CSC2420 Fall 2012: Algorithm Design, Analysis and Theory
Lecture 9

Allan Borodin

March 13, 2016

Lecture 9

- Announcements
  1. I have the assignments graded by Lalla.
  2. I have now posted five questions for Assignment 2 and the assignment is now complete. The due date is March 27.
- Todays agenda
  1. Sublinear time and space algorithms.

Sublinear time and sublinear space algorithms

We now consider contexts in which randomization is provably more essential. In particular, we will study sublinear time algorithms and then the (small space) streaming model.

- An algorithm is sublinear time if its running time is $o(n)$, where n is the "length" of the input. As such an algorithm must provide an answer without reading the entire input.
- Thus to achieve non-trivial tasks, we almost always have to use randomness in sublinear time algorithms to sample parts of the inputs and/or to randomly keep "snapshots" of what we have seen.
- The subject of sublinear time and sublinear space algorithms are extensive topics and we will only present a very small selection.
- The general flavour of sublinear time results will be a tradeoff between the accuracy of the solution and the time bound. There is some relation between this topic and distributed local algorithms.
- Sublinear space algorithm (e.g. streaming algorithms) have some relation to online algorithms.
- These topics will take us beyond search and optimization problems.

A deterministic exception: estimating the diameter in a finite metric space

- We first conisder an exception of a "sublinear time" algorithm that does not use randomization. (Comment: "sublinear in a weak sense".)
- Suppose we are given a finite metric space M (with say n points x_i) where the input is given as n^2 distance values $d(x_i, x_j)$. The problem is to compute the diameter D of the metric space, that is, the maximum distance between any two points.
- For this maximum diameter problem, there is a simple $O(n)$ time (and hence sublinear in n^2, the number of distances) algorithm; namely, choose an arbitrary point $x \in M$ and compute $D = \max_j d(x, x_j)$. By the triangle inequality, D is a 2-approximation of the diameter.
- I say sublinear time in a weak sense because in an implicitly represented distance function (such as d dimensional Euclidean space), the points could be explicitly given as inputs and then the input size is n and not n^2.

Sampling the inputs: some examples

- The goal in this area is to minimize execution time while still being able to produce a reasonable answer with sufficiently high probability.
- Recall that by independent trials, we can reduce the probability of error.
- We will consider the following examples:
  1. Finding an element in an sorted (doubly) linked list [Chazelle,Liu,Magen]
  2. Estimating the average degree in a graph [Feige 2006]
  3. Estimating the size of some maximal (and maximum) matching [Nguyen and Onak 2008] in bounded degree graphs.
  4. Examples of property testing, a major topic within the area of sublinear time algorithms. See Dana Ron's DBLP for many results and surveys.
  5. In many cases, the algorithms will be "simple" or "reasonably natural" but the analysis might be quite non-trivial.

Finding an element in a sorted list of distinct elements.

- Suppose we have an array $A[i]$ for $1 \leq i \leq n$ where each $A[i]$ is a triple (x_i, p_i, s_i) where the $\{p_i, s_i\}$ constitute a doubly linked list.
- That is, $p_i = j : argmax\{j | x_j < x_i\}$ if such an x_j exists and similarly $q_i = argmin\j | x_j > x_i\}$.
- We would like to determine if a given value x occurs in a doubly linked list and if so, output the index j such that $x = x_j$.

A $\sqrt{n}$ algorithm for searching in a sorted linked list

Let $R = \{j_i | 1 \leq i \leq \sqrt{n}\}$ be $\sqrt{n}$ randomly chosen indices.
Access these $\{A[j_i]\}$ to determine the predecessor and successor of x amongst these randomly chosen elements of the list. (There may not be both a predecssor and successor.) Then (alternately) do a brute force linked search (or resp. search for $\sqrt{n}$ steps) in both directions of the linked list to determine whether or not x_k exists.

Finding an element in a sorted list (continued)

Claim:

This is a zero sided (resp, one-sided error algorithm) that runs in expected time $O(\sqrt{n})$ (resp. has constant probability of not find x if it exists).

- Using the Yao principle this expected time can be shown to be optimal.
- The same can be done for a singly linked list if the list is “anchored” ; i.e., we have the index of the smallest element in the list.
- Similar results were shown by Chazelle, Liu and Magen for various geometric problems such as determining whether or not two convex polygons (represented by doubly linked lists of the vertices) intersect.
- Note that most sublinear time algorithms are either randomized 1-sided or 2-sided error algorithms and not 0-sided algorithms that always compute a correct answer but whose running time is bounded in expectation.

Estimating average degree in a graph

- Given a graph $G = (V, E)$ with $|V| = n$, we want to estimate the average degree d of the vertices. We can assume that G is connected and hence there are at least $n - 1$ edges.
- We want to construct an algorithm that approximates the average degree within a factor less than $(2 + \epsilon)$ with probability at least $3/4$ in time $O(\frac{\sqrt{n}}{poly(\epsilon)})$. We will assume that we can access the degree d_i of any vertex v_i in one step.
- Again, we note that like a number of results in this area, the algorithm is simple but the analysis requires some care.

The (simplified) Feige algorithm Czumaj and Sohler survey.

Sample $8/\epsilon$ random subsets S_i of V each of size (say) $\frac{\sqrt{n}}{\epsilon^3}$
Compute the average degree a_i of nodes in each S_i.
The output is the minimum of these $\{a_i\}$.

The analysis of the approximation

Since we are sampling subsets to estimate the average degree, we might have estimates that are too low or too high. But it can be shown that with high probability these estimates will not be too bad. More precisely, we need:

1. Lemma 1: $Prob[a_i < \frac{1}{2}(1-\epsilon)\bar{d}] \leq \frac{\epsilon}{64}$
2. Lemma 2: $Prob[a_i > (1+\epsilon)\bar{d}] \leq 1 - \frac{\epsilon}{2}$

The probability bound in Lemma 2 follows from the Markov inequality which is then amplified as usual by the repeated $8/\epsilon$ trials so that the probability that all of the a_i are bigger than $(1+\epsilon)\bar{d}$ is at most $(1-\epsilon/2)^{8/\epsilon} = (1-1/t)^{4t} \leq (1/e)^{4t}$ letting $t = 2/\epsilon$.

The analysis of the average degree (continued)

From Lemma 1, we fall outside the desired bound if any of the repeated trials gives a very small estimate of the average degree but by the union bound this is no worse than the sum of the probabilities for each trial.

It remains to sketch a proof of Lemma 1. Let H be the set of the $\sqrt{\epsilon n}$ highest degree vertices in V and $L = V \setminus H$. Then

- $\sum_{v \in L} d_v \geq (\frac{1}{2} - \epsilon) \sum_{v \in V} d_v$ since there can be at most $\epsilon \cdot n$ edges within H and every edge adjacent to L contribues at least 1 to the sum of degrees of vertices adjacent to L and thereare at least $n-1$ edges. The $\frac{1}{2}$ is becasue we are possibly double counting the contribution of edges within L.
- For a lower bound on the average degree of vertices in a sample set S, the worst case is if all the sampled vertices are in L.

Conclusion of proof sketch for Lemma 1

- Let X_j be the random variable corresonding to the ith sampled vertex in a sampled set S where each such S has size s. By Hoeffding's generalization of the Chernoff bound, we have

$$Prob[(1/s)(\sum_{j}^{s} X_j \leq (1-\epsilon)(1/|L|)[\sum_{v \in L} deg(v)]$$

  is exponentially small; that is, the probability that the average degree in a sampled set is $(1-\epsilon)$ less than the average degree in L is exponentially small.
- But the average degree of a vertex in L is at least $(1/2-\epsilon)$ times the average degree in the graph so that being less than $(1/2-\epsilon)$ the average degree is exponentially small.
- Feige's more detailed analysis shows that a $(2+\epsilon)$ approximation can be obtained using time (i.e., queries) $O(\sqrt{n/d_0}/\epsilon)$ for graphs with average degree at least d_0.

Understanding the input query model

- As we initially noted, sublinear time algorithms almost invariably sample (i.e. query) the input in some way. The nature of these queries will clearly influence what kinds of results can be obtained.
- Feige's [2006] algorithm for estimating the average degree uses only "degree queries"; that is, "what is the degree of a vertex v".
- Feige shows that in this degree query model, any (randomized) algorithm that acheives a $(2 - \epsilon)$ approximation requires $\Omega(n)$ queries.

Understanding the input query model

- As we initially noted, sublinear time algorithms almost invariably sample (i.e. query) the input in some way. The nature of these queries will clearly influence what kinds of results can be obtained.
- Feige's [2006] algorithm for estimating the average degree uses only "degree queries"; that is, "what is the degree of a vertex v".
- Feige shows that in this degree query model, any (randomized) algorithm that acheives a $(2-\epsilon)$ approximation requires $\Omega(n)$ queries.
- In contrast, Goldreich and Ron [2008] consider the same average degree problem in the "neighbour query" model; that is, upon a query (v,j), the query oracle returns the j^{th} neighbour of v or a special symbol indicating that v has degree less than j. A degree query can be simulated by $\log n$ neighbour queries.
- Goldreich and Ron show that in the neighbour query model, that the average degree $\bar{d}$ can be $(1+\epsilon)$ approximated (with one sided error probability 2/3) in time $O(\sqrt{n}/poly(\log n, \frac{1}{\epsilon}))$
- They show that $\Omega(\sqrt{(}n/\epsilon))$ queries is necessary to achieve a $(1+\epsilon)$ approximation. in this neighbourhood model.

Approximating the size of a maximum matching in a bounded degree graph

- We recall that the size of any *maximal* matching is within a factor of 2 of the size of a maximum matching. Let m be smallest possible maximal matching.
- Our goal is to compute with high probability a *maximal* matching in time depending only on the maximum degree D.

Nguyen and Onak Algorithm

Choose a random permutation p of the edges $\{e_j\}$
% Note: this will be done "on the fly" as needed
The permutation determines a maximal matching M as given by the greedy algorithm that adds an edge whenever possible.
Choose $r = O(D/\epsilon^2)$ nodes $\{v_i\}$ at random
Using an "oracle" let X_i be the indicator random variable for whether or not vertex v_i is in the maximal matching.
Output $\tilde{m} = \sum_{i=1...r} X_i$

Performance and time for the maximal matching

Claims

1. $m \leq \tilde{m} \leq m + \epsilon$ n where $m = |M|$.
2. The algorithm runs in time $2^{O(D)}/\epsilon^2$

- This immediately gives an approximation of the *maximum* matching m^* such that $m^* \leq \tilde{m} \leq 2m^* + \epsilon n$
- A more involved algorithm by Nguyen and Onak yields the following result:

Nguyen and Onak maximum matching result

Let $\delta, \epsilon > 0$ and let $k = \lceil 1/\delta \rceil$. There is a randomized one sided algorithm (with probability $2/3$) running in time $\frac{2^{O(D^k)}}{\epsilon^{2^{k+1}}}$ that outputs a maximium matching estimate $\tilde{m}$ such that $m^* \leq \tilde{m} \leq (1+\delta)m^* + \epsilon n$.

Property Testing

- Perhaps the most prevalent and useful aspect of sublinear time algorithms is for the concept of property testing. This is its own area of research with many results.
- Here is the concept: Given an object G (e.g. a function, a graph), test whether or not G has some property P (e.g. G is bipartite) or is in some sense far away from that property.
- The tester determines with sufficiently high probability (say 2/3) if G has the property or is "ϵ-far" from having the property. The tester can answer either way if G does not have the property but is "ϵ-close" to having the property.
- We will usually have a 1-sided error in that we will always answer YES if G has the property.
- We will see what it means to be "ϵ-far" (or close) from a property by some examples. See also question 5 in assignment 2.

Tester for linearity of a function

- Let $f : Z_n -> Z_n$; f is linear if $\forall x, y\ f(x+y) = f(x) + f(y)$.
- Note: this will really be a test for group homomorphism
- f is said to be ϵ-close to linear if its values can be changed in at most a fraction ϵ of the function domain arguments (i.e. at most ϵn elements of Z_n) so as to make it a linear function. Otherwise f is said to be ϵ-far from linear.

The tester

```
**Repeat** $4/\epsilon$ times
Choose $x, y \in Z_n$ at random
   **If** $f(x) + f(y) \neq f(x+y)$
   **then** Output $f$ is not linear
**End Repeat** If all these $4/\epsilon$ tests succeed then Output linear
```

- Clearly if f is linear, the tester says linear.
- For $\epsilon < 2/9$, if f is ϵ-far from being linear then the probability of detecting this is at least $2/3$.

Testing a list for monotonicity

- Given a list $A[i] = x_i, i = 1 \dots n$ of distinct elements, determine if A is a monotone list (i.e. $i < j \Rightarrow A[i] < A[j]$) or is ϵ-far from being monotone in the sense that more than $\epsilon * n$ list values need to be changed in order for A to be monotone.
- The algorithm randomly chooses $2/\epsilon$ random indices i and performs binary search on x_i to determine if x_i in the list. The algorithm reports that the list is monotone if and only if all binary searches succeed.
- Clearly the time bound is $O(\log n/\epsilon)$ and clearly if A is monotone then the tester reports monotone.
- If A is ϵ-far from monotone, then the probability that a random binary search will succeed is at most $(1 - \epsilon)$ and hence the probability of the algorithm failing to detect non-monotonicity is at most $(1 - \epsilon)^{\frac{2}{\epsilon}} \leq \frac{1}{e^2}$

Graph Property testing

- Graph property testing is an area by itself. There are several models for testing graph properties.
- Let $G = (V, E)$ with $n = |V|$ and $m = |E|$.
- Dense model: Graphs represented by adjacency matrix. Say that graph is ϵ-far from having a property P if more than ϵn^2 matrix entries have to be changed so that graph has property P.
- Sparse model, bounded degree model: Graphs represented by vertex adjacency lists. Graph is ϵ-far from property P is at least ϵm edges have to be changed.
- In general there are substantially different results for these two graph models.

The property of being bipartite

- In the dense model, there is a constant time one-sided error tester. The tester is (once again) conceptually what one might expect but the analysis is not at all immediate.

Goldreich, Goldwasser,Ron bipartite tester

Pick a random subset S of vertices of size $r = \Theta(\frac{\log(\frac{1}{\epsilon})}{\epsilon^2})$
Output bipartite iff the induced subgraph is bipartite

- Clearly if G is bipartite then the algorithm will always say that it is bipartite.
- The claim is that if G is ϵ-far from being bipartite then the algorithm will say that it is not bipartite with probability at least $2/3$.
- The algorithm runs in time proportional to the size of the induced subgraph (i.e. the time needed to create the induced subgraph).

Testing bipartiteness in the bounded degree model

- Even for degree 3 graphs, $\Omega(\sqrt{n})$ queries are required to test for being bipartite or ϵ-far from being being bipartite. Goldreich and Ron [1997]
- There is an algorithm that uses $O(\sqrt{n} \cdot poly(\log n/\epsilon))$ queries. The algorithm is based on random walks in a graph and utilizes the fact that a graph is bipartite iff it has no odd length cycles.

Goldreich and Ron [1999] bounded degree algorithm

```
Repeat $O(1/\epsilon)$ times
    Randomly select a vertex $s \in V$
    If algorithm $OddCycle(s)$ returns cylce found then REJECT
End Repeat
If case the algorithm did not already reject, then ACCEPT
```

- *OddCycle* performs $poly(\log n/\epsilon)$ random walks from s each of length $poly(\log n/\epsilon)$. If some vertex v is reached by both an even length and an odd length prefix of a walk then report cycle found; else report odd cycle not found

Sublinear space: A slight detour into complexity theory

- Sublinear space has been an important topic in complexity theory since the start of complexity theory. While not as important as the $P = NP$ or $NP = co - NP$ question, there are two fundamental space questions that remain unresolved:
  1. Is $NSPACE(S) = DSPACE(S)$ for $S \geq \log n$?
  2. Is P contained in $DSPACE(\log n)$ or $\cup_k SPACE(\log^k n)$? Equivalently, is P contained in polylogarthmic parallel time.
- Savitch [1969] showed a non deterministic S space bounded TM can be simulated by a deterministic S^2 space bounded machine (for space constructible bounds S).
- Further in what was considered a very surprising result, Immerman [1987] and independently Szelepcsényi [1987] $NSPACE(S) = co - NSPACE(S)$. (Savitch's result was also considered suprising by some researchers when it was announced.)

USTCON vs STCON

We let *USTCON* (resp. *STCON*) denote the problem of deciding if there is a path from some specified source node s to some specified target node t in an unidrected (resp. directed) graph G.

- As previously mentioned the Aleliunas' et al [1979] random walk result showed that *USTCON* is in $RSPACE(\log n)$ and after a sequence of partial results about *USTCON*, Reingold [2008] was eventually able to show that *USTCON* is in $DSPACE(\log n)$
- It remains open if
  1. *STCON* (and hence $NSPACE(\log n)$) is in $RSPACE(\log n)$ or even $DSPACE(\log n)$.
  2. $STCON \in RSPACE(S)$ or even $DSAPCE(S)$ for any $S = o(\log^2 n)$
  3. $RSPACE(S) = DSPACE(S)$.

The streaming model

- In the data stream model, the input is a sequence A of inputs $a_1, \ldots, a_m$ where say each $a_i \in \{1, 2, \ldots, n\}$; the stream is assumed to be too large to store in memory.
- We usually assume that m is not known and hence one can think of this model as a type of online or dynamic algorithm that is maintaining (say) current statistics.
- The space available $S(m, n)$ is some sublinear function. The input streams by and one can only store information in space S.
- In some papers, space is measured in bits (which is what we will usually do) and sometimes in words, each word being $O(\log n)$ bits.
- It is also desirable that that each input is processed efficiently, say $\log(m + n)$ and perhaps even in time O(1) (assuming we are counting operations on words as $O(1)$).

The streaming model continued

- The initial (and primary) work in streaming algorithms is to approximately compute some function (say a statistic) of the data or identify some particular element(s) of the data stream.
- Lately, the model has been extended to consider "semi-streaming" algorithms for optimization problems. For example, for a graph problem such as matching for a graph $G = (V, E)$, the goal is to obtain a good approximation using space $\tilde{O}(|V|)$ rather than $O(|E|)$.
- Most results concern the space required for a one pass algorithm. But there are other results concerning the tradeoff between the space and number of passes.

An example of a deterministic streaming algorithms

As in sublinear time, it will turn out that almost all of the results in this area are for randomized algorithms. Here is one exception.

The missing element problem

Suppose we are given a stream $A = a_1, \ldots, a_{n-1}$ and we are promised that the stream A is a permutation of $\{1, \ldots, n\} - \{x\}$ for some integer x in $[1, n]$. The goal is to compute the missing x.

- Space n is obvious using a bit vector $c_j = 1$ iff j has occured.
- Instead we know that $\sum_{j \in A} = n(n+1)/2 - x$.
  So if $s = \sum_{i \in A} a_i$, then $x = n(n+1)/2 - s$.
  This uses only $2 \log n$ space and constant time/item.

Generalizing to k missing elements

Now suppose we are promised a stream A of length $n-k$ whose elements consist of a permutation of $n-k$ distinct elements in $\{1, \ldots, n\}$. We want to find the missing k elements.

- Generalizing the one missing element solution, to the case that there are k missing elements we can (for example) maintain the sum of j^{th} powers $(1 \leq j \leq k)$ $s_j = \sum_{i \in A} (a_i)^j = c_j(n) - \sum_{i \notin A} x_i^j$. Here $c_j(n)$ is the closed form expression for $\sum_{i=1}^{n} i^j$. This results in k equations in k unknowns using space $k^2 \log n$ but without an efficient way to compute the solution.
- As far as I know there may not be an efficient small space streaming algorithm for this problem.
- Using randomization, much more efficient methods are known; namely, there is a streaming alg with space and time/item $O(k \log k \log n)$; it can be shown that $\Omega(k \log(n/k))$ space is necessary.

Some well-studied streaming problems

- Computing frequency moments. Let $A = a_1 \dots a_m$ be a data stream with $a_i \in [n] = \{1, 2, \dots n\}$. Let m_i denote the number of occurences of the value i in the stream A. For $k \geq 0$, the k^{th} frequency moment is $F_k = \sum_{i \in [n]} (m_i)^k$. The frequency moments are most often studied for integral k.
  1. $F_1 = m$, the length of the sequence which can be simply computed.
  2. F_0 is the number of distinct elements in the stream
  3. F_2 is a special case of interest called the repeat index (also known as Ginis homogeneity index).
- Finding k-heavy hitters; i.e. those elements appearing at least n/k times in stream A.
- Finding rare or unique elements in A.

What is known about computing F_k?

Given an error bound ϵ and confidence bound δ, the goal in the frequency moment problem is to compute an estimate F'_k such that $Prob[|F_k - F'_k| > \epsilon F_k] \leq \delta$.

- The seminal paper in this regard is by Alon, Matias and Szegedy (AMS) [1999]. AMS establish a number of results:
    1. For $k \geq 3$, there is an $\tilde{O}(m^{1-1/k})$ space algorithm. The $\tilde{O}$ notation hides factors that are polynomial in $\frac{1}{\epsilon}$ and polylogarithmic in $m, n, \frac{1}{\delta}$.
    2. For $k = 0$ and every $c > 2$, there is an $O(\log n)$ space algorithm computing F'_0 such that $Prob[(1/c)F_0 \leq F'_0 \leq cF_0$ does *not* hold$] \leq 2/c$.
    3. For $k = 1$, $\log n$ is obvious to exactly compute the length but an estimate can be obtained with space $O(\log \log n + 1/\epsilon)$
    4. For $k = 2$, they obtain space $\tilde{O}(1) = O(\frac{\log(1/\delta}{\epsilon^2})(\log n + \log m))$
    5. They also show that for all $k > 5$, there is a (space) lower bound of $\Omega(m^{1-5/k})$.

Results following AMS

- A considerable line of research followed this seminal paper. Notably settling conjectures in AMS:
- The following results apply to real as well as integral k.
  1. An $\tilde{\Omega}(m^{1-2/k})$ space lower bound for all $k > 2$ (Bar Yossef et al [2002]).
  2. Indyk and Woodruff [2005] settle the space bound for $k > 2$ with a matching upper bound of $\tilde{O}(m^{1-2/k})$
- The basic idea behind these randomized approximation algorithms is to define a random variable Y whose expected value is close to F_k and variance is sufficiently small such that this r.v. can be calculated under the space constraint.
- We will just sketch the (non optimal) AMS results for F_k for $k > 2$ and the result for F_2.

The AMS F_k algorithm

Let $s_1 = (\frac{8}{\epsilon^2} m^{1-\frac{1}{k}})/\delta^2$ and $s_2 = 2\log\frac{1}{\delta}$.

AMS algorithm for F_k

The output Y of the algorithm is the median of s_2 random variables $Y_1, Y_2, ..., Y_{s_2}$ where Y_i is the mean of s_1 random variables $X_{ij}, 1 \leq j \leq s_1$. All X_{ij} are independent identically distributed random variables. Each $X = X_{ij}$ is calculated in the same way as follows: Choose random $p \in [1, \ldots, m]$, and then see the value of a_p. Maintain $r = |\{q|q \geq p \text{ and } a_q = a_p\}|$. Define $X = m(r^k - (r-1)^k)$.

- Note that in order to calculate X , we only require storing a_p (i.e. $\log n$ bits) and r (i.e. at most $\log m$ bits). Hence the Each $X = X_{ij}$ is calculated in the same way using only $O(\log n + \log n)$ bits.
- For simplicity we assume the input stream length m is known but it can be estimated and updated as the stream unfolds.
- We need to show that $\mathbf{E}[X] = F_k$ and that the variance $Var[X]$ is small enough so as to use the Chebyshev inequality to show that $Prob[|Y_i - F_k| > \epsilon F_k$ is small.

AMS analysis sketch

- Showing $E[X] = F_k$.

$$\frac{m}{m}[(1^k + (2^k - 1^k) + \ldots + (m_1^k - (m_1 - 1)^k))+$$

$$(1^k + (2^k - 1^k) + \ldots + (m_2^k - (m_2 - 1)^k)) + \ldots \ldots +$$
$$(1^k + (2^k - 1^k) + \ldots + (m_n^k - (m_n - 1)^k))]$$

(by telescoping)

$$= \sum_i^n m_i^k$$

$$= F_k$$

AMS analysis continued

- Y is the median of the Y_i. It is a standard probabilistic idea that the median Y of identical r.v.s Y_i (each having constant probability of small deviation from their mean F_k) implies that Y has a high probability of having a small deviation from this mean.
- $E[Y_i] = E[X]$ and $Var[Y_i] \leq Var[X]/s_1 \leq E[X^2]/s_1$.
- The result needed is that $Prob[|Y_i - F_k| > \epsilon F_k] \leq \frac{1}{8}$
- The Y_i values are an average of independent $X = X_{ij}$ variables but they can take on large vales so that instead of Chernoff bounds, AMS use the Chebyshev inequality:

$$Prob[|Y - E[Y]| > \epsilon E[Y]] \leq \frac{Var[Y]}{\epsilon^2 E[Y]}$$

- It remains to show that $E[X^2] \leq kF_1F_{2k-1}$ and that $F_1F_{2k-1} \leq n^{1-1/k}F_k^2$

Sketch of F_2 improvement

- They again take the median of $s_2 = 2\log(\frac{1}{\delta})$ random variables Y_i but now each Y_i will be the sum of only a constant number $s_1 = \frac{16}{\epsilon^2}$ of identically distibuted $X = X_{ij}$.
- The key additional idea is that X will not maintain a count for each particular value separately but rather will count an appropriate sum $Z = \sum_{t=1}^{n} b_t m_t$ and set $X = Z^2$.
- Here is how the vector $< b_1, \ldots, b_n > \in \{-1, 1\}^n$ is randomly chosen.
- Let $V = \{v_1, \ldots, v_h\}$ be a set of $O(n^2)$ vectors over $\{-1, 1\}$ where each vector $v_p =< v_{p,1}, \ldots, v_{p,n} > \in V$ is a 4-wise independent vector of length n.
- Then p is selected uniformly in $\{1, \ldots, h\}$ and $< b_1, \ldots, b_n >$ is set to v_p.