Meet-in-the-middle
Professor Wladimir

Problema SUBSUMS - Subset Sums

Dado um conjunto $S = \{S_1, S_2, \ldots, S_N\}$ de N números inteiros, onde $1 \leq N \leq 34$ e $-20{,}000{,}000 \leq S_i \leq 20{,}000{,}000$, deseja-se determinar quantos subconjuntos de S (incluindo o conjunto vazio) possuem uma soma total S_{sub} tal que:

$$A \leq S_{sub} \leq B,$$

onde A e B são inteiros que satisfazem $-500{,}000{,}000 \leq A \leq B \leq 500{,}000{,}000$.

Em outras palavras, o problema consiste em contar o número de subconjuntos de S cuja soma está dentro do intervalo $[A, B]$.

Referência: https://www.spoj.com/problems/SUBSUMS

Soma de todos subconjuntos

A função `subconjuntos` recebe um vetor de inteiros v e gera todos os possíveis subconjuntos desse vetor, armazenando as somas correspondentes no vetor `subsets`.

Para gerar todos os subconjuntos de um conjunto S de tamanho n, utilizamos uma estratégia eficiente baseada na **representação binária de números inteiros**. Cada número inteiro entre 0 e $2^n - 1$ pode ser interpretado como um subconjunto de S, onde cada *bit* indica a presença ou ausência de um elemento.

Mais precisamente, o j-ésimo bit do número indica se o elemento $v[j]$ pertence ao subconjunto correspondente. Assim, o número 0 representa o conjunto vazio, enquanto o número $2^n - 1$ representa o conjunto completo.

Por exemplo, considere o conjunto $S = \{1, 2, 3\}$. Os números de 0 a $2^3 - 1 = 7$ correspondem a todos os subconjuntos possíveis de S, conforme ilustrado na Tabela 1.

Número	Representação Binária	Subconjunto
0	000	$\emptyset$
1	001	$\{1\}$
2	010	$\{2\}$
3	011	$\{1, 2\}$
4	100	$\{3\}$
5	101	$\{1, 3\}$
6	110	$\{2, 3\}$
7	111	$\{1, 2, 3\}$

Tabela 1: Representação binária dos subconjuntos de $S = \{1, 2, 3\}$.

```
void subconjuntos(vector<int> &v, vector<int> &subsets){
    int n = v.size();
    for (int mask = 0; mask < (1 << n); mask++) {
        long long int soma = 0;
        for (int j = 0; j < n; j++) {
            if (mask & (1 << j))
                soma += v[j];
        }
        subsets.push_back(soma);
    }
}
```

Solução do problema

A função `subconjuntos` percorre todos os 2^n subconjuntos possíveis e, para cada um deles, realiza uma soma envolvendo até n elementos. Assim, a complexidade dessa etapa é:

$$O(n \cdot 2^n)$$

Após a geração de todas as somas dos subconjuntos, o vetor resultante é ordenado para permitir a contagem eficiente da quantidade de subconjuntos cuja soma pertence ao intervalo $[A, B]$.

A complexidade da ordenação do vetor de soma de tamanho 2^n é dada por:

$$O(2^n \log(2^n)) = O(2^n \cdot n).$$

A ordenação possibilita o uso das funções `lower_bound` e `upper_bound`, que realizam buscas binárias para localizar os limites inferior e superior do intervalo de interesse, reduzindo o custo da contagem para $O(n)$.

Portanto, a complexidade total do algoritmo é dominada pela geração dos subconjuntos:

$$O(n \cdot 2^n)$$

```cpp
vector <int> sub1;
    subconjuntos(v1, sub1);
    sort(sub1.begin(), sub1.end() );
    auto low = lower_bound( sub1.begin(), sub1.end(), a);
    auto up  = upper_bound( sub1.begin(), sub1.end(), b);
    long long int cont = up-low;
    cout << cont;
```

Embora o limite de $n \leq 34$ pareça relativamente pequeno, a abordagem que gera todos os 2^n subconjuntos é computacionalmente inviável tanto em tempo quanto em memória. Isso ocorre porque a complexidade espacial dessa abordagem é da ordem de $O(2^n)$

Para ilustrar, quando $n = 34$, o número de subconjuntos é:

$$2^{34},$$

Cada soma de subconjuntos é armazenada em uma variável do tipo `long long int`) que ocupa 8 bytes. Logo, o total de bytes será $2^{34} * 2^3 = 2^{37}$ (aproximadamente 128 GB).

Esse valor ultrapassa em muito o limite de memória disponível no problema (1.5 GB), tornando a solução inviável.

Para contornar essa limitação, utiliza-se a técnica conhecida como *meet in the middle*, que reduz a complexidade de memória para aproximadamente:

$$O(2^{n/2}).$$

Essa técnica consiste em dividir o conjunto S em duas partes de tamanhos próximos, gerar todas as somas possíveis dos subconjuntos de cada metade separadamente e, em seguida, combinar os resultados de forma eficiente. Dessa forma, é possível resolver o problema de forma viável dentro dos limites de tempo (1 segundo) e memória (1536 MB) especificados.

Meet-in-the-Middle

Dividindo o vetor em duas partes

Lê-se o valor de n, os limites a e b, e o vetor de n inteiros. O vetor é dividido em duas partes: a primeira com $s_1 = \lfloor n/2 \rfloor$ elementos e a segunda com $s_2 = n - s_1$.

```
cin >> n >> a >> b;
vector <int> v1;
vector <int> v2;
v1.resize(n/2);
v2.resize(n-n/2);
for(auto & x : v1)
    cin >> x;
for(auto & x : v2)
    cin >> x;
```

Solução

```
subconjuntos(v1, sub1);
subconjuntos(v2, sub2);
sort(sub2.begin(), sub2.end() );
long long int cont = 0;
for(auto & x : sub1){
    auto low = lower_bound( sub2.begin(), sub2.end(), a-x);
    auto up  = upper_bound( sub2.begin(), sub2.end(), b-x);
    long long int qtd = up-low;
    cont += qtd;
}
```

Para cada metade, a soma de todos subconjuntos são gerados utilizando a função subconjuntos: As somas são armazenadas em vetores $sub1$ e $sub2$. Em seguida, o vetor $sub2$ é ordenado para a utilização da busca binária.

Para cada soma $s_1 \in sub1$, deseja-se encontrar o número de somas $sub2 \in v_2$ que satisfaçam:

$$a \leq s_1 + s_2 \leq b \quad \iff \quad a - s_1 \leq s_2 \leq b - s_1.$$

Para isso, o programa utiliza:

- `lower_bound(v2.begin(), v2.end(), a - s_1)`
  que retorna o primeiro índice em v_2 com valor $\geq a - s_1$.
- `upper_bound(v2.begin(), v2.end(), b - s_1)`
  que retorna o primeiro índice em v_2 com valor $> b - s_1$.

O número de somas válidas para aquele s_1 é:

$$\text{count} += \text{upper} - \text{lower}.$$

A soma total acumulada é impressa, representando o número de subconjuntos cuja soma está no intervalo $[a, b]$.

Complexidade

A complexidade temporal é:

$$O\big(n/2 \cdot 2^{n/2}\big),$$

tanto para a ordenação quanto para as buscas binárias.

Exercícios

1. 888E - Maximum Subsequence (Codeforces)

   Dado um vetor a contendo $n \leq 35$ números inteiros e um inteiro m, o objetivo é escolher uma subsequência de índices

   $$b_1, b_2, \ldots, b_k \quad \text{com} \quad 1 \leq b_1 < b_2 < \cdots < b_k \leq n$$

   de forma que o valor de uma função associada à subsequência seja maximizado.

   A subsequência escolhida pode ser vazia. O problema consiste em determinar o **valor máximo possível** da função considerando todas as subsequências válidas.

   Em outras palavras, devemos selecionar alguns elementos S do vetor para maximizar $\sum_{i \in S} a_i \mod m$, levando em conta o inteiro m.

   https://codeforces.com/problemset/problem/888/E

Resumo das Habilidades exercitadas

Habilidade	**Como é treinada**
Bitmasking	Geração de todos os subconjuntos de cada metade.
Meet in the middle	Divisão do vetor e combinação de resultados parciais.
Busca binária	Contagem eficiente de intervalos em vetor ordenado.
Análise de complexidade	Redução de $O(2^n)$ para $O(2^{n/2} \cdot n)$.

Tabela 2: Habilidades treinadas no problema de SubSums com bitmasking e meet in the middle.