CS526-HW3

Zongqi Cui

October 2024

Problem 1:Solution to Exercise 2-2

Part (a): Proving 2-Universality of $\mathcal{H}$

Definition of 2-Universal Hash Family:

A hash family $\mathcal{H}$ from X to Y is 2-universal if for any distinct $x, x' \in X$,

$$\Pr_{h \in \mathcal{H}}[h(x) = h(x')] = \frac{1}{|Y|}$$

where the probability is over the random choice of h from $\mathcal{H}$.

Proof:

1. **Given:**

- $x, x' \in X$ with $x \neq x'$.
- a, b are chosen uniformly at random from F_{2^n}.

2. **Compute $h_{a,b}(x) - h_{a,b}(x')$:**

$$h_{a,b}(x) - h_{a,b}(x') = (a \cdot x + b) - (a \cdot x' + b) = a \cdot (x - x')$$

3. **Set $x_\Delta = x - x'$:** Since $x \neq x'$, $x_\Delta \neq 0$ in F_{2^n}.

4. **Determine when $h_{a,b}(x) = h_{a,b}(x')$:** This occurs if and only if $a \cdot x_\Delta = 0$ in F_{2^n}. Since $x_\Delta \neq 0$, the equation $a \cdot x_\Delta = 0$ holds if and only if $a = 0$.

5. **Compute the Probability:**

- Total number of elements in F_{2^n} is 2^n.
- Only one value of a (specifically $a = 0$) satisfies $a \cdot x_\Delta = 0$.

- Therefore:
$$\Pr[a \cdot x_\Delta = 0] = \Pr[a = 0] = \frac{1}{2^n}$$

6. **Conclusion:** The probability that $h_{a,b}(x) = h_{a,b}(x')$ is $\frac{1}{2^n}$. Since $Y = X$ and $|Y| = 2^n$, this matches $\frac{1}{|Y|}$. Therefore, $\mathcal{H}$ is a 2-universal hash family.

Summary:
For any distinct $x \neq x'$ in X,
$$\Pr_{a,b}[h_{a,b}(x) = h_{a,b}(x')] = \frac{1}{2^n}$$
which confirms that $\mathcal{H}$ is 2-universal.

Part (b): Number of Bits Required to Represent $h_{a,b}$

Understanding the Representation:

1. **Elements a and b:**

- Both a and b are elements of F_{2^n}.
- Each element of F_{2^n} can be represented by an n-bit string (since $|F_{2^n}| = 2^n$).

2. **Total Bits Required:**

- **For a:** n bits.
- **For b:** n bits.
- **Total:** $n + n = 2n$ bits.

Answer:
To represent a function $h_{a,b}$ from $\mathcal{H}$, you need $2n$ **bits**.

Part (c): Comparison with Another 2-Universal Hash Family $\mathcal{G}$

Assuming $\mathcal{G}$ from Exercise 2-1 is Defined as Follows:

- **Hash Function $g_{A,b}(x) = Ax + b$:**

- A is a $k \times n$ binary matrix.
- b is a binary vector of length k.
- Operations are over GF(2), the finite field with 2 elements.

Number of Bits Required for $\mathcal{G}$:
1. **Bits for Matrix A:**

- A has $k \times n$ entries.
- Each entry is 1 bit (since it's binary).
- Total bits for A: kn.

2. **Bits for Vector b:**

- b has k entries.
- Total bits for b: k.

3. **Total Bits Required:**

$$kn + k = k(n+1) \text{ bits.}$$

Comparison:

- **For $\mathcal{H}$:** $2n$ bits.
- **For $\mathcal{G}$:** $k(n+1)$ bits.

If $k = n$:

$$\text{Bits for } \mathcal{G} : n(n+1).$$

In this case, $\mathcal{H}$ uses significantly fewer bits ($2n$ vs. $n(n+1)$).
Conclusion:

- **Efficiency:** The hash family $\mathcal{H}$ is more space-efficient than $\mathcal{G}$ when n is large.
- **Trade-offs:** While $\mathcal{G}$ may have other advantages (like simplicity over GF(2)), $\mathcal{H}$ offers a compact representation.

Problem 2:Comparison of BJKST and CVM Algorithms (Problem 2)

(a) Space Complexity

BJKST Algorithm:

$$O\left(\log(n) + \frac{1}{\epsilon^2}\left(\log\left(\frac{1}{\epsilon}\right) + \log\log(n)\right)\right)$$

BJKST uses optimized space by storing only hash values instead of all tokens, reducing the total space required.

CVM Algorithm: The CVM algorithm typically uses more space as it lacks the space-optimization features of BJKST. Exact space complexity is generally higher than that of BJKST.

Conclusion: BJKST is more space-efficient than CVM, especially for large data streams and small values of ϵ.

(b) Randomness Used

BJKST Algorithm: Uses fewer random bits due to efficient 2-universal hash functions h and g, which manage randomness more effectively.

CVM Algorithm: Likely requires more randomness to store and process data without the same space-optimization techniques.

Conclusion: BJKST requires less randomness than CVM.

(c) Other Differences

- **Accuracy:** BJKST offers better accuracy due to the "median trick" and optimized sampling.
- **Efficiency:** BJKST is more efficient in terms of processing due to adaptive bucket resizing.

Conclusion: BJKST provides advantages in terms of accuracy, adaptability, and efficiency.

Problem 3: Join Estimate

Definitions and Setup:

- Let $\|f\|^2 = \sum_j f_j^2$ and $\|g\|^2 = \sum_j g_j^2$, which are the squared 2-norms of the vectors f and g, respectively. - Using the Tug-of-War sketch, we can estimate $\|f\|^2$, $\|g\|^2$, and $\|f+g\|^2$.

Approach Idea 1: Using the Combined Stream $\sigma_1 \circ \sigma_2$

1. If the Tug-of-War sketches for f and g use the same hash functions, then adding the corresponding z-counters from the two sketches yields the sketch for the combined frequency vector $f+g$.

2. Using the Tug-of-War sketch for $f+g$, estimate $\|f+g\|^2$:

$$\|f+g\|^2 = \sum_j (f_j+g_j)^2 = \|f\|^2 + \|g\|^2 + 2(f \cdot g).$$

3. Rearrange this formula to isolate $f \cdot g$:

$$f \cdot g = \frac{1}{2}\left(\|f+g\|^2 - \|f\|^2 - \|g\|^2\right).$$

—

Approach Idea 2: Using Separate Counters for σ_1 and σ_2

1. Apply the Tug-of-War sketch to both streams separately, yielding two counter values z_1 for σ_1 and z_2 for σ_2.

2. Directly compute the estimated inner product $f \cdot g$ using the formula:

$$f \cdot g \approx \mathbb{E}[z_1 z_2].$$

3. This works because the Tug-of-War sketch is designed to ensure that the counters z_1 and z_2 are unbiased estimators of the dot product.

—

Error Analysis

1. The error bounds for this estimation are additive. Specifically:

$$\text{Error} = O(\|f\|^2 + \|g\|^2),$$

since the Tug-of-War sketch estimates each norm and the combined norm with a small additive error.

2. Additive error is often sufficient in practice, even if relative error bounds may not be achievable for cases where $f \cdot g$ is small.

—

Final Algorithm

- **Input:** Frequency vectors f and g from streams σ_1 and σ_2.
- **Steps:**
    1. Construct Tug-of-War sketches for f and g.
    2. Estimate $\|f\|^2$, $\|g\|^2$, and $\|f+g\|^2$ using the sketches.
    3. Compute the estimated inner product:

$$f \cdot g \approx \frac{1}{2}\left(\|f+g\|^2 - \|f\|^2 - \|g\|^2\right).$$

- **Output:** Approximate value of $f \cdot g$, which corresponds to the join size.

—

Conclusion

Using the Tug-of-War sketch, we can efficiently estimate the inner product $f \cdot g$ and thus the size of the join between streams σ_1 and σ_2. The estimator is unbiased, and the error is additive, proportional to $\|f\|^2$ and $\|g\|^2$. This approach is suitable for large-scale streaming scenarios with limited memory.

Problem 4 :Solution to QKD Problems (B92 Protocol)

Overview of B92 Protocol:

In the B92 protocol, Alice encodes her random string x as:

$$0 \mapsto |0\rangle, \quad 1 \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle).$$

Bob randomly selects a bit y:

- If $y_i = 0$, Bob measures the i-th qubit in the Hadamard basis $\{|+\rangle, |-\rangle\}$.
- If $y_i = 1$, Bob measures the i-th qubit in the standard basis $\{|0\rangle, |1\rangle\}$.

Based on the results of his measurements, Bob discards ambiguous measurements and communicates with Alice to finalize the shared key.

—

(a) When Are the Two Bits Equal with Certainty?

The two bits (Alice's encoded bit and Bob's measured bit) are guaranteed to be equal when:

- Alice encodes 0 ($|0\rangle$), and Bob measures in the standard basis.
- Alice encodes 1 ($\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$), and Bob measures in the Hadamard basis.

In these cases:

- If Bob measures $|0\rangle$ or $|1\rangle$ in the standard basis, the result will correspond to Alice's original bit x_i.
- Similarly, if Bob measures in the Hadamard basis, the result corresponds to Alice's bit due to the properties of the $|+\rangle$ state.

Conclusion: The two bits are equal with certainty when Bob's measurement matches Alice's encoding basis.

—

(b) When May the Two Bits Be Unequal?

The two bits may differ when:

- Bob measures in a basis different from the one used by Alice to encode the bit.
- For example:
  - If Alice encodes 0 ($|0\rangle$), but Bob measures in the Hadamard basis, the result will not deterministically match x_i.
  - If Alice encodes 1 ($\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$), but Bob measures in the standard basis, the result will not deterministically match x_i.

Conclusion: The two bits may differ when Bob's measurement basis does not align with Alice's encoding.

—

(c) Detecting Eavesdropping

Suppose an eavesdropper (Eve) measures each qubit in either the standard basis or the Hadamard basis and forwards her results to Bob. This introduces the possibility of detection:

- If Eve measures in the same basis as Alice's encoding, her measurement will not disturb the qubit.
- If Eve measures in a different basis, her measurement collapses the qubit into a state consistent with her measurement, introducing a discrepancy if Bob measures in a different basis.

Probability of Detection:

- If Alice and Bob check s randomly chosen bits of their strings x' and y', the probability of detecting a discrepancy caused by Eve is at least:

$$P_{\text{detect}} = 1 - \left(\frac{3}{4}\right)^s .$$

- This arises because there is a $\frac{1}{4}$ probability that Bob's measurement will reveal an inconsistency for a single bit.

Conclusion: Eve's interference is detectable with high probability by comparing a subset of the shared key bits.

—

Summary

1. (a) The two bits are equal with certainty when Bob's measurement basis matches Alice's encoding.
2. (b) The two bits may differ when Bob's measurement basis differs from Alice's encoding.
3. (c) Eavesdropping introduces a discrepancy with a probability of at least $\frac{1}{4}$ per bit, making Eve's presence detectable with high probability when Alice and Bob compare enough key bits.

Problem 5: Reversible Circuits

Approach

To construct C', we need to ensure reversibility, which means the circuit must preserve all input information. This is achieved through the following steps:

1. Forward Simulation of Circuit C

- C computes $f(x)$ using gates $g_1, g_2, \ldots, g_T$, where T is the number of gates in C. - In C', simulate C gate by gate:

- Use the ancilla bits $z_1, z_2, \ldots, z_k$ to store intermediate results of C during computation.
- At each gate g_i, compute the output of g_i and store it in an appropriate ancilla bit.

- After the simulation, the output $f(x)$ is stored in the ancilla bits.

—

2. XOR Output into y

- After the forward simulation, XOR $f(x)$ into the y-bits:

$$y \to y \oplus f(x).$$

- At this point, the state of the system is $(x, y \oplus f(x), z_1, z_2, \ldots, z_k)$.
—

3. Erase Intermediate Results (Uncompute Ancilla Bits)

- To make C' reversible, we must reset all ancilla bits to their original state (i.e., zero). - This is achieved by **reversing the computation** of C:

- For each gate g_i, apply the inverse operation of g_i in reverse order.
- Since C is a classical Boolean circuit, every operation (e.g., NOT, CNOT, Toffoli) has an inverse that can be applied to "uncompute" the ancilla bits.

- After uncomputing, the state becomes $(x, y \oplus f(x), 0^k)$.
—

Reversibility and Gate Count

1. **Reversibility:** - The circuit C' preserves all input information because:

- x remains unchanged.
- y is XORed with $f(x)$, which is a reversible operation.
- All ancilla bits are reset to zero after uncomputing.

2. **Gate Count:** - The forward simulation of C requires $O(T)$ gates, where $T = \#\text{gates}(C)$. - The uncomputing step also requires $O(T)$ gates, as each gate is applied in reverse. - Therefore, the total gate count for C' is:

$$\#\text{gates}(C') = 2T = O(\#\text{gates}(C)).$$

—

Conclusion

The reversible circuit C' computes $(x, y, 0^k) \mapsto (x, y \oplus f(x), 0^k)$ with the following properties:

- C' is reversible, preserving all input information.
- The ancilla bits are reset to zero after computation.
- The gate count satisfies $\#\text{gates}(C') = O(\#\text{gates}(C))$, ensuring that C' has a similar size to C.