streaming algorithm Written Assignment ♯2

0656124 劉承順

2018/4/23

1. **Algorithm**:
   input: x
   compute A(x) n times. save the results.
   if the results have more than $\frac{1}{2}n$ "Yes", claim x is a "Yes-instance", output "Yes-instance".
   otherwise, claim x is a "No-instance", output "No-instance".

   The time complexity is $O(log\frac{1}{\epsilon}\times$ time complexity of $A(x))$.
   proof:
   Suppose x is a "Yes-instance" w.l.o.g.
   Let $X_i = 1$ if A(x)="Yes" at ith time.($X_i = 0$ if A(x)="No")
   Let $X = \Sigma X_i$.
   Because A have 2/3 chance to be correct, Pr(A(x)="Yes")=$\frac{2}{3}$.
   Then $E(X) = \mu = \frac{2}{3}n$.
   By Chernoff bounds,
   $\Pr[X \le (1-\delta)\mu] \le e^{-\mu\frac{\delta^2}{2}}$.
   Substitute δ with $\frac{1}{4}$, μ with $\frac{2}{3}n$:
   $\Pr[X \le \frac{1}{2}n] \le e^{-\frac{1}{48}n} = \frac{1}{a^n}$, for some constant $a > 1$.
   We need $\frac{1}{a^n} \le \epsilon$, just let $n \ge log\frac{1}{\epsilon}$.
   And we have $\Pr[X \ge \frac{1}{2}n] \ge 1-\epsilon$, that is, the probability that this algorithm is correct is more than $1-\epsilon$.

2. **input**: stream of n elements(real number).
   Algorithm:

   for every two pass{ //until find k-th smallest number
   randomly find $n^{\frac{1}{c}}$ elements as pivots at first pass;

let $c_i(0 \le i \le n^{\frac{1}{c}})$ be the number of elements which is bigger than i-th pivot and smaller than $i+1$-th pivot. obtain all c_i at second pass.

suppose $\Sigma_{j=0}^{i-1} c_j < k \le \Sigma_{j=0}^{i} c_j$, than the k-th smallest number must between i-th pivot and $i+1$-th pivot. so in next iteration, only consider elements between i-th pivot and $i+1$-th pivot.
}

This algorithm runs O(c) pass in best case and average case. $O(n^{1-\frac{1}{c}})$ pass in worst case. Because every c_i cost $O(logn)$ space, the total space complexity is $O(n^{\frac{1}{c}} logn)$.

3. (a)
In case $k = 2$, X_1 and X_2 are pairwise independent *iff* they are mutually independent. In case $k = 3$, define x$\epsilon\{1, 2, 3, 4\}$ with same probability for each element.
Let $X_1 = \begin{cases} 1 & if \quad x\epsilon\{1,2\} \\ 0 & otherwise \end{cases}$
$X_2 = \begin{cases} 1 & if \quad x\epsilon\{1,3\} \\ 0 & otherwise \end{cases}$
$X_3 = \begin{cases} 1 & if \quad x\epsilon\{1,4\} \\ 0 & otherwise \end{cases}$
we have $P(X_1 = 1)P(X_2 = 1) = \frac{1}{4} = P(X_1 = 1 \wedge X_2 = 1)$,
$P(X_1 = 1)P(X_3 = 1) = \frac{1}{4} = P(X_1 = 1 \wedge X_3 = 1)$,
$P(X_2 = 1)P(X_3 = 1) = \frac{1}{4} = P(X_2 = 1 \wedge X_3 = 1)$.
but $P(X_1 = 1)P(X_2 = 1)P(X_3 = 1) = \frac{1}{8} \neq P(X_1 = 1 \wedge X_2 = 1 \wedge X_3 = 1) = \frac{1}{4}$.
So pairwise independence does not imply mutually independence.
In case $k \ge 4$, by induction, if case $k-1$ is proved, add new variable X_k.
Let Pr[$X_k = 0$]=1, X_k is independent with $X_1, ..., X_{k-1}$.
$X_1, ..X_k$ are still pairwise independent, but X_1, X_2, X_3 are not mutually independent.
So case k is also proved.

(b)
cov(X,Y)=E[XY]-E[X]E[Y].
If X,Y are independent,
E[X]E[Y]=[$\Sigma_i p_i X_i$][$\Sigma_j p_j Y_j$]

$=\Sigma_{i,j} p_i p_j X_i Y_j$
$=\Sigma_{i,j} Pr[X = X_i \wedge Y = Y_j] X_i Y_j$
=E[XY].
So, cov(X,Y)=0.
If X_1,...,X_n are pairwise independent,
Var(X=ΣX_i)=ΣVar(X_i)+$\Sigma_{i \neq j}$cov(X_i,X_j)
=ΣVar(X_i).

4. (a)
Let F be one of $MSF(G)$.
Replace $F \cap E_r$ with F_r.(They are both MSF(E_r))
that is, $F' = (F \setminus E_r) \cup F_r$.
F' is $MSF(G)$, because the total weight of F' holds, and F' is still in G. And because both $F \cap E_r$ and F_r connects all connected components of E_r, F' is a MSF.
F' is also $MSF(G')$, so $\mu(G) = \mu(G')$

(b)
Use disjoint-set forest with Union-by-rank.

Input: a sequence of edges $e_i = \{u_i, v_i\}$ and the edge weight $w(e_i)$.
Algorithm:
$F \leftarrow \emptyset$;

for each node x{ //init disjoint-set forest
$p[x] \leftarrow x$, //set parent
$t[x] \leftarrow 0$ //weight
};
for each incoming edge e_i {
if (F $\cup$ e is acyclic){
F $\leftarrow$ F $\cup$ e_i;
Union(u_i, v_i);
If tree of u_i is higher than tree of v_i, $p(root(v_i)) = root(u_i)$, $t[root(v_i)] = w(e_i)$, vice versa.;
}else{
discard e_k whose weight $w(e_k)$ is largest among all edges on the cycle; //to find it, travel from u_i to $root(u_i)$, and v_i to $root(v_i)$,find the minimum t[x]. }
}

output F;

this algorithm runs in $O(mlogn)$ time, because trees of disjoint-set forest are of O(log n) height.

5. (a) I don't know.