International Journal of Latest Trends in Engineering and Technology
Vol.(7)Issue(4), pp.218-224
DOI: http://dx.doi.org/10.21172/1.74.029
e-ISSN:2278-621X

INTERPOLATION SEARCH: A MEMOIZED APPROACH

Deepti Verma[1] and Dr. Kshama Paithankar[2]

Abstract: While dealing with a large amount of data, methods/ techniques are to be used to perform searching appropriate data. Time is the core concern while evaluating performance of these techniques. While addressing this issue, for further time minimization, an efficient method as an extension of binary search namely Interpolation Search technique has been emerged. Observations led to move towards optimization of the performance of this algorithm. Literature reveals optimization of time in various algorithms achieved through an approach of memoization. Hence, interpolation search technique is implemented with memorization in this paper. The results indicate a significant time reduction and thus achieve the objective of improving performance of interpolation search technique.

Keywords: Interpolation Search, Memoization, Execution Time.

I. INTRODUCTION

Searching is the process of finding a particular item in an available collection of items. Literature and study material is evident of existence of many searching methods/techniques especially in the area of computer science. While dealing with a large amount of data, such process is the core to access the selective items in processing. Mainly, linear search, binary search etc. are available to perform searching of data. The performance of any searching algorithm is evaluated primarily in terms of time required to seek data item or specific value. It has been observed that this seek time is a major issue with existing searching algorithms. However, binary searching algorithm proves better among the available algorithms. This issue has been taken up for further time minimization and hence an efficient method as an extended version of binary search namely Interpolation Search technique has been emerged.

Verification of the performance of Interpolation Search led to move towards optimization of the performance of this algorithm. It has been observed that for optimization of time in various algorithms, memoization has proved to be a method to achieve this objective[1]. Memoization is also known as function caching. It was first introduced in 1968 in the context of Artificial Intelligence [2]. It is a way for machines to learn from past experiences. Here, it is proposed to develop an algorithm of interpolation search *MemIPS()* with memoized approach to optimize the time of element searching.

[1] *Shri Vaishnav^{SM} Institute of Management, Indore*
[2] *Shri Vaishnav^{SM} Institute of Management, Indore*

The concept of memoization is described in section 2. Section 3 includes the formal description of Non-memoized Interpolation search function *NIPS()*. Formal description of memoized Interpolation Search function *MemIPS()* is presented in section 4. Section 5 deals with case study to show performance of *MemIPS()* and *NIPS()*. Section 6 covers discussion on results in different cases. Finally, we end-up with the conclusion and future scope in section 7.

II. MEMOIZATION

Memoization proves to be a technique for reducing the execution time of program [3]. Memoized function stores the output and provides it when user calls same function again with same value [4]. In other words, using memoization programs could “recall” previous computations and thus avoid repeated work [5][6]. The key idea behind this is to speed up the execution of a function by maintaining the cache of its previous computations and look-up into the cache instead of computing data repeatedly.

Normally, if functions are being called more than once, its computational code is executed and the results of each iteration are to be stored. It consumes time and affects the memory utilization as well. Excluding for the first time, in every next step, results are looked up in cache resulting in reducing redundancy, computation time and thus improvement of performance and efficiency [7][8].

III. FORMAL DESCRIPTION OF *NIPS()*

Interpolation search is a classical method for searching through ordered random data[9]. It is retrieving a desired record by key in an ordered file by using the value of the key and the statistical distribution of the keys. It works on Divide and Conquer method[10]. Here, the algorithm *NIPS()* is presented.

Algorithm *NIPS()*

```
/*arr[] represent the data set*/
/*n1 contains the size of data set*/
/*item is an integer type variable which holds the key value which we want to search */
/*mid is a class level integer variable*/
/*last and first are the integer type variables to hold the first and last position value*/
Start
Step 1: int inpoSearch (int arr[],int n1,int item)
/*Function declaration with the parameters of data set and key value and size of data set.*/
Step 2:  first = 0 ;
         last = n1-1;
 /* initializes first with 0 and size of data set is stored in last.*/
Step 3: while(first <= last)
        {mid = first + (last - first)*((item   - arr[first]) / (arr[last] - arr[first]));
         if (arr[mid] == item)
         {return 1;}
```

```
/*To find the mid, and if this element matches the item then return 1*/
Step 4:     if (item < arr[mid])
        last = mid -1;
        else
        first = mid + 1;
/*Updating values of first and last as per the condition. */
Step 5: if (first > last)
return 0;
/*returning the value 0 if no search*/
End
```

IV. FORMAL DESCRIPTION OF *MemIPS()*

In the previous section, regular *NIPS()* has been discussed. Now in this section, *MemIPS()* algorithm is being presented. *MemIPS()* accepts the values entered by the user and verifies the output. The value found is stored in variable and whenever needed, this variable will be extracted as store value. The process continues till the result is available.

Algorithm *MemIPS()*

```
/* v is an integer type variable*/
/* st1 and ed1 are  integer type local variables*/
/* b is an integer type local variable used to retrieve output*/
Start
Step 1: st1=Systemtime
/* to stores starting time */
Step 3: b=d1.memoIps();
/* Retrieving output */
ed1=Systemtime;
/*getting ending time */
Step 4:   tot1=(ed1-st1);
Message ("Memo Time "+tot1);
/*Showing total time with memoization*/
Step 5: if(v==1)
Message ("Value Found= "+b);
/*When the value is found then show the output with time*/
End
```

V. CASE STUDY

The performance of proposed algorithm *MemIPS()* and that of *NIPS()* has been evaluated using three cases. Case 1 includes the study for data size 10 whereas Case 2 deals with data size as 25. Data size 50 was considered for study in Case 3. These different cases have been studied for the *NIPS()* as well.

Case 1: Study for data size 10

Here, the performance of *NIPS()* and *MemIPS()* is discussed with the list containing 10 values. Table-1 illustrates the performance in terms of time whereas Figure-1 represents the trend of time including that memoized function improve the performance consistently.

Table-1: Dataset of 10 values (Values from 10 to 100)

Search Value	Non Memoized Time	Memoized Time	Difference
10	2265	1359	906
20	2718	1359	1359
30	3170	1358	1812
40	3171	1359	1812
50	3623	1359	2264
60	3623	1359	2264
70	4982	1359	3623
80	4529	1359	3170
90	4528	1359	3169
100	2264	1358	906
Not Found	2717	1812	905

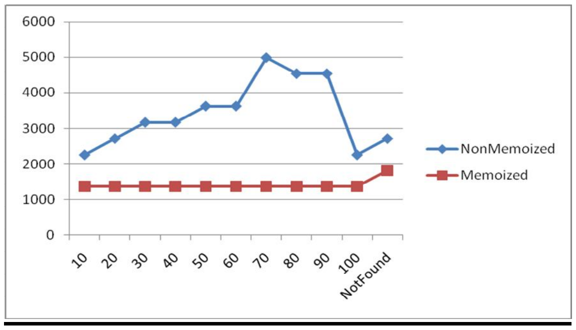

Figure-1: Performance of *MemIPS()* vs *NIPS()* in Case 1.

Case 2: Study for data size 25

In this Case, the list of dataset containing 25 values is experimented with proposed *MemIPS()* and *NIPS()* as well as shown in Table-2. Similarly, Figure-2 represents the trend of time. In this Case increasing the size of the dataset also results in improving time and thus the performance of algorithm using memoization.

Table-2: Dataset of 25 values (Values from 10 to 250)

Search Value	Non Memoized Time	Memoized Time	Difference	Range

50	7699	1358	6341	10-100
80	4076	1359	2717	10-100
120	4982	1359	3623	100-150
140	5888	1359	4529	100-150
170	5888	1811	4077	150-200
190	6340	1359	4981	150-200
220	7693	1358	6335	200-250
240	7699	1359	6340	200-250
110	4076	1358	2718	random
210	6793	1358	5435	random
Not Found	7699	1812	5887	

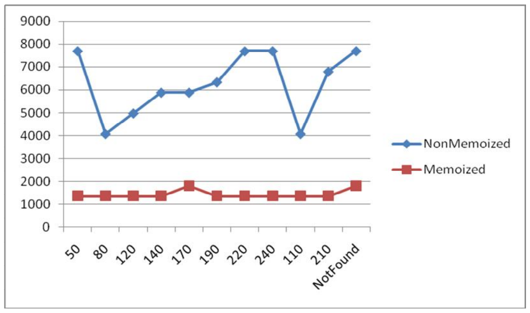

Figure-2: Performance of *MemIPS()* vs *NIPS()* in Case 2.

Case 3: Study for data size 50

A list containing 50 values for implementing *NIPS()* and *MemIPS()* is used in this Case. Table-3 highlights the outcome in terms of time and Figure-3 represents the trend of this time. Here, it is observed that the performance of *MemIPS()* still follows the trend of optimization of time to improve performance of searching.

Table-3: Dataset of 50 values (Values 10 to 500)

Search Value	**Non Memoized Time**	**Mem oized Time**	**Diffe rence**	**range**
50	3170	1359	1811	10-100
80	4982	1359	3623	10-100
120	4982	1359	3623	100-200
180	5887	1359	4528	100-200

210	6794	1359	5435	200-300
250	7699	1359	6340	200-300
350	9511	1358	8153	300-400
370	9963	1359	8604	300-400
420	11322	1358	9964	400-500
460	11775	1359	10416	400-500
300	8152	1359	6793	random
440	11775	1359	10416	random
NotFound	2717	1359	1358	NotFound

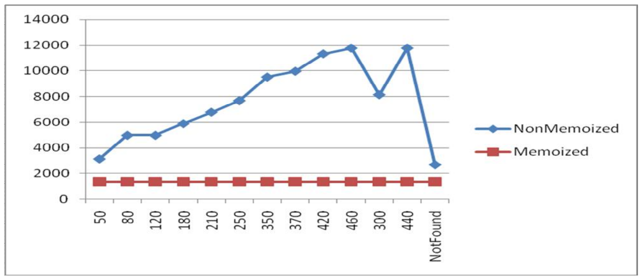

Figure-3: Performance of *MemIPS()* vs *NIPS()* in Case 3.

VI. DISCUSSION

With the help of three cases, it has been noticed that *MemIPS()* really improves the performance of searching as compare to regular *NIPS()*. For instance, in Case 1 the time recorded to search the first element is 226 5ms using *NIPS()* whereas using *MemIPS()* it is noted 1359 ms thereby reducing the time by 906 ms clearly as shown in Table-4.

Table-4: Comparative Performance in different Cases

case	**Index**	**Search Value**	**Non Memoized Time**	**Memoized Time**	**Difference**
1	First	10	2265	1359	906
	Mid	50	3623	1359	2264
	Last	100	2264	1358	906
2	First	50	7699	1358	6341
	Mid	170	5888	1811	4077
	Last	210	6793	1358	5435
3	First	50	3170	1359	1811
	Mid	210	6794	1359	5435
	Last	460	11775	1359	10416

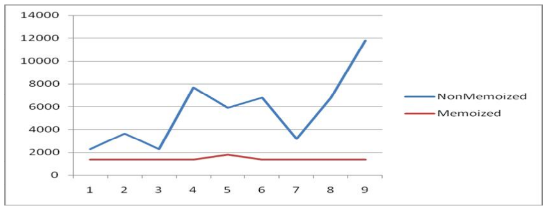

Figure-4: Cumulative bracket of *MemIPS()* and *NIPS()*.

Similarly, for mid element, time recorded is 3623 ms using *NIPS()* and in *MemIPS()* it is 1359 ms. Here, time is reducing by 2264 ms and in the case of last element the searching time is recorded using *NIPS()* is 2264 ms where as using *MemIPS()* it is computed as1358 ms thereby reducing by 906 ms. This trend is observed in Case 2 and Case 3 also as represented by Figure-4. Presently, the performance of *MemIPS()* is evaluated for first, middle and last elements of the lists. However, the trends of performance indicate that it may surely be applicable for the element at any position in the list.

VII. CONCLUSION

In this paper, the proposed *MemIPS()* proves to be a effective algorithm that optimizes the execution time with optimum utilization of memory and thus improving the overall performance of the Regular Interpolation Search algorithm. However, there exists scope of further enhancement in many aspects. Presently a single data type has been used where as multiple data types may be worked in order to improve the performance proportionally. Further, data types long, float and double may be used leading to more accuracy of result.

REFERENCES

[1] Purey, J., Paithankar, K., *Memoization: a Technique to optimize Performance of Searching*, National Conference on "Challenges of Globalization and Strategies for Competitiveness" Shri Vaishnav Institute of Management, Indore, January, 2015, pp 486-491.

[2] Norvig,P. "*Techniques for Automatic Memoization with Applications to Context-Free Parsing*", University of California, Volume 17,Issue1, March 1991, pp 91-98 .

[3] Pfeffer,A. "*Sampling with Memoization*", School of Engineering and Applied Sciences, Harvard University, 2007.

[4] Crockford, D. "*Java Script The Good Parts*", O'Reilly, May 2008, pp 44-45.

[5] Ziarek, L; Sivaramakrishnan,K.C.and Jagannathan, S. "*Partial Memoization of Concurrency and Communication*", Department of Computer Science Purdue University, Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming. Edinburgh, Scotland, ACM, pp. 161 172.

[6] Acar,U.A.; Blelloch,G.E. and Harper,R. "*Selective Memoization*", School of Computer Science Carnegie Mellon University Pittsburgh, PA 15213, 2003.

[7] Brown,.D. and Cook, W.R. "*Monadic Memoization Mixins*", Department of Computer Sciences, University of Texas at Austin, 2006.

[8] Purey,J and Muley,K. "*Auto Response Memoization using JAVA*", National Conference on Emerging Technologies in Electronics, Mechanical and Computer Engineering (ETEMC) April 2010.

[9] Graham, S.L. and Rivest, R.L. "*Interpolation Search A Log LogN Search*", 3 July 1993.

[10] Demaine, E.D.; Jones, T. and Patrascu,M. "*Interpolation Search for Non- Independent Data*", Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA) , 2004, pp 529–530.