Advanced Randomization

Haoqiang Fan
IIIS, Tsinghua

Hello

- More stories about Xiaoqiang and Ameba!

Recovering Linear Function

- mod 2
- $f(x_1,x_2,\ldots,x_n)=x_1\oplus x_2\oplus x_5$
- given access to a noisy version of f
- $\tilde{f}(x)=f(x)\oplus g(x)$
- g is non-zero on only 30% inputs
- can you recover f?

Little Secret

- Familiar?
- If you only have **random** samples
- Last year's CTSC. Collision + FFT, time complexity is
- $O(n\gamma^{-2^{\frac{n}{\log m}-1}})$
- where the noise ratio is 1/2-γ
- not polynomial
- let's forget about it

Little Secret

- If we can query arbitrary point
- $\tilde{f}(x)$
- can we do better?
- Polynomial?

Yes

- The answer is Goldreich-Levin
- One of the most amazing algorithm (at least in theory)

Another Application

- Given **black box** access to a function
- You want to decide whether it only depends on only a few parameters
- $f(x_{1..n})=h(x_3,x_5,x_8)$
- or, whether it is close to such a thing
- $\tilde{f}(x_{1..n})=h(x_3,x_5,x_8)\oplus g(x)$

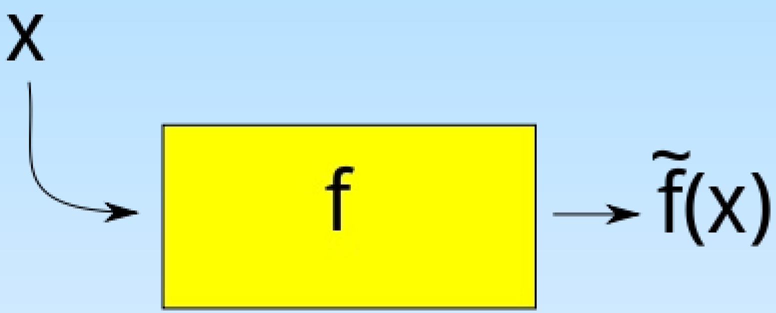
When do we encounter Black Box?

- complied binary

- obfuscation v.s. reverse engineering

Goldreich-Levin

- Let's only talk about the science part
- Given black-box and noisy access to a function,



- can you reverse-engineer it?

Fourier Analysis

- the linear attack
- any (boolean) function can be written as a weighted combination of linear (mod 2) functions
- $a \wedge b = 0.5\, a + 0.5\, b - 0.5\, a \oplus b$
- $a \vee b \vee c = 0.25\, a + 0.25\, b + 0.25\, c - 0.25\, a \oplus b - 0.25\, a \oplus c - 0.25\, b \oplus c + 0.25\, a \oplus b \oplus c$

Know How

- How do you find the coefficients?
- $\neg a \vee (b \wedge c)$

$$
\begin{aligned}
= &-0.75\, a + 0.25\, b + 0.25\, c \\
&-0.25\, a \oplus b + 0.25\, b \oplus c - 0.25\, a \oplus c \\
&+ 0.25\, a \oplus b \oplus c \\
&+ 1
\end{aligned}
$$

- Fourier Coefficients

Fourier Coefficients

- f(a,b,c)=... **-0.25** a⊕b + ...
- There are totally 2^n potential Fourier Coefficients, one for each frequency
- 2^n equations and 2^n unknowns
- more insightful explanation:
- correlation coefficient
- $2\left(\frac{1}{2^3}\sum_{a,b,c}[f(a,b,c)=a\oplus b]\right)-1$

Check it

a	b	c	f(a,b,c)	$a \oplus b$
0	0	0	1	0
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

	$a \oplus b=0$	$a \oplus b=1$
f(a,b,c)=0	1	2
f(a,b,c)=1	3	2

3/8*2-1=**-0.25**

Fourier Transform

- For each possible frequency (variable subset), compute
- $\hat{f}(S) = \frac{1}{2^n} \sum_{x} f(x)(-1)^{\sum_{i \in S} x_i}$
- e.g.

$$\hat{f}(\{1,2\}) = \frac{1}{2^3} \sum_{x_1, x_2, x_3} [f(x_1, x_2, x_3) \cdot (-1)^{x_1 + x_2}]$$

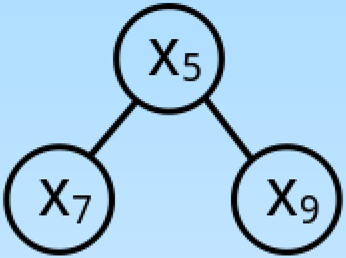
- Then $2\hat{f}(\{1,2\})$ will be the coefficient behind $x_1 \oplus x_2$

Fourier Coefficients

- $a \wedge b$ = **0.5** a + **0.5** b **-0.5** $a \oplus b$
- The sum of square of the coefficients <=1 (Parseval's Inequality)
- If we want to find all coefficients >=0.25, there will be at most 16 of them
- If the function only depends on a few variables, the Fourier coefficients will be sparse.

Fourier Coefficients

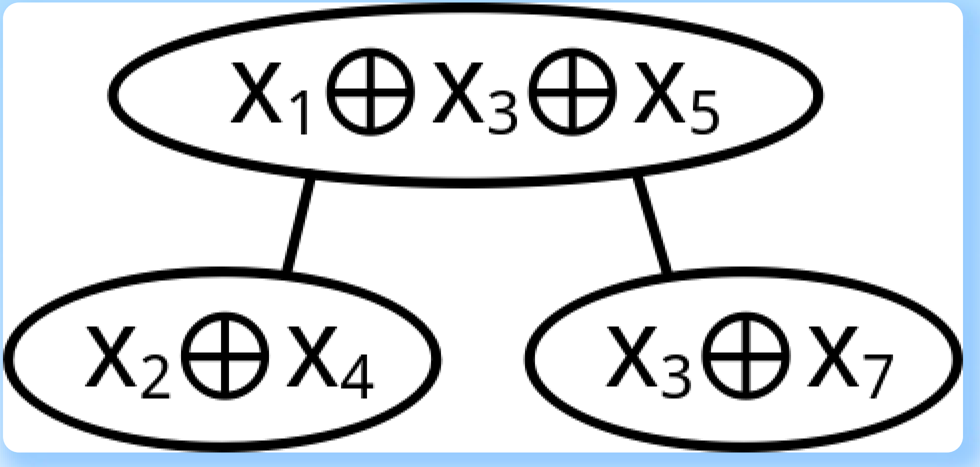
- ```
  int f(int x[]){
      if (x[5]){
          if (x[7]) return 0;
          else return 1;
      }else{
          if (x[9]) return 1;
          else return 0;
      }
  }
  ```



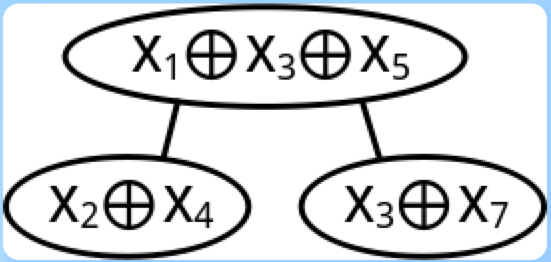
- Possible non-empty frequencies:
- x_5, x_7, $x_5 \oplus x_7$, x_9, $x_5 \oplus x_9$
  ```

So

Theoretically

- How to reverse-engineer a (simple) function
- S1: Somehow find the frequencies a, b, a⊕b
- S2: Compute correlations Pr[f(a,b,...)!=a⊕b]*2-1
- S3: Write the original function as a linear combination

Example

- Black box f(x[1..100]). We somehow know the non-empty frequencies are x_2, $x_1 \oplus x_2$, x_3, $x_1 \oplus x_3$.
- Randomly choose 10000 samples, compute the correlations
- $\Pr[f(x)=x_2]=0.675$
  $\Pr[f(x)=x_1 \oplus x_2]=0.675$
  $\Pr[f(x)=x_3]=0.675$
  $\Pr[f(x)=x_1 \oplus x_3]=0.375$
- Now we know that the function is
  $\mathbf{0.25}\ x_1 + \mathbf{0.25}\ x_1 \oplus x_2 + \mathbf{0.25}\ x_3 - \mathbf{0.25}\ x_1 \oplus x_3$

This is $x_1?x_2:x_3$

Now comes the key part

- How to find the non-empty frequencies?
- Goldreich-Levin
- Polynomial time complexity
- (but does not directly work in practice)

The key

- Consider what is
- $g(x_2,x_3,...)=(f(0,x_2,x_3,...)+f(1,x_2,x_3,...))/2.0$
- For f's Fourier component **α** $x_a \oplus x_b \oplus ...$
- If x_1 is inside the frequency, it disappears
- $(0 \oplus x_b \oplus x_c + 1 \oplus x_b \oplus x_c)/2=0.5$ ← **constant**
- Otherwise, it is untouched
- $(x_a \oplus x_b \oplus x_c + x_a \oplus x_b \oplus x_c)/2=x_a \oplus x_b \oplus x_c$
- So, g retains a subset of all frequencies.

The filter

- What is
- $g(x_4,\ldots)=$

+0.125 $f(0,0,0,x_4,\ldots)$ **-0.125** $f(0,0,1,x_4,\ldots)$

-0.125 $f(0,1,0,x_4,\ldots)$ **+0.125** $f(0,1,1,x_4,\ldots)$

+0.125 $f(1,0,0,x_4,\ldots)$ **-0.125** $f(1,0,1,x_4,\ldots)$

-0.125 $f(1,1,0,x_4,\ldots)$ **+0.125** $f(1,1,1,x_4,\ldots)$

The filter

- What is
- $g(x_4,\dots)=$

  +0.125 $f(0,0,0,x_4,\dots)$ **-0.125** $f(0,0,1,x_4,\dots)$

  -0.125 $f(0,1,0,x_4,\dots)$ **+0.125** $f(0,1,1,x_4,\dots)$

  +0.125 $f(1,0,0,x_4,\dots)$ **-0.125** $f(1,0,1,x_4,\dots)$

  -0.125 $f(1,1,0,x_4,\dots)$ **+0.125** $f(1,1,1,x_4,\dots)$
- $x_4 \oplus x_5 \oplus x_6$

The filter

- What is
- $g(x_4,\ldots)=$

  $+0.125\ f(0,0,0,x_4,\ldots)\ -0.125\ f(0,0,1,x_4,\ldots)$

  $-0.125\ f(0,1,0,x_4,\ldots)\ +0.125\ f(0,1,1,x_4,\ldots)$

  $+0.125\ f(1,0,0,x_4,\ldots)\ -0.125\ f(1,0,1,x_4,\ldots)$

  $-0.125\ f(1,1,0,x_4,\ldots)\ +0.125\ f(1,1,1,x_4,\ldots)$
- $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6$

The filter

- What is
- $g(x_4,\ldots)=$

  +0.125 $f(0,0,0,x_4,\ldots)$ **-0.125** $f(0,0,1,x_4,\ldots)$

  -0.125 $f(0,1,0,x_4,\ldots)$ **+0.125** $f(0,1,1,x_4,\ldots)$

  +0.125 $f(1,0,0,x_4,\ldots)$ **-0.125** $f(1,0,1,x_4,\ldots)$

  -0.125 $f(1,1,0,x_4,\ldots)$ **+0.125** $f(1,1,1,x_4,\ldots)$
- $\cancel{x_1 \oplus} x_2 \oplus x_3 \oplus \ldots$

The Filter

- If we want to know whether f contains a (large) frequency with prefix $x_a \oplus x_b \oplus \ldots \oplus x_d$
- We just need to see if $g(x_k,\ldots)=E_{x_1,x_2,\ldots,x_{k-1}}[f(x)(-1)^{x_a \oplus x_b \oplus \ldots \oplus x_d}]$ is (close to) constant
- The extremely clever reformulation
- $E_{x,x'_{1..k}}[f(x)f(x'_{1..k}x_{k+1..n})\,\chi_S(x_{1..k})\chi_S(x'_{1..k})]$
- χ_S is the "mask"

Goldreich-Levin

- Binary search
- see if there are frequencies beginning with 0
- see if there are frequencies beginning with 00
- see if there are frequencies beginning with 000
- ......
- $E_{x,x'_{1..k}} [f(x)f(x'_{1..k}x_{k+1..n})\ \chi_S(x_{1..k})\chi_S(x'_{1..k})]$

GL

- Very good in theory. Used to prove Goldreich and Levin's Hard Core Predicate.
- Not useful in (OI) practice because of its huge constant factor.
- $O(n\varepsilon^{-6}\log(n/\varepsilon))$

However

- It can be optimized in practice.
- And we need randomization to do it.

Hashing

- Suppose the frequencies' first three digits are all different
- 000???
- 001???
- ...
- 110???
- 111???
- Then we can dig them out one by one

Isolating

- define $g(x_4,\ldots)=(f(0,0,0,x_4,\ldots)+f(0,0,1,x_4,\ldots)+$
  $f(0,1,0,x_4,\ldots)+f(0,1,1,x_4,\ldots)+$
  $f(1,0,0,x_4,\ldots)+f(1,0,1,x_4,\ldots)+$
  $f(1,1,0,x_4,\ldots)+f(1,1,1,x_4,\ldots))/8$
- If we are lucky, g will be (close to) **linear**

Isolating

- How to recover a linear function's coefficients
- at the presence of a tiny amount of noise?
- Influence
- $\Pr[g(x_1,\ldots,x_{k-1},0,x_{k+1},\ldots,x_n) \cdot g(x_1,\ldots,x_{k-1},1,x_{k+1},\ldots,x_n)<0]$
- So long as < 0.25 of all g's sign are reversed, we will be able to decide if the uncorrupted g depends on x_k

And

- $g(x_4,\ldots)=(f(0,0,0,x_4,\ldots)-f(0,0,1,x_4,\ldots)+f(0,1,0,x_4,\ldots)-f(0,1,1,x_4,\ldots)-f(1,0,0,x_4,\ldots)+f(1,0,1,x_4,\ldots)-f(1,1,0,x_4,\ldots)+f(1,1,1,x_4,\ldots))/8$
- selects the 101??? frequency out
- and so on

One Issue

- What if the frequencies only differ at the last digits?
- ???000
  ???001
  ???010
  ...
  ???111
- We should somehow transform them to the "good cases"

Randomization

- Randomly choose a (reversible) linear substitution
- $y_1=x_2 \oplus x_3$
- $y_2=x_1$
- $y_3=x_1 \oplus x_3$
- $f(x_1,x_2,x_3)=u(y_1,y_2,y_3)$
- Find the Fourier components of u, then do back substitution
- u will (probably) be good

Optimization?

- Works when the noise is small enough
- Now talk about implementation details

```
vector<vec> genFreq(){
    vec U[MMax],V[MMax];
    getOrthBasis(U,V);
    int pt=0;
    for (int i=0;i<N;i++){
        for (int j=0;j<RMax;j++){
            vec u;
            for (int k=0;k<N;k++)u[k]=rand()&1;
            for (int z=0;z<=1;z++){
                u[i]=z;
                for (int l=0;l<(1<<M);l++){
                    vec x=u;
                    for (int m=0;m<M;m++)if ((1<<m)&l)x=x^V[m];
                    int key=0;
                    for (int m=0;m<M;m++)
                        key|=dotproduct(U[m],x)<<m;
                    trials[pt+key]=x;
                }
                pt+=(1<<M);
            }
        }
    }
    evaluatetrials(trials,trialresults,N*RMax*2*(1<<M));
```

```
vector<vec> found;
for (int i=0;i<(1<<M);i++){
    vec freq;
    for (int j=0,pt=0;j<N;j++){
        int s=0;
        for (int k=0;k<RMax;k++){
            int t[2]={0,0};
            for (int z=0;z<2;z++){
                for (int l=0;l<(1<<M);l++){
                    if (__builtin_popcount(l&i)&1)
                        t[z]-=trialresults[pt++];
                    else t[z]+=trialresults[pt++];
                }
            }
            s+=((t[0]>0)!=(t[1]>0));
        }
        freq[j]= s*2-RMax > 0;
    }
    for (int j=0;j<M;j++)if ((1<<j)&i)freq^=U[j];
    int mcnt=0;
    for (int j=0;j<TMax;j++)
        mcnt+=(testresults[j]==dotproduct(testset[j],freq));
    if (abs(mcnt*2-TMax)>=TMax*0.040)
        found.push_back(freq);
}
```

Implementation Detail

- How to generate orthogonal basis?
- Brute-force is OK

```
while (true){
    for (int i=0;i<M;i++)for (int j=0;j<N;j++)U[i][j]=rand()&1;
    bitset<MMax> found;
    for (int i=0;i<3*M*(1<<M);i++){
        vec x;
        for (int j=0;j<N;j++)x[j]=rand()&1;
        int s0=0,s1=0;
        for (int j=0;j<M;j++)
            if (dotproduct(U[j],x))
                s0++,s1+=j;
        if (s0==1 && !found[s1]){
            found[s1]=1;
            V[s1]=x;
            if (found.count()==M)break;
        }
    }
    if (found.count()==M)break;
}
```

Implementation Detail

- ```
  typedef bitset<NMax> vec;
  int dotproduct(const vec & a,const vec & b){
      return (a&b).count()&1;
  }
  ```
- Run multiple times until enough energy is collected
- How large should the test set be?
- $O(\varepsilon^{-2}\log n)$
  ```

Application

- Reverse a depth 2 Parity Decision Tree



Application

- 

- discovered frequencies
- $x_1 \oplus x_3 \oplus x_5$

  $x_1 \oplus x_3 \oplus x_5 \oplus x_2 \oplus x_4$

  $x_2 \oplus x_4$

  $x_1 \oplus x_3 \oplus x_5$

  $x_1 \oplus x_5 \oplus x_7$

  $x_3 \oplus x_7$

Application

- Discovered a 3-dim subspace, totally 8 possible frequencies
- brute force search 8^3 possible trees

Enough about it

- Let's talk about something fun

Interactive Proof

- How to prove that two graphs are isomorphic?

Interactive Proof

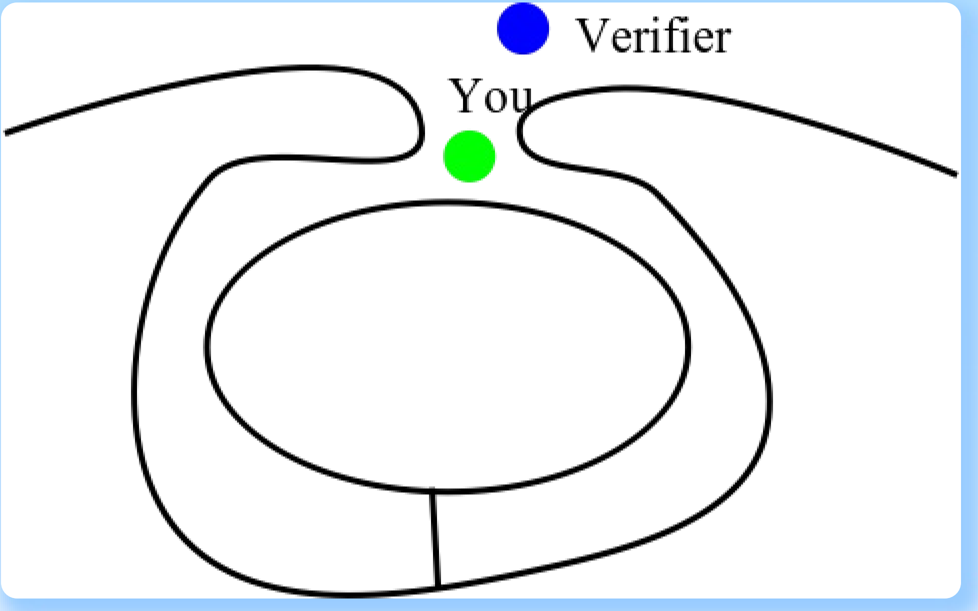
- How to prove that two graphs are not isomorphic?

Interactive Proof

- Zero knowledge proofs
- Prove to someone (verifier) that you know something without revealing it

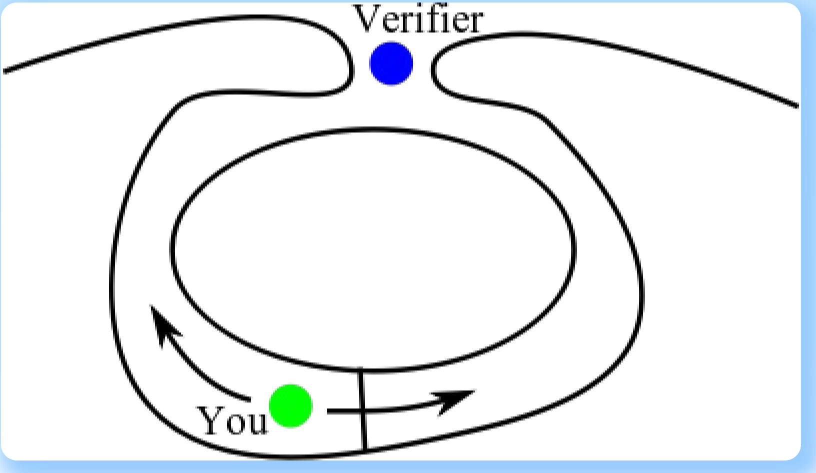
One Example

- How to prove that you can cross a wall?



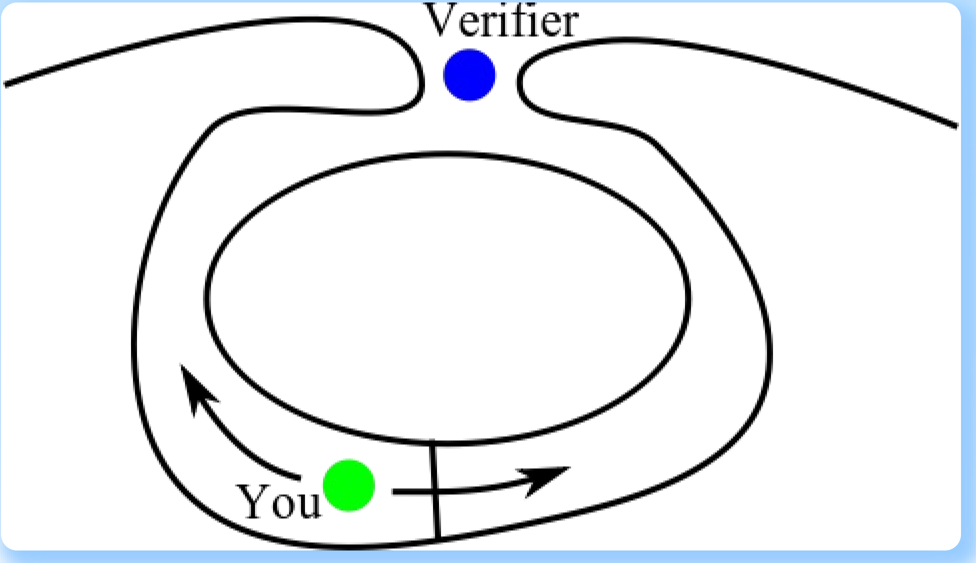
One Example

- The prover goes into the cave, chooses a direction and goes into it
- The verifier goes into the cave, requests the prover comes from one of the directions.



One Example

- Zero knowledge: The verifier cannot even prove to others that he knows the prover can do this.
- He can make the video record himself



Discrete Logarithm

- how to prove that you know an x such that $g^x \bmod p = C$

Discrete Logarithm

- $g^x \bmod p = C$
- **prover** randomly chooses r, sends $k=g^r \bmod p$ to verifier
- **verifier** either requests prover to send r or $(x+r) \bmod p-1$ and checks this
- $g^{(x+r) \bmod p-1}=kC$

Sudoku

- How to prove that you have solved a Sudoku?

Sudoku

- write numbers on chess pieces and put them upside-down on a board
- the **verifier** chooses a row/column/block
- the **prover** picks those pieces up from the board, shuffle them and give them to the verifier
- repeat for many rounds

3 coloring

- How to prove that you know a 3-coloring of a graph?
- Of particular interest because graph coloring is NP-hard.

graph coloring

- randomly shuffle your coloring
- cover them with chess pieces
- verifier requests to reveal two adjacent pieces
- repeat

Bit Commitment

- How to do the choose-and-reveal step?
- prover sends hash(c_i,$salt_i$) for all i
- verifier requests c_u,c_v,$salt_u$,$salt_v$ for adjacent u and v

NP-hard?

- The implication of NP-hardness
- Any NP problem can be reduced to it
- Especially, you can prove that you know a proof to a math theorem
- because MATH $\in$ NP

More on cryptography

- randomness + hardness assumption + interaction

Flip a coin through telephone

- How to flip a coin through a telephone line?
- "I guess your coin is head"
- "Let me see... Sorry, it is tail"

Quadratic Residual

- Alice chooses p,q, sends m=pq to Bob
- Bob chooses x_1, sends $y=x_1^2$ mod m
- $y=x^2$ mod m has two roots x_1 and x_2

How to compute square root

- How to compute square root of y mod pq?
- compute the square root of y mod p and mod q, then CRT
- How to compute square root of y mod p?
- Tonelli Shanks / Cipolla
- $\left(a + \sqrt{a^2 - y}\right)^{(p+1)/2} \mod p$
- All operations are mod p. a^2-y is chosen so that it is not an quadratic residual.

Flip Coin

- Alice sends $t=x_1^2$ mod m to Bob
- Bob knows how to compute sqrt of t, but he does not know which root is Alice's
- He arbitrarily chooses one and sends it to Alice.
- If he happens to send x_2 to Alice, Alice now knows the factorization of m
- $x_1^2-x_2^2=0$ mod m
- Otherwise, Alice still does not know how to factor m
- A fair coin flip

Do you want more cryptography?

- Perhaps not
- Then let's move back to proofs
- The PCP Theorem
- NP=PCP(O(log n),O(1))

PCP Theorem

- Any NP language admits polynomial length proof
- that can be checked **probabilistically** by looking at only constant number of bits
- "New Short Cut Found For Long Math Proofs"
- 1992 April 7, New York Times

How it works

- A certificate of an instance belonging to a language
- A randomized verifier only looks at constant number of bits in the proof
- and has constant probability of rejecting fake proofs

The PCP Theorem

- I'll cheat you a little bit
- Only prove that NP=PCP(poly(n),O(1)) because it is much easier but still interesting
- Heavily relies on randomization

NP-Hard

- Solving quadratic boolean equations is NP-Hard
- $x_2x_3+x_1x_1=0 \bmod 2$
  $x_1x_2+x_1x_3+x_2x_3=1 \bmod 2$
  ......
- Now I'll give a 2^{n^2} length **P**robabilistically **C**heckable **P**roof to it.

The proof

- computes $x_i x_j$ for all i,j
- writes all linear combinations of them in the proof
- totally 2^{n^2} bits
- how should I check it?

Step 1

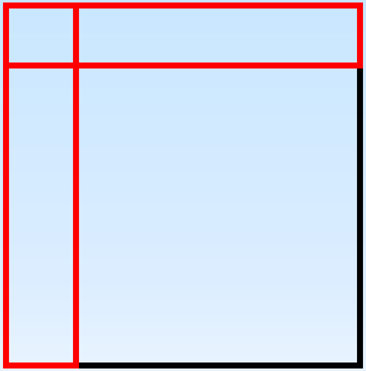
- check that the original equations are satisfied
- I cannot check the m equations one by one, because this requires reading more than O(1) bits
- randomly choose a subset of equations and check their sum
- $x_2x_3+x_1x_1=0 \bmod 2$
  $x_1x_2+x_1x_3+x_2x_3=1 \bmod 2$
  ......
- $x_1x_1+x_1x_2+x_1x_3 = 1$?

Step 1

- because the chosen subset is random
- if there are at least one unsatisfied equations
- my chance to detect it is >= 0.5
- 001000000001100000 010100101010101101

Step 2

- Check that $x_i x_j$ is really x_i times x_j
- The same trick
- randomly choose two subsets A,B
- checks if
$$\sum_{i\in A} x_i^2 \sum_{i\in B} x_i^2 = \sum_{i\in A, j\in B} x_i x_j$$
- at least 1/4 prob. to reject wrong proofs



Step 3

- Check that the 2^{n^2} bits encodes after all the linear combination of n^2 variables.
- Equal to testing whether a function is linear
- 0011001100110011

The linearity test

- Randomly choose a,b
- Checks if f(a+b)=f(a)+f(b) mod 2
- probability of passing is $\sum_S \hat{f}^3(S) \leq \max \hat{f}(S)$
- So the function is **close** to linear if it passes many random tests

Are we done?

- Wait, there is one issue
- Independence

Self Correcting

- Using the linearity test, we can use ?000 tests to ensure that the truth table is 0.999 close to **some** linear function (although we don't know which one it is)
- However, no guarantee is given on which bits are corrupted
- And the queries in Step 1,2 are not uniform

Self Correcting

- It is possible that although only a small portion of bits are corrupted in the linear function
- but they are the bits that are queried in Step 1 and 2
- Solution: self correcting
- $f(x)=f(a)+f(x+a)$
- query $f(a)$, $f(x+a)$ whenever we want to get $f(x)$ in later steps
- the queries are uniform now

Putting everything together

- NP=PCP(O(n^2),O(1))
- Can be optimized to PCP(O(log n),O(1)) by using more advanced techniques
- enough about proofs!

A New Page

- Let's talk about something interesting (and useful for OI)
- Advanced Randomization
- Sketching

Sketch

- A short summary (hash) of a large object that (probabilistically) preserves some of its properties
- Randomization */+ Approximation

Sketch of String Equality

- Suppose both you and your friend have a (long) 01 string
- You want to decide whether they are same
- by exchanging only 1 bit
- with accuracy requirement >= 2/3

Which bit to send?

- The first bit?
- The last bit?
- The xor of all bits?

1 bit sketch

- Assume you have shared randomness
- randomly choose a subset of bits (must be the same subset for both of you)
- send the xor sum
- If not the same, >=1/2 probability of detecting it

1 bit sketch

- same: 100% - 0%
- not same: 50% - 50%
- The trick: if the bits match, answer with 2/3 prob. that the strings are the same
- same: 67% - 33%
- not same: 67% - 33%

Better Accuracy

- Just increase the number of bits
- by repeating this process
- k bits $\rightarrow$ 2^{-k} failure prob.

Sketch of set size

- support union operation
- the MinHash algorithm
- f(X)=min((ax+b)%p for x in X)

Review of MinHash

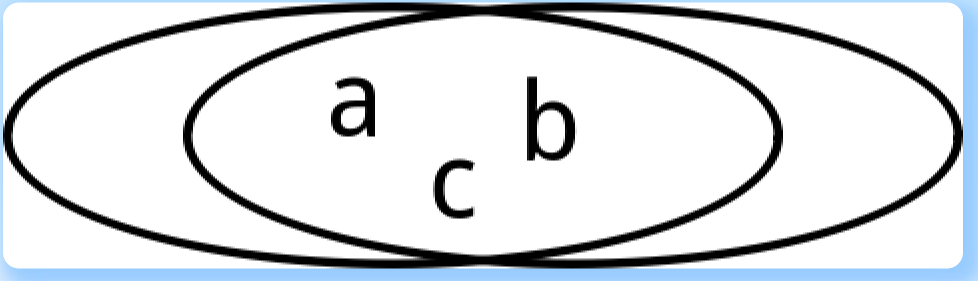
- pairwise independence, Chernoff bound, ...
- improved version: multi-scale buffer
- store a buffer of k^2 items
- only store items whose hash values' last p digits are zero
- size=#elements in buffer * 2^p
- relative error <= O(1/k)
- merge sort to merge two sets

BJOI

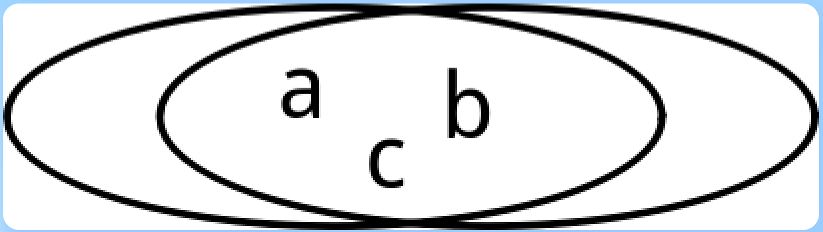
- maintain many sets
- each time, create a new set as the union of two previously created sets
- return the size of the newly created set
- probably (>=95%) approximately (±25%) correct

Sketch of set intersection

- decide whether two sets are close enough to each other
- close : intersection / union is large
- MinHash
- the k smallest distinct hash values



Sketch of set intersection

- 

- k smallest hash values
- assume intersection >= p union
- prob. of equal : p^k

Thu Training 2014

- Ameba mutation
- find similar N-element set pairs in $2N^2$ subsets of a N^2 universe
- randomly generated
- intersection of close pairs: N/2
- p=1/3 for close pairs, 1/N for irrelevant pairs
- choose k=2
- #subsets shrinks by 1/9 after each iteration
- $O(N^2)$ false positives

Implementation Detail

- do not explicitly choose k
- randomly relabel the elements
- sort alphabetically, check adjacent pairs
- using a reverted index, this can be done in sub-linear time!
- Overall time complexity: linear

Sketch of Hamming Distance

- sketch used to estimate the Hamming Distance of two 01 strings
- $d(x,y) \approx f(h(x),h(y))$
- randomly choose p bits, compute xor sum
- enumerate $p=1,2,4,\ldots,n/2,n$
- multiple runs
- multi-scale Equality sketch

Multi-scale Equality Sketch

- assume the real Hamming distance is qN
- p bits: equal prob. = $(1-q)^p$
- O(1) distortion

Nearest Neighbour Search

- In Hamming cube
- coarse-to-fine hashes based on sampling and Equality
- Efficient Search for Approximate Nearest Neighbour in High Dimensional Space
- Kushilevtz, Ostrovsky, Rabani

Sketch of Edit Distance?

- Sketch of string's edit distance?
- Possible! Distortion $2^{O(\sqrt{\log n \log \log n})}$
- patented, not very useful in OI (at least now)

Sketch of frequency

- Implement a multiset that supports count()
- for the most frequent elements
- naive idea: randomly sample a population
- works if the portion of the element is high

Sketch of frequency

- Improved idea
- Hash each element to a unit vector in k-dim sphere
- Record the sum, compute the dot product
- Works in expectation
- $1/\sqrt{N}$,1/N,1/N,...,1/N
- better than the naive idea!

Signal to Noise Ratio

- signal: x_i
- noise: $\sqrt{\sum_j x_j^2}$
- works if the portion of second-order moment is high
- always better for the most frequent element

sketch of most frequent element

- hash the elements to k bins
- inside each bin, map each element to a unit vector (or just ±1), compute the sum
- take the maximum absolute value
- linear sketch!

sketch of second order moment

- $x_1^2+x_2^2+...+x_n^2$
- The amazing AMS algorithm
- map each element to ±1
- compute the sum
- square it

How it works

- $1+1+...+1=E[(\pm1\pm1...\pm1)^2]$

Let me analyze it

- $X=x_1+...+x_n$
- $E[X^2]=x_1{}^2+x_2{}^2+...+x_n{}^2=F_2$
- $E[(X^2-F_2)^2]<=E[X^4+F_2{}^2]<=2F_2{}^2$
- Chebyshev bound
- ε^{-2} repetitions
- 4-wise independence

How to get 4-wise independence?

- $ax^3+bx^2+cx+d \bmod p$

Sketch endless

- verification
- database search

Best is Endless.

www.faceplusplus.com