Design and Analysis of Algorithms

Session: Jan-May, 2020, (Course ID: CS2800)

Assignment 2: Decremental Design

Date of Submission : 17 Feb, 2020

1. **Binary GCD algorithm**: Most computers can perform the operations of subtraction, testing the parity (odd or even) of a binary integer, and halving more quickly than computing remainders. This problem investigates the binary gcd algorithm, which avoids the remainder computations used in Euclid's algorithm.

(a) Prove that if a and b are both even, then $\mathsf{gcd}(a, b) = 2 \cdot \mathsf{gcd}(a/2, b/2)$.

(b) Prove that if a is odd and b is even, then $\mathsf{gcd}(a, b) = \mathsf{gcd}(a, b/2)$.

(c) Prove that if a and b are both odd, then $\mathsf{gcd}(a, b) = \mathsf{gcd}((a - b)/2, b)$.

(d) Design an efficient binary gcd algorithm for input integers a and b, where $a \geq b$, that runs in $O(\lg a)$ time. Assume that each subtraction, parity test, and halving takes unit time. [$\lg a$ is the same as $\log_2 a$.]

2. **Analysis of bit operations in Euclid's algorithm**.

(a) Consider the ordinary "paper and pencil" algorithm for long division: dividing a by b, which yields a quotient q and remainder r. Show that this method requires $O((1 + \lg q) \lg b)$ bit operations.

(b) Define $\mu(a, b) = (1 + \lg a)(1 + \lg b)$. Show that the number of bit operations performed by $\textsc{Euclid}(a, b)$ in reducing the problem of computing $\mathsf{gcd}(a, b)$ to that of computing $\mathsf{gcd}(b, a \bmod b)$ is at most $c(\mu(a, b) - \mu(b, a \bmod b))$ for some sufficiently large constant $c > 0$.

(c) Show that $\textsc{Euclid}(a, b)$ requires $O(\mu(a, b))$ bit operations in general and $O(\beta^2)$ bit operations when applied to two β-bit inputs.

(d) If $a > b \geq 0$, show that the call $\textsc{Euclid}(a, b)$ makes at most $1 + \log_\phi b$ recursive calls. Improve this bound to $1 + \log_\phi(b/\mathsf{gcd}(a, b))$.

3. Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values $A[1], A[2], \ldots, A[n]$ is unimodal: For some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted

points would rise until x-value p, where they'd achieve their maximum, and then fall from there on.)

You'd like to find the "peak entry" p without having to read the entire array-in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most $O(\log n)$ entries of A.

4. Consider an n-node complete binary tree T, where $n = 2^d - 1$ for some d. Each node v of T is labelled with a real number x_v. You may assume that the real numbers labelling the nodes are all distinct. A node v of T is a *local minimum* if the label x_v is less than the label x_w for all nodes w that are joined to v by an edge.

You are given such a complete binary tree T, but the labelling is only specified in the following *implicit* way: for each node v, you can determine the value x_v by probing the node v. Show how to find a local minimum of T using only $O(\log n)$ probes to the nodes of T.

5. Suppose now that you're given an $n \times n$ grid graph G. (An $n \times n$ grid graph is just the adjacency graph of an $n \times n$ chessboard. To be completely precise, it is a graph whose node set is the set of all ordered pairs of natural numbers (i, j), where $1 \leq i \leq n$ and $1 \leq j \leq n$; the nodes (i, j) and (k, l) are joined by an edge if and only if $|i - k| + |j - l| = 1$.

We use some of the terminology of the previous question. Again, each node v is labeled by a real number x_v; you may assume that all these labels are distinct. Show how to find a local minimum of G using only $O(n)$ probes to the nodes of G. (Note that G has n^2 nodes.)

6. **Majority Element**.

(a) Let $T[1 \ldots n]$ be an array of n elements. An element x is said to be a *majority* element in T if $|\{i \mid T[i] = x\}| > n/2$. Give an algorithm that can decide whether an array $T[1 \ldots n]$ includes a majority element (it cannot have more than one), and if so find it. Your algorithm must run in linear time.

(b) Rework the above problem with the supplementary constraint that the only comparisons allowed between elements are tests of *equality*. You may therefore not assume that an order relation exists between the elements.

7. Suppose $S = \{1, 2, \ldots, n\}$ and $f : S \to S$. If $R \subset S$, define $f(R) = \{f(x) \mid x \in R\}$. Device an $O(n)$ algorithm for determining the largest $R \subset S$, such that $f(R) = R$.

8. You are a contestant on the hit game show "Beat Your Neighbours!" You are presented with an $m \times n$ grid of boxes, each containing a unique number. It costs \$100 to open a box. Your goal is to find a box whose number is larger than its neighbours in the grid (above, below, left, and right). If you spend less money than any of your opponents, you win a week-long trip for two to Las Vegas and a year's supply of Rice-A-Roni$^{\text{TM}}$, to which you are hopelessly addicted.

(a) Suppose $m = 1$. Describe an algorithm that finds a number that is bigger than any of its neighbours. How many boxes does your algorithm open in the worst case?

(b) Suppose $m = n$. Describe an algorithm that finds a number that is bigger than any of its neighbours. How many boxes does your algorithm open in the worst case?

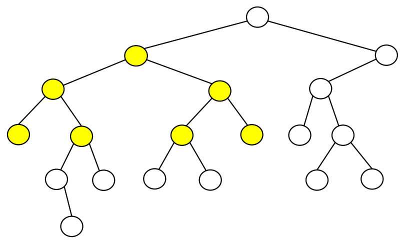
(c) Prove that your solution to part (b) is optimal up to a constant factor.

9. Suppose we are given two sorted arrays $A[1 \ldots n]$ and $B[1 \ldots n]$ and an integer k. Describe an algorithm to find the k-th smallest element in the union of A and B in $O(\log n)$ time. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, your algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicate elements. [Hint: First solve the special case $k = n$.]

(a) Now suppose we are given three sorted arrays $A[1 \ldots n]$, $B[1 \ldots n]$, and $C[1 \ldots n]$, and an integer k. Describe an algorithm to find the k-th smallest element in $A \cup B \cup C$ in $O(\log n)$ time.

(b) Finally, suppose we are given a two dimensional array $A[1 \ldots m][1 \ldots n]$ in which every row $A[i][\ ]$ is sorted, and an integer k.Describe an algorithm to find the k-th smallest element in A as quickly as possible. How does the running time of your algorithm depend on m? [Hint: Use the linear-time Select algorithm as a subroutine.]

10. For this problem, a *subtree* of a binary tree means any connected subgraph. A binary tree is *complete* if every internal node has two children, and every leaf has exactly the same depth. Describe and analyze a recursive algorithm to compute the *largest complete subtree* of a given binary tree. Your algorithm should return the root and the depth of this subtree.



The largest complete subtree of this binary tree has depth 2.

11. You are given a sorted array of numbers where every value except one appears exactly twice; the remaining value appears only once. Design an efficient algorithm for finding which value appears only once.

Here are some example inputs to the problem:

1 1 2 2 3 4 4 5 5 6 6 7 7 8 8
10 10 17 17 18 18 19 19 21 21 23
1 3 3 5 5 7 7 8 8 9 9 10 10

Clearly, this problem can be solved in $O(n)$ time, where n is the number of elements in the array, by just scanning across the elements of the array one at a time and looking for one that isn't paired with the next or previous array element. But can we do better?

12. You are given n balls with some of them coloured red and some black. Your mission is to identify a ball with a majority colour (if it exists). Here, majority means strictly more than other colour. You may pick two balls and ask if these two balls have same colour. What is the minimum number of queries needed for the task?

13. You are given n identical looking gold coins where some are genuine and some are fake. Coins of same type have same weight, i.e., all fakes have weight and all genuine ones have weight b, but you do not know the values or relative values of a or b. You are given a balance which can be used to test if two coins have same weight or not. It is known that number of genuine coins is more than number of false coins. Find a genuine coin in as few weighings as possible.

14. The problem consists of finding the lowest floor of a building from which a box would break when dropping it. The building has n floors, numbered from 1 to n, and we have k boxes. There is only one way to know whether dropping a box from a given floor will break it or not. Go to that floor and throw a box from the window of the building. If the box does not break, it can be collected at the bottom of the building and reused. The goal is to design an algorithm that returns the index of the lowest floor from which dropping a box will break it. The algorithm returns $n+1$ if a box does not break when thrown from the n^{th} floor. The cost of the algorithm, to be kept minimal, is expressed as the number of boxes that are thrown (note that re-use is allowed).

(a) For $k > \log(n)$, design an algorithm with $O(\log(n))$ boxes thrown.

(b) For $k < \log(n)$, design an algorithm with $O(k + \frac{n}{2^{k-1}})$ boxes thrown.

(c) For $k = 2$, design an algorithm with $O(n)$ boxes thrown.

15. Alice controls 2 cops and Bob controls one thief in an $n \times n$ rectangular grid. On a move Alice can move each cop to a neighbour or leave there without moving. Bob can do that for the thief. Show how Alice can nab the thief in $3n$ moves.

16. We have n threads with each one having a bead. We want to line up all the beads vertically by moving them along the threads. The objective is to minimize the total distance moved. Given n numbers denoting the positions of beads, design a linear time algorithm to find the minimum total distance the beads have to be moved in order to line them up.

17. Let F be an array of size $n \geq 1$ whose elements are 0 or 1. A section $[i, \ldots, j]$ of consecutive elements of F, with $1 \leq i \leq j \leq n$, is balanced if it contains as many 0 as 1 elements. The length of a balanced section $[i, \ldots, j]$ is its number of elements, $j - i + 1$. Design an algorithm to find the longest balanced section of F.

18. Professor Diogenes has n supposedly identical integrated-circuit chips that in principle are capable of testing each other. The professor's test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the professor cannot trust the answer of a bad chip. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	Both are good or both are bad
B is good	A is bad	At least one is bad
B is bad	A is good	At least one is bad
B is bad	A is bad	At least one is bad

(a) Show that if more than $n = 2$ chips are bad, the professor cannot necessarily determine which chips are good using any strategy based on this kind of pairwise test. Assume that the bad chips can conspire to fool the professor.

(b) Consider the problem of finding a single good chip from among n chips, assuming that more than $n = 2$ of the chips are good. Show that $\lfloor n/2 \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size.

(c) Show that the good chips can be identified with $\Theta(n)$ pairwise tests, assuming that more than $n = 2$ of the chips are good. Give and solve the recurrence that describes the number of tests.

19. You are given a sorted array of numbers where every value except one appears exactly twice; the remaining value appears only once. Design an efficient algorithm for finding which value appears only once. Clearly, this problem can be solved in $O(n)$ time, where n is the number of elements in the array, by just scanning across the elements of the array one at a time and looking for one that isn't paired with the next or previous array element. But can we do better?

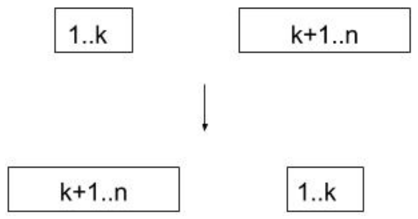
20. There are n identical boxes in which $2n$ balls are equally distributed. The balls are labelled from 1 to $2n$. We don't know which ball is in which box, but do know that each box contains two balls. The objective is to learn the arrangement of balls. For any set of balls $S \subseteq \{1, \ldots, 2n\}$ we can ask a query of the form 'How many boxes contain the balls in S?'. Prove that we can learn the distribution of balls by asking $O(n \log n)$ queries. Note that we cannot tell which ball is in which box. We only need to report the set of n pairs of balls.

21. Given a function $f : S \to S$ (that maps a natural number from a finite set S to another natural number in the same finite set S and an initial value $x_0 \in N$, the sequence of iterated function values: $\{x_0, x_1 = f(x_0), x_2 = f(x_1), \ldots, x_i = f(x_{i-1}), \ldots\}$ must eventually use the same value twice, i.e. $\exists i \neq j$ such that $x_i = x_j$. Once this happens, the sequence must then repeat the cycle of values from x_i to x_{j-1}. Let μ (the start of cycle) be the smallest index i and λ (the cycle length) be the smallest positive integer such that $x_\mu = x_{\mu+\lambda}$. The cycle-finding problem is defined as the problem of finding μ and λ given $f(x)$ and x_0.

22. Given an array $A[1 \cdots n]$ of, say n integers and an integer k, $1 \leq k \leq n-1$, your mission is to swap the segments $A[1 \cdots k]$ and $A[k+1 \cdots n]$, in place without using any auxiliary array.

Eg:- $A[1 \cdots 8] = [10, 20, 30, 40, 50, 60, 70, 80]$.

$k = 3$, output must be $A[1 \cdots 8] = [40, 50, 60, 70, 80, 10, 20, 30]$.



What is the exact complexity of your algorithm? { Number of swap operations }

23. How many times is the print statement executed?

```
for $i_1 = 1$ to $n$ do
    for $i_2 = 1$ to $i_1$ do
        for $i_3 = 1$ to $i_2$ do
            .
                .
                    .
                        for $i_k = 1$ to $i_{k-1}$ do
                            print $i_1, i_2, \cdots, i_k$
```

Notice that each line in the output consists of k integers

$$i_1, i_2, \cdots i_k,$$

where

$$n \geq i_1 \geq i_2 \geq \cdots \geq i_k \geq 1.$$