Course based on MIT 6.046J & Clemson CPSC 840
Soliton Technologies, Bangalore

Design and Analysis of Algorithms

Problem Set: 4 — Due Date: Oct 26, 12:00am
Instructor: Sumod K Mohan — Schedule : MWTh 5:30pm-7:00pm

1 Hashing Practise: 30 Pts

1.1 Anagram Detection

Two length-n strings S_1 and S_2 are said to be *anagrams* if we can permute the contents of S_1 to obtain S_2. We have seen previously, how to do this $O(n \log n)$ worst-case time in the comparison model. Using universal hash table, please show how to test whether S_1 and S_2 are anagrams in only $O(n)$ expected time. The "universal hash table" supports the dictionary operations *insert* and *delete* in $O(1)$ amortized time, and *find* in $O(1)$ expected time.

```
AnagramDetection(s1,s2)
Input : Strings s1 and s2
Output : Whether s1 and s2 are anagrams or not
Data Structure used : HashMap H with universal hash function
                    Operations : Insert(H,x)
                                 Delete(H,x)
                                 Search(H,x) (all in O(1) time)

Initialise Hashmap H
if length(s1) != length(s2) // checking length of the strings
       return false
for i <-1 in length(s1)
       if Search(H,s1[i]) == false // if s1[i] is not in hashmap, insert it
              Insert(H,s1[i])
       else                        // if s1[i] is already in hashmap, remove it
              Delete(H,s1[i])
       if Search(H,s2[i]) == false // if s2[i] is not in hashmap, insert it
              Insert(H,s2[i])
       else
              Delete(H,s2[i])      // if s2[i] is already in hashmap, remove it
// check whether hashmap is empty or not
// if it is empty, s1 and s2 are anagrams
if isEmpty(H)
       return true
return false
```

1.2 Grouping Equal Elements

Given an array $A[1...n]$, sow how to permute its contents so that equal elements are grouped together, although not necessarily in sorted order in $O(n)$ worst-case running time using universal hash table.

```
GroupEqualElements(A,n)
Input : Array A and n no.of elements in A
Ouput : Array in which all equal elements are grouped

Initialise Hashmap H
for i<-1 to n
       if (Search(H,s[i]) == False)
              value[i] <- 1
       else
              value[i]++
for i<-1 to n
       value <- getValue[s[i]]
       for value to 0
              b[i] <- value
              value[i]--
return B
```

1.3 Farthest and Closest Duplicate Elements

Given an array $A[1...n]$, find the maximum value of $|i - j|$ such that $A[i] = A[j]$. For slightly more of a challenge, find the minimum value of $|i - j|$ such that $A[i] = A[j]$.

```
FarthestDuplicate(A,n)
Input : Array A and no.of elements in A
Output : Maximum distance between two elements in A

Initialise Hashmap H
// key -> character in an array and value -> (start index,end index)
for i <- 1 to n
       // if A[i] is not in hashmap, insert its start index
       if Search(H,A[i]) == false
              Insert(H,A[i],(i,0))// as its position and end index as 0
       else
              start <- getStartIndex(H,A[i])
              Insert(H,A[i],start,i)     // update end index to current position
max <- 0
for i <- 1 to n
       start <- getStartIndex(H,A[i])
       end <- getEtartIndex(H,A[i])
       diff <- getDifference(start,end)
       if diff > max
              max <- diff
return max
```

```
ClosestDuplicate(A,n)
Input : Array A and no.of elements in A
Output : Mimimum distance between two elements in A

Initialise Hashmap H
// key -> character in an array and value -> (start index,end index)
for i <- 1 to n
       // if A[i] is not in hashmap, insert its start index
       if Search(H,A[i]) == false
              Insert(H,A[i],(i,0))   // as its position and end index as 0
       else
              end <- getEndIndex(H,A[i])
              Insert(H,A[i],end,i)   // update start to end and end to current
position
min <- 0
for i <- 1 to n
       start <- getStartIndex(H,A[i])
       end <- getEtartIndex(H,A[i])
       diff <- getDifference(start,end)
       if diff < min
              min <- diff
return min
```