Sequential Selection Algorithm

Dr.N.Sairam & Dr.R.Seethalakshmi
School of Computing,
SASTRA Univeristy,
Thanjavur-613401.

Contents

1. Sequential Selection Algorithm

Selection algorithm is used to find the Kth smallest element in the data set. The selection algorithm takes two parameters: the selection element and an array or list. Let the array in which the data is to be selected be unsorted. Then linear search is used for selection. The array is searched from the first, comparing each element with the selection element until the desired selection element or sub list is found. The time complexity of this algorithm is O(kn) where k is the kth smallest or kth largest element to be detected.

Algorithm sequential_select(a[n],k)

```
for i from 1 to k
mIndex = i
mValue = a[i]
for j from i+1 to n
if a[j] <mValue
mIndex = j
mValue = a[j]
exchange a[i] and a[mIndex]
return a[k]
```

This locates the kth smallest element in a much easier way.

Partition based selection algorithm

Selection is most complex if the data is not sorted. First we proceed by sorting the data. There are several sorting techniques like heap sort, quick sort, bubble sort and so on. The complexity of such algorithm ranges from O(nlogn) to O(n^2). We ignore the bubble sort and heap sort since they are not suitable in detecting the data. The quick sort plays a major role in selection.

In quick selection we have a routine called partition which places the key such a way that to the left of the key all elements are less and to the right of the key all elements are greater. Then it is sufficient at every step to migrate to either left half or right half. Thus the time taken for locating the selection element is O(logn). This will be comparable to the sequential selection method explained above when k is very large. Quick selection method is hence advantageous because in selection we follow only one half in contrast to quick sort method. The algorithms are written below.

```
function partition(a[], lt, rt, pIndex)
pValue := a[pIndex]
```

```
exchange a[pIndex] and a[rt]
sIndex := lt
for i from lt to rt-1
if a[i] <pValue
swap a[sIndex] and a[i]
incrementsIndex
swap a[rt] and a[sIndex]
return sIndex
function select(a[], lt, rt, k)
loop
selectpIndex between lt and rt
pNewIndex := partition(a, lt, rt, pIndex)
pDist := pNewIndex - lt + 1
if pDist = k
return a[pNewIndex]
else if k <pDist
rt := pNewIndex - 1
else
k := k - pDist
lt := pNewIndex + 1
```

The quick selection algorithm depends upon the pivot that is chosen. If good pivots are there the algoirthm could run better. If bad pivots are consistently chosen the selection will take the worst case time. The sequential selection algorithm finds application in many computer areas like mining, prediction and so on.