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.

Nischay Nagar(IIT2019198), Jaidev Das(IIT2019197) and Priyanshu Singh(IIT2019196)
4th Semester B.Tech, Department Of IT, Indian Institute Of Information Technology Allahabad, Prayagraj, India

Abstract — In this paper, we have approached the problem of finding k elements from the array such that the minimum distance between any two (consecutive points among the k points) is maximized. Although this problem can be solved by checking for every possible subset of a given array, we have used divide and conquer technique to solve said problem because of its higher efficiency. In the paper, first we discuss our algorithm, then we go through the pseudo code on which our source code for solving the problem is based on and then we do a time complexity analysis of our algorithm.

Keywords

- Arrays
- Divide and Conquer algorithm
- Binary Search

Algorithm

1.First sort given array in ascending order
2.Take minDist = -1
3.Take left = 0 and right = maximum distance between elements + 1
4.Find midpoint of(left + right) (let's call it mid)
5.Check if k elements can be placed with minimum distance between any two consecutive elements equal to mid.If this is true , update left to mid + 1 and midDist to mid, store the k elements, and repeat from step 4.Else, update right to mid and repeat from step 4.
6.Keep repeating above steps while left is less than right

7.When above condition fails , return minDist and the stored k elements

Pseudo Code

```
maxMinDist(arr,k):
    sort(arr)
    minDist = -1
    points = []
    left = 0
    right = arr[last] + 1
    while left < right:
        mid = (l + r) / 2
        r = findPoints(arr,k,mid)
        if (r):
           minDist = mid
           points = r
           left = mid + 1
        else:
           right = mid

    print(minDist)
    print(points)



findPoints(arr,k,mid):
      curr = arr[first]
      count = 1
      points = [[arr[first]]
      for el in arr[2nd to last]:
            if el - curr >= mid:
                  curr = el
                  points.append(curr)
                  count = count + 1
                  if count = k:
                        return points

      return False
```

Time Complexity

Here, arr is the array of given elements, n is the elements in arr.
In the while loop we are halving the maximum distance between 2 points in the given array (let's call it d) in each iteration (d = arr[last element] - arr[first element if arr is sorted in ascending order)

So the while loop runs log(d) times.

Inside this loop, there's a call to another function which employs a loop which runs for maximum n-1 times. So this function runs for n-1 times in the worst case.

So T(n) = (n-1)log(d) = O(nlgd)
Where d is the maximum distance between given elements.

Conclusion

The method discussed in the paper is sufficient for concluding that the efficient approach for solving the problem of finding k elements from the array such that the minimum distance between any two (consecutive points among the k points) is maximized is based on biinary search.

References

- https://www.geeksforgeeks.org/binary-search/
- https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/
- https://www.geeksforgeeks.org/divide-and-conquer/