Tópicos Especiais em Algoritmos
Estruturas de Dados Associativas e Vetores de Bits
Editorial

Daniel Saad Nogueira Nunes

AtCoder ABC206C: Swappable

Uma forma de resolver este problema é utilizar um mapeamento *count* que armazenará, para cada número v_i da entrada (chave), o número de vezes que ele ocorre (valor).

Após isto, basta percorrer o mapeamento, basta, para cada elemento k do mapeamento, acumular o resultado de $count[k] \cdot (n - count[k])$.

O único cuidado que precisa-se ter é remover os pares duplicados da conta, bastando dividir o resultado acumulado por 2.

Complexidade

Como cada operação sobre o mapeamento leva tempo $O(\lg n)$, temos que a complexidade total do algoritmo, no pior caso, é $O(n \lg n)$, sendo n o tamanho da entrada.

Codeforces 1598C: Delete Two Elements

Seja $A = (a_1, \ldots, a_n)$ a sequência de entrada. A média matemática é definida como:

$$mean = \frac{1}{n} \cdot \sum_{i=1}^{n} A[i]$$

Temos que achar todos os pares ($A[k]$,$A[l]$), $k < l$, tal que

$$mean' = \frac{1}{n-2} \left(\sum_{i=1}^{n} A[i] - A[k] - A[l] \right) = mean$$

Resolvendo $mean = mean'$, temos:

$$(n-2) \cdot \sum_{i=1}^{n} A[i] = n \cdot \left(\sum_{i=1}^{n} A[i] - A[k] - A[l] \right)$$
$$(n-2) \cdot \sum_{i=1}^{n} A[i] = n \cdot \sum_{i=1}^{n} A[i] + n \cdot (-A[k] - A[l])$$
$$-2 \cdot \sum_{i=1}^{n} A[i] = n \cdot (-A[k] - A[l])$$
$$A[k] + A[l] = \frac{2 \cdot \sum_{i=1}^{n} A[i]}{n}$$

Seja $sum = \frac{2 \cdot \sum_{i=1}^{n} A[i]}{n}$, temos que encontrar todos os pares ($A[k]$,$A[l]$), $k < l$, tal que $A[k] + A[l] = sum$.

Se $2 \cdot \sum_{i=1}^{n} A[i]$ não é divisível por n, não temos como formar pares que atendam esta condição, visto que os elementos de A são inteiros, logo a resposta é 0.

Caso contrário, podemos contar a quantidade de vezes que cada elemento $A[i]$ ocorre através de um mapeamento map e usar o mapeamento para obter a resposta. Para cada elemento (k, v) do mapeamento, em que k é a chave e v o valor, isto é, a quantidade de vezes que k ocorre, basta verificar a quantidade de vezes que $sum - v$ ocorre, e isto está em $map[sum - v]$. Com posse desta informação, basta somar a uma variável acumuladora $total$ o resultado de $v \cdot map[sum - v]$.

Antes de finalizar, precisamos de alguns ajustes:

- Caso exista algum elemento $A[i]$ que seja igual a $\frac{sum}{2}$, então $A[i]$ pode parear com ele próprio, desta forma, temos que descontar $map[A[i]]$ da solução.
- Temos que dividir a quantidade de pares por 2, visto que se (a_i, a_j) foi contabilizado, (a_j, a_i) também foi, e queremos somente os pares em que $i < j$.

Complexidade

A complexidade da solução tem tempo $O(n \lg n)$, uma vez que as operações sobre mapeamentos levam tempo $O(\lg n)$.

UVA 10264: The Most Potent Corner

Se n é a dimensão do cubo, ele tem 2^n vértices, com cada vértice v_i podendo ser representado por uma sequência $b_i = i$ de n bits. Outra observação que pode ser feita é que os vértices adjacentes a v_i são os vértices v_j cuja representação b_j difere de b_i por apenas 1 bit.

Podemos, primeiramente, pré-computar a potência de cada vértice ao fixar um vértice v_i e gerar todos os seus adjacentes v_j ao realizar uma operação de xor bit-a-bit de i com a máscara `1<< j`.

Com a potência de cada vértice calculada, basta agora gerar todos os pares de vértices adjacentes, que pode ser feito usando a mesma estratégia anterior através das operações bit a bit, e selecionar o par cuja soma de potências é máxima.

Complexidade

Pré-computar a potência dos vértices e gerar todos os pares adjacentes para selecionar aqueles de maior potência gasta tempo $O(n \cdot 2^n)$. Como $n \leq 15$ isso não é um problema.

Detalhes de Implementação

O fim da entrada deve ser detectado ao ler `EOF`. Em `C++` basta testar `while(cin>>n){...}`, enquanto em `C`, podemos fazer `while(scanf("%d",&n)){...}`.

AtCoder ABC216B: Same Name

Para resolver este problema, podemos usar um conjunto para armazenar os pares $(family\_name, given\_name)$. Para cada par lido, basta verificar se o mesmo já se encontra no conjunto para responder de maneira afirmativa. Caso não seja o caso para todos os pares lidos, a resposta é negativa.

Complexidade

A complexidade da solução tem tempo $O(n \lg n)$, em que n é número de pares dados na entrada, visto que, operações sobre conjuntos levam tempo $O(\lg n)$. Para esta análise, estamos assumindo que o tamanho de cada string é constante.