

ChatGPT 5.2 ▾

   

what is isomorphic in competitive programming
how to check if array and string are isomorphic
string and strings are isomorphic
array and array are isomorphic
explain in detail with intuition and give codes
with proper proof in details
then make a final cpp file and combine all code [give codes
in cpp this time only]

Below is a **complete CP-style explanation** of **isomorphic structures**, with **deep intuition**, **formal conditions**, **proof of correctness**, and **clean C++ implementations**.

You can treat this as a **reference note + template** for contests.

Isomorphic in Competitive Programming

1 What does *Isomorphic* mean?

Two sequences are **isomorphic** if there exists a **one-to-one correspondence** (bijection) between their elements **preserving order and equality pattern**.

👉 Pattern matters, actual values don't

2 Formal Definition (Very Important)

Given two sequences `A` and `B` of the same length `n`:

They are **isomorphic iff:**

Condition 1 — Forward Mapping

mathematica

```
If A[i] == A[j] → B[i] == B[j]
```

 Copy code

Condition 2 — Reverse Mapping

mathematica

```
If B[i] == B[j] → A[i] == A[j]
```

Copy code

- 📌 Both directions are **mandatory**
 - 📌 This guarantees **one-to-one mapping (bijection)**
-

3 Why both conditions are needed?

✗ Without reverse condition

ini

```
A = [1, 2]
B = [a, a]
```

Copy code

- Forward holds ✓
- Reverse fails ✗ (two A-values map to same B)

So NOT isomorphic

4 Intuition (How to think in contests)

Think in terms of **patterns**:

bash

```
cat    →  0 1 2
egg   →  0 1 1
foo    →  0 1 1
paper →  0 1 0 2 3
title →  0 1 0 2 3
```

Copy code

Two sequences are isomorphic iff their pattern representation is identical.

✓ Types of Isomorphism Problems

5 String ↔ String Isomorphic

Example

ini

```
s = "egg"
t = "add"
```

Copy code

Mapping:

css

e → a
g → d

Copy code

- ✓ One-to-one
 - ✓ Order preserved
 - ✓ Isomorphic
-

✗ Counter Example

ini

```
s = "foo"
t = "bar"
```

Copy code

pgsql

f → b
o → a
o → r conflict

Copy code

✓ C++ Logic (Hash Maps)

cpp

```
bool isIsomorphicString(string s, string t) {
    if (s.size() != t.size()) return false;

    unordered_map<char, char> mp1, mp2;

    for (int i = 0; i < s.size(); i++) {
        if (mp1.count(s[i]) && mp1[s[i]] != t[i]) return false;
        mp1[s[i]] = t[i];
    }
}
```

```

    if (mp2.count(t[i]) && mp2[t[i]] != s[i]) return false;

    mp1[s[i]] = t[i];
    mp2[t[i]] = s[i];
}

return true;
}

```

 Copy code

Proof of Correctness

- `mp1` ensures **forward condition**
- `mp2` ensures **reverse condition**
- Conflict in either \Rightarrow bijection broken
- Thus mapping exists **iff function returns true**

✓ Correct

6 Array \leftrightarrow Array Isomorphic

Example

ini

```

A = [1, 2, 1, 3]
B = [5, 6, 5, 7]

```

 Copy code

Mapping:

```

1 → 5
2 → 6
3 → 7

```

 Copy code

✓ Isomorphic

✗ Counter Example

ini

```
A = [1, 2]
B = [3, 3]
```

 Copy code

Two distinct values → same target ×

✓ C++ Code

cpp

```
bool isIsomorphicArray(vector<int>& a, vector<int>& b) {
    if (a.size() != b.size()) return false;

    unordered_map<int, int> mp1, mp2;

    for (int i = 0; i < a.size(); i++) {
        if (mp1.count(a[i]) && mp1[a[i]] != b[i]) return false;
        if (mp2.count(b[i]) && mp2[b[i]] != a[i]) return false;

        mp1[a[i]] = b[i];
        mp2[b[i]] = a[i];
    }
    return true;
}
```

 Copy code

🧠 Proof

Same as string case

Only type differs (`int` instead of `char`)

- ✓ Guarantees bijection
 - ✓ Order preserved
-

7 String ↔ Array Isomorphic

Example

ini

```
s = "abbc"
a = [1, 2, 2, 3]
```

Copy code

Mapping:

css

```
a → 1
b → 2
c → 3
```

Copy code

✓ Isomorphic

✗ Counter Example

ini

```
s = "ab"
a = [1, 1]
```

Copy code

Two chars → same int ✗

✓ C++ Code

cpp

```
bool isIsomorphicStringArray(string s, vector<int>& a) {
    if (s.size() != a.size()) return false;

    unordered_map<char, int> mp1;
    unordered_map<int, char> mp2;

    for (int i = 0; i < s.size(); i++) {
        if (mp1.count(s[i]) && mp1[s[i]] != a[i]) return false;
        if (mp2.count(a[i]) && mp2[a[i]] != s[i]) return false;

        mp1[s[i]] = a[i];
        mp2[a[i]] = s[i];
    }
}
```

Copy code

```

    return true;
}

```

8 Pattern-Encoding Method (Advanced & Elegant)

Convert any sequence into first-occurrence index pattern.

Example

nginx

```

paper → 0 1 0 2 3
title → 0 1 0 2 3

```

Copy code

Same pattern ⇒ Isomorphic

Pattern Function

cpp

```

template <typename T>
vector<int> encodePattern(const vector<T>& v) {
    unordered_map<T, int> mp;
    vector<int> pattern;
    int id = 0;

    for (auto &x : v) {
        if (!mp.count(x)) mp[x] = id++;
        pattern.push_back(mp[x]);
    }
    return pattern;
}

```

Copy code

Usage

cpp

```

bool isIsomorphicPattern(vector<int>& a, vector<int>& b) {
    return encodePattern(a) == encodePattern(b);
}

```

}

- ✓ Very useful in CP
 - ✓ Clean logic
 - ✓ Strong conceptual clarity
-



Time & Space Complexity

Method	Time	Space
Map-based	$O(n)$	$O(n)$
Pattern-based	$O(n)$	$O(n)$



FINAL COMBINED C++ FILE (CP Ready)

cpp

```
#include <bits/stdc++.h>
using namespace std;

/* String ↔ String */
bool isIsomorphicString(string s, string t) {
    if (s.size() != t.size()) return false;
    unordered_map<char, char> mp1, mp2;
    for (int i = 0; i < s.size(); i++) {
        if (mp1.count(s[i]) && mp1[s[i]] != t[i]) return false;
        if (mp2.count(t[i]) && mp2[t[i]] != s[i]) return false;
        mp1[s[i]] = t[i];
        mp2[t[i]] = s[i];
    }
    return true;
}

/* Array ↔ Array */
bool isIsomorphicArray(vector<int>& a, vector<int>& b) {
    if (a.size() != b.size()) return false;
    unordered_map<int, int> mp1, mp2;
    for (int i = 0; i < a.size(); i++) {
        if (mp1.count(a[i]) && mp1[a[i]] != b[i]) return false;
        if (mp2.count(b[i]) && mp2[b[i]] != a[i]) return false;
        mp1[a[i]] = b[i];
        mp2[b[i]] = a[i];
    }
    return true;
}
```

```

        if (mp2.count(b[i]) && mp2[b[i]] != a[i]) return false;
        mp1[a[i]] = b[i];
        mp2[b[i]] = a[i];
    }
    return true;
}

/* String ↔ Array */
bool isIsomorphicStringArray(string s, vector<int>& a) {
    if (s.size() != a.size()) return false;
    unordered_map<char, int> mp1;
    unordered_map<int, char> mp2;
    for (int i = 0; i < s.size(); i++) {
        if (mp1.count(s[i]) && mp1[s[i]] != a[i]) return false;
        if (mp2.count(a[i]) && mp2[a[i]] != s[i]) return false;
        mp1[s[i]] = a[i];
        mp2[a[i]] = s[i];
    }
    return true;
}

/* Pattern Encoding */
template <typename T>
vector<int> encodePattern(const vector<T>& v) {
    unordered_map<T, int> mp;
    vector<int> pattern;
    int id = 0;
    for (auto &x : v) {
        if (!mp.count(x)) mp[x] = id++;
        pattern.push_back(mp[x]);
    }
    return pattern;
}

bool isIsomorphicPattern(vector<int>& a, vector<int>& b) {
    return encodePattern(a) == encodePattern(b);
}

int main() {
    // Example usage
    string s = "paper", t = "title";
    cout << isIsomorphicString(s, t) << "\n";
}

```

```

vector<int> a = {1, 2, 1}, b = {3, 4, 3};
cout << isIsomorphicArray(a, b) << "\n";

return 0;
}

```

Copy code

🔑 Final Takeaway (CP Mindset)

- Isomorphism = same equality pattern
- Always enforce two-way mapping
- Think in bijections, not values
- Pattern-encoding is a power tool

If you want:

- Lambda version
- Interview-style explanation
- LeetCode / Codeforces variants

just tell me

...

cover all advance cases and please use map always from now no unordered map

Perfect — this is a very CP-mature request

I'll now:

1. Cover ALL advanced / tricky cases
2. Explain WHY each case matters
3. Use `map` ONLY (no `unordered_map`, as you asked)
4. Give formal correctness proofs
5. End with ONE final combined C++ file (CP-ready)

From now on → `map` only

ISOMORPHIC — ADVANCED & COMPLETE GUIDE (CP Level)

1 What exactly must hold? (Formal Model)

Two sequences `A` and `B` are **isomorphic** iff there exists a **bijection**

`f : elements(A) → elements(B)` such that:

python

```
B[i] = f(A[i])  for all i
```

 Copy code

Bijection =

- ✓ Injective (no two A map to same B)
- ✓ Surjective (every used B has exactly one A)

This translates to **two constraints**:

```
A[i] == A[j]  ⇔  B[i] == B[j]
```

 Copy code

 BOTH directions are mandatory

2 ALL ADVANCED EDGE CASES (Very Important)

✓ Case 1: Length mismatch

ini

```
A = [1, 2]
```

```
B = [a]
```

 Copy code

✗ Impossible → return false immediately

✓ Case 2: Single element

ini

```
A = [5]  
B = [100]
```

Copy code

- ✓ Always isomorphic (trivial bijection)
-

✓ Case 3: All elements same

```
ini
```

```
A = [1, 1, 1]  
B = [2, 2, 2]
```

Copy code

- ✓ Valid

But:

```
ini
```

```
A = [1, 1, 1]  
B = [2, 3, 2]
```

Copy code

✗ Violates forward condition

✓ Case 4: Distinct elements only

```
ini
```

```
A = [1, 2, 3]  
B = [4, 5, 6]
```

Copy code

- ✓ Always isomorphic
-

✗ Case 5: Many-to-one mapping (MOST COMMON BUG)

```
ini
```

```
A = [1, 2]  
B = [3, 3]
```

Copy code

Forward mapping passes ✗

Reverse mapping fails ✗

✗ Case 6: Conflicting remap

ini

```
A = [1, 1, 2]
B = [3, 4, 3]
```

 Copy code

1 → 3 then 1 → 4 ✗

✗ Case 7: Same pattern but wrong order

ini

```
A = [1, 2, 1]
B = [3, 3, 4]
```

 Copy code

Patterns differ → ✗

✓ Case 8: Negative / large numbers

ini

```
A = [-1, 1000000000, -1]
B = [5, 6, 5]
```

 Copy code

✓ Works fine (map handles it)

✓ Case 9: Mixed domains (string ↔ array)

ini

```
s = "abba"
a = [10, 20, 20, 10]
```

 Copy code

✓ Valid

3 WHY `map` IS SAFER IN CP

Feature	<code>map</code>
Deterministic	✓
No hacking risk	✓
Ordered traversal	✓
Stable in stress	✓

Time: $O(n \log n)$ → perfectly acceptable

🧠 CORE LOGIC (UNIVERSAL)

For every index `i`:

1. If `A[i]` seen before → must map to same `B[i]`
2. If `B[i]` seen before → must map to same `A[i]`
3. Else → create new mapping

This enforces **bijection**

4 STRING ↔ STRING (map only)

cpp

```
bool isIsomorphicString(string s, string t) {
    if (s.size() != t.size()) return false;

    map<char, char> fwd, rev;

    for (int i = 0; i < (int)s.size(); i++) {
        if (fwd.count(s[i]) && fwd[s[i]] != t[i]) return false;
        if (rev.count(t[i]) && rev[t[i]] != s[i]) return false;

        fwd[s[i]] = t[i];
        rev[t[i]] = s[i];
    }
}
```

Copy code

```
    return true;
}
```

✓ Correctness Proof

- `fwd` enforces **A → B consistency**
- `rev` enforces **injectivity**
- Conflict ⇒ bijection impossible
- No conflict ⇒ bijection exists

✓ Correct

5 ARRAY ↔ ARRAY (map only)

cpp

```
bool isIsomorphicArray(const vector<long long>& a,
                      const vector<long long>& b) {
    if (a.size() != b.size()) return false;

    map<long long, long long> fwd, rev;

    for (int i = 0; i < (int)a.size(); i++) {
        if (fwd.count(a[i]) && fwd[a[i]] != b[i]) return false;
        if (rev.count(b[i]) && rev[b[i]] != a[i]) return false;

        fwd[a[i]] = b[i];
        rev[b[i]] = a[i];
    }
    return true;
}
```

 Copy code

Handles:

- negatives
- large values
- duplicates
- all CP constraints

6 STRING ↔ ARRAY (map only)

cpp

```

bool isIsomorphicStringArray(const string& s,
                            const vector<long long>& a) {
    if (s.size() != a.size()) return false;

    map<char, long long> fwd;
    map<long long, char> rev;

    for (int i = 0; i < (int)s.size(); i++) {
        if (fwd.count(s[i]) && fwd[s[i]] != a[i]) return false;
        if (rev.count(a[i]) && rev[a[i]] != s[i]) return false;

        fwd[s[i]] = a[i];
        rev[a[i]] = s[i];
    }
    return true;
}

```

 Copy code

7 ADVANCED PATTERN-BASED METHOD (map only)

Why this is powerful?

- Converts any domain → integer pattern
- Simplifies comparison
- Great for **multiple sequences**

Pattern Encoder

cpp

```

template <typename T>
vector<int> encodePattern(const vector<T>& v) {
    map<T, int> id;
    vector<int> pattern;
    int cur = 0;

    for (auto &x : v) {

```

```

    if (!id.count(x)) id[x] = cur++;
    pattern.push_back(id[x]);
}
return pattern;
}

```

 Copy code

Usage

cpp

```

bool isIsomorphicByPattern(const vector<long long>& a,
                           const vector<long long>& b) {
    return encodePattern(a) == encodePattern(b);
}

```

 Copy code

Proof

- First appearance gets new ID
- Same values → same ID
- Equality pattern preserved exactly
- Equal patterns \Leftrightarrow isomorphic

✓ Correct



COMPLEXITY ANALYSIS

Method	Time	Space
Two-map bijection	$O(n \log n)$	$O(n)$
Pattern encoding	$O(n \log n)$	$O(n)$



FINAL COMBINED C++ FILE (MAP ONLY)

cpp

```

#include <bits/stdc++.h>
using namespace std;

/* STRING ↔ STRING */
bool isIsomorphicString(const string& s, const string& t) {
    if (s.size() != t.size()) return false;

    map<char, char> fwd, rev;
    for (int i = 0; i < (int)s.size(); i++) {
        if (fwd.count(s[i]) && fwd[s[i]] != t[i]) return false;
        if (rev.count(t[i]) && rev[t[i]] != s[i]) return false;
        fwd[s[i]] = t[i];
        rev[t[i]] = s[i];
    }
    return true;
}

/* ARRAY ↔ ARRAY */
bool isIsomorphicArray(const vector<long long>& a,
                      const vector<long long>& b) {
    if (a.size() != b.size()) return false;

    map<long long, long long> fwd, rev;
    for (int i = 0; i < (int)a.size(); i++) {
        if (fwd.count(a[i]) && fwd[a[i]] != b[i]) return false;
        if (rev.count(b[i]) && rev[b[i]] != a[i]) return false;
        fwd[a[i]] = b[i];
        rev[b[i]] = a[i];
    }
    return true;
}

/* STRING ↔ ARRAY */
bool isIsomorphicStringArray(const string& s,
                            const vector<long long>& a) {
    if (s.size() != a.size()) return false;

    map<char, long long> fwd;
    map<long long, char> rev;
    for (int i = 0; i < (int)s.size(); i++) {
        if (fwd.count(s[i]) && fwd[s[i]] != a[i]) return false;
        if (rev.count(a[i]) && rev[a[i]] != s[i]) return false;
        fwd[s[i]] = a[i];
    }
}

```

```

        rev[a[i]] = s[i];
    }
    return true;
}

/* PATTERN METHOD */
template <typename T>
vector<int> encodePattern(const vector<T>& v) {
    map<T, int> id;
    vector<int> pattern;
    int cur = 0;
    for (auto &x : v) {
        if (!id.count(x)) id[x] = cur++;
        pattern.push_back(id[x]);
    }
    return pattern;
}

bool isIsomorphicByPattern(const vector<long long>& a,
                           const vector<long long>& b) {
    return encodePattern(a) == encodePattern(b);
}

int main() {
    string s = "paper", t = "title";
    cout << isIsomorphicString(s, t) << "\n";

    vector<long long> a = {1,2,1}, b = {3,4,3};
    cout << isIsomorphicArray(a, b) << "\n";
    return 0;
}

```

 Copy code

🔥 FINAL CP TAKEAWAY

- Isomorphism = **bijection**
- Always check **both directions**
- **map** is safest in contests
- Pattern method = elite technique

If you want:

- Lambda version
- Test-case generator
- LeetCode / CF problem list
- Formal math proof style

Just say the word 

     ...

give lambda fn based code also

Perfect 

Below is a **clean, CP-style LAMBDA-FUNCTION based solution** for **ALL isomorphic cases**, using `map` **only** (as you strictly asked).

This is **contest-ready, safe**, and **advanced-case proof**.

ISOMORPHIC — LAMBDA FUNCTION APPROACH (`map` only)

We'll write **generic lambdas** so that:

- Same logic works for **string–string**
- **array–array**
- **string–array**
- **any comparable type**

Core Idea (Lambda Logic)

Inside lambda:

1. Maintain two maps
 - forward map (`A → B`)
 - reverse map (`B → A`)
2. For each index:
 - If mapping exists → must match
 - Else → create mapping
3. Any conflict → `false`

This enforces **bijection ✓**

1 Generic Lambda (Most Powerful)

Works for ANY two sequences with `operator[]` and `size()`

cpp

```
auto isIsomorphicGeneric = [](const auto& A, const auto& B) -> bool {
    if (A.size() != B.size()) return false;

    map<typename decay<decltype(A[0])>::type,
        typename decay<decltype(B[0])>::type> fwd;
    map<typename decay<decltype(B[0])>::type,
        typename decay<decltype(A[0])>::type> rev;

    for (int i = 0; i < (int)A.size(); i++) {
        if (fwd.count(A[i]) && fwd[A[i]] != B[i]) return false;
        if (rev.count(B[i]) && rev[B[i]] != A[i]) return false;

        fwd[A[i]] = B[i];
        rev[B[i]] = A[i];
    }
    return true;
};
```

 Copy code

✓ Usage

cpp

```
string s1 = "egg", s2 = "add";
cout << isIsomorphicGeneric(s1, s2) << "\n";

vector<long long> a = {1,2,1}, b = {3,4,3};
cout << isIsomorphicGeneric(a, b) << "\n";
```

 Copy code

2 Lambda for String ↔ String (Explicit)

cpp

```
auto isIsomorphicString = [](<const string& s, const string& t) -> bool {
    if (s.size() != t.size()) return false;

    map<char, char> fwd, rev;

    for (int i = 0; i < (int)s.size(); i++) {
        if (fwd.count(s[i]) && fwd[s[i]] != t[i]) return false;
        if (rev.count(t[i]) && rev[t[i]] != s[i]) return false;

        fwd[s[i]] = t[i];
        rev[t[i]] = s[i];
    }

    return true;
};
```

Copy code

3 Lambda for Array \leftrightarrow Array

cpp

```
auto isIsomorphicArray = [](<const vector<long long>& a,
                           const vector<long long>& b) -> bool {
    if (a.size() != b.size()) return false;

    map<long long, long long> fwd, rev;

    for (int i = 0; i < (int)a.size(); i++) {
        if (fwd.count(a[i]) && fwd[a[i]] != b[i]) return false;
        if (rev.count(b[i]) && rev[b[i]] != a[i]) return false;

        fwd[a[i]] = b[i];
        rev[b[i]] = a[i];
    }

    return true;
};
```

Copy code

4 Lambda for String \leftrightarrow Array

cpp

```

auto isIsomorphicStringArray =
[]([](const string& s, const vector<long long>& a) -> bool {
    if (s.size() != a.size()) return false;

    map<char, long long> fwd;
    map<long long, char> rev;

    for (int i = 0; i < (int)s.size(); i++) {
        if (fwd.count(s[i]) && fwd[s[i]] != a[i]) return false;
        if (rev.count(a[i]) && rev[a[i]] != s[i]) return false;

        fwd[s[i]] = a[i];
        rev[a[i]] = s[i];
    }
    return true;
};

```

 Copy code

5 Lambda Pattern-Encoding (Advanced & Elegant)

Pattern Encoder Lambda

cpp

```

auto encodePattern = []([](const auto& v) {
    map<typename decay<decltype(v[0])>::type, int> id;
    vector<int> pattern;
    int cur = 0;

    for (auto &x : v) {
        if (!id.count(x)) id[x] = cur++;
        pattern.push_back(id[x]);
    }
    return pattern;
};

```

 Copy code

Pattern-Based Isomorphism Lambda

cpp

```
auto isIsomorphicByPattern =
[&](const auto& A, const auto& B) -> bool {
    if (A.size() != B.size()) return false;
    return encodePattern(A) == encodePattern(B);
};
```

 Copy code

6 FINAL COMBINED C++ FILE (LAMBDA + MAP ONLY)

cpp

```
#include <bits/stdc++.h>
using namespace std;

int main() {

    /* GENERIC LAMBDA */
    auto isIsomorphicGeneric = [] (const auto& A, const auto& B) -> bool {
        if (A.size() != B.size()) return false;

        map<typename decay<decltype(A[0])>::type,
            typename decay<decltype(B[0])>::type> fwd;
        map<typename decay<decltype(B[0])>::type,
            typename decay<decltype(A[0])>::type> rev;

        for (int i = 0; i < (int)A.size(); i++) {
            if (fwd.count(A[i]) && fwd[A[i]] != B[i]) return false;
            if (rev.count(B[i]) && rev[B[i]] != A[i]) return false;
            fwd[A[i]] = B[i];
            rev[B[i]] = A[i];
        }
        return true;
    };

    /* PATTERN METHOD */
    auto encodePattern = [] (const auto& v) {
        map<typename decay<decltype(v[0])>::type, int> id;
        vector<int> pattern;
        int cur = 0;
        for (auto &x : v) {
            if (!id.count(x)) id[x] = cur++;
            pattern.push_back(id[x]);
        }
        return pattern;
    };
}
```

```

        pattern.push_back(id[x]);
    }
    return pattern;
};

auto isIsomorphicByPattern =
[&](const auto& A, const auto& B) -> bool {
    if (A.size() != B.size()) return false;
    return encodePattern(A) == encodePattern(B);
};

/* TESTS */
string s1 = "paper", s2 = "title";
cout << isIsomorphicGeneric(s1, s2) << "\n";

vector<long long> a = {1,2,1}, b = {3,4,3};
cout << isIsomorphicGeneric(a, b) << "\n";

cout << isIsomorphicByPattern(a, b) << "\n";

return 0;
}

```

 Copy code

🔥 WHY THIS IS CP-LEVEL ELITE

- ✓ Handles **all edge cases**
- ✓ Uses **map only** (deterministic)
- ✓ Generic lambda = reusable everywhere
- ✓ Pattern trick = CF / ICPC favorite
- ✓ Clean, readable, safe

If you want next:

- **Lambda inside class Solution**
- **STL-only ultra-short version**
- **Interview vs CP differences**
- **Test case generator**
- **Formal mathematical proof**

Just tell me 🚀



Condition 1 — Forward Mapping

mathematica

If $A[i] == A[j]$ → $B[i] == B[j]$

Copy code

Condition 2 — Reverse Mapping

<https://chatgpt.com/c/693e9377-be00-8320-a2cf-3dac6e5654ed>

1/20

1/14/25, 4:13 PM

Advanced map usage

If $B[i] == B[j]$ → $A[i] == A[j]$

Copy code

how u will describe this in words



You can describe those two conditions **clearly and formally in words** like this 👇

📌 Verbal Description (Clean & CP-ready)

Condition 1 — Forward Mapping

If two positions in the first sequence contain the **same value**, then the corresponding positions in the second sequence must also contain the **same value**.

In words:

"Equal elements in A must map to equal elements in B ."

Condition 2 — Reverse Mapping

If two positions in the second sequence contain the **same value**, then the corresponding positions in the first sequence must also contain the **same value**.

In words:

"Equal elements in B must come from equal elements in A ."

🔁 Combined One-Line Description (Best for Interviews / CP)

Two sequences are **isomorphic** if and only if **equality relationships are preserved in both directions** between corresponding positions.

OR more explicitly:

Two sequences are isomorphic when **each distinct element in the first sequence maps to exactly one distinct element in the second sequence, and vice versa**, while preserving the order.

Intuition (Very Important)

- **Forward condition** prevents:
 - One element mapping to multiple different values
- **Reverse condition** prevents:
 - Multiple elements mapping to the same value

Together, they enforce a **bijection (one-to-one correspondence)**.

Why both are required (simple explanation)

If you check **only forward mapping**, you might allow:

ini

 Copy code

```
A = [1, 2]  
B = [3, 3]
```

- Forward condition holds
- Reverse condition fails
 - ✗ Not isomorphic

Hence **both conditions are mandatory**.

Ultra-short CP definition (you can memorize)

Isomorphic means: same pattern of equality.

If you want, I can also:

- Rewrite this in **math-proof style**
- Convert this into **comment block for your code**
- Give **one-sentence explanation for Codeforces editorials**

Just say 

     ...