Brute Force, Prefix and Difference Arrays, Binary Search

Ieuan David Vinluan

August 2024

Brute Force

- The one strategy that works for all problems

Brute Force

- The one strategy that works for all problems
- Straightforward: just try all possibilities and see which ones work!

Brute Force

- The one strategy that works for all problems
- Straightforward: just try all possibilities and see which ones work!
- 100% sure we're correct because there's no other possibility we haven't checked

Brute Force

- The one strategy that works for all problems
- Straightforward: just try all possibilities and see which ones work!
- 100% sure we're correct because there's no other possibility we haven't checked
- Complete search

Example: Codeforces 214A

A. System of Equations

time limit per test: 2 seconds
memory limit per test: 256 megabytes

Furik loves math lessons very much, so he doesn't attend them, unlike Rubik. But now Furik wants to get a good mark for math. For that Ms. Ivanova, his math teacher, gave him a new task. Furik solved the task immediately. Can you?

You are given a system of equations:

$$\begin{cases} a^2 + b = n \\ a + b^2 = m \end{cases}$$

You should count, how many there are pairs of integers (a, b) $(0 \le a, b)$ which satisfy the system.

Input
A single line contains two integers n, m $(1 \le n, m \le 1000)$ — the parameters of the system. The numbers on the line are separated by a space.

Output
On a single line print the answer to the problem.

Brute Force

- You can probably try some fancy solution involving factoring

Brute Force

- You can probably try some fancy solution involving factoring
- However, the easiest solution to code would be one that just tries all possible (a, b)

Brute Force

- You can probably try some fancy solution involving factoring
- However, the easiest solution to code would be one that just tries all possible (a, b)

```
int n, m, ans = 0;
cin >> n >> m;
for (int i = 0; i <= n; i++) {
    if (i * i > n) break;
    for (int j = 0; j <= m; j++) {
        if (j * j > m) break;
        ans += (i * i + j == n and i + j * j == m);
    }
}
cout << ans << endl;
```

Another example: Codeforces 798B

B. Mike and strings

time limit per test: 2 seconds
memory limit per test: 256 megabytes

Mike has n strings $s_1, s_2, \ldots, s_n$ each consisting of lowercase English letters. In one move he can choose a string s_i, erase the first character and append it to the end of the string. For example, if he has the string "`coolmike`", in one move he can transform it into the string "`oolmikec`".

Now Mike asks himself: what is minimal number of moves that he needs to do in order to make all the strings equal?

Input
The first line contains integer n $(1 \le n \le 50)$ — the number of strings.

This is followed by n lines which contain a string each. The i-th line corresponding to string s_i. Lengths of strings are equal. Lengths of each string is positive and don't exceed 50.

Output
Print the minimal number of moves Mike needs in order to make all the strings equal or print -1 if there is no solution.

Brute Force

- Again, no need to figure out anything crazy!

Brute Force

- Again, no need to figure out anything crazy!
- Try using all of the strings as a goal

Brute Force

- Again, no need to figure out anything crazy!
- Try using all of the strings as a goal
- For each string, find the number of moves needed to make all other strings equal to it

Brute Force

```
int ans = 1e9;
for (int i = 0; i < n and ans != -1; i++) {
    string goal = s[i];
    int cur = 0;
    for (int j = 0; j < n; j++) {
        if (i == j) continue;
        int index = (s[j] + s[j]).find(goal);
        if (index == -1) {
            ans = -1;
        } else {
            cur += index;
        }
    }
    ans = min(ans, cur);
}
cout << ans << endl;
```

Brute Force

More notes:

Brute Force

More notes:

- Of course, we know that brute force solutions are usually too slow for most problems, and we'll have to use some other technique to solve it (e.g., dynamic programming).

Brute Force

More notes:

- Of course, we know that brute force solutions are usually too slow for most problems, and we'll have to use some other technique to solve it (e.g., dynamic programming).
- It's still important to know how to brute force problems though. Some competitions will give you partial points for brute force solutions that aren't completely correct (like NOI), while others will punish you for doing so (DISCS-PRO, CF, AtCoder).

More notes:

- Of course, we know that brute force solutions are usually too slow for most problems, and we'll have to use some other technique to solve it (e.g., dynamic programming).
- It's still important to know how to brute force problems though. Some competitions will give you partial points for brute force solutions that aren't completely correct (like NOI), while others will punish you for doing so (DISCS-PRO, CF, AtCoder).
- With enough practice, this will be easy :3

Prefix Arrays

Given a binary operation $\oplus$ and an array of integers $A = \{a_1, a_2, \ldots, a_n\}$, we can precompute the prefix array for $(a_1 \oplus a_2)$, $(a_1 \oplus a_2 \oplus a_3)$, and so on until $(a_1 \oplus a_2 \oplus \ldots \oplus a_n)$ with the following code:

Prefix Arrays

Given a binary operation $\oplus$ and an array of integers $A = \{a_1, a_2, \ldots, a_n\}$, we can precompute the prefix array for $(a_1 \oplus a_2)$, $(a_1 \oplus a_2 \oplus a_3)$, and so on until $(a_1 \oplus a_2 \oplus \ldots \oplus a_n)$ with the following code:

```
vector<int> pre(n);
pre[0] = A[0];
for (int i = 1; i < n; i++) {
    // do operation with pre[i - 1] and A[i]
    pre[i] = operation(pre[i - 1], A[i]);
}
```

Prefix Arrays

For example, if we want to precompute the sum of the elements from index 0 to i:

Prefix Arrays

For example, if we want to precompute the sum of the elements from index 0 to i:

```
vector<int> pre(n);
pre[0] = A[0];
for (int i = 1; i < n; i++) {
    pre[i] = pre[i - 1] + A[i];
}
```

Or, if we want to precompute the GCD of all elements from index 0 to i:

Prefix Arrays

Or, if we want to precompute the GCD of all elements from index 0 to i:

```
vector<int> pre(n);
pre[0] = A[0];
for (int i = 1; i < n; i++) {
    pre[i] = gcd(pre[i - 1], A[i]);
}
```

Example: AtCoder Beginner Contest 125 C

Problem Statement

There are N integers, $A_1, A_2, ..., A_N$, written on the blackboard.

You will choose one of them and replace it with an integer of your choice between 1 and 10^9 (inclusive), possibly the same as the integer originally written.

Find the maximum possible greatest common divisor of the N integers on the blackboard after your move.

Constraints

- All values in input are integers.
- $2 \leq N \leq 10^5$
- $1 \leq A_i \leq 10^9$

Output

Input is given from Standard Input in the following format:

N
$A_1 \; A_2 \; ... \; A_N$

Output

Print the maximum possible greatest common divisor of the N integers on the blackboard after your move.

Prefix Arrays

How can one go about this problem?

How can one go about this problem?

- Idea: get the GCD of $n - 1$ elements

Prefix Arrays

How can one go about this problem?

- Idea: get the GCD of $n - 1$ elements
- Replace the one remaining element with the GCD of those $n - 1$ elements

Prefix Arrays

How can one go about this problem?

- Idea: get the GCD of $n-1$ elements
- Replace the one remaining element with the GCD of those $n-1$ elements
- The answer is the maximum GCD of a certain subset of $n-1$ elements

The most obvious way to code this would be to just use brute force:

Prefix Arrays

The most obvious way to code this would be to just use brute force:

```
int N, ans = 1;
cin >> N;
vector<int> A(N);
for (int i = 0; i < N; i++) cin >> A[i];
for (int i = 0; i < N; i++) {
    int current = 0;
    for (int j = 0; j < N; j++) {
        if (i == j) continue;
        current = gcd(current, A[j]);
    }
    ans = max(ans, current);
}
cout << ans << endl;
```

Time complexity: $O(N^2)$

But, considering that we can evaluate the expression:

Prefix Arrays

But, considering that we can evaluate the expression:

$$\gcd(A_1,\ A_2,\ \ldots,\ A_k,\ A_{k+2},\ A_{k+3},\ \ldots,\ A_n)$$

Prefix Arrays

But, considering that we can evaluate the expression:

$$\gcd(A_1,\ A_2,\ \ldots,\ A_k,\ A_{k+2},\ A_{k+3},\ \ldots,\ A_n)$$

By just getting the GCD of the GCDs of the separate continuous subsections:

Prefix Arrays

But, considering that we can evaluate the expression:

$$\gcd(A_1,\ A_2,\ \ldots,\ A_k,\ A_{k+2},\ A_{k+3},\ \ldots,\ A_n)$$

By just getting the GCD of the GCDs of the separate continuous subsections:

$$\gcd(\gcd(A_1,\ A_2,\ \ldots,\ A_k),\ \gcd(A_{k+2},\ A_{k+3},\ \ldots,\ A_n))$$

Prefix Arrays

But, considering that we can evaluate the expression:

$$\gcd(A_1,\ A_2,\ \ldots,\ A_k,\ A_{k+2},\ A_{k+3},\ \ldots,\ A_n)$$

By just getting the GCD of the GCDs of the separate continuous subsections:

$$\gcd(\gcd(A_1,\ A_2,\ \ldots,\ A_k),\ \gcd(A_{k+2},\ A_{k+3},\ \ldots,\ A_n))$$

We can use prefix arrays!

Prefix Arrays

We can implement this as follows:

Prefix Arrays

We can implement this as follows:

```
int N, ans = 1;
vector<int> A(N);
for (int i = 0; i < N; i++) cin >> A[i];
vector<int> left(N + 1), right(N + 1);
for (int i = N - 1; i >= 0; i--) {
    right[i] = gcd(right[i + 1], A[i]);
}
for (int i = 1; i <= N; i++) {
    left[i] = gcd(left[i - 1], A[i - 1]);
}
for (int i = 1; i <= N; i++) ans = max(ans, gcd
    (left[i - 1], right[i]));
cout << ans << endl;
```

Prefix Arrays

We can implement this as follows:

```
int N, ans = 1;
vector<int> A(N);
for (int i = 0; i < N; i++) cin >> A[i];
vector<int> left(N + 1), right(N + 1);
for (int i = N - 1; i >= 0; i--) {
    right[i] = gcd(right[i + 1], A[i]);
}
for (int i = 1; i <= N; i++) {
    left[i] = gcd(left[i - 1], A[i - 1]);
}
for (int i = 1; i <= N; i++) ans = max(ans, gcd
    (left[i - 1], right[i]));
cout << ans << endl;
```

Time complexity: $O(N)$

Some notes:

Prefix Arrays

Some notes:

- You can use prefix arrays for other binary operations, depending on what your problem requires

Prefix Arrays

Some notes:

- You can use prefix arrays for other binary operations, depending on what your problem requires
- Again, ensure that the array does not change; otherwise, precomputing will be pointless

Difference Arrays

- Difference Arrays take the differences of consecutive elements in an array

Difference Arrays

- Difference Arrays take the differences of consecutive elements in an array
- Consider an array $A = \{a_1,\ a_2,\ \ldots,\ a_n\}$

Difference Arrays

- Difference Arrays take the differences of consecutive elements in an array
- Consider an array $A = \{a_1,\ a_2,\ \ldots,\ a_n\}$
- Its difference array would be:

$$D = \{a_1,\ a_2 - a_1,\ a_3 - a_2, \ldots,\ a_n - a_{n-1},\ -a_n\}$$

Difference Arrays

- They let us do range updates in constant $O(1)$ time

Difference Arrays

- They let us do range updates in constant $O(1)$ time
- For example: consider the array $A = \{3, 1, 4, 1, 5\}$

Difference Arrays

- They let us do range updates in constant $O(1)$ time
- For example: consider the array $A = \{3, 1, 4, 1, 5\}$

3	1	4	1	5

- They let us do range updates in constant $O(1)$ time
- For example: consider the array $A = \{3, 1, 4, 1, 5\}$

3	1	4	1	5

- Given (one-based) indices l, r, and another integer x, add x to all elements with indices in the range $[l, r]$

Difference Arrays

- They let us do range updates in constant $O(1)$ time
- For example: consider the array $A = \{3, 1, 4, 1, 5\}$

3	1	4	1	5

- Given (one-based) indices l, r, and another integer x, add x to all elements with indices in the range $[l, r]$
- For this example, let's take $(l, r, x) = (1, 3, 2)$

Difference Arrays

- They let us do range updates in constant $O(1)$ time
- For example: consider the array $A = \{3, 1, 4, 1, 5\}$

3	1	4	1	5

- Given (one-based) indices l, r, and another integer x, add x to all elements with indices in the range $[l, r]$
- For this example, let's take $(l, r, x) = (1, 3, 2)$
- Then, the end result of our range update would be:

Difference Arrays

- They let us do range updates in constant $O(1)$ time
- For example: consider the array $A = \{3, 1, 4, 1, 5\}$

3	1	4	1	5

- Given (one-based) indices l, r, and another integer x, add x to all elements with indices in the range $[l, r]$
- For this example, let's take $(l, r, x) = (1, 3, 2)$
- Then, the end result of our range update would be:

5	3	6	1	5

Difference Arrays

To make our implementation easier, we can "pad" our array with one 0 at the beginning and at the end.

Difference Arrays

To make our implementation easier, we can "pad" our array with one 0 at the beginning and at the end.

```
vector<int> A = {0, 3, 1, 4, 1, 5, 0};
vector<int> dif(A.size() - 1);
for (int i = 0; i < A.size() - 1; i++)
    dif[i] = A[i + 1] - A[i];
int l = 1, r = 3, x = 2;
dif[l - 1] += x;
dif[r] -= x;
for (int i = 0; i < A.size() - 1; i++)
    A[i + 1] = A[i] + dif[i];
for (int i = 1; i < A.size() - 1; i++)
    cout << A[i] << " ";
```

Difference Arrays

Quick summary!

Difference Arrays

Quick summary!

- You can add a number x to elements of an array with indices in the range $[l, r]$ by adding x to two elements in its difference array D: D_l and D_{r+1}

Difference Arrays

Quick summary!

- You can add a number x to elements of an array with indices in the range $[l, r]$ by adding x to two elements in its difference array D: D_l and D_{r+1}
- Again, make sure that the elements of the array do not change outside of our range updates; otherwise, doing range updates with difference arrays won't work.

Binary Search

- Given a sorted array, check the middle element, and depending on if it's greater or less than the target element, search the appropriate half of the array

Binary Search

- Given a sorted array, check the middle element, and depending on if it's greater or less than the target element, search the appropriate half of the array
- Uses two pointers (left and right) to track the part of the array that still needs to be searched

Binary Search

- Given a sorted array, check the middle element, and depending on if it's greater or less than the target element, search the appropriate half of the array
- Uses two pointers (left and right) to track the part of the array that still needs to be searched
- Repeat until you either find the element or the two pointers cross (depends on your implementation)

Binary Search

- Given a sorted array, check the middle element, and depending on if it's greater or less than the target element, search the appropriate half of the array
- Uses two pointers (left and right) to track the part of the array that still needs to be searched
- Repeat until you either find the element or the two pointers cross (depends on your implementation)
- "Thanos Search" - Vernon Gutierrez, 2023

Binary Search

Given an array of integers A with n elements, we can implement binary search like so, treating our left and right pointers as the (inclusive) boundaries of our search space:

Binary Search

Given an array of integers A with n elements, we can implement binary search like so, treating our left and right pointers as the (inclusive) boundaries of our search space:

```
int l = 0, r = n - 1;
while (l <= r) {
    int m = (l + r) / 2;
    if (target == A[m]) {
        return m;
    } else if (target < A[m]) {
        r = m - 1;
    } else {
        l = m + 1;
    }
}
return -1; // not found
```

Binary Search

A more reliable way to implement Binary Search would redefine our left and right pointers.

A more reliable way to implement Binary Search would redefine our left and right pointers. We will use invariant-based binary search.

Binary Search

A more reliable way to implement Binary Search would redefine our left and right pointers. We will use invariant-based binary search.

- Instead of making our bounds inclusive (i.e., indices l and r are included in our search space), we can make them exclusive.

Binary Search

A more reliable way to implement Binary Search would redefine our left and right pointers. We will use invariant-based binary search.

- Instead of making our bounds inclusive (i.e., indices l and r are included in our search space), we can make them exclusive.
- Avoids the use of $m \pm 1$, which helps us be more sure of the correctness of our algorithm

Binary Search

A more reliable way to implement Binary Search would redefine our left and right pointers. We will use invariant-based binary search.

- Instead of making our bounds inclusive (i.e., indices l and r are included in our search space), we can make them exclusive.
- Avoids the use of $m \pm 1$, which helps us be more sure of the correctness of our algorithm
- How will that change our implementation?

Binary Search

A more reliable way to implement Binary Search would redefine our left and right pointers. We will use invariant-based binary search.

- Instead of making our bounds inclusive (i.e., indices l and r are included in our search space), we can make them exclusive.
- Avoids the use of $m \pm 1$, which helps us be more sure of the correctness of our algorithm
- How will that change our implementation?

Binary Search

This will be our new implementation:

This will be our new implementation:

```
// should be outside the range of indices
int l = -1, r = n;

while (l + 1 < r) {
    int m = (l + r) / 2;
    if (target == A[m]) {
        return m;
    } else if (target < A[m]) {
        r = m;
    } else {
        l = m;
    }
}
return -1; // not found
```

How does this work?

Binary Search

How does this work?

- Our two pointers separate our search space: the elements at the indices in $[1, l]$ are less than our answer, and the elements at the indices in $[r, n]$ are greater than our answer

Binary Search

How does this work?

- Our two pointers separate our search space: the elements at the indices in $[1, l]$ are less than our answer, and the elements at the indices in $[r, n]$ are greater than our answer
- Our search space is (l, r)

Binary Search

How does this work?

- Our two pointers separate our search space: the elements at the indices in $[1, l]$ are less than our answer, and the elements at the indices in $[r, n]$ are greater than our answer
- Our search space is (l, r)
- If we check an element at index m and find that our target is not equal to it, we know that that element can be immediately excluded from our search space

Then, why should we use invariant-based binary search?

Then, why should we use invariant-based binary search?

Don't have to deal with $m \pm 1$ — we can avoid errors better with this implementation and ensure that not a single element in the array is missed.

Then, why should we use invariant-based binary search?

Don't have to deal with $m \pm 1$ — we can avoid errors better with this implementation and ensure that not a single element in the array is missed.

Though, of course, there is a time and place for both implementations. How you use them is up to you!

Homework :3

Check the Reboot website for the homework problems !!