Tutorial H - Subregional 2025

Paulo Miranda

September 14, 2025

Solução por Convolução: $O(K \cdot (N + M + 2^K))$

A ideia da solução é modelar o problema como uma Convolução de OR, usando Fast Walsh–Hadamard Transform.

Primeiramente, definimos os seguintes polinômios:

$$F_1(x) = \sum x^{p_i}$$

$$F_2(x) = conv_{or}(F(x), F(x))$$

$$F_3(x) = conv_{or}(F_2(x), F(x))$$

Agora, consideramos o polinômio:

$$F_3(x) = \sum a_k \cdot x^k$$

O coeficiente a_k representa a quantidade de trios (com repetição de elementos permitida) cuja OR (bit a bit) seja k. No entanto, queremos apenas as equipes válidas, ou seja, formadas por **3 alunos distintos**. Portanto, é necessário ajustar essa contagem.

Para isso, usamos outros dois polinômios auxiliares:

$$F_2(x) = \sum b_k \cdot x^k$$

Aqui, o coeficiente b_k representa a quantidade de trios em que **pelo menos dois dos três elementos são iguais** e cuja OR (bit a bit) seja k.

$$F_1(x) = \sum c_k \cdot x^k$$

Neste caso, o coeficiente c_k corresponde à quantidade de trios onde **os três elementos são iguais** e cuja OR (bit a bit) seja k.

Correção das Contagens

Agora que temos as contagens para todos os casos, precisamos isolar as combinações com **3 elementos distintos**. Para isso, aplicamos o princípio da inclusão-exclusão.

- a_k inclui todas as combinações, inclusive aquelas com repetições de elementos.
- b_k conta as combinações com pelo menos 2 elementos iguais.
- c_k conta as combinações com todos os elementos iguais.

A quantidade de trios com 3 elementos distintos é então dada por:

$$resposta_k = \frac{a_k - 3 \cdot b_k + 2 \cdot c_k}{6}$$

A justificativa é a seguinte:

- Subtraímos $3 \cdot b_k$ porque um trio com dois elementos iguais pode ter o elemento distinto em qualquer uma das três posições.
- Porém, ao fazer isso, acabamos subtraindo excessivamente os casos com todos os elementos iguais (que aparecem em c_k), pois eles foram subtraídos 3 vezes e só deveriam ser subtraídos 1 vez. Por isso, somamos de volta $2 \cdot c_k$.
- Por fim, dividimos o resultado por 6 para corrigir as permutações dos elementos ($3! = 6$), já que estamos interessados apenas em conjuntos de alunos distintos, e não em suas ordenações.

Resumo

A estratégia computacional envolve:

1. Construir os polinômios $F_1(x)$, $F_2(x)$ e $F_3(x)$.
2. Precomputar para todos os valores possíveis usando as convoluções, que podem ser feitas de forma eficiente com FWHT.
3. Acessar a resposta para cada número mágico E_i.

Isso permite resolver o problema de maneira eficiente mesmo com restrições grandes.

Complexidade

A complexidade esperada da solução é:

- $O(K \cdot (N + M + 2^K))$, onde K é a quantidade de dígitos, N é o número de alunos e M é a quantidade de números mágicos.

Solução por transformada (FWHT ou SOS-DP): $O(K \cdot (N + M + 2^K))$

Notação

Vamos definir a terminologia para formalizar a solução:

- K: O número total de habilidades distintas.
- N: O número total de alunos.
- Uma **máscara** (mask) é um inteiro de K bits, que representa um subconjunto de habilidades.
- $f[mask]$: A quantidade de alunos cujo conjunto de habilidades é *exatamente* representado por `mask`.
- $F[mask]$: A quantidade de alunos cujo conjunto de habilidades é um *subconjunto* de `mask`. Matematicamente, $F[mask] = \sum_{sub \subseteq mask} f[sub]$.
- $g[mask]$: A resposta final. O número de equipes de 3 alunos distintos cuja união de habilidades é *exatamente* `mask`.
- $G[mask]$: O número de equipes de 3 alunos distintos cuja união de habilidades é um *subconjunto* de `mask`. Matematicamente, $G[mask] = \sum_{sub \subseteq mask} g[sub]$.

Passo 1: Contagem Inicial dos Alunos

Primeiro, processamos a entrada para popular o vetor f. Iteramos pelos N alunos e, para cada aluno com conjunto de habilidades H_i, incrementamos $f[H_i]$.

```
f = array de tamanho 2^K inicializado com zeros
Para cada aluno i de 1 a N:
    mask_aluno = converte a string H_i para inteiro
    f[mask_aluno]++
```

Passo 2: DP Sobre Subconjuntos (SOS DP) - Ida

O próximo passo é calcular $F[mask]$ para todas as 2^K máscaras possíveis. A definição é $F[mask] = \sum_{sub \subseteq mask} f[sub]$. Uma maneira ingênua seria iterar por todas as sub-máscaras para cada máscara, resultando em uma complexidade de $O(3^K)$, o que é inviável para $K = 20$.

Usamos a DP Sobre Subconjuntos, que calcula esses valores em $O(K \cdot 2^K)$. A ideia é iterar sobre cada bit i de 0 a $K-1$ e, para cada máscara, adicionar a contribuição das máscaras que diferem apenas no i-ésimo bit.

```
// Inicialmente, F é uma cópia de f
F = f

// Itera sobre cada bit
Para i de 0 a K-1:
    // Itera sobre todas as máscaras
    Para mask de 0 a (2^K - 1):
        // Se o i-ésimo bit de mask está ligado
        Se (mask >> i) & 1:
            // Adiciona a contribuição da máscara sem o i-ésimo bit
            F[mask] += F[mask ^ (1 << i)]
```

Ao final deste processo, $F[mask]$ conterá o número total de alunos cujas habilidades são um subconjunto de `mask`.

Passo 3: Calculando Combinações de Equipes

Agora, vamos calcular $G[mask]$. Se a união das habilidades de uma equipe de 3 alunos é um subconjunto de `mask`, então cada um dos 3 membros deve, individualmente, possuir um conjunto de habilidades que é um subconjunto de `mask`.

O número de alunos que satisfazem essa condição é exatamente $F[mask]$. Portanto, o número de maneiras de formar uma equipe de 3 pessoas a partir deste grupo de alunos é dado pela combinação de $F[mask]$ escolhe 3.

$$G[mask] = F[mask]3 = \frac{F[mask] \cdot (F[mask] - 1) \cdot (F[mask] - 2)}{6}$$

Calculamos $G[mask]$ para todas as máscaras de 0 a $2^K - 1$.

Passo 4: Invertendo a Transformada (SOS DP - Volta)

Neste ponto, temos $G[mask] = \sum_{sub \subseteq mask} g[sub]$. Nosso objetivo é encontrar $g[mask]$. Para isso, aplicamos o Princípio da Inclusão-Exclusão, que pode ser implementado como a operação inversa da SOS DP.

A relação para obter g é:

$$g[mask] = G[mask] - \sum_{sub \subset mask} g[sub]$$

Isso pode ser implementado de forma eficiente com uma estrutura de loop semelhante à da ida, mas com subtração.

```
// Inicialmente, g é uma cópia de G
g = G

// Itera sobre cada bit
Para i de 0 a K-1:
    // Itera sobre todas as máscaras
    Para mask de 0 a (2^K - 1):
        // Se o i-ésimo bit de mask está ligado
        Se (mask >> i) & 1:
            // Remove a contribuição das sub-máscaras
            g[mask] -= g[mask ^ (1 << i)]
```

Após a execução deste algoritmo, o vetor g conterá a resposta para cada possível subconjunto mágico.

Finalização

Para cada uma das M strings de consulta E_i, convertemos a string para sua representação inteira `mask_magica` e imprimimos o valor de $g[mask\_magica]$.

Complexidade Total

- **Passo 1 (Contagem):** $O(N \cdot K)$
- **Passo 2 (SOS DP - Ida):** $O(K \cdot 2^K)$
- **Passo 3 (Cálculo de G):** $O(2^K)$
- **Passo 4 (SOS DP - Volta):** $O(K \cdot 2^K)$
- **Consultas:** $O(M \cdot K)$

A complexidade dominante é $O(K \cdot (N + M + 2^K))$. Com $N = 10^5, K = 20, M = 5 \cdot 10^4$, esta abordagem é eficiente o suficiente para ser aceita.

Bonus: FWHT

Além do método SOS-DP, a etapa de soma sobre subconjuntos também pode ser implementada de forma equivalente usando a Fast Walsh-Hadamard Transform (FWHT). A FWHT é uma transformada discreta que, quando aplicada com a operação de OR, computa exatamente a soma sobre subconjuntos. A transformada inversa da FWHT realiza a operação oposta, que é o que precisamos para o último passo. Ou seja, podemos resolver da mesma forma usando FWHT.