IJCSE International Journal of Computer Sciences and Engineering Open Access

Research Paper Vol.-7, Issue-7, July 2019 E-ISSN: 2347-2693

Octonary (O) Search Algorithm

Bhavesh R Maheshwari

DNV International Education Academy, Gandhidham-Kachchh, Gujarat, India

Corresponding Author: Maheshwari.bhavesh27@gmail.com

DOI: https://doi.org/10.26438/ijcse/v7i7.252256 | Available online at: www.ijcseonline.org

Accepted: 15/Jul/2019, Published: 31/Jul/2019

Abstract: Searching is the process of finding a given value position in a list of values. It is fundamental operation in computer science.

It is achieved by different searching methods which require number of iterations to reach at desired piece of data. In this research paper, another octonary (o) search algorithm is developed to compare & make searching even faster than algorithms like sequential, binary, ternary, & quaternary search.

Keyword **-** Search Algorithm**,** Searching.

I. INTRODUCTION

A fundamental operation of computer is to store large amount of information & retrieval of information as quickly as possible. It is achieved by different searching methods. All these algorithms iterate loop to find out the desired input data form a collection of stored data. For e.g. an Array. In this research paper, an algorithm is developed to make searching faster in large collection of data & this algorithm is also compared with different algorithms based on an approximate number of search loops. This algorithm is named as Octonary (O) Search because it divides sorted array into eight parts. After division of an array seven indexes are obtained which represents an array element & are compared to the key which is inputted from user to be searched. If the inputted key value matches any of the values represented by seven indexes then the search is successful. Else Algorithm will find any one part's range of indexes where availability of key is possible. This process will be repeated until key is found or a list is exhausted & search is unsuccessful.

II. WORKING OF DIFFERENT SEARCH METHODS

A search algorithm is the step-by-step procedure used to locate specific data among a collection of data. It is considered a fundamental procedure in computing.

Sequential Search: Sequential search is a method for finding a target value within a list. It sequentially checks each element of the list for the target value until a match is found or until all the elements have been searched.

Binary Search: Binary search that finds the position of a target value within a sorted array. Binary search compares the target value to the middle element of the array [1].

Ternary Search: In binary search, the sorted array is divided into two parts while in ternary search, it is divided into three parts and then algorithm determines in which part the element exists [2].

Quaternary Search: A four search locates an item in a sorted array by repeatedly dividing the search interval into four equal parts.

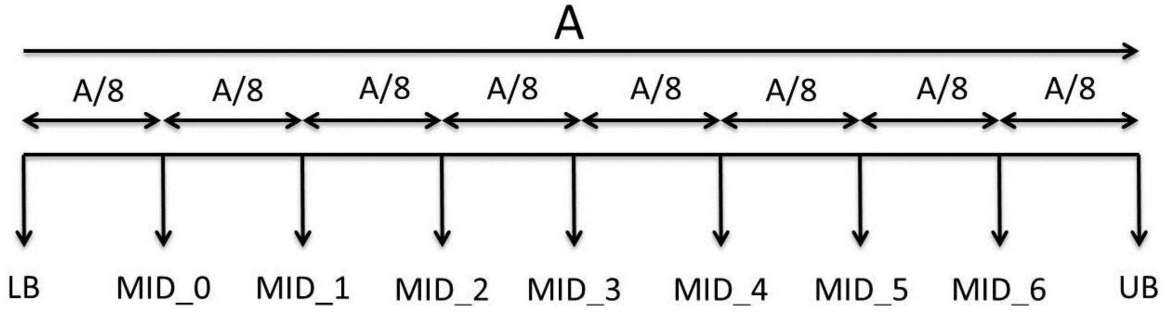
III. WORKING OF OCTONARY(O) SEARCH

In these algorithm eight searches locates an element in a sorted array by repeatedly dividing the search interval into eight equal parts. In other words this algorithm works by partitioning the array into eight equal parts. After partitioning seven elements are obtained. The input key is now compared to these seven elements. If any one of them matches with input key then the location of that element is obtained else algorithm will find new range of indexes from any one of eight parts where availability of key is possible. This procedure is repeated until a key is found or an array is exhausted.

Goal

We need a general formula for partitioning the array into eight equal parts for obtaining seven values which will be used for comparison. Given below diagram shows partitions of an array.

Diagram



Proof

Let the leftmost & the rightmost index of an array A is LB (lower bound) & UB (upper bound) respectively.
Let the middle element at ½ distances from 'LB' to 'UB' be 'MID_3'.
Let the element at 1/2 distances from 'LB' to 'MID_3' be 'MID_1'.
Let the element at 1/2 distances from 'LB' to 'MID_1' be 'MID_0'.
Let the element at 1/2 distances from 'MID_1' to 'MID_3' be 'MID_2'.
Let the element at 1/2 distances from 'MID_3' to 'UB' be 'MID_5'.
Let the element at 1/2 distances from 'MID_3' to 'MID_5' be 'MID_4'.
Let the element at 1/2 distances from 'MID_5' to 'UB' be 'MID_6'.
Now,
MID_3 = (LB+UB) / 2
MID_1 = (LB+MID_3) / 2
MID_0 = (LB+MID_1) / 2
MID_2 = (MID_1+MID_3) / 2
MID_5 = (MID_3+UB) / 2
MID_4 = (MID_3+MID_5) / 2
MID_6 = (MID_5+UB) / 2

For Example,
We assume that array indexes starts from 1.
A[100]={1,2,3,4,5,6,7,8,9,10,
11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25,26,27,28,29,30,
31,32,33,34,35,36,37,38,39,40,
41,42,43,44,45,46,47,48,49,50,
51,52,53,54,55,56,57,58,59,60,
61,62,63,64,65,66,67,68,69,70,
71,72,73,74,75,76,77,78,79,80,
81,82,83,84,85,86,87,88,89,90,
91,92,93,94,95,96,97,98,99,100}

The number of times the loop executes is given by Log_8^n where n is the number of elements of the array. This is because the array is divided into eight equal parts every time the loop is executed.

Number to be searched (KEY) = 22
Lower bound (LB) = 1
Upper bound(UB) = 100
By use of our equations values obtained are,
MID_0 = 13
MID_1 = 25
MID_2 = 37
MID_3 = 50
MID_4 = 62
MID_5 = 75
MID_6 = 87
Key is not equal to MID_0, MID_1, MID_2, MID_3, MID_4, MID_5 or MID_6.
Now, MID_0<Key< MID_1
Hence, the search is repeated between MID_0 & MID_1.
Now,
LB = MID_0 + 1
UB =MID_1 - 1
LB = 14
UB = 24
Key = 22
Therefore,
MID_0 = 15
MID_1 = 16
MID_2 = 17
MID_3 = 19
MID_4 = 20
MID_5 = 21
MID_6 = 22
Now, Key is equal to MID_6 & index of MID_6 is obtained.

Working of Octonary (O) search is explained in given below algorithm.
This algorithm is divided into two parts.

1. Calling recursive octonary function and passing initial leftmost and rightmost indexes and a key.
2. A recursive function octonary where mid values are obtained by leftmost and rightmost values and search will be performed.

IV. ALGORITHM OCTONARY (O) SEARCH

Part 1. Calling Octonary.

Input: A [1….N] is an array of N elements and K is the key to be searched, where N is the total number of elements in an array A.

MID is an array that stores mid values in it.

Output: A successful message and the location of the array element where K matches, otherwise unsuccessful message and return -1.

Remarks: All elements in are arranged in ascending order and array indexes ranges from 1 to N. L is the leftmost index of an array and R is the rightmost index of an array.

Steps:

```
1.  START
2.  L = 1, R = N
3.  I = OCTONARY(L,R,K)
4.  IF (I != -1) THEN
5.  PRINT "Successful" at Location
6.  I
7.  ELSE
8.  PRINTF "Unsuccessful"
9.  END IF
10. STOP
```

Part 2. Octonary Procedure.

Input: Leftmost and rightmost indexes from an array and a key to be searched.

Output: Returning an array index to the

Remark: LB and UB are lower and upper bound index ranges of an array where search will be performed. Initially LB=L, UB=R.

Proof that octonary search is faster than binary, ternary, and quaternary search (the amount of time taken by if-else statement is unconsidered).

Given below table shows approximate number of search loops & comparison between different search algorithms.

Table 1

Number of Elements (n)	BIN Log_2^n	TERN Log_3^n	QUATER Log_4^n	OCTO Log_8^n
10	3.32	2.09	1.16	1.10
100	6.64	4.19	3.32	2.21
1000	9.96	6.28	4.98	3.32
10000	13.28	8.38	6.64	4.42
100000	16.60	10.47	8.83	5.53
1000000	19.93	12.57	9.96	6.64
10000000	23.25	14.67	11.62	7.75
100000000	26.57	16.76	13.28	8.85

It is seen that number of loops runs for octonary search is

Steps:

```
1.  START
2.  IF (UB>=LB) THEN
3.  MID[3]=(LB+UB)/2
4.  MID[1]=(LB+MID[3])/2
5.  MID[0]=(LB+MID[1])/2
6.  MID[2]=(MID[1]+MID[3])/2
7.  MID[5]=(MID[3]+UB)/2
8.  MID[4]=(MID[3]+MID[5])/2
9.  MID[6]=(MID[5]+UB)/2
10. IF(A[MID[0]]==K)
11. RETURN MID[0]
12. ELSE IF (A[MID[1]]==K)
13. RETURN MID[1]
14. ELSE IF (A[MID[2]]==K)
15. RETURN MID[2]
16. ELSE IF (A[MID[3]]==K)
17. RETURN MID[3]
18. ELSE IF (A[MID[4]]==K)
19. RETURN MID[4]
20. ELSE IF (A[MID[5]]==K)
21. RETURN MID[5]
22. ELSE IF (A[MID[6]]==K)
23. RETURN MID[6]
24. ELSE IF(K<A[MID[0]])
25. RETURN
    OCTONARY(LB,MID[0]-1,K)
26. ELSE IF(K<A[MID[1]])
27. RETURN
.   OCTONARY(MID[0]+1,MID[1]-1,K)
28. ELSE IF(K<A[MID[2]])
29. RETURN
    OCTONARY(MID[1]+1,MID[2]-1,K)
30. ELSE IF(K<A[MID[3]])
31. RETURN
    OCTONARY(MID[2]+1,MID[3]-1,K)
32. ELSE IF(K<A[MID[4]])
33. RETURN
    OCTONARY(MID[3]+1,MID[4]-1,K)
34. ELSE IF(K<A[MID[5]])
35. RETURN
    OCTONARY(MID[4]+1,MID[5]-1,K)
36. ELSE IF(K<A[MID[6]])
37. RETURN
    OCTONARY(MID[5]+1,MID[6]-1,K)
38. ELSE
39. RETURN
.   OCTONARY(MID[6]+1,UB,K)
40. END IF
41. RETURN -1.
42. END IF
43. STOP
```

Above table shows that octonary search

Is terminated after 6th iteration. Where binary, ternary & quaternary search terminated after 14th, 9th and 8th iteration respectively. By observation it is also seen that in octonary

three times lesser than binary search, approximately two times lesser than ternary search and one and half time lesser than quaternary search.
Given below table shows different search algorithms applied on 10000 elements.
It shows total number elements which can be found in a particular iteration.

Table 2

Iteration Number.	BIN	TERN	QUATER	OCTO
1.	1	2	3	7
2.	2	6	12	56
3.	4	18	48	448
4.	8	54	192	3584
5.	16	162	768	4608
6.	32	486	3072	1297
7.	64	1458	4305	□
8.	128	4374	1600	
9.	256	3440	□	
10.	512	□		
11.	1024			
12.	2048			
13.	4096			
14.	1809			
15.	□			

□ denotes unsuccessful search.

```
//Function Octonary
int octo(int lb, int ub,int k)
{
if(ub>=lb)
{
mid[3]=(lb+ub)/2;
mid[1]=(lb+mid[3])/2;
mid[0]=(lb+mid[1])/2;
mid[2]=(mid[1]+mid[3])/2;
mid[5]=(mid[3]+ub)/2;
mid[4]=(mid[3]+mid[5])/2;
mid[6]=(mid[5]+ub)/2;
if(a[mid[0]]==k)
return mid[0];
else if(a[mid[1]]==k)
return mid[1];
else if(a[mid[2]]==k)
return mid[2];
else if(a[mid[3]]==k)
return mid[3];
else if(a[mid[4]]==k)
return mid[4];
else if(a[mid[5]]==k)
return mid[5];
else if(a[mid[6]]==k)
return mid[6];
```

search maximum number of elements could be found in 5th iteration where maximum number of elements could be found in 13th iteration of binary search, 8th iteration of ternary search and 7th iteration of quaternary search.

Where can Octonary Search be Helpful?

It can be used when user have large set of data. For example: user wants to search a name of particular customer from a table which contains details about millions of customers.

An Implementation of Octonary Search Using C Language.

```
#include<stdio.h>
#include<conio.h>
int a[10000],i,j,key,l=0,r,mid[7],res,c,n;
void main()
{
printf("Enter Total Number of Elements in List:");
scanf("%d",&n);
r=n;
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Enter a Key To Be Searched:");
scanf("%d",&key);
//Calling Function for Finding Mid Values & //Storing in Array of Mids
res=octo(l,r,key);
if(res!=-1)
{
printf("\n#~~Sucessful: %d is Available at %d~~#",a[res],res);
}
else
{
printf("\n#~~Unsucessful~~#");
 }
getch();
}
```

V. CONCLUSION

The Octonary (O) search algorithm is designed to search desired of inputted data from large pool of storage faster than other search algorithms mentioned above. This project will help to perform search quickly in huge database.

REFERENCES

[1] D Samanta, Classic Data Structure(2nd Edition), PHI Learning Private Limited ISBN - 10: 9788120337312 PP. 722-724
[2] Taranjit Khokhar, September 2016, IJIRT, Volume-3, Issue-4, PP.143908. ISSN: 2349-6002.

```
else if(k<a[mid[0]])
return octo(lb,mid[0]-1,k);
else if(k<a[mid[1]])
return octo(mid[0]+1,mid[1]-1,k);
else if(k<a[mid[2]])
return octo(mid[1]+1,mid[2]-1,k);
else if(k<a[mid[3]])
return octo(mid[2]+1,mid[3]-1,k);
else if(k<a[mid[4]])
return octo(mid[3]+1,mid[4]-1,k);
else if(k<a[mid[5]])
return octo(mid[4]+1,mid[5]-1,k);
else if(k<a[mid[6]])
return octo(mid[5]+1,mid[6]-1,k);
else
return octo(mid[6]+1,ub,k);
}
return -1;
}
```

Author's Profile

Mr. Bhavesh R Maheshwari pursed Bachelor of Computer Application from KSKV Kachchh University of Kachchh-Gujarat in 2010 and Master of Science in Information & Communication Technology from Veer Narmad South Gujarat University in year 2012. He is UGC NET Qualified and currently working as Assistant Professor in DNV International Education Academy, Gandhidham - Kachchh, Gujarat, India since 2013 his main research work focuses on Data Structure Algorithms, Information & Communication Technology. He has 6 years of teaching experience and 1 year of Research Experience.