CHỌN TỔNG

Cho dãy số nguyên dương $A = (a_1, a_2, \dots, a_n)$ hãy chỉ ra một dãy con của dãy A có tổng bằng m.

Dữ liệu: Vào từ file văn bản SUBSETSUM.INP

- Dòng 1 chứa hai số nguyên dương $n \le 10^5; m \le 10^5$
- Dòng 2 chứa n số nguyên dương $a_1, a_2, \dots, a_n$ $(\sum_{i=1}^{n} a_i \le 10^5)$

Kết quả: Ghi ra file văn bản SUBSETSUM.OUT

- Dòng 1 ghi từ YES nếu tồn tại dãy con của A có tổng bằng m, ngược lại ghi từ NO
- Nếu dòng 1 ghi từ YES, dòng 2 ghi chỉ số các phần tử được chọn theo thứ tự tăng dần.

Ví dụ

SUBSETSUM.INP	SUBSETSUM.OUT
6 99 11 44 33 55 77 88	YES 1 3 4

Thuật toán

Cho dãy $A = (a_1, a_2, \dots, a_n)$. Cần chọn một dãy con có tổng bằng m.

Với một số nguyên không âm $x \le m$.

Gọi $f[x]$ là chỉ số i nhỏ nhất thỏa mãn: Có thể chọn trong i phần tử đầu của dãy A: $(a_1, a_2, \dots, a_i)$ ra một dãy con có tổng bằng x. Nếu không tồn tại chỉ số i như vậy ta coi như $f[x] = +\infty$.

Ví dụ với $n = 4$. Dãy A cho như sau

i	1	2	3	4
a_i	8	2	1	3

Mảng f sẽ là

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
$f[x]$	0	3	2	3	4	4	4	$+\infty$	1	3	2	3	4	4	4	$+\infty$

Trước hết ta thấy rằng $f[0] = 0$ theo định nghĩa. Vì ta có thể chọn trong 0 phần tử đầu của dãy A (dãy $\emptyset$) ra một dãy con (chính nó) có tổng bằng 0.

Ta xây dựng công thức tính $f[x]$ trong điều kiện đã biết $f[0], f[1], \dots, f[x-1]$.

Nếu $f[x] = i$ thì khi chọn một dãy con của $(a_1, a_2, \dots, a_i)$ để được tổng bằng x, dĩ nhiên ta phải chọn a_i (nếu không thì vi phạm định nghĩa $f[x] = i$ nhỏ nhất thỏa mãn…) và như vậy a_i phải $\le x$.

Ngoài ra khi bỏ a_i ra khỏi dãy các phần tử được chọn, ta sẽ thu được một dãy khác có tổng bằng $x - a_i$ và dãy này phải là dãy con của $(a_1, a_2, \dots, a_{i-1})$. Nói cách khác, ta phải có cách chọn một dãy con của $(a_1, a_2, \dots, a_{i-1})$ để được tổng bằng $x - a_i$. Tiêu chuẩn này tức là $f[x - a_i]$ tồn tại và nhỏ hơn i. Ta chỉ cần kiểm tra bằng điều kiện $f[x - a_i] < i$ do tiêu chuẩn $f[x - a_i]$ tồn tại nghĩa là $f[x - a_i] < +\infty$ được bao hàm trong điều kiện $f[x - a_i] < i$ rồi.

Vậy cách tính $f[x]$ là tìm chỉ số i nhỏ nhất thỏa mãn: $a_i \le x$ && $f[x - a_i] < i$ rồi gán $f[x] = i$.

```
f[0] = 0;
for (x = 1; x <= m; ++x)
{   //Tính f[x];
    f[x] = n + 1; //+∞;
    for (i = 1; i <= n; ++i)
        if (a[i] <= x && f[x - a[i]] < i) //Thấy i đầu tiên thỏa mãn
        {
            f[x] = i; break; // gán luôn cho f[x] và dừng
        }
}
```

Khi tính xong mảng f, để chỉ ra cách chọn cho tổng bằng m, ta bắt đầu với phần tử $a[i]$ với $i = f[m]$. Phần tử $a[i]$ này chắc chắn phải chọn.

Chọn xong phần tử này thì lặp lại để chỉ ra cách chọn cho tổng bằng $m -= a[i]$, cứ lặp đến khi $m = 0$ thì không cần chọn tiếp nữa.

```
while (m > 0)
{
    i = f[m];
    «Thông báo chọn a[i]»;
    m -= a[i];
}
```

Độ phức tạp tính toán của thuật toán này là $O(m \times n)$. Tuy là thuật toán đúng nhưng chưa đạt yêu cầu về tốc độ.

Gọi S là tổng các phần tử trong dãy a. Một nhận xét nhỏ cho phép cải tiến độ phức tạp tính toán xuống còn $O(m\sqrt{S})$: Nếu trong dãy a có 3 phần tử giống nhau, chẳng hạn 3 số v, v, v, khi đó có thể thay 2 số v bởi một số có giá trị $2v$ (còn 2 phần tử $v, 2v$) mà không ảnh hưởng đến việc chọn được/không chọn được tổng bằng m.

Bằng cách thay thế như vậy, không có một giá trị nào xuất hiện > 2 lần trong dãy a. Ta sẽ chỉ ra rằng số phần tử của dãy a bây giờ là một đại lượng $O(\sqrt{S})$.

Thật vậy, nếu sắp xếp tăng dần dãy A: $a_1 \leq a_2 \leq \cdots \leq a_n$. Vì dãy A nguyên dương và không có giá trị nào xuất hiện > 2 lần nên

$a_1, a_2 \geq 1$;

$a_3, a_4 \geq 2$;

…

Tức là $a_i \geq \frac{i}{2}$ $(\forall i = 1,2, \dots, n)$

Vậy

$$S = \sum_{i=1}^{n} a_i \geq \sum_{i=1}^{n} \frac{i}{2} = \frac{n(n+1)}{4}$$

Vậy $n < \sqrt{4S}$ hay $n < 2\sqrt{S}$

Để chỉ ra cách chọn, từ dãy A ban đầu, ta xây dựng các danh sách $pos[1 \dots S]$: Trong đó $pos[v]$ chứa những vị trí xuất hiện của giá trị v trong dãy A. Căn cứ vào thuật toán chọn ở trên, mỗi khi cần chọn 1 giá trị v:

- Nếu $pos[v] \neq \emptyset$, ta chọn một chỉ số bất kỳ trong danh sách $L[v]$ và xóa chỉ số đó khỏi $L[v]$
- Nếu $pos[v] = \emptyset$, ta quy về việc chọn 02 giá trị $v/2$ bằng cách tương tự.

Độ phức tạp của phép truy vết vẫn là $O(n)$ nếu cài đặt $L[v]$ là kiểu danh sách cho phép xóa một phần tử tự chọn trong thời gian $O(1)$: vector, list, stack, queue, ...

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;

const int maxN = 1e5 + 1;
const int maxM = 1e5;

int n, m, k;
int a[maxN], b[maxN], f[maxM + 1];
vector<int> pos[maxM];
bool Selected[maxN];


void ReadInput()
{
    cin >> n >> m;
    fill(f + 1, f + maxM + 1, 0);
    for (int i = 1; i <= n; ++i)
    {
        cin >> a[i];
        ++f[a[i]];
        pos[a[i]].push_back(i);
    }
    k = 0;
    for (int v = 1; v <= maxM; ++v)
    {
        if (f[v] > 2)
        {
            int carry = (f[v] - 1) / 2;
            f[v * 2] += carry;
            f[v] -= 2 * carry;
        }
        while (f[v]-- > 0)
            b[++k] = v;
    }
}

void Solve()
{
    f[0] = 0;
    for (int x = 1; x <= m; ++x)
    {
        f[x] = k + 1;
        for (int i = 1; i <= k; ++i)
        {
            if (b[i] > x) break;
            if (f[x - b[i]] < i)
            {
                f[x] = i;
                break;
            }
        }
    }
}

```

```
void DoSelect(int Value, int Count)
{
    while (Count > 0)
        if (!pos[Value].empty())
        {
            Selected[pos[Value].back()] = true;
            --Count;
            pos[Value].pop_back();
        }
        else
        {
            Value /= 2;
            Count *= 2;
        }
}

void Print()
{
    if (f[m] > k)
    {
        cout << "NO";
        return;
    }
    cout << "YES\n";
    fill(Selected + 1, Selected + n + 1, false);
    while (m > 0)
    {
        int i = f[m];
        DoSelect(b[i], 1);
        m -= b[i];
    }
    for (int i = 1; i <= n; ++i)
        if (Selected[i]) cout << i << ' ';
}

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(nullptr);
    ReadInput();
    Solve();
    Print();
}
```