# Alchemy

Keza is magically creating materials in her workshop. She has $N$ types of materials, numbered $0$ to $N-1$. For each material $i$ ($0 \leq i < N$), she initially has $A[i]$ units of it.

Keza can magically create some number of materials (possibly zero) using recipes. For each material $i$, she can use one unit of material $i$ to produce one unit of each of the materials specified in the array $B[i]$.

She repeats the following sequence of operations a total of $T$ times:

- For each material $i$ from $0$ to $N-1$ in increasing order: if she currently has at least one unit of material $i$, she immediately consumes one unit of it to produce one unit of each of the materials specified in $B[i]$ (which become available immediately).

So, in total, she is going to perform $T \cdot N$ operations.

Your task is to determine the final amount of each material after Keza is finished performing the operations.

## Implementation Details

### C++/Python Interface

For C++/Python submisisons, you should implement the following procedure:

```
int64[] transform_materials(int32 N, int64 T, int64[] A, int32[][] B)
```

- `N`: the number of materials.
- `T`: the number of iterations.
- `A`: an array of length $N$ specifying the initial amount of each material.
- `B`: an array of length $N$ of arrays, where `B[i]` contains the materials produced by transforming one unit of material $i$.
- The procedure should return an array of length $N$, where the $i$-th element is the final amount of material $i$ after $T$ iterations.
- This procedure is called exactly once for each testcase.

### C Interface

For C submissions, include the provided header `alchemy_c.h` and implement the following function:

```
void transform_materials(int N, long long T, long long A[],
    int B_sizes[], int *B[], long long result[]);
```

- `B_sizes`: an array of length $N$, where `B_sizes[i]` is the number of entries in `B[i]`.
- `B`: an array of $N$ pointers. For each $i$, if `B_sizes[i]` is positive, `B[i]` points to an array of that length containing the materials produced by transforming one unit of material $i$. If `B_sizes[i]` is $0$, `B[i]` may be `NULL`.
- `result`: an array of length $N$ provided by the grader. The function should write the final amount of material $i$ to `result[i]` for every $0 \le i < N$.
- The function returns `void`; the answer is communicated through `result`.
- The grader owns `A`, `B_sizes`, `B`, every array pointed to by `B[i]`, and `result`. Your function may modify these buffers, but it must not free any of them.

## Constraints

- $1 \le N \le 200\,000$
- $1 \le T \le 10^{12}$
- $1 \le A[i] \le 10^9$ for each $i$ such that $0 \le i < N$.
- $0 \le B[i][j] < N$ for each $i, j$ such that $0 \le i < N$ and $0 \le j <$ length of $B[i]$.
- The sum of the lengths of $B[i]$ over all $0 \le i < N$ does not exceed $200\,000$.

## Subtasks

| Subtask | Score | Additional Constraints |
|---|---|---|
| 1 | 10 | $B[i][j] = i$ for $0 \le i < N$ and for $0 \le j <$ length of $B[i]$. |
| 2 | 13 | $B[i][j] > i$ for $0 \le i < N$ and for $0 \le j <$ length of $B[i]$. |
| 3 | 15 | $N \le 2000$, $T \le 2000$, and the sum of the lengths of $B[i]$ over all $0 \le i < N$ does not exceed $2000$. |
| 4 | 23 | $T \le 100\,000$. |
| 5 | 39 | No additional constraints. |

# Examples

## Example 1

Consider the following call:

```
transform_materials(2, 3, [100, 50], [[], [0, 0]])
```

In this example:

- Material $0$ is consumed to produce nothing (the length of $B[0]$ is $0$).
- Material $1$ is consumed to produce two units of material $0$ ($B[1] = [0, 0]$).

The following table shows the amount of each material after $T$ iterations:

| $T$ | $0$ | $1$ | $2$ | $3$ |
|---|---|---|---|---|
| $A[0]$ | $100$ | $101$ | $102$ | $103$ |
| $A[1]$ | $50$ | $49$ | $48$ | $47$ |

The procedure should return $[103, 47]$.

## Example 2

Consider the following call:

```
transform_materials(2, 53, [100, 50], [[], [0, 0]])
```

The configuration and recipes are identical to Example 1. At iteration $T = 50$, material $1$ runs out. After this, only material $0$ is transformed. Since transforming $1$ unit of material $0$ produces nothing, the amount of material $0$ decreases by $1$ in each subsequent iteration. Thus, after $T = 53$ iterations, the amount of material $0$ is $147$.

The procedure should return $[147, 0]$.

---

## Example 3

Consider the following call:

```
transform_materials(2, 1000000000000, [100, 50], [[0, 0], [0, 0]])
```

In this example, both materials are consumed to produce two units of material $0$. After $50$ iterations, material $1$ runs out, and only material $0$ is transformed. Since transforming $1$ unit of material $0$ produces $2$ units of material $0$, the net amount of material $0$ increases by $1$ in each subsequent iteration.

The procedure should return: $[1000000000200, 0]$.

## Sample Grader

Input format:

```
N T
A[0] A[1] ... A[N-1]
L[0] B[0][0] B[0][1] ... B[0][L[0]-1]
L[1] B[1][0] B[1][1] ... B[1][L[1]-1]
...
L[N-1] B[N-1][0] B[N-1][1] ... B[N-1][L[N-1]-1]
```

Here, $L[i]$ is the length of $B[i]$.

Output format:

```
R[0]
R[1]
...
R[N-1]
```

Here, $R[i]$ is the array returned by `transform_materials`.