1. Study Group

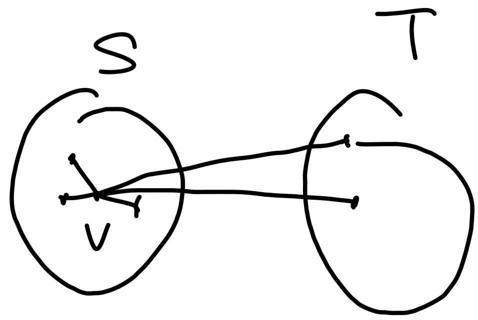
Jialiang Tang (myself), SID: 3039758308

Jessi Wen, SID: 3039751912

2\. (a) $|E|$. After one iteration, the # of edges across the cut has increased and won't exceed $|E|$.

(b) Fact: When moving $v$ from $S$ to $T$, $v$ has more neighbours in $S$ than $T$.

Proof: # of new crossing edges




$-$ # of old crossing edges

$=$ # of neighbors of $v$ in $S$

$-$ # of neighbors of $v$ in $T$

Since we move $v$ from $S$ to $T$,

# of new crossing edges $>$ # of old crossing edges

So # of neighbors of $v$ in $S$ $>$ # of neighbors of $v$ in $T$.

$\square$

By the fact, after Alg terminates, for each $v \in S$ (or $v \in T$), $v$ has more neighbors in $T$ (or $S$)

So # of edges containing $v$ in $S$ (or $T$)

$<$ # of edges containing $v$ crossing the cut, for $\forall v \in S$ (or $T$)

Thus we have # of edges crossing the cut $>$ # of edges inside $S$ and $T$ by making summation and applying inequivalance.

3. (a) Just assign each variable uniformly randomly by 0 or 1!

Denote random variable $C_i = \begin{cases} 1 & \text{if clause } i \text{ is satisfied} \\ 0 & \text{o.w.} \end{cases}$

In 3-SAT, we have 3 distinct literals in each clause, so

$$\Pr[C_i = 1] = 1 - \tfrac{1}{2} \cdot \tfrac{1}{2} \cdot \tfrac{1}{2} = \frac{7}{8}.$$

$$\Rightarrow E[C_i] = \frac{7}{8}$$

$$\text{So } E\left[\sum_i^c C_i\right] = \sum_i^c E[C_i] = \frac{7}{8}c.$$

(b) Smallest Value: $\frac{7}{8}c$.

For each instance of 3-SAT, our randomized alg in (a) ensures a expectation of $\frac{7}{8}c$. So since a random variable must have at least one possible value not less than its expectation, there must exist one assignment for the instance satisfying $\frac{7}{8}c$ clauses. So $\min_I OPT_I \geq \frac{7}{8}$

On the other hand, consider this 3-SAT:

$$I = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3})$$
$$\wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

$\Rightarrow OPT_I = 7 = \frac{7}{8}c$. The lower bound is tight!

So $\min_I OPT_I = \frac{7}{8}$

Description of Algorithm:

4. (a) Initially, set a reservoir $p=0$ and $x=0$.

Each time we receive a number, $p+=1$ and replace $x$ with the new number by probability $\frac{1}{p}$.

Proof of correctness:

At time $t$, for $\forall t$ we have

$$Pr[x = x_t] = 1 \cdot \frac{1}{2} \cdot \frac{2}{3} \cdot \cdots \cdot \frac{t-1}{t} = \frac{1}{t}.$$

Space Analysis:

Only need two reservoirs, taking $O(\log n + \log M)$ space.

(b) $S$ has $2n$ integers in $[n]$, so at least one number has $\geq 2$ appearance. Otherwise there are at most $n$ integers in $S$.

Description of algorithm:

Use $\log n$ copies of alg in (a) (meaning $\log n$ reservoirs which hold a universal randomly selected element. Noted as $q_1, \cdots, q_{\log n}$, initially assigned by $0$. Also, a counter reservoir $p$ is needed.

Each time we see an integer, do a query on $q_1, \cdots, q_{\log n}$

to search for the same element as the integer. If found, output the integer. Otherwise go into next stream. If stream is over, failed.

$x_1$ $\boxed{q_1}$ $\cdot$ —— $\boxed{q_{\log n}}$

$\boxed{P}$

## Proof of correctness:

There are at most $n$ indices $t$ s.t. $x_t$ never occurs after $t$.

$$Pr[\text{failed}] = \prod_{i=1}^{2n} Pr[\text{fail to find same element at time } i]$$

$$\leq \left(\frac{n-1}{n}\right)^{\log n} \cdot \left(\frac{n}{n+1}\right)^{\log n} \cdot \left(\frac{n+1}{n+2}\right)^{\log n} \cdot \cdots \cdot \left(\frac{2n-2}{2n-1}\right)^{\log n}$$

$$= \left(\frac{n-1}{2n-1}\right)^{\log n} < \left(\frac{1}{2}\right)^{\log n} = \frac{1}{n} \quad \text{(By the hint)}$$

So $Pr[\text{succeed}] > 1 - \frac{1}{n}$.

**Space Complexity:** $\log n$ copies of (a) takes $O(\log^2 n)$.

5. (a) Using one bit reservoir $\hat{j}$. Initially $\hat{j}=0$.
Each time we get a number $x_i$, we modify $\hat{j}$:

$\hat{j} = (\hat{j} + x_i) \bmod 2$, If $j=1$, output "odd". Otherwise "even".

<u>Proof</u>: $\hat{j}$ represents the parity of sum of $x_i$ before. By arithmatic rule it's obviously correct. It takes 1 bit.

(b) Setting a N-bit reservoir $j$. Initially $j=0$.
Each time we get $x_i$, we modify $\hat{j}$:

$j = (j + x_i) \bmod N$. If $j=0$, output "devisible". Otherwise output "undevisible".

<u>Proof</u>: $\hat{j}$ represents the module value of $\sum_i^{\text{so far}} x_i$ about $N$. If $j=0$, it means $\sum_i^{\text{so far}} x_i \equiv 0 \pmod{N}$ so $N \mid \sum_i^{\text{so far}} x_i$. Otherwise $N \nmid \sum_i^{\text{so far}} x_i$. It takes $O(\log N)$ bit space (Assume $x_i$ is in $O(N^c)$ for each $i$ so we only need $O(\log N)$ to store $x_i$)

(c) We only need to store the streaming $x_i$ and result (0/1)

Each time we get $x_i$, judge whether $N \mid x_i$. If so, output "Yes". Otherwise after checking all data so far, ~~go to~~ next stream. without outputing "Yes", output "No".

<u>Proof</u>: Since $N$ is prime, $N \mid \prod_{i=1}^{\text{so far}} x_i \Leftrightarrow N \mid x_j$ for some $j$ so far.

So we only need to check each $x_i$, instead of computing the product of all of them. Assume $x_i$ is in $O(N^c)$, we only need $O(\log n)$ bits space. Except that we only need 1 bit to store result ( "undivisable" or "divisable")
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad$ 0 $\quad\quad\quad\quad\quad\quad\quad$ 1

↳ Initially it's 0.

(d) Using $r$ registers $w_1, w_2, \dots, w_r$. Initially $w_i = k_i$.

Each time we get $x_i$, for $i$, if $p_i \mid x_i$, compute $m$ s.t. $p^m \mid x_i$ but $p^{m+1} \nmid x_i$, then modify $w_i$ : $w_i = \max(0, w_i - m)$.

After streams so far, output "divisable" if all $w_i = 0$.

Otherwise output "undivisable".

<u>Proof</u>: It keeps track of order of $p_i$ of $\prod_{i=1}^{\text{so far}} x_i$

and $w_i = 0 \iff \prod_{i=1}^{\text{so far}} x_i$ has factor $p_i^{k_i}$.

It takes $O(r \log(\max_i k_i))$