Java → Implementation of basic algorithms → Various data structures → Hash table in Java

Theory: Hash table in Java

🕒 1 hour 0 / 5 problems solved

Skip this topic Start practicing

523 users solved this topic. Latest completion was 1 day ago.

A **hash table** is a structure that allows us to efficiently perform *insert*, *find*, and *remove* operations with data. In Java, this structure is represented by the `Hashtable<K, V>` class from the standard collections. In this topic, we will implement our own simplified version of a hash table to get a general idea of how it works under the hood.

§1. The structure of a hash table in Java

For simplicity, we will implement a hash table with the following properties:

- keys are integers, values might be of arbitrary type;
- the maximum size of a table is fixed;
- the *linear probing* technique is used to resolve collisions.

First, let's implement a class for storing table entries:

```
class TableEntry<T> {
    private final int key;
    private final T value;

    public TableEntry(int key, T value) {
        this.key = key;
        this.value = value;
    }

    public int getKey() {
        return key;
    }

    public T getValue() {
        return value;
    }
}
```

The `TableEntry<T>` is a generic class with two private fields. The first is an integer `key`, the other is a `value` of a generic type `T`. Also, the class has a constructor and getters for the fields.

Now, let's start implementing a hash table itself. It will be a public class with one generic parameter:

```
public class HashTable<T>
```

The class will contain two private fields:

```
private final int size;
private TableEntry[] table;
```

Since we assume that the size of a table is fixed, the corresponding field is specified as final.

A constructor of the class looks like this:

```
public HashTable(int size) {
    this.size = size;
    table = new TableEntry[size];
}
```

Current topic:

Hash table in Java ···

Topic depends on:

✗ Algorithms in Java ···

✗ Generics and Object ···

✗ Generic methods ···

✓ Hash table ···

Table of contents:

↑ Hash table in Java

§1. The structure of a hash table in Java

§2. Basic methods

§3. Overriding toString

§4. Example

§5. Summary

Feedback & Comments

It takes one parameter that stores the size of a table. The corresponding field of the class is initialized by that size and then a new array of the same size is allocated and assigned to the `table` field.

§2. Basic methods

The first method to implement is `findEntryIndex`. This is a private helper method that finds the index of entry with a specified key in a table. It will be used as a subroutine in other methods. Its implementation is the following:

```
private int findEntryIndex(int key) {
    int hash = key % size;

    while (!(table[hash] == null || table[hash].getKey() == key)) {
        hash = (hash + 1) % size;

        if (hash == key % size) {
            return -1;
        }
    }

    return hash;
}
```

The method uses the modulo division hash function and the *linear probing* technique to resolve collisions. It stops the searching either if the current entry is *null* or the specified key is found. Then, it returns a hash value that corresponds to the index of the found entry. If the table is full, the method returns -1.

Next, let's implement a `put` method, that inserts a new entry to a hash table:

```
public boolean put(int key, T value) {
    int idx = findEntryIndex(key);

    if (idx == -1) {
        return false;
    }

    table[idx] = new TableEntry(key, value);
    return true;
}
```

First, the method finds a place to insert a new entry using the `findEntryIndex` method. Then, if such a place is found, it puts a new entry to the table and returns ***true***. Otherwise, the method returns ***false*** indicating that the insertion is failed.

A `get` method finds and returns an entry with a specified key. It can be implemented as follows:

```
public T get(int key) {
    int idx = findEntryIndex(key);

    if (idx == -1 || table[idx] == null) {
        return null;
    }

    return (T) table[idx].getValue();
}
```

If the searching is successful, the method returns the value associated with the key. Otherwise, it returns `null`.

§3. Overriding toString

To conveniently print the content of a hash table, we will also override the `toString` method:

```
@Override
public String toString() {
    StringBuilder tableStringBuilder = new StringBuilder();

    for (int i = 0; i < table.length; i++) {
        if (table[i] == null) {
            tableStringBuilder.append(i + ": null");
        } else {
            tableStringBuilder.append(i + ": key=" + table[i].getKey()
                                   + ", value=" + table[i].getValue());
        }

        if (i < table.length - 1) {
            tableStringBuilder.append("\n");
        }
    }

    return tableStringBuilder.toString();
}
```

§4. Example

Let's consider an example of how the described hash table can be used:

```
HashTable<String> table = new HashTable(5);

table.put(21, "John");
table.put(33, "Tom");
table.put(42, "Alice");
table.put(10, "Mike");
table.put(54, "Kate");

System.out.println(table);
```

Here, we create a table of size 5 and put 5 entries to the table. After that, the content of the table looks like this:

```
0: key=10, value=Mike
1: key=21, value=John
2: key=42, value=Alice
3: key=33, value=Tom
4: key=54, value=Kate
```

Now, let's try to get some value from the table:

```
String name = table.get(42);
System.out.println(name); // Alice
```

Then, let's update some value that is already in the table:

```
if (table.put(21, "Ann")) {
    System.out.println(table);
}
```

This gives the following:

```
0: key=10, value=Mike
1: key=21, value=Ann // updated value
2: key=42, value=Alice
3: key=33, value=Tom
4: key=54, value=Kate
```

§5. Summary

In this topic, we have considered one possible implementation of a hash table in Java. Keep it in mind that our example is simplified. Real hash tables are implemented using not only a generic value but a generic key as well. Also, real implementations use more sophisticated and efficient approaches to resolve collisions. However, the provided example is enough to get a general idea of how hash tables work.

Report a typo

70 users liked this theory. **6** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (8) Hints (0) Useful links (1) Show discussion