Majority problem

Introduction

The algorithm which we use to detect the element that occurs most times in a given sequence is known as the majority algorithm.

In simple word we can find the majority element of a set S when we find the number of occurrences of an element is ≥ |S|/2.

There are two versions of the algorithm we should consider. One is the algorithm for which we will count the majority element as the element with the number of occurrences > |S|/2. The other algorithm will count the majority element as the element with the number of occurrences ≥ |S|/2.

We can consider a real world example. James stares at the pile of papers in front of him. His class has just finished voting in the election of a class rep. All pupils have written the name of their preferred candidate on a piece of paper, and James has volunteered to count the votes and determine the result of the election. Prior to the election the class agreed that a candidate should become class rep only if more than half the class voted for him or her. If none of the candidates wins the absolute majority of the votes, the election will have to be repeated. James’s task is now to find out whether any candidate has received more than half of all the votes.

How should James approach this task? He doesn’t think about it much and decides to use the most straightforward method, namely putting down the names of all candidates that receive votes on a piece of paper, and keeping a tally of how many votes each of them has received. He picks up each of the ballot papers in turn to see which name has been written on it. If the name is not yet on his sheet, he writes down the name and puts one tally mark next to it. If the name is already on his sheet, he simply adds an extra tally mark next to that name.

But he thinks something different.He thinks that the complexity can be improved.so he come up with a algorithm that will reduce the complexity.He divides the votes into two halves.He first calculated which candidate get more votes in the first half and respectively in the second half.If the candidate is same then he/she becomes winner.If not then he calculates whether the majority candidate in the first half gets the votes greater than total number of votes.If that occurs that candidate will become the winner.If not check the same for the majority candidate for the second half.If that occurs then that candiadate will be the winner.If not then there will be again reelection.This procedure reduces the complexity and overhead to a great extent.

The constraint of the first algorithm is that we can determine only the equality of the elements in the array.We can only determine whether one element is equal to the other element in the array but we can say whether one element is greater or less than the other element.

Algorithm for Majority equal or more than half

Procedure FetchMajorityElement(a[1..n])

Input:Array a objects

Output:Majority Element a

If n=1: return a[1] //array consists of single element

k=└n/2┘ //Divide and Conquer Method(We take the lower bound of k)

elemldiv = GetMajorityElement(a[1...k]) //Function to check the candidate of majority

elemrdiv =GetMajorityElement(a[k+1...n])

ifelemldiv=elemrdiv

returnelemldiv

leftcount=GetCounts(a[1..n],elemIdiv) //Function to check the candidate is greaterthan n/2 elements

rightcount=GetCounts(a[1...n],elemrdiv)

ifleftcount> k+1:

returnelemldiv

else if rightcount> k+1:

returnelemrdiv

else

return NO-MAJORITY-ELEMENT

GetCounts computes the number of times an element (elemldiv or elemrdiv) appears in the given array a[1...n].

procedureGetMajorityElement(a[1...n])

Input: Array a of objects Output:

Goal is to find the Majority element of a

if n = 2:

if a[1] = a[2] return a[1]

else return NO-MAJORITY-ELEMENT

Create a temporary array temp for i = 1 to n:

if a[i] = a[i+1]:

Insert a[i] into temp

i = i+1 return GetMajorityElement(temp)

Time Complexity

The recursion relationship in this algorithm is T(n)=2T(n/2)+0(n)

In Masters theorem we know T(n)=aT(n/b)+O(nc)

Comparing the equation we get c=1,b=2,a=2

c=log ab

Computing we get O(nlogn) by Masters theorem.

Algorithm for Majority more than half.

Let A is the set of elements.

Divide A into two sets each of size A/2

Look at each pair.

Compare each element of one pair with the elements of the other pair

If two elements are different in each pair discard both of them

Else keep one of them.

A/2 elements are left .

The condition must be such that if they must have a majority element if A does.

Time Complexity=O(n)

Reference-Algorithms UnPluggedby Berthold Vocking