Finding k elements from a given array of positions such that the minimum distance between any two consecutive points among them is maximized.

Naveen Kumar - IIT2019066,
Slok Aks - IIT2019067,
Simhachalam Anirudh - IIT2019068

Date: 11-03-2021

Abstract

In this paper, we have devised an algorithm to find k elements from the array such that the minimum distance between any two(consecutive points among the k points) is maximized, in a given array representing n positions along a straight line.

1 Problem

Given an array representing n positions along a straight line. Find k (where $k <= n$) elements from the array such that the minimum distance between any two (consecutive points among the k points) is maximized.

2 Keywords

positions, straight, points, consecutive, array.

3 Introduction

Array: An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value.

Straight-line: A straight line is the set of all points between and extending beyond two points.

Sorting: A Sorting Algorithm is used to re-arrange a given array or list elements according to a comparison operator on the elements. In this assignment, we sort the array in ascending order.

4 Algorithm Design

To find the k elements (where k<=n), from the array such that the minimum distance between any two (consecutive points among the k points) is maximized, we implement the following steps in our algorithm :

1. Sort the input array.
2. Set lower = 1, which is the lowest possible distance, and set upper = arr[n-1] - arr[0], which is the maximum possible distance.
3. Now, we apply binary search over lower to upper.
    - Check isFeasible() for the mid until lower < upper, i.e; if it is possible to select 'k' elements using the mid as minimum distance.

- If feasible, then check for higher distance next, i.e; more than mid and upto upper.
- Else, check for lower distance next, i.e; less than mid and from lower.

4. So, we get the required distance.
5. Now, we apply a modified version of isFeasible(), named printElem() to get the final 'k' selected elements with the maximum minimum distance, which was required.

5 Algorithm Analysis

(I) Function to check for feasibility of a chosen distance.

```
bool isFeasible(int mid, int arr[], int n,
    int k)

   int current = arr[0];
   int element_count = 1;

   for (int i = 1; i < n; i++)
      if (arr[i] - current >= mid)
         current = arr[i];
         element_count++;

         if (element_count == k)
            return true;

   return false;
```

(II) Function to find maximised minimum distance.

```
int maximised_min_dist(int arr[], int n,
    int k)

   sort(arr, arr + n);
   int result = -1;

   int lower = 1, upper = arr[n - 1] -
       arr[0];

   while (lower < upper)
      int mid = (lower + upper) / 2;

      if (isFeasible(mid, arr, n, k))
         result = max(result, mid);
         lower = mid + 1;
      else
         upper = mid;

   return result;
```

(III) Function to print the selected 'k' elements for which we get maximised minimum distance.

```
void printElem(int arr[], int maxMinDiff,
     int k, int n)

    int prevPlaced = arr[0];
    cout<<prevPlaced<<" ";

    int numPlaced = 1;

    for(int i = 1; i < n; i++)
        if(arr[i] - prevPlaced >= maxMinDiff)
            prevPlaced = arr[i];
            cout<<prevPlaced<<" ";

            numPlaced++;

            if(numPlaced == k)
                break;
   return;
```

5.1 Time Complexity Analysis

In the above algorithm, the time complexity of isFeasible() function is O(N) as in the worst case, the for loop will have N - 1 iterations.

In the function maximised_min_dist(), the while loop iterates until "lower" becomes equal to "upper". When lower becomes equal to upper, then

$$D/2^m = 1$$

where m is number of iterations and D is maximum possible distance between any two points among N points.
=> D = 2^m
=> m = logD

But in each iteration of that while loop, isFeasible() function is called once, and also there is sort function before while loop in

maximised_min_dist() whose time complexity is O(NlogN). Therefore Overall time complexity is O(NlogN + NlogD)

$$=> O(Nlog(ND))$$

5.2 Space Complexity:

In the above algorithm, only one array of size N is used to store input array and some integer variables are used. So, the Overall space complexity is O(N).

6 Experimental Analysis

From the graph from the Fig.1, it is clear that as the Time complexity is logarithmic in nature, the graph of Running time Vs Iterations is also in logarithmic in nature. Hence this graph proves the time complexity.

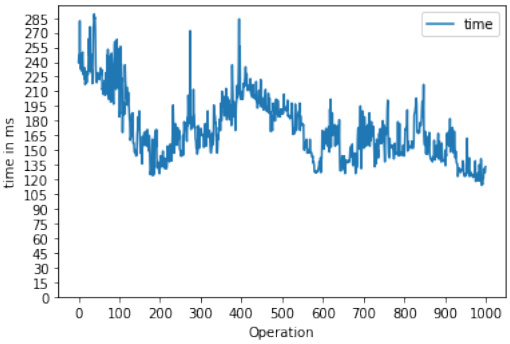

*Fig.*1

6.1 Conclusion

In this paper, we have discussed an algorithm that will print the k elements in the array such that the minimum distance between two consecutive elements (among the k points) is maximized. The time complexity of the algorithm is $O(Nlog(ND)$ and space complexity is $O(N)$, where d = maximum distance between any two points. This algorithm can be considered as the optimal solution to this problem.

7 References

[1] GeeksforGeeks, 'Introduction to Arrays', GeeksforGeeks, 2018. [Online]. [Accessed: 5-Mar-2021]

[2] GeeksforGeeks, 'Place k elements such that minimum distance is maximized)', GeeksforGeeks, 2018. [Online]. [Accessed: 5-Feb-2021]

[3] GeeksforGeeks, 'Sorting Algorithm', GeeksforGeeks, 2018. [Online]. [Accessed: 5-Mar-2021]