M1941 - Flawed Four

Percy Wong {percywtc}

Authored by percywtc, prepared by kctung

Slides by percywtc

The Problem: tl;dr

Given **N** "**L**-bit" binary numbers, find a subset s.t.:

- the size of the subset is either 1, 2, 3, or 4
- "the xor sum of the subset" = 0

e.g. "`000 = 0`"

"`01 xor 01 = 0`"

"`01 xor 10 xor 11 = 0`"

SAMPLE TESTS

	Input	Output
1	5 4 1000 0100 0010 0001 1111	-1
2	2 3 000 111	1 1
3	4 2 01 01 10 11	2 1 2
4	4 2 01 01 10 11	3 1 3 4

This input is the same as the previous one.

How to read the input?

```
long long num = 0;       // 64-bit
for (int i = 0; i < L; i++) {
  scanf("%1d", &x);      // scanning only 1 digit
  num = (num << 1) + x; // left-shift by 1 bit
}
```

Subtask 1

N ≤ 20, **L** ≤ 6

We can just exhaust every subset possibility, and check if it meets all constraints:

- the size of the subset is either 1, 2, 3, or 4
- “the xor sum of the subset” = 0

Exhaustion can be done via recursion / iteration

there are 2^N possible subsets,
total time complexity = $O(2^N)$ or $O(N \times 2^N)$ depends on implementation

Subtask 1

N ≤ 20, **L** ≤ 6

We can just exhaust every subset with “1 ≤ size ≤ 4”, by using:

- for() {}
- for() for() {}
- for() for() for() {}
- for() for() for() for() {}

total time complexity = $O(N^4)$

Subtask 2

L ≤ 6

Observe that it's not necessary to use the same value 3 times or more

- because: `x xor x xor x` = x

Therefore, although **N** is large (2500), we can reduce the no. of numbers we have

- reduce the occurences of every number to AT MOST 2

N is now small (at most $2^6 \times 2 = 128$)

We can just do the same $O(N^4)$ exhaustion just like the previous slide

Subtask 3

Notice that if we fix 3 numbers, we need not exhausting the 4th number

- if `a xor b xor c xor d = 0`, then `a xor b xor c = d`

Instead of finding the 4th number using the 4th for loop, we can just

- try to use a faster time to know if there exists the required number

How to do so? Boolean array? Frequency array?

- the numbers are too large (2^{64}), we have to use some data structures
- `std::map` in C++

total time complexity = $O(N^3 \log N)$

Subtask 4

What if we just exhaust two numbers and find the required one/two numbers

- if `a xor b xor c xor d = 0`, then `a xor b = c xor d`

We can first precompute all possibilities of `num[i] xor num[j]` (where `i != j`)

Store them in a data structure (e.g. `std::map`),
just exhaust two numbers, find if the required “xor sum” exists and available or not

total time complexity = $O(N^2 \log N^2)$

Subtask 5

Hash table is the king! (e.g. `std::unordered_map`)

total time complexity = $O(N^2)$

HKOI Mini-Competition III
M1942 - Connectedness Calculation

Alex Tung
alex20030190[at]yahoo.com.hk

13 April, 2019

Problem Statement

- Given a complete graph with $N \leq 50$ nodes.
- Each edge has a probability $q := \frac{p}{100}$ of being removed.
- Edge removals are independent events.
- Calculate the expected number of connected components of the remaining graph.

Solution 1

- Dynamic Programming; but how?

Solution 1

- Dynamic Programming; but how?
- Simulate the "expansion" of a connected component as follows:
  - Maintain a set of nodes in the connected component (call them "active")
  - For each step, pick an active node, draw some number of edges to inactive nodes, then remove the node
  - When there are no more active nodes, start anew

DP Formulation for Solution 1

- Precompute $binom[e][f]$, the probability that, among e edges, f edges will remain. The value equals $\binom{e}{f} q^{e-f}(1-q)^f$.
- Let $dp[r][a][c]$ denote the probability that:
  - r nodes remain
  - a nodes are active
  - There are c connected components
- Base case: $dp[N][1][1] = 1$

DP Formulation for Solution 1

- Precompute $binom[e][f]$, the probability that, among e edges, f edges will remain. The value equals $\binom{e}{f} q^{e-f}(1-q)^f$.
- Let $dp[r][a][c]$ denote the probability that:
  - r nodes remain
  - a nodes are active
  - There are c connected components
- Base case: $dp[N][1][1] = 1$
- Transition, in decreasing r:
  - For $c > 0$ increase $dp[r][1][c+1]$ by $dp[r][0][c]$ (start anew when there are no active nodes)
  - For $a > 0$, $c > 0$, $0 \leq d \leq r+1-a$, increase $dp[r][a+d-1][c]$ by $dp[r+1][a][c] \times binom[r+1-a][d]$ (pick one active node, draw d edges to the $r+1-a$ inactive nodes)
- The answer is $\sum_{c=1}^{n} c \times dp[0][0][c]$.

Analysis of Solution 1

- We need to iterate over
  - r (number of remaining nodes),
  - a (number of active nodes),
  - c (number of connected components), and
  - d (number of edges drawn to inactive nodes)
- Time complexity is $O(N^4)$ per query, with a really small constant.
- Can we do better?

Solution 2

- Still DP, but more efficient
- Observe that the c dimension is pretty "static"
- Let $prob[r][a]$ denote the probability that:
  - r nodes remain
  - a nodes are active
- Let $ev[r][a]$ denote the expected number of connected components **given that** r nodes remain and a nodes are active, **multiplied by** $prob[r][a]$.
- Base case: $prob[N][1] = 1$, $ev[N][1] = 1$

Transition Formula for Solution 2

Transition, in decreasing r:

- Handling the case with no active nodes:
  - $ev[r][1] += ev[r][0] + prob[r][0]$
  - $prob[r][1] += prob[r][0]$
- Draw edges to inactive nodes, for $a > 0, 0 \leq d \leq r + 1 - a$:
  - $ev[r][a + d - 1] += ev[r + 1][a]$
  - $prob[r][a + d - 1] += prob[r + 1][a]$
- Time complexity is $O(N^3)$ per query, with a really small constant.

Why set $N \leq 50$?

For N large, answer is very close to 1 unless $p \approx 100$.
For example, below shows answer for $N = 50$ corrected to 7 d.p.:

- $p = 50$: `1.0000000`
- $p = 60$: `1.0000000`
- $p = 70$: `1.0000013`
- $p = 80$: `1.0008922`
- $p = 90$: `1.2914863`
- $p = 95$: `5.6574515`
- $p = 97$: `15.8914262`
- $p = 98$: `25.9389519`
- $p = 99$: `37.7800382`
- $p = 100$: `50.0000000`

HKOI Mini-Competition III
M1943 - Food Poisoning

Alex Tung
alex20030190[at]yahoo.com.hk

13 April, 2019

Problem Statement

- K dishes out of $1..N$ are "bad".
- N is known but K is not.
- Handle 1000 queries.
- Each query consists of a subset Q of $1..N$ and you are to guess if Q contains any bad dishes.
- Correct answer provided after making every guess.
- Target:
  - Subtask 1: make at most 6 mistakes for $1 \leq N \leq 100$, $K = 1$
  - Subtask 2: make at most 90 mistakes for $1 \leq N \leq 500$, $1 \leq K \leq 5$.

Solution 1

- Define $isBad[1..N]$:
  - ($isBad[i] = 1$: bad dish)
  - $isBad[i] = -1$: not bad dish
  - $isBad[i] = 0$: unsure
- For each query Q:
  - If Q contains at least one j with $isBad[j] \geq 0$ (bad or unsure), return `true`.
  - Else, return `false`.

Solution 1

- Define $isBad[1..N]$:
  - ($isBad[i] = 1$: bad dish)
  - $isBad[i] = -1$: not bad dish
  - $isBad[i] = 0$: unsure
- For each query Q:
  - If Q contains at least one j with $isBad[j] \geq 0$ (bad or unsure), return `true`.
  - Else, return `false`.
- False negative (guess `false`, answer is `true`) can never occur.
- If false positive (guess `true`, answer is `false`), for all $j \in Q$ set $isBad[j] := -1$.

Solution 1

- Define $isBad[1..N]$:
  - ($isBad[i] = 1$: bad dish)
  - $isBad[i] = -1$: not bad dish
  - $isBad[i] = 0$: unsure
- For each query Q:
  - If Q contains at least one j with $isBad[j] \geq 0$ (bad or unsure), return `true`.
  - Else, return `false`.
- False negative (guess `false`, answer is `true`) can never occur.
- If false positive (guess `true`, answer is `false`), for all $j \in Q$ set $isBad[j] := -1$.
- There can be at most $N - K$ wrong guesses.
- $score = 10 + 15.08 = 25.08$.

Solution 2

Works for Subtask 1 only.

- Also use $isBad[1..N]$.
- For each query Q:
  - Among all dishes i such that $isBad[i] \geq 0$, count how many of them are in Q.
  - If **at least half** are in Q, return `true`.
  - Else, return `false`.

Solution 2

Works for Subtask 1 only.

- Also use $isBad[1..N]$.
- For each query Q:
  - Among all dishes i such that $isBad[i] \geq 0$, count how many of them are in Q.
  - If **at least half** are in Q, return `true`.
  - Else, return `false`.
- If false negative, for all dishes i **not** in Q set $isBad[i] := -1$.
- If false positive, for all dishes i in Q set $isBad[i] := -1$.

Solution 2

Works for Subtask 1 only.

- Also use $isBad[1..N]$.
- For each query Q:
  - Among all dishes i such that $isBad[i] \geq 0$, count how many of them are in Q.
  - If **at least half** are in Q, return `true`.
  - Else, return `false`.
- If false negative, for all dishes i **not** in Q set $isBad[i] := -1$.
- If false positive, for all dishes i in Q set $isBad[i] := -1$.
- After each mistake, number of dishes with $isBad[i] \geq 0$ drops to at most half its original size. Number of mistakes $\leq \log_2(N) \approx 6.6$.
- $score = \mathbf{30} + 0 = 30$:)

A "Theoretical" Solution

- Maintain a set S of *potential combinations* (hypotheses) of bad dishes. Initially S contains $\binom{N}{1} + \binom{N}{2} + \binom{N}{3} + \binom{N}{4} + \binom{N}{5}$ combinations.
- For each query Q:
  - If **at least half** of the dish combinations in S contains some dishes in Q, return `true`.
  - Else, return `false`.

A "Theoretical" Solution

- Maintain a set S of *potential combinations* (hypotheses) of bad dishes. Initially S contains $\binom{N}{1} + \binom{N}{2} + \binom{N}{3} + \binom{N}{4} + \binom{N}{5}$ combinations.
- For each query Q:
  - If **at least half** of the dish combinations in S contains some dishes in Q, return `true`.
  - Else, return `false`.
- By a similar analysis, Number of mistakes $\leq \log_2(\binom{N}{1} + \binom{N}{2} + \binom{N}{3} + \binom{N}{4} + \binom{N}{5}) \approx 37.9$.
- But this solution cannot fit within time or memory limit...
- Think of it as a rough lower bound for the number of mistakes :)

Solution 3

- For each query Q:
  - If $isBad[i] = 1$ for some $i \in Q$, return `true`;
  - Otherwise, let *unsure* be the number of $i \in Q$ with $isBad[i] = 0$.
  - Return `true` if *unsure* ≥ 2; return `false` otherwise.

Solution 3

- For each query Q:
  - If $isBad[i] = 1$ for some $i \in Q$, return `true`;
  - Otherwise, let *unsure* be the number of $i \in Q$ with $isBad[i] = 0$.
  - Return `true` if *unsure* ≥ 2; return `false` otherwise.
- False negative: Q contains just one "unsure" dish i, so it has to be bad. Set $isBad[i] := 1$.
- False positive: for all $i \in Q$ set $isBad[i] := -1$.

Solution 3

- For each query Q:
  - If $isBad[i] = 1$ for some $i \in Q$, return `true`;
  - Otherwise, let *unsure* be the number of $i \in Q$ with $isBad[i] = 0$.
  - Return `true` if *unsure* ≥ 2; return `false` otherwise.
- False negative: Q contains just one "unsure" dish i, so it has to be bad. Set $isBad[i] := 1$.
- False positive: for all $i \in Q$ set $isBad[i] := -1$.
- There are at most K false negatives and $\frac{N-K}{2}$ false positives.
- $score = 17.99 + 34.84 = 52.83$.

Solution 4

Let L be a threshold parameter.

- For each query Q:
  - If $isBad[i] = 1$ for some $i \in Q$, return `true`;
  - Otherwise, let *unsure* be the number of $i \in Q$ with $isBad[i] = 0$.
  - If $unsure \geq L$, return `true`.
  - Else return `false`.

Solution 4

Let L be a threshold parameter.

- For each query Q:
  - If $isBad[i] = 1$ for some $i \in Q$, return `true`;
  - Otherwise, let *unsure* be the number of $i \in Q$ with $isBad[i] = 0$.
  - If $unsure \geq L$, return `true`.
  - Else return `false`.
- False negative: set $isBad[i] := 1$ for all $i \in Q$ (Note that the conclusion may be wrong, resulting in more false postives!)
- False positive: set $isBad[i] := -1$ for all $i \in Q$

Solution 4

Let L be a threshold parameter.

- For each query Q:
  - If $isBad[i] = 1$ for some $i \in Q$, return `true`;
  - Otherwise, let *unsure* be the number of $i \in Q$ with $isBad[i] = 0$.
  - If $unsure \geq L$, return `true`.
  - Else return `false`.
- False negative: set $isBad[i] := 1$ for all $i \in Q$ (Note that the conclusion may be wrong, resulting in more false postives!)
- False positive: set $isBad[i] := -1$ for all $i \in Q$
- There can be at most $\frac{N}{L} + K \times (L - 2)$ false positives and K false negatives.
- Best to set $L \approx 10$.
- Score:
  - $L = 10$: $score = 22.88 + 68.52 = 91.40$
  - $L = 11$: $score = 22.88 + 69.16 = 92.04$

Solution 5

- Introduce auxiliary variables $a_1, \ldots, a_n$. Initially, $a_i := 1$.
- For each query Q:
  - Compute $x := \sum_{i \in Q} a_i$.
  - If $x \geq n$, return `true`
  - Else, return `false`

Solution 5

- Introduce auxiliary variables $a_1, \ldots, a_n$. Initially, $a_i := 1$.
- For each query Q:
  - Compute $x := \sum_{i \in Q} a_i$.
  - If $x \geq n$, return `true`
  - Else, return `false`
- False negative: for each $i \in Q$ **double** the value of a_i.
- False positive, for each $i \in Q$ set $a_i := 0$.

Analysis of Solution 5

- Let fn be the number of FNs and fp the number of FPs.

Analysis of Solution 5

- Let fn be the number of FNs and fp the number of FPs.
- fn is bounded by $K \times ceil(\log_2 N)$, because for each bad dish i, a_i gets doubled at most $ceil(\log_2 N)$ times.
- fp is bounded by fn, because $\sum_i a_i$ increases by at most $n - 1$ for each FN and decreases by at least n for each FP, and the sum is always nonnegative.

Analysis of Solution 5

- Let fn be the number of FNs and fp the number of FPs.
- fn is bounded by $K \times ceil(\log_2 N)$, because for each bad dish i, a_i gets doubled at most $ceil(\log_2 N)$ times.
- fp is bounded by fn, because $\sum_i a_i$ increases by at most $n - 1$ for each FN and decreases by at least n for each FP, and the sum is always nonnegative.
- Therefore, the number of mistakes $\leq 2K \times ceil(\log_2 N)$.
- For $K \leq 5, N \leq 500$, number of mistakes ≤ 90 using Solution 5.
- $score = 24.35 + \mathbf{70} = 94.35$:)

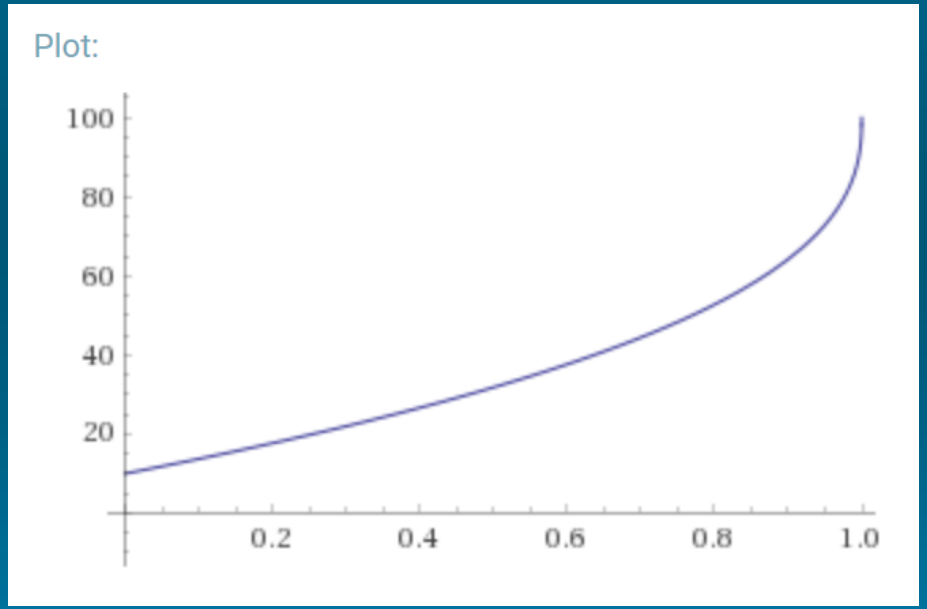
Call For Nice Solutions

If you find a nice solution, do let me know!

Prime Scrabble

Tony Wong
Hong Kong Olympiad in Informatics
2019-04-13

Scoring Formula



Case 1

- 2346788
- A 7-digit prime cannot end with 2 4 6 or 8
- So the last digit must be 3 or 7
- Write a program to exhaust the permutations

Case 2

- 023444599
- Similarly, the number must end with 3, 5 or 9

Case 3

- 00133458
- Why s_target = 2?
- Because sum of digits = 24
- So if you use all digits form a single number, it must be a multiple or 3

Case 5

- 00000001234444444666666667788888888
- Input has only 4 odd numbers, how to form 5 primes?
- Well, 2 is a prime

Precomputing isPrime[x]

- We can use Sieve of Eratosthenes to precompute isPrime, but it takes ~10 seconds to compute up to 10^9.
- Store the result into a file in binary format
- Your programs can then read the file back into memory. (< 1 second)

Bit	0	1	2	3	4	5	6	7	8	9	10	11
x	5	7	11	13	17	19	23	25	29	31	35	37

- In this way, space required = 10^9 / 8 / 6 * 2 < 40MB

Using the other digits

- We need to use the digits other than 2 3 5 7
- Prioritize utilizing 0 1 4 6 8 9,
  e.g. by forming 641, 809
  because they cannot be eliminated individually
- Solution (20.042 points)
  - Make as many 809 as possible
  - Make as many 641 as possible
  - Use 2 3 5 7 to form primes

Primes with many of the same digit

- Examples are:
  - 100000007, 111111181, 222222227, 333333313, 44444443, 55555553, 666666667, 777767777, 888888883, 999499999
- Solution (35.322 points)
  - Make as many 100000007 as possible
  - Make as many 111111181 as possible
  - .....
  - Use 2 3 5 7 to form primes

The solution can be extended to use more different primes (~60 points)

Randomized solution

- Put all the digits into a string, like
  - 000000111112222333333344455556677777999
- For `len` = 9 down to 1:
  - Repeat 1000 times
    - Shuffle the string
    - Find a substring of length `len` that is a prime
    - Output the prime
    - Splice (remove) the substring from the string
- Expected score: ~80

Intuition:
The primes formed will have similar digit proportions. Thus the algorithm work well for all inputs.

Try again

- It’s a kind of greedy solution.
- The score largely depends on the last few steps
- Given 9 random digits, what’s the probability that it has a permutation that is a prime?
  - < 1/3 because if the digit sum is a multiple of 3, then no permutation would be prime.
  - Otherwise, there’s very high chance that it has a prime permutation
- So 1/3 chance you will get stuck in the last step
- We can avoid this by trying many times (different seed)
- Expected score: 100