LeetCode 771: Jewels and Stones
Brute Force → Optimal (C++, Java, Python)

Problem Summary: You are given strings J (types of jewels) and S (stones you have). Count how many stones in S are jewels.

Approach 1 — Brute force (nested loops)

Idea: For every stone, scan all jewels and check equality. Time: O(|J| * |S|), Space: O(1)

C++:

```
class Solution {
public:
    int numJewelsInStones(string J, string S) {
        int count = 0;
        for (char s : S) {
            for (char j : J) {
                if (s == j) {
                    count++;
                    break;
                }
            }
        }
        return count;
    }
};
```

Java:

```
public class Solution {
    public int numJewelsInStones(String J, String S) {
        int count = 0;
        for (int i = 0; i < S.length(); i++) {
            char s = S.charAt(i);
            for (int j = 0; j < J.length(); j++) {
                if (s == J.charAt(j)) {
                    count++;
                    break;
                }
            }
        }
        return count;
    }
}
```

Python:

```
class Solution:
    def numJewelsInStones(self, J: str, S: str) -> int:
        count = 0
        for s in S:
            for j in J:
                if s == j:
                    count += 1
                    break
        return count
```

Approach 2 — Hash set (common optimal in practice)

Idea: Put J characters in a hash set, then iterate S and test membership. Time: O(|J| + |S|), Space: O(|J|)

C++:

```
#include <unordered_set>
class Solution {
public:
    int numJewelsInStones(string J, string S) {
        unordered_set<char> jewels;
        for (char j : J) jewels.insert(j);
        int count = 0;
        for (char s : S) if (jewels.count(s)) count++;
        return count;
    }
};
```

Java:

```
import java.util.HashSet;
import java.util.Set;

public class Solution {
    public int numJewelsInStones(String J, String S) {
        Set<Character> jewels = new HashSet<>();
        for (char c : J.toCharArray()) jewels.add(c);
        int count = 0;
        for (char s : S.toCharArray()) if (jewels.contains(s)) count++;
        return count;
    }
}
```

Python:

```
class Solution:
    def numJewelsInStones(self, J: str, S: str) -> int:
        jewels = set(J)
        count = 0
        for s in S:
            if s in jewels:
                count += 1
        return count
```

Approach 3 — ASCII count / fixed-size lookup

Idea: If characters are ASCII, use an array of size 128 to mark jewels; then iterate S. Time: O(|J| + |S|), Space: O(1)

C++:

```
class Solution {
public:
    int numJewelsInStones(string J, string S) {
        int mark[128] = {0};
        for (char c : J) mark[(int)c] = 1;
        int count = 0;
        for (char s : S) if (mark[(int)s]) count++;
        return count;
    }
};
```

Java:

```
public class Solution {
    public int numJewelsInStones(String J, String S) {
        boolean mark[] = new boolean[128];
        for (char c : J.toCharArray()) {
            if (c < 128) mark[c] = true;
        }
        int count = 0;
        for (char s : S.toCharArray()) {
            if (s < 128 && mark[s]) count++;
        }
        return count;
    }
}
```

Python:

```
class Solution:
    def numJewelsInStones(self, J: str, S: str) -> int:
        mark = [0] * 128
        for c in J:
            if ord(c) < 128:
                mark[ord(c)] = 1
        count = 0
        for s in S:
            if ord(s) < 128 and mark[ord(s)]:
                count += 1
        return count
```

Example:

Input: J = "aA", S = "aAAbbbb" → Output: 3
Explanation: Stones 'a', 'A', 'A' are jewels (3 total).