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.

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.

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 $A[i] = A[j]$.

2 Finding Near Neighbours : 15 Pts

Given an array $A[1...n]$ of integers, we have already seen how to solve the element uniqueness problem (determine if any two array elements are equal) in $O(n)$ expected time using hashing. Consider now a generalization of this problem where we ask whether or not there are any two elements whose values are within some specified distance k of each other. Please assume that now we have available a "universal hash table" that

supports the dictionary operations *insert* and *delete* in $O(1)$ amortized time, and find in $O(1)$ expected time.

1. Please show how to solve this problem also in $O(n)$ expected time and $O(n)$ space.
2. Suppose we are given n points in the plane $(x_1, y_1), ...(x_n, y_n)$ with integer coordinates and asked whether or not there are any two points within distance k of each-other. Please show how to solve this problem in O(n) expected time and O(n) space as well.

3 Prog: Line Fitting : 20 Pts

Given a two dimensional space with points on it, find a line which passes the most number of points. The first line of input is n, the number of points. The following n lines are the (x, y) coordinates of the points. The output should be the slope (double) and y-intercept.

Sample input.

```
6
1 10
2 20
3 30
1 4
2 8
2 5
```

4 Prog: Priority Queue Implementation : 35 Pts

The standard *queue* data structure exhibit a First-In-First-Out (FIFO) behaviour, where those entering the queue first are also the first to leave. There are situations like in a hospital emergency room, where patients waiting for treatment are typically considered in an order according to urgency of their condition. This is known as a *priorityqueue*, where elements enter with associated priorities, and the elements with highest priority is always the first to leave the queue.

To be considered a priority queue, a data structure must at the very least support two basic operations:

- $Insert(e, k)$: Insert a new element e with key k.
- $Remove - Min$: Remove the element with minimum key.

The usual practise is that elements with low key values have high priority. As defined above, the priority queue is an *abstractdatastructure*. Only the basic skeleton is provided and not the functioning. In this problem, we will implement a abstract class that captures the abstract *priorityqueue* properties. Then we will implement two variants of the priority queue: One based on unsorted vectors and other using heaps.

1. Implement interface PQueue Class.

```
class PQueue
{ public:
   PQueue();
   ~PQueue();
   int size();
   bool IsEmpty();
   void Enqueue(int new_element);
   int RemoveMin();
}
```

2. Now implement the actual data structure using unsorted arrays. The simplest enqueue strategy is to add the new elements to the end. When it comes to RemoveMin, it uses a linear time search to find the minimum.

3. Final implementation using Heap. A heap is a binary tree that has these two properties:

   - It is a complete binary tree, except the bottom-most level which is filled from left to right with no gaps.
   - The value of each node is greater than or equal to the value of its children.

   Please refer to your textbook to understand the same in detail. One way to create the binary heap would be to recreate the tree structure using pointers. However, one can exploit the completeness of the tree and create a simple array representation and avoid the complexity and space overhead of pointers. Implement the priority queue operations using the heap represented using arrays. We would not want to keep an upper-bound on the size of your heap, which means it must grow dynamically as per the problem size.

4. Extra Points : If you feel ambitious, you could template the input parameters and the comparison function.

PS: For all the programming problems, please use the appropriate STL containers.