]=a[j];
                }
        }
        printf("\n--A ---- B ---");
        printf("\n--------------");
        for(i=0;i<=7;i++)
                printf("\n%d%d",b[i],b[i+8]);
        getch();
        flushall();
}
```

The shuffle function will mix the elements belonging to two different buckets based on the searcher's input for the place of the key in search. If the input is A then the elements are loaded into the array B by taking the one element from each bucket from the bottom end and placed with bucket A's element first and then bucket B's element till last element of each bucket is loaded to array B. This process will be repeated three times. Then the key is displayed as the result when final time bucket name provided. The key after 3rd shuffle will always be the 3rd element of the user opted bucket.

The result of the code for one set of choices is as given below.

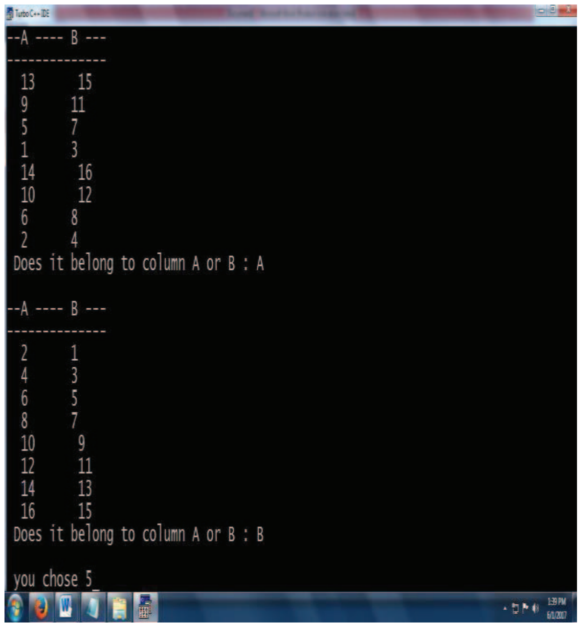


```
Guess any no from 2 lists and let us know
--A ---- B ---
-------------
 1     9
 2     10
 3     11
 4     12
 5     13
 6     14
 7     15
 8     16
Does it belong to column A or B : A

--A ---- B ---
-------------
 8     4
 16     12
 7     3
 15     11
 6     2
 14     10
 5     1
 13     9
Does it belong to column A or B : A_
```

Figure 1(a) showing the output before 2nd input from the user.

The option was for key 5 which belongs to bucket A for the first time. After the first shuffle the key is present in the bucket A so the bucket name A is input.



```
--A ---- B ---
-------------
 13     15
 9     11
 5     7
 1     3
 14     16
 10     12
 6     8
 2     4
Does it belong to column A or B : A

--A ---- B ---
-------------
 2     1
 4     3
 6     5
 8     7
 10     9
 12     11
 14     13
 16     15
Does it belong to column A or B : B

you chose 5_
```

Figure 1(b) showing the output 3rd input from the user and bucket selection.

After the second shuffle the key is present in the bucket A so the input is A. After 3rd shuffle the key is present in the bucket B so the input is B. Once the 3rd shuffle performed the user selects the bucket B because key 5 present in bucket B and the 3rd element of the bucket B is the key that was opted.

Applications:

1. To identify the intrusion in the Advanced Persistent Threats [4] where the intrusion does happen after a very long trial for intrusion.
2. In network of nodes [3] where one of the node is indulging in non-ethical activity, to fix the node.

Future Enhancement:

The approach is being demonstrated for 16 elements but could be enhanced with slight variation in the approach's data. Where the number of inputs required may change.

Conclusion:

The Shuffle-Selective Search could be very effective in intrusion detection and miss-behaving node [3] identification in a network. Since no comparisons of key element with the set of elements this approach could be used to provide faster and efficient selective searching areas.

References:

[1]. "SORTING AND SEARCHING ALGORITHMS" by Thomas Niemann

[2]. "Efficient Search for Robust Solutions by Means of Evolutionary Algorithms and Fitness Approximation" Ingo Paenke, Jürgen Branke, Member, IEEE, and Yaochu Jin, Senior Member, IEEE.

[3]. "Detection of misbehaving node using Secure Acknowledgement in MANET" Rasika R. Mali, Sudhir T. Bagade Department of Computer Science and Technology, Usha Mittal Institute of Technology, SNDT University, Mumbai - 400049, India.

[4]. "Enhanced Protection for Big Data using Intrusion Kill Chain and Data cience" by Abdul Khadar A, Dr. Shrishail Math, Srinivasamurthy, Karnataka, India, CSI Communications, ISSN 0970-647X, Volume No. 41, Issue No. 1, April 2017.