unstop | Design HashMap

Problem | Solution | Submissions

Problem Statement

Bob is an arrogant person and got into an argument with the developer of programming languages. In his fit of rage, he declared that he won't use any built-in libraries from now on.

Later that day, he started to implement several functions and data structures on his own, but he had trouble in implementing HashMap. After failing several times, he asked you for help.

He wants his HashMap to map unique integer keys to integer values and perform 3 basic operations in response to the following queries:

- Insert – Insert a given (key, value) pair into the map. If the key already exists, update its corresponding value. No need to print anything in this type of query.
- Get – Print the value corresponding to the given key, and print -1 if the key is not present.
- Delete – Delete the key and its corresponding value if the key is present in the map. No need to print anything in this type of query.

Help Bob implement his own HashMap without using any built-in Hash table libraries.

Input Format

- The first line of the input contains n, the number of queries.
- Each of the next n lines starts with integer t, the type of query.
- For type = 1, the line further contains 2 integers denoting the key and its corresponding value to be inserted.
- For type = 2, the line further contains a single integer denoting the key whose value is to be printed.
- For type = 3, the line further contains a single integer denoting the key that is to be deleted.

Output Format

- For queries of type 2, print a single integer – value corresponding to the given key.

Constraints

- $1 <= n <= 10^4$
- $0 <= key, value <= 10^9$
- All the queries are orderly dependent.
- There will be at least one type 2 query.
- In type 3 queries, it is guaranteed that the key given is present in the map at the moment.

Sample Testcase 0

Testcase Input

```
8
1 1 1
1 2 2
2 1
2 3
1 2 1
2 2
3 2
2 2
```

Testcase Output

```
1
-1
```

```
1
-1
```

Explanation

For query 1, we insert (1, 1). Our map is now [(1, 1)]
For query 2, we insert (2, 2). Our map is now [(1, 1), (2, 2)]
For query 3, we print value corresponding to key 1, i.e., 1
For query 4, we do not have the key 3 in our map, hence we print -1
For query 5, we update the value corresponding to key 2 to 1. Out map now is [(1, 1), (2, 1)]
For query 6, we print the value corresponding to key 2, i.e., 1
For query 7, we delete the key 2 from the map. Out map now is [(1, 1)]
For query 8. we do not have the key 2 in our map anymore, hence we print -1

Sample Testcase 1

Testcase Input

```
9
1 4 2
1 25 6
2 4
1 4 8
2 25
3 25
2 25
1 25 1
2 25
```

Testcase Output

```
2
6
-1
1
```

Explanation

For query 1, we insert (4, 2). Our map is now [(4, 2)]
For query 2, we insert (25, 6). Our map is now [(4, 2), (25, 6)]
For query 3, we print value corresponding to key 4, i.e., 2
For query 4, we update the value corresponding to key 4 to 8. Out map now is [(4, 8), (25, 6)]
For query 5, we print the value corresponding to key 25, i.e., 6
For query 6, we delete the key 25 from the map. Out map now is [(4, 8)]
For query 7, we do not have the key 25 in our map anymore, hence we print -1
For query 8. we insert (25, 1). Our map is now [(4, 8), (25, 1)]
For query 9, we print the value corresponding to key 25, i.e., 1

CODE

```
#include <iostream>
#include <vector>

using namespace std;

const int N = 10010; // N is the size of the hash table

vector<pair<int, int>> h[N]; // 'h' is an array of vectors, where each vector stores key-value pairs

// This function calculates the hash value for a given key
int hashFunc(int key) {
    return key % N;
}
```

```
// Insert function: Insert a key-value pair into the HashMap
void insert(int key, int value) {
    int idx = hashFunc(key); // Calculate the hash index for the given key
    for (int i = 0; i < h[idx].size(); i++) {
        if (h[idx][i].first == key) { // Check if the key already exists in the vector
            h[idx][i].second = value; // Update the value if the key is found
            return;
        }
    }
    h[idx].push_back({key, value}); // If the key is not found, add it as a new pair
}

// Get function: Retrieve the value associated with a given key
int get(int key) {
    int idx = hashFunc(key); // Calculate the hash index for the given key
    for (int i = 0; i < h[idx].size(); i++) {
        if (h[idx][i].first == key) { // Check if the key exists in the vector
            return h[idx][i].second; // Return the corresponding value
        }
    }
    return -1; // If the key is not found, return -1 to indicate that it doesn't exist
}

// Delete function: Remove a key-value pair from the HashMap
void del(int key) {
    int idx = hashFunc(key); // Calculate the hash index for the given key
    for (int i = 0; i < h[idx].size(); i++) {
        if (h[idx][i].first == key) { // Check if the key exists in the vector
            h[idx].erase(h[idx].begin() + i); // Remove the key-value pair from the vector
            return;
        }
    }
}

int main() {
    int n;
    cin >> n; // Read the number of queries

    while (n--) {
        int type, key, value;
        cin >> type; // Read the type of query (1, 2, or 3)

        if (type == 1) {
```

```
            cin >> key >> value; // For type 1 query, read the key and value
            insert(key, value); // Insert the key-value pair into the HashMap
        } else if (type == 2) {
            cin >> key; // For type 2 query, read the key
            cout << get(key) << endl; // Print the value associated with the key (or -1 if not found)
        } else {
            cin >> key; // For type 3 query, read the key
            del(key); // Delete the key-value pair from the HashMap
        }
    }

    return 0;
}
```