**G-Hay Points** [**UVA - 10295**](https://vjudge.net/problem/UVA-10295/origin)**https://vjudge.net/problem/UVA-10295**

#include <iostream>

#include <unordered\_map>

#include <string>

#include <sstream>

#include <cstdio>

using namespace std;

int main() {

    int m, n;

    scanf("%d %d", &m, &n);

    unordered\_map<string, int> hayDict;

    char word[21];

    int value;

    // Reading the dictionary

    for (int i = 0; i < m; ++i) {

        scanf("%s %d", word, &value);

        hayDict[string(word)] = value;

    }

    // Clear newline character after last input

    cin.ignore();

    // Reading job descriptions

    string line;

    int total = 0;

    while (n--) {

        total = 0;

        while (getline(cin, line)) {

            if (line == ".") break;

            stringstream ss(line);

            string token;

            while (ss >> token) {

                if (hayDict.count(token)) {

                    total += hayDict[token];

                }

            }

        }

        printf("%d\n", total);

    }

    return 0;

}

**H-The Department of Redundancy Department**[**UVA - 484**](https://vjudge.net/problem/UVA-484/origin)**(https://vjudge.net/problem/UVA-484)**

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

int main() {

    unordered\_map<int, int> freqMap;

    vector<int> order;

    int num;

    while (cin >> num) {

        if (freqMap.count(num) == 0) {

            order.push\_back(num);

        }

        freqMap[num]++;

    }

    for (int val : order) {

        cout << val << " " << freqMap[val] << "\n";

    }

    return 0;

}

**I-Babelfish** [**UVA - 10282**](https://vjudge.net/problem/UVA-10282/origin)[**https://vjudge.net/problem/UVA-10282**](https://vjudge.net/problem/UVA-10282)

#include <iostream>

#include <unordered\_map>

#include <string>

using namespace std;

int main() {

    unordered\_map<string, string> dict;

    string line, english, foreign;

    // Read dictionary entries

    while (getline(cin, line) && !line.empty()) {

        size\_t spacePos = line.find(' ');

        english = line.substr(0, spacePos);

        foreign = line.substr(spacePos + 1);

        dict[foreign] = english;

    }

    // Read and translate the foreign message

    while (getline(cin, foreign)) {

        if (dict.find(foreign) != dict.end()) {

            cout << dict[foreign] << "\n";

        } else {

            cout << "eh\n";

        }

    }

    return 0;

}

**Swappable**[**AtCoder - abc206\_c**](https://vjudge.net/problem/AtCoder-abc206_c/origin)[**https://vjudge.net/problem/AtCoder-abc206\_c**](https://vjudge.net/problem/AtCoder-abc206_c)

**Efficient Approach**

**Step-by-step:**

1. **Total pairs** (without conditions):

total\_pairs=N×(N−1) / 2

1. **Subtract invalid pairs (A[i] == A[j])**:
   * Count frequency of each element.
   * For every value with count f, the number of bad pairs is:

f×(f−1) / 2

* + Subtract those from total\_pairs

#include <iostream>

#include <unordered\_map>

using namespace std;

int main() {

    ios::sync\_with\_stdio(false);

    cin.tie(NULL);

    long long n;

    cin >> n;

    unordered\_map<int, long long> freq;

    int x;

    for (int i = 0; i < n; ++i) {

        cin >> x;

        freq[x]++;

    }

    long long total\_pairs = n \* (n - 1) / 2;

    for (auto it = freq.begin(); it != freq.end(); ++it) {

        long long count = it->second;

        total\_pairs -= count \* (count - 1) / 2;

    }

    cout << total\_pairs << '\n';

    return 0;

}

**Step-by-Step Complexity:**

Step 1: Reading Input and Counting Frequencies

Time: O(n)

Each element is read once and inserted/updated in the hash map.

Average-case insertion into an unordered\_map is O(1).

Space: O(k)

Where k is the number of distinct elements in the array.

Worst case: k = n (all values are unique).

Time: O(1)

Space: O(1)

Time: O(k)

We loop once over the distinct elements.

Space: O(1) extra (since we're using existing map structure).

Overall Complexity

Operation Time Complexity Space Complexity

Reading & frequency count O(n) O(k)

Compute total pairs O(1) O(1)

Subtract duplicates O(k) O(1)

Total O(n) O(k)

Where k is the number of distinct integers (k ≤ n).

Summary:

The algorithm runs in linear time relative to input size: O(n).

It uses linear space in the number of distinct integers.