Advanced Algorithm: Assignment #1

Due on Sept 26, 2017 at 12:00 pm

Professor Yitong Yin

Hao Li DZ1833013

Problem 1

Recall that in class we show by the probabilistic method how to deduce a $\frac{n(n-1)}{2}$ upper bound on the number of distinct min-cuts in any multigraph G with n vertices from the $\frac{n(n-1)}{2}$ lower bound for success probability of Karger's min-cut algorithm.
Also recall that the $FastCut$ algorithm taught in class guarantee to return a min-cut with probability at least $\Omega(1/\log n)$. Does this imply a much tighter $O(log\ n)$ upper bound on the number of distinct min-cuts in any multigraph G with n vertices? Prove your improved upper bound if your answer is "yes", and give a satisfactory explanation if your answer is "no".

Solution
part 1
the Karger's theorem is:

$$\text{For any min-cut } C,\ Pr[C\ is\ returned] \geq \frac{2}{n(n-1)}.$$

Assume G has $\boldsymbol{r}$ min-cut, which is $c_1, c_2, \ldots, c_r$. For each $c_i \in \{c_1, c_2, \ldots, c_r\}$, use the Karger's theorem, we have:

$$Pr[c_i\ is\ returned] \geq \frac{2}{n(n-1)} \tag{1}$$

then we have:

$$Pr[\bigcup_{i=1}^{r} c_i\ is\ returned] = \sum_{i=1}^{r} \frac{2}{n(n-1)} \geq \frac{2\times r}{n(n-1)} \tag{2}$$

And obviously, we know $Pr[\bigcup_{i=1}^{r} c_i\ is\ returned] \leq 1$, so:

$$\frac{2\times r}{n(n-1)} \leq 1 \tag{3}$$

then:

$$r \leq \frac{n(n-1)}{2} \tag{4}$$

that is the proof.

part 2
First, I can give a counter example. For a complete graph with n vertices, due to its symmetry, the number of distinct min-cuts must be upper than n. So it cannot be bound by $\boldsymbol{O}(log\ n)$. So my answer is "no".
Then I wanna to find the reason of the gap:
Assume C is a min-cut, and $Pr[C = Fastcut(G)] \geq \frac{1}{\boldsymbol{O}(1/log\ n)}$. Then we also assume that there are r min-cuts in the graph, which is $c_1, c_2, \ldots, c_r$. then we also have $Pr[\text{c}_i\text{=Fastcut(G)}] \geq \frac{1}{\boldsymbol{O}(1/log\ n)}$. But the operation is not at one times, so we don't have $Pr[\bigcup_{i=1}^{r} c_i = Fastcut(G)] = \sum_{i=1}^{r} Pr[c_i = Fastcut(G)]$, that is where the gap is.

Problem 2

Two rooted trees T_1 and T_2 are said to be *isomorphic* if there exists a bijection ϕ that maps vertices of T_1 to those of T_2 satisfying the following condition: for each internal vertex v of T_1 with children $\{\phi(u_1), \phi(u_2), \dots, \phi(u_k)\}$, no ordering among children assumed.
Give an efficient randomized algorithm with bounded one-sided error (false positive), for testing isomorphism between rooted trees with n vertices. Analyze your algorithm.

Solution
First I wanna to sort a tree and find a unique expression of a tree using invariant 0 and 1. Then using the fingerprint algorithm to check the identity.
we use the AHUSORT algorithm to solve the first part. The algorithm is shown below. The core idea of the algorithm is: give a rooted tree with n vertices. and we wanna to express the tree to a unique expression using invariant 0 and 1; there are two points here, that is unique and invariant. So we need to sort a tree to a unique expression what make two isomorphic trees express the same. In this algorithm, line 9 do this thing. Line 1-8 is a recursive process to name a vertice. If the vertex is a leaf, then name it '10'. If the vertex has children, then concatenate all the children to tmp, then return '1tmp0'. For example, vertex v_0 have 3 children, which is v_1, v_2, v_3, and v_1 is named as '110100', v_2 is named as '1100', v_3 is named as '11010100'. By sort, we transform the order v_1, v_2, v_3 to v_2, v_1, v_3. Then we can concatenate the name of all children to '110011010011010100', then we use tmp to represent it. Then we get the name of v_0. That's the process of the algorithm.
Then I wanna to analysis the complexity of the algorithm. The time complexity of sorted operation is $\boldsymbol{O}(nlog\ n)$. And the traverse is $\boldsymbol{O}(n)$. So the time complexity is $\boldsymbol{O}(\mathrm{n}^2 log\ n)$.

Algorithm 1: Tree isomorphism (AHU) AHUSORT(v)

```
input :
        a rooted tree with n vertices;
output:
        a unique expression of the tree using invariant 0 and 1;
1  if v is a leaf then
2      Name v '10';
3  end
4  else
5      for all child w of v do
6          AHUSORT(w);
7      end
8  end
9  sort the names of the children of v;
10 concatenate the names of all children of v to temp;
11 return 1temp0
```

Then we use Polynomial Identity Testing algorithm introduced in the lesson to check the identity. The problem is given 2 unique expression of two rooted trees, which is called a, b, and $a \in 0, 1^{2n}, b \in 0, 1^{2n}$. And output whether $a \equiv b$.
We define $f = \sum_{i=0}^{2n-1} a_i x^i$,$g = \sum_{i=0}^{2n-1} b_i x^i$. Let $k = \lceil log_2(4n) \rceil$ We choose a prime $p \in [2^k, 2^{k+1}]$, let $f, g \in Z_p[x]$. Then we pick a random $r \in [p]$, check whether $f(r) = g(r)$. The time complexity of the algorithm is $\boldsymbol{O}(log\ n)$, and the one side error is less than $\frac{1}{2}$.

Problem 3

Fix a universe U and two sunset $A, B \subset U$, both with size n. We create both Bloom filters $F_A(F_B)$ for $A(B)$, using the same number of bits m and the same k hash functions.

- **Let $F_C = F_A \bigcap F_B$ be the Bloom filter formed by computing the bitwise AND of F_A and F_B. Argue that F_C may not always be the same as the Bloom filter that are created for $A \bigcap B$.**
- **Bloom filters can be used to estimate set differences. Express the expected number of bits where F_A and F_B differ as a function of m, n, k and $|A \bigcap B|$.**

Solution
part 1
First, I wanna to do some definitions. We define $S = \{i|F_A[i] = 1\}$, and $T = \{i|F_B[i] = 1\}$, $R = \{i|F_{A \bigcap B}[i] = 1\}$.Then we consider this condition:
For $x_i \in A, x_i \notin B$, and $y_i \notin A, y_i \in B$. But in the hash function, $h_j(x_i) = h_j(y_i)$,and $h_j(x_i) \notin R$. Then $F_A[h_j(x_i)] = 1, F_B[h_j(y_i)] = 1$, but $F_{A \bigcap B}[h_j(x_i)] = 0$.
So F_C may not be always the same as the Bloom filter that are created for $A \bigcap B$.

part 2
First, we define a indicate function $I(i)$:

$$I(i) = \begin{cases} 1 & if F_A[i] \neq F_B[i] \\ 0 & o.w. \end{cases}$$

Define the number of bits where F_A differs with F_B is D. $D = \sum_{i=1}^{m} I(i)$.
When $I(i) = 1$ occurs when:

$$\text{Case 0: For all } x \in A \bigcap B,\ h_j(x) \neq i$$

and one of the following case occurs.

$$\text{Case 1: } [\exists x \in A - (A \bigcap B), h_j(x) = i] \cap [x \in B - (A \bigcap B), h_j(x) \neq i];$$
$$\text{Case 2: } [\exists x \in B - (A \bigcap B), h_j(x) = i] \cap [x \in A - (A \bigcap B), h_j(x) \neq i];$$

Note, case 0 represent no elements in $A \cap B$ will hash to i, case 1 represent some elements in A will hash to i, but no elements in B will hash to i, case 2 represents some elements in B will hash to i, but no elements in A will hash to i. Iff $[(case0) \cap (case1)] \cup [(case0) \cap (case2)]$ occurs, I(i)=1.
Then we separately consider the 3 cases. Assume $t = |A \cap B|$.
First for case 0, no elements in $A \cap B$ will hash to i during k independent hashes. So:

$$Pr[case0] = (1 - \frac{1}{m})^{t \times k} \tag{5}$$

For case 1, some elements in A will hash to i, but no elements in B will hash to i. We have:

$$\begin{aligned} Pr[case\ 1] &= Pr[[\exists x \in A - (A \cap B), h_j(x) = i] \cap [x \in B - (A \cap B), h_j(x) \neq i]] \\ &= (1 - (1 - \frac{1}{m})^{(n-t) \times k}) \times (1 - \frac{1}{m})^{(n-t) \times k} \end{aligned}$$

Note, some elements in A will hash to i is exact the exclusively event of no elements in A will hash to i. So $Pr[\exists x \in A - (A \cap B), h_j(x) = i] = 1 - Pr[x \in A - (A \cap B), h_j(x) \neq i]$. And the equation is obvious.

Case 2 is the same as case 1, because only A transform to B, B transform to A.
Thus, we get this:

$$\begin{aligned}
Pr[I(i) = 1] &= Pr[case\ 0 \cap (case \cup case)] \\
&= Pr[case\ 0] \times (Pr[case\ 1] + Pr[case\ 2]) \\
&= Pr[case\ 0] \times 2 \times Pr[case\ 1] \\
&= 2 \times (1 - \frac{1}{m})^{t \times k} \times (1 - (1 - \frac{1}{m})^{(n-t)\times k}) \times (1 - \frac{1}{m})^{(n-t)\times k} \\
&= 2 \times (1 - \frac{1}{m})^{n \times k} \times (1 - (1 - \frac{1}{m})^{(n-t)\times k})
\end{aligned}$$

So:

$$\begin{aligned}
E(D) &= E(\sum_{i=1}^{m} I(i)) \\
&= \sum_{i=1}^{m} E(I(i)) \\
&= m \times Pr[I(i) = 1] \\
&= 2m \times (1 - \frac{1}{m})^{n \times k} \times (1 - (1 - \frac{1}{m})^{(n-t)\times k})
\end{aligned}$$

Among it:
m is the bit number;
n is the size of A or B;
k is number of hash functions;
t is the size of $A \cap B$.