Design and Analysis of Algorithms Assignment

Department of Information Technology,

Indian Institute of Information Technology, Allahabad 211015, India

Sangam(IIT2019094), Sanjana(IIT2019095) , Riya Goyal(IIT2019096)

Abstract:Given a large binary string, we need to find the length of substring which is having the maximum difference of number of 0s and number of 1s. In this paper, we have solved it using dynamic programming (or kadane algorithm which can be viewed as a simple example of dynamic programming). Using kadane algorithm, the algorithm achieves its worst time complexity in O(n), which reduces the time to a much significant extend. Thus our kadane algorithm approach speeds up the mechanism of finding the length of substring which is having the maximum difference of number of 0s and number of 1s.

Index Terms: Strings, Kadane algorithm, Implementation

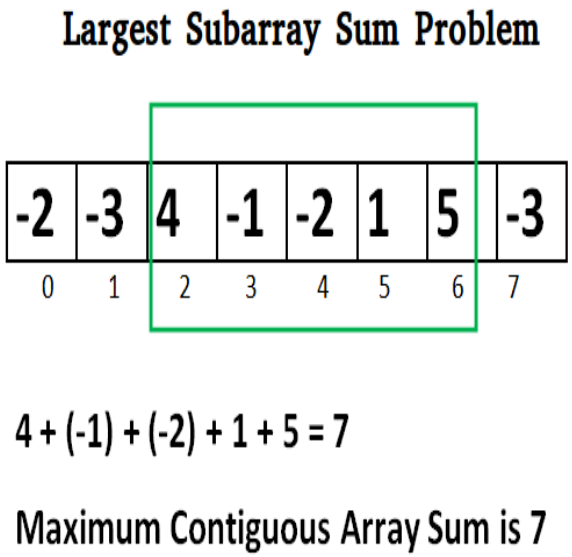

Figure 1: Kadane-Algorithm.png

INTRODUCTION

We have been given a binary string, we have to find the length of substring which is having the maximum difference of number of 0s and number of 1s. In this paper we have used the concept of Kadane algorithm, which has its prime importance, as it reduces the time complexity to a much significant extent.Because of the way this algorithm uses optimal substructures (the maximum subarray ending at each position is calculated in a simple way from a related but smaller and overlapping subproblem: the maximum subarray ending at the previous position) this algorithm can be viewed as a simple example of dynamic programming. Our paper is based on dynamic programming and Kadane algorithm is a part of it, so we will talk more about Kadane algorithm.

Kadane Algorithm The simple idea of Kadane's algorithm is to look for all positive contiguous segments of the array (current-sum is used for this). And keep track of maximum sum contiguous segment among all positive segments (max-Sum is used for this). Each time we get a positive-sum compare it with max-Sum and update max-Sum if the positive-sum is greater than max-Sum

Advantages of Kadane Algorithm The Kadane Algorithm generally has following mentioned advantages over Brute Force Algorithms.

Time Efficiency .
This approach generally reduces the running time of the algorithm because the running time of algorithm based on Brute Force is in general high order in nature say quadratic. But when we are using the approach based on Kadane , this running time generally decreases and becomes linear in nature.
Space Efficiency: Space complexity is way less here, we only use O(1) space. Kadane Algorithm doesn't use much space, it just keeps track of maximum sum contiguous segment among all positive segments .

This report further contains:

- Algorithm Designs
- Algorithm Analysis
- Experimental Study and Profiling
- Conclusion

- References
- Appendix

ALGORITHM DESIGN

Algorithmic Steps:

1. Input the string s.
2. Find the length of the string and store it in a variable named n.
3. Create a function length which will find the maximum difference between the number of zeroes and number of ones in the string.
4. Create two variables, lets call the first one current-sum and the other one max-sum.
5. Iterate the loop from 0 to the length of the string.
6. If the current character in string is '0' we decrease current-sum and if the current character is '1' we increase current-sum and after each iteration we check if the current-sum is less than 0 which means that there is no '1's present till that character we initialize current-sum=0 and then we check if current-sum is greater than max-sum and if it is, then we change the value of max-sum to current sum.
7. Finally, we check if max-sum is 0 which means there is no '1's, then we have to make max-sum = -1 and we return the max-sum to the main function.

```
int:
Function length(s,n)
    int current_sum = 0
        int max_sum = 0


        for (int i = 0; i < n; i++) {


                if(s[i]=='0')
                  current_sum ++
                else
                   current_sum --

                if (current_sum < 0)
                current_sum = 0


        max_sum = max(current_sum, max_sum)
        }
    if(max_sum==0)
    max_sum = -1
    return max_sum
```

```
Int:
Function main()
    int n
    Input n
    string s
    Input s

    //Calling the function
        int ans=Length(s, n)
        printing the maximum difference
        between no of zeroes and no of ones
        in the string
```

ALGORITHM ANALYSIS

APRIORI ANALYSIS: Let T(n) and S(n) is the time and space respectively with input parameters defined above.

TIME COMPLEXITY DERIVATION: Let Time complexity of above algorithm be T(n). Let us assume that we take O(n) times as we are iterating the size of the string given and we assume that its length is n . so T(n) = O(n) .So T(n) can be expressed as follows:

TIME COMPLEXITY DERIVATION:

T(n) = O(n)
Using above relation, we get for T/2, T/8 etc as:
T(n/2) = T(n/2)
T(n/4) = T(n/4)
T(n/8) = T(n/8) and so on.....
.
.
.
.
Thus on combining we get the overall time complexity as:T(n) = O(n)

DIFFERENT CASES: Now let us consider our algorithm in different scenarios.

BEST CASE/AVERAGE CASE/WORST CASE: The time and space complexity of our approach approach in all the three cases is same , i.e, Time Complexity is O(n), as we are always iterating a loop of the size of string. Space complexity is the O(1) because we need are using constant space in each case.

PROFILING

So, after the above analysis, let us have the glimpse of space and time graph and then comparison between both the approaches as a follow-up.

TIME ANALYSIS: Following is the graph representing the time complexity of the algorithm.

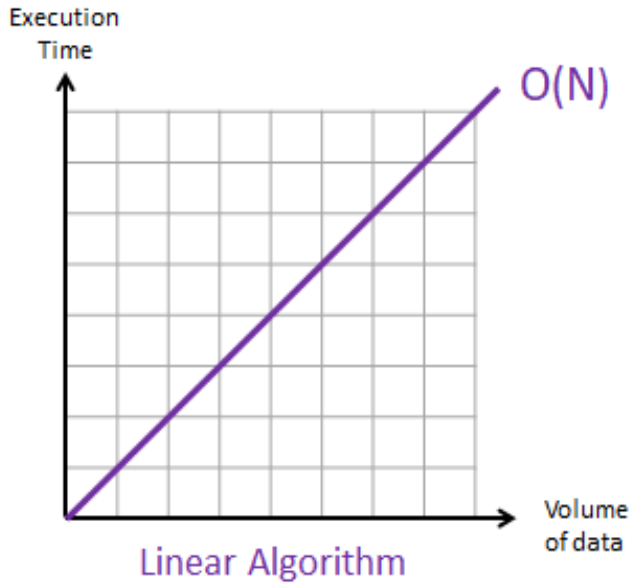

Figure 2: Time Complexity Graph

By the experimental analysis, we found that the more is the value of n, the more time it takes.

SPACE ANALYSIS: Following is the graph representing the space complexity of the algorithm.

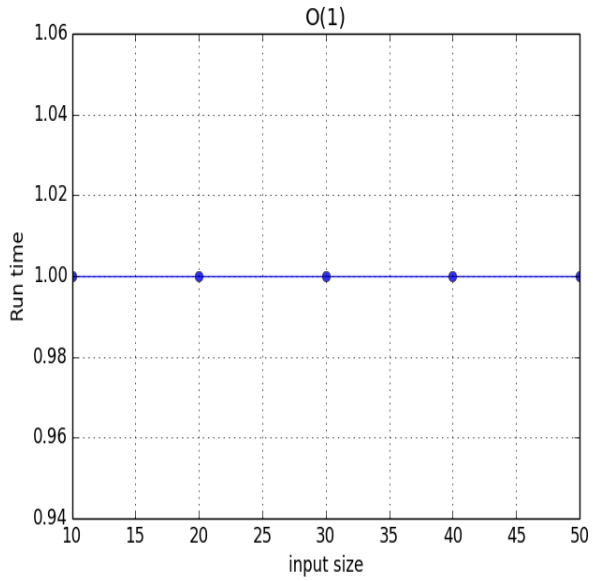

Figure 3: Space Complexity Graph

By the experimental analysis, we found that in any case, space taken is constant.

APPLICATIONS

Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems. Some of the applications of Dynamic Programming are:

1. **Matrix Chain Multiplication:**The order of matrices are given and we need to find the minimum no. of operations we need to do and the order in which we multiply the matrices
2. **Longest Common Subsequence:** It is an algorithm which helps to find the longest common subsequence present in both strings.
3. **Shortest Path Problem:** This algorithm helps to find the shortest path between two points in a 2D-plane using optimizaton.

CONCLUSION

So, with the above mentioned algorithms and their profiling, we come to the conclusion that this problem of finding the length of substring which is having maximum difference of number of 0s and number of 1s is achieving its best time complexity of O(n) and space complexity of O(1).

We have used kadane's algorithm which proved to be the most efficient algorithm here.

ACKNOWLEDGMENT

We are very much grateful to our Course instructor Dr Mohammed Javed and our mentor, Bulla Rajesh, who have provided the great opportunity to do this wonderful work on the subject of Data Structure and Algorithm Analysis specifically on Dynamic Programming.

REFERENCES

1. Introduction to Dynamic Programming: https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/
2. Introduction to Algorithms by Cormen,Charles, Rivest and Stein. https://web.ist.utl.pt/ fabio.ferreira/material/asa

APPENDIX

To run the code, follow the following procedure:

1. Download the code(or project zip file) from the github repository.
2. Extract the zip file downloaded above.
3. Open the code with any IDE like Sublime Text, VS Code, Atom or some online compilers like GDB.
4. If required, save the code with your own desirable name and extension is .cpp
5. Run the code following the proper running commands(vary from IDE to IDE)
   (a) **For VS Code:** Press Function+F6 key and provide the input on the terminal.
   (b) **For Sublime Text:** Click on the Run button and provide the input.

Code for Implementation is:

```
#include <bits/stdc++.h>
using namespace std;

int Length(string s, int n)
{
        int current_sum = 0;
        int max_sum = 0;


        for (int i = 0; i < n; i++) {


                if(s[i]=='0')
                  current_sum ++;
                else
                   current_sum --;

                if (current_sum < 0)
                        current_sum = 0;


                max_sum = max(current_sum, max_sum);
        }
    if(max_sum==0)
    max_sum=-1;
    return max_sum;

}

int main()
{
        string s;
        cin>>s;

        int n;
        n=s.size();

        int ans=Length(s, n);
        cout << ans << endl;
        return 0;
}
```