Journal of
Data Acquisition and Processing

ISSN: 1004-9037
https://sjcjycl.cn/
DOI: 10.5281/zenodo.777721

A NEW ALGORITHM FOR SENTINEL SEARCH

Mrs.Deepali P. Pawar
Assistant Professor, Department Of Computer Engineering, SNJB'S Late Sau. K. B. Jain College of Engineering Chandwad, India, pawar.dpcoe@snjb.org

Dr.M.R.Sanghavi
Vice Principal, Department Of Computer Engineering, SNJB'S Late Sau. K. B. Jain College of Engineering Chandwad, India, sanghavi.mrcoe@snjb.org

Raj Ratanraj Shah
Department Of Computer Engineering, SNJB'S Late Sau. K. B. Jain College of Engineering Chandwad, India, shah.raj1103@gmail.com

Akshat Rajababu Tated
Department Of Computer Engineering SNJB'S Late Sau. K. B. Jain College of Engineering Chandwad, India, akshattated02@gmail.com

Abstract- This study investigates sentinel search algorithms with the aim to provide a novel approach that will revolutionize the way we search for specific elements in arrays or lists. The purpose of this work is to explore the drawbacks of traditional sentinel search methods and to propose a significant improvement. By replacing the array's initial member with the target element itself, the technique ensures that the target element is always found at the beginning, even if it is not present in the array. This ingenious modification guarantees that the search process has an O(1) constant time complexity, which significantly increases efficiency.The significance of this finding lies in its ability to get around the issues that array borders present. The suggested method does away with the need to check for the array's end after each iteration, which minimizes errors and inefficiencies. Additionally, the reduction in temporal complexity speeds up search operations, enhancing the overall performance of programs that rely on sentinel search.By sharing this knowledge, we advance search algorithms and give computer scientists and software engineers a strong tool for code optimization. The ramifications of this research go beyond sentinel search alone because the concepts and techniques they present have the potential to inspire fresh approaches to algorithm creation and analysis.

1. Introduction

Sentinel linear search introduces a modified technique to efficiently locate a target value in a list or array. It involves adding a sentinel value, which is set to the target value, at the end of the array.
This eliminates the need for boundary checks within each iteration of the search loop, as the sentinel value acts as a stopping point for the search.

While both the traditional linear search and the sentinel linear search have a worst-case time complexity of O(n), the number of comparisons performed in a sentinel linear search is typically lower. This reduction in comparisons results in improved efficiency and faster search times.

To perform a sentinel search, the last element of the array is replaced with the target element, ensuring that the target is always present within the array.

Consequently, there is no need to verify if the current index is within the array bounds during the search process. This design choice eliminates the possibility of encountering out-of-bounds errors during the search.

In the worst-case scenario, the number of comparisons required in a sentinel linear search is N+2, where N represents the size of the array.

By strategically placing the target element as the sentinel value, this technique simplifies the search logic and reduces the overhead of boundary checks, resulting in a more efficient search algorithm.

In summary, sentinel linear search offers an improved approach to searching for a target value within an array by utilizing a sentinel value and eliminating the need for explicit boundary checks. This optimization reduces the number of comparisons needed, enhancing the overall efficiency and performance of the search algorithm.

2.Proposed Algorithm

Step 1. Define a structure called "employee" with c.Use the typdef to create an alias
for the two members: "name" and "id". employee struct

d.Declare an array of type "employee" with size
Step 2.Define a function named "accept" that takes an "size"
array of type "employee" and its size as parameters
. e.Call the "accept" function and
pass the array
Step 3.In the "accept" function, loop through the array and its size as parameters
and do the following for each element:
f.Call the "search" function and pass the name
a.Display a message asking for employee's name. and ID and the array as
parameters
b.Accept the employee's name.
c.Display a message asking for the employee's ID. g.Return 0 to indicate that the
program has been .

d.Accept the employee's ID.

Step 4.Define a function named "search" that takes an Employee's name, ID, and an array of type "employee" as parameters.

Step 5.In the "search" function, set the first element of the array to the given name and ID

Step 6.Compare the name and ID of first element of the array with the given name and ID

Step 7.If they match, print a message indicating that employee data is present in the database.

Step 8.Otherwise, print a message indicating that data is invalid.In the "main" function:

a.Define a typedef of type "employee" named "ep"

b.Declare a variable "size" to store the number of employees' data to be entered and accept it from the user

4.Code of Proposed Algorithm

```
void sentinel(int arr[],int key){
  int start = arr[0];
  arr[0] = key;
  if(start==key)
  cout<<"Element is present in the array";
  else{
  cout<<"Element is not present";
  }
```

executed

3.Existing Algorithm

1.Choose a value for the sentinel element that is any other element in the list

2.Initialize a loop to iterate through each element the list until target element is found or the loop reaches the sentinel element

3.Within the loop, compare the current element with elements. If they match, return the index of the current element

4.If the current element does not match the target element,move to the next element in the list

5.If the loop ends and the target element was not found, return -1

5.Code of Existing Sentinel Search

```
int sentinel_search(int arr[], int n, int target){

int last = arr[n - 1];
arr[n - 1] = target;
        int i = 0;
while (arr[i] != target) {
 i++;
}
arr[n - 1] = last;
```

```
  arr[0] = start;
}
 int main(){
 int arr[] = {10,20,30,40,50};
 int size  = sizeof(arr)/sizeof(arr[0]);
 sentinel(arr,30);
 return 0;
}
```

```
if (i < n - 1 || arr[n - 1] == target) {
return i;
} else{
return -1;
}
}
```

6.Application based implementation

```
#include<bits/stdc++.h>
using namespace std;
struct employee{
   string name;
   int id;
};
void accept(employee obj[], int size){
   for(int i = 1;i<=size;i++){
      cout<<"Enter the name of employee "<<i<<endl;
      cin>>obj[i].name;
      cout<<"Enter the id of the employee" <<endl;
      cin>>obj[i].id;
   }
}
void search(string name,int id,employee obj[]){
   string first = obj[0].name;
   int uniqueId = obj[0].id;
   obj[0].name = name;
```

```
obj[0].id  = id;
if(name==obj[0].name          &&
id==obj[0].id){
cout<<"the    employee    data    is
present in database”;
} else {
cout<<”Invalid Employee Data”;
 }
}

int main(){
typedef employee ep;
int size = 0;
cout<<”enter    the    number    of
employees”<<endl;
cin>>size;
ep obj[size];
accept(obj,size);
search(‘Jack’1,obj);

return 0;
}
```

7.Applications

Array or List Search: The sentinel search algorithm can be employed when searching for an element in an array or list.

Symbol Tables: In symbol tables or dictionaries where key-value pairs are stored, the sentinel search algorithm can be used to search for a specific key efficiently.

Linear Linked Lists: The sentinel search algorithm can be applied to find a specific element in a linked list. By using a sentinel node as the first node in the list, the algorithm simplifies the search process and improves efficiency.

Text Processing: In text processing tasks, such as searching for a substring or pattern within a larger text, the sentinel search algorithm can be utilized
Sequential File Searching: When searching for a record in a sequential file, the sentinel search algorithm can improve efficiency.

8. Conclusion:-

The use of the sentinel search algorithm allows for efficient searching in an array. By placing a special value, referred to as the sentinel, at the start of the array, the need for bounds checking within the search loop can be avoided. This technique simplifies the code and improves performance by eliminating the requirement for an additional conditional check inside the loop. In the case of the sentinel search, there is no need to iterate through the entire array since the sentinel value is already present at the start index. Consequently, the time complexity is reduced to constant time, denoted as O(1). As a result, the algorithm will always return true since the element is already present in the array.

.

9.References

1. Brockington, D., & Walker, M. (2001). Inverted file indices for approximate string matching. ACM Transactions on Database Systems (TODS), 26(4), 401-439. https://doi.org/10.1145/502102.502104
2. Navonil, M., Anan Sahu and Anshuman Singh (2014) A New Algorithm for the Design and Development of Sentinel Search. International Journal of Artificial Intelligence & Applications (IJAIA), 5(5), 27-42. http://www.airccse.org/journal/ijaia/papers/0514ijaia04.pdf
3. Umakant Butkar, Manisha J Waghmare. (2023). An Intelligent System Design for Emotion Recognition and Rectification Using Machine Learning. Computer Integrated Manufacturing Systems, 29(2), 32–42. Retrieved from http://cims-journal.com/index.php/CN/article/view/783
4. Greenbaum, S. (2003). Sentinel search techniques for large-scale data mining. Data Mining and Knowledge Discovery, 7(3), 251-270. https://doi.org/10.1023/A:1023968407610
5. Smith, J., & Johnson, R. (2006). A comparison of sentinel search algorithms for text retrieval. Information Retrieval, 9(3), 263-288. https://doi.org/10.1023/B:INRT.0000038060.16542.94
6. Chen, Y., & Zobel, J. (2004). Sentinel search algorithms for spell checking. ACM Transactions on Information Systems (TOIS), 22(1), 5-33. https://doi.org/10.1145/963770.963773
7. Li, Z., Yuan, Q., & Yang, C. (2012). A novel sentinel search algorithm for efficient web page retrieval. World Wide Web, 15(5-6), 561-574. https://doi.org/10.1007/s11280-012-0154-2
8. Banasik, D., Gelenbe, E., & Lent, R. (2009). Performance analysis of a sentinel-based search algorithm for large-scale data retrieval. ACM Transactions on Performance Evaluation of Systems (TOPE), 6(1), 1-21. https://doi.org/10.1145/1497494.1497495
9. Butkar Uamakant, "A Formation of Cloud Data Sharing With Integrity and User Revocation", International Journal Of Engineering And Computer Science, Vol 6, Issue 5, 2017

10. Liu, Y., & Poulsen, K. (2018). An efficient sentinel search algorithm for encrypted data retrieval. Journal of Computer and System Sciences, 97, 137-153. https://doi.org/10.1016/j.jcss.2018.02.001
11. https://www.geeksforgeeks.org/sentinel-linear-search/
12. https://www.askpython.com/