INDIAN STATISTICAL INSTITUTE

Mid Semestral Examination

M. Tech (CS) - II Year, 2012-2013 (Semester - IV)

Topics in Algorithms and Complexity

Date : 28.02.2013 Maximum Marks : 60 Duration : 2.5 Hours

Note: Answer as much as you can, but the maximum you can score is 60.

(Q1) Given an undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges. Show that there exists a *partition* of V into two disjoint sets V_1 and V_2 such that at least $m/2$ edges cross the partition. [10]

(Ans:) We construct sets V_1 and V_2 by randomly and independently assigning vertices in V to either V_1 or V_2. This surely creates a partition of V as (V_1, V_2). Consider any edge in $e = (x, y) \in E$. Both x and y can belong to either V_1 or V_2; in these two cases e does not cross the partition. In the other two cases, with $x \in V_1$ and $y \in V_2$ and vice-versa, e crosses the partition. So, the probability that e crosses the partition is $1/2$. For $i = 1, \ldots, m$, define X_i such that

$$X_i = \begin{cases} 1 & \text{if edge } e_i \text{ crosses the partition;} \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that $E[X_i] = \frac{1}{2}$. Let $C(V_1, V_2)$ be a random variable that denotes the number of edges going across the partition (V_1, V_2). Therefore,

$$E[C(V_1, V_2)] = E[\sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} E[X_i] = \frac{m}{2}.$$

The expectation of the random variable $C(V_1, V_2)$ is $m/2$. This, in turn, implies that there exists a partition of V so that at least $m/2$ edges cross the partition.

(Q2) Consider the randomized algorithm for the two dimensional linear programming studied in class.

(a) Extend the algorithm so that it works for d dimensions, where $d \geq 2$.

(b) Deduce the time complexity as a function of n and d, where n is the number of linear constraints.

$[5 + 6 = 11]$

(Ans:) See David Mount's lecture notes for the answer.

Let $\mathcal{H} = \{h_1, \ldots, h_n\}$ be a set of n closed halfspaces in dimension d and let $\overrightarrow{c}$ is a vector in d-dimensional space corresponding to the objective function as discussed in the class. The algorithm is as follows:

Algorithm 1: Randomized solution to higher dimensional LP.

LP$(\mathcal{H}, n, d)$

Input: $\mathcal{H} = \{h_1, \ldots, h_n\}$, a set of n closed halfspaces in dimension d and a vector $\overrightarrow{c}$ corresponding to the objective function.

Output: The optimal d-dimensional point.

if *(the dimension* $d == 1$*)* **then**
  solve the 1-dimensional LP;
end

Randomly permute $\mathcal{H}$ to get a sequence $\{h_1, \ldots, h_n\}$ of halfspaces. We assume $h_1, \ldots, h_d$ denote a bounded LP w.r.t. $\overrightarrow{c}$; let v_d denote the intersection point of the d halfspaces;

for *(*$i = (d+1)$ **to** n*)* **do**
  if *(*$v_{i-1} \in h_i$*)* **then**
    $v_i = v_{i-1}$;
  end
  else
    project $\{h_1, \ldots, h_{i-1}\}$ to the hyperplane l_i that defines the hyperspace h_i to get a set $\mathcal{X}$ and call **LP**$(\mathcal{X}, i-1, d-1)$ to solve the $d-1$-dimensional LP.;
    if *(the LP is infeasible)* **then** report that the LP is infeasible;
    else report the optimal point till now as v_i
  end
end

return v_n as the optimal point;

Choose d halfspaces whose feasible region is bounded w.r.t. $\overrightarrow{c}$ to get the vertex v_d. Next, we add halfspaces one by one in random order. There are two cases:

Case I: If the current optimal vertex is feasible (above) w.r.t. the current halfspace, there is no change in the optimal vertex and it can be checked in $O(d)$ time.

Case II: Else, let l_i denote the hyperplane that defines the hyperspace h_i. We project all the halfspaces on l_i, and then solve the resulting $d-1$ dimensional LP problem with $i-1$ halfspaces recursively.

As to the recurrence, let $T(n, d)$ denote the expected time taken for solving a d-dimensional LP with n constraints given as n halfspaces. Let us consider the case after the i-th halfspace ($i > d$) has been inserted. Case I takes $O(d)$ time and Case II takes $T(i-1, d-1)$ time. As to the probability of Case II happening, it is $\frac{d}{i}$ as if and only if any one of the d hyperspaces that define the optimal vertex, was the last one inserted, the smaller LP would be called. The probability of Case I happening is thus $\frac{i-d}{i}$. Thus, the recurrence is

$$T(n,d) = \begin{cases} \sum_{i=d}^{n} \left(\frac{i-d}{i} O(d) + \frac{d}{i} T(i-1, d-1) \right) & \text{if } d \geq 2; \\ n & \text{if } d = 1; . \end{cases}$$

(Q3) Consider an instance of SAT with m clauses, where every clause has exactly k literals. Design a Las Vegas randomized algorithm that finds an assignment satisfying at least $m(1 - 2^{-k})$ clauses. Analyze the expected running time of the algorithm. $[10 + 5 = 15]$

(Ans:) Let X be the random variable that denotes the number of satisfied clauses. So, $X \leq m$.

The probability that a clause is satisfied is $(1 - 2^{-k})$. Let $c = (1 - 2^{-k})$. So, $E[X] = \sum_{i=1}^{m} c = mc$.

To use the *waiting for success time* bound, we need to find the probability $\Pr[X \geq mc]$. Let us denote this probability as p. So, in the expected case, we have to make at most $1/p$ assignments before we get an assignment satisfying at least $m(1-2^{-k})$ clauses.

We know focus on X taking values less than and greater than mc. More precisely,

$$\begin{aligned} E[X] &= \sum_{i=1}^{m} i \cdot \Pr[X=i] \\ &= \sum_{i \leq mc-1} i \cdot \Pr[X=i] + \sum_{i \geq mc} i \cdot \Pr[X=i] \\ &\leq (1-p)(mc-1) + mp \\ mc &\leq (1-p)(mc-1) + mp \\ p &\geq \frac{1}{1+m-mc} \\ p &\geq \frac{1}{1+m \cdot 2^{-k}} \end{aligned}$$

So, we have to make at most $1+m \cdot 2^{-k}$ assignments and each assignment would take $O(km)$ time to evaluate for satisfiability. Thus, overall time taken in the expected case is polynomial in m and k.

(Q4) Consider the following randomized version of quick sort.

Algorithm 2: Randomized quick sort algorithm.

Input: An array $A[1 \ldots n]$ of n unique numbers.

Output: The elements of A sorted in increasing order.

1 Choose an element a uniformly at random from A;
2 Compare each element of A with a and determine the set A_1 of all numbers less than a and the set A_2 of all numbers greater than a ;
3 Recursively sort A_1 and A_2;
4 Output the sorted version of A_1, followed by a, followed by A_2;

Analyze the expected number of comparisons the above algorithm does. [12]

(Ans:) You can see Randomized Algorithms by Motwani and Raghavan for this analysis.

Let the sorted order of the array $A = [a_1, \ldots, a_n]$ be the sequence $B = [b_1, \ldots, b_n]$, where $b_1 < \cdots < b_n$. Let X_{ij} be a 0-1 random variable such that

$$X_{ij} = \begin{cases} 1 & \text{if } b_i \text{ and } b_j \text{ are compared;} \\ 0 & \text{otherwise.} \end{cases}$$

So, the total number of comparisons is $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$. The expected number of comparisons, in which we are interested, is $E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij}$, as X_{ij} is a 0-1 random variable. We now need to find out what is p_{ij}?

View the recursive sorting process as a binary tree, where a is at the root and A_1 and A_2 form the left and right subtrees. Note that, henceforth, no element in A_1 would be compared to an element in A_2. So, for any b_i and b_j to be compared, they must have a parent-child relationship in this tree. Note that, any two numbers can be compared at most once (pretty obvious!). To find p_{ij}, look into the set $B_{ij} = \{b_i, \ldots, b_j\}$. b_i and b_j are compared if and only if either b_i or b_j is selected as the first pivot from the set B_{ij}. This is because b_i and b_j would still be in the

same sub-list and would be compared. If any b_k, $i < k < j$, is selected as the first pivot, then b_i and b_j will go into different sub-lists and would not be compared. This observation coupled with the fact that pivots are chosen uniformly at random (meaning that each element is equally likely to be picked) leads to finding $p_{ij} = \frac{2}{j-i+1}$.

So, the expected number of comparisons is

$$\begin{aligned} E\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} p_{ij} \\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1} \\ &= 2\sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1} \frac{1}{k} \\ &\leq 2\sum_{i=1}^{n}\sum_{k=1}^{n} \frac{1}{k} \\ &\leq 2nH_n \\ &\leq 2n(\ln n + \theta(1)) \end{aligned}$$

(Q5) Let $G = (V, \mathcal{E})$ be a graph where we assign any of the three colors $\{\mathsf{R}, \mathsf{B}, \mathsf{G}\}$ to vertices in G. We say an edge (u, v) is *conflict free* if the vertices u and v are assigned different colors. Consider a coloring that maximizes the number of *conflict free* edges and let this number be c^*. Notice that this is an optimization version of the 3-COLORING problem's decision version which is NP-hard.

Consider the following randomized polynomial time algorithm to solve the above problem approximately. We pick any one of the three colors uniformly at random and color a vertex. We do this for all vertices in G.

Using the above algorithm, what will be the expected number of edges that would be *conflict free*? Explain your result. [10]

(Ans:) Focus on an edge $(u, v) = e \in \mathcal{E}$, $u, v \in V$. The vertices u and v are colored uniformly at random using any one of the three colors $\{\mathsf{R}, \mathsf{B}, \mathsf{G}\}$. Out of the $3^2 = 9$ possible assignments to u and v, there are three that leads to a conflict; the rest assignments are conflict free. So, the probability of an edge to have a conflict free coloring is $\frac{6}{9} = \frac{2}{3}$. Now, let $X_i = 1$ if edge $e_i \in \mathcal{E}$ has a conflict free coloring; 0, otherwise. Let $X = \sum_{i=1}^{|\mathcal{E}|} X_i$. Now, the expected number of edges that are conflict free is $E[X] = \sum_{i=1}^{|\mathcal{E}|} E[X_i] = \frac{2}{3}|\mathcal{E}| \geq \frac{2}{3}c^*$, as $c^* \leq |\mathcal{E}|$.

(Q6) Suppose we roll a standard fair die 200 times. Let X be the sum of the numbers that appear over the 200 rolls. Use Chebyshev's inequality to bound $\mathbf{Pr}[X \geq 750]$. [5]

(Ans:) Let X_i be the random variable that denotes the value obtained in the i^{th} roll of the die, X_i can take any integral value in the range $[1, 6]$ with equal probability. So, $E[X_i] = \frac{7}{2}$ and $\text{Var}[X_i] = E[X_i^2] - (E[X_i])^2 = \frac{91}{6} - \frac{49}{4} = \frac{35}{12}$. Since $X = \sum_{i=1}^{200} X_i$, $E[X] = 200 \times \frac{7}{2} = 700$ and as X_i's are independent, $\text{Var}[X] = 200 \times \frac{35}{12} = \frac{1750}{3}$. Now using Chebyshev's inequality, we have

$$\begin{aligned} \Pr[X \geq 750] &\leq \Pr[|X - 700| \geq 50] \\ &\leq \frac{\text{Var}[X]}{50^2} \end{aligned}$$

$$= \frac{1750}{3 \times 2500}$$
$$= \frac{7}{30}.$$

(Q7) Suppose that we have an algorithm that takes as input a string of n bits. We know that the expected running time is $O(n^2)$ if the input bits are chosen independently and uniformly at random. What can Markov's inequality tell us about the worst-case running time of this algorithm on inputs of size n? [12]

(Ans:) Markov's inequality tells us that for a random variable X that assumes only non-negative values and $\forall a > 0$,

$$\Pr(X \geq a) \leq \frac{E[X]}{a}.$$

For our case, let X be the random variable that represents the running time of the algorithm. By the question statement, $E[X] = O(n^2)$, i.e. $E[X] \leq an^2$ for some constant a. Let us see what Markov's inequality tells us about the running time being any T. Applying Markov's inequality, we get

$$\Pr(X \geq T) \leq \frac{E[X]}{T} \leq \frac{an^2}{T}.$$

Let us set $T = bn^2$, where $b > a$. Then,

$$\Pr(X \geq bn^2) \leq \frac{an^2}{bn^2} = \frac{a}{b}.$$

If $b \geq ka$, then

$$\Pr(X \geq bn^2) \leq \frac{a}{b} \leq \frac{1}{k}.$$

This tells us that

$$\begin{aligned} \Pr[X \geq bn^2] &\leq \frac{1}{k} \\ 1 - \Pr[X \geq bn^2] &\geq 1 - \frac{1}{k} \\ \Pr[X \leq bn^2] &\geq 1 - \frac{1}{k}. \end{aligned}$$

With a sufficient large value of k, $\Pr[X \leq bn^2]$ is almost equal to 1, i.e. with high probability X, the running time of the algorithm in the worst case is $O(n^2)$.