DS 8^{th} homework

58121102 Jiang Yuchu

Jan 14, 2023

P_{475} 3. Write a C++ function to delete the pair with key k from a hash table that uses linear probing.Show that simply setting the slot previously occupied by the deleted pair to empty does not solve the problem.How must `Get` Program 8.4 be modified so that a correct search is made in the situation when deletions are permitted? Where can a new key be inserted?

Answer:

1. The loop condition of the for loop in the get function is `ht[j]&&hj[j]->first!=k`. If you simply set the slot to be empty, then the for loop executes to the slot, and there is no loop anymore. If the k value you are looking for is originally sorted after the deleted value, then the problem arises.
2. A possible solution is to set the key in the deleted slot to a value that will never appear. Assuming that the value that will not appear is IMPOSSIBLE, the code for deletion is as follows:

```
template <class K,class E>
void del_key(const K& k) {
    int i = h(k);
    for (int j = i; ht[j] && ht[j]->first != k;) {
        j = (j + 1) & b;
        if (j == i) return 0;
    }
    if (ht[j]->first == k)
        ht[j] = IMPOSSIBLE;
}
```

3. In the Get function, the judgment condition of the for loop should be changed to `ht[j] && ht[j] != IMPOSSIBLE && ht[j] != k`
4. New keywords can be inserted into empty buckets and buckets where the keyword is IMPOSSIBLE

Experiment

P_{475} 6.

1 Experiment Goal

Develop a C++ hash table class in which overflows are resolved using a binary search tree. Use the division hash function with an odd divisor D and array doubling whenever the loading density exceeds a prespecified amount. Recall that in this context array doubling actually increases the size of the hash table from its current size $b = D$ to $2b + 1$.

2 Basic Idea

My C++ code implements a hash table class, in which overflow conflicts in the hash table are resolved using a binary search tree. The hash function used is the division hash function, with an odd divisor D. The hash table size is doubled using array doubling whenever the loading density exceeds a pre-specified threshold.

Specifically, my hash table class defines a private member variable called table, which is an array of type vector, and each element of the array is a pointer to the root node of a binary search tree. I used the vector and algorithm libraries of the STL to simplify the code.

The hash table class has three public member functions: insert, search, and remove. The insert function is used to insert a new key-value pair into the hash table, or update the value if the key already exists. The search function is used to query whether a key exists in the hash table, returning true if it exists, or false otherwise. The remove function is used to delete a key-value pair. When deleting a key-value pair, if a binary search tree exists in the corresponding bucket, the function needs to search for and remove the key in the tree.

- In the implementation of the hash table class, I use a helper function called resize to expand the size of the hash table when the loading density exceeds the threshold. This function reallocates memory and reinserts all the elements of the old hash table into the new hash table. In addition, I also define a TreeNode structure to represent a node in a binary search tree. It contains a key, a value, and two pointers to left and right child nodes.
- In the insert function, I first calculate the hash value of the given key, then find the corresponding bucket. If there is no binary search tree in the bucket, the key-value pair is directly inserted into the bucket; otherwise, the key-value pair is inserted into the binary search tree. When inserting a key-value pair into a binary search tree, I recursively search for the node according to the rules of the binary search tree until an empty node is found, and then insert the new node at that position.
- In the search function, I first calculate the hash value of the given key, then find the corresponding bucket. If there is no binary search tree in the bucket, the function returns false; otherwise, it recursively searches the binary search tree until the corresponding key is found or a leaf node is reached.
- In the remove function, I first calculate the hash value of the given key, then find the corresponding bucket. If there is no binary search tree in the bucket, the function returns; otherwise, it searches for the corresponding key in the binary search tree, and then deletes the node according to the deletion rule of the binary search tree. If the binary search tree is empty after deleting the node, the corresponding bucket is cleared.

3 Procedure

In the main function, I define a hash table object and demonstrate its use by inserting, searching, and removing key-value pairs. Otherwise, I defined D as 11.

```
1       int        D = 11;
2       HashTable hash_table(D);
3       hash_table.insert(7);
4       hash_table.insert(25);
5       hash_table.insert(18);
6       hash_table.insert(10);
7       hash_table.insert(13);
8       hash_table.insert(5);
9       hash_table.insert(4);
10      hash_table.insert(2);
11      hash_table.insert(3);
12      hash_table.insert(12);
13      hash_table.insert(8);
14      hash_table.insert(15);
15      hash_table.insert(14);
16      hash_table.insert(17);
17      hash_table.insert(9);
18      hash_table.insert(16);
19      hash_table.insert(19);
20      hash_table.insert(21);
21      hash_table.insert(24);
```

```
22      hash_table.insert(23);
23      hash_table.insert(20);
24      hash_table.insert(1);
25      hash_table.insert(22);
26      cout << hash_table.search(15) << endl;  // output: 1 (true)
27      hash_table.remove(15);
28      cout << hash_table.search(15) << endl;  // output: 0 (false)
```

4 Result and Analysis

See before. The code is in *p475-6.cpp*