Data Structures and Algorithms in Java

Searching: Symbol Tables

Outline

1. What is a Symbol Table?

2. API

What is a Symbol Table?

What is a Symbol Table?

A symbol table is a data structure for key-value pairs that supports two operations: insert (put) a new pair into the table and search (get) the value associated with a given key

What is a Symbol Table?

A symbol table is a data structure for key-value pairs that supports two operations: insert (put) a new pair into the table and search (get) the value associated with a given key

Applications

Application	Purpose	Key	Value
dictionary	find definition	word	definition
book index	find relevant pages	term	list of page numbers
file share	find song to download	name of song	computer ID
web search	find relevant web pages	keyword	list of page names
compiler	find type and value	variable name	type and value

What is a Symbol Table?

What is a Symbol Table?

Conventions:

What is a Symbol Table?

Conventions:

- No duplicate keys are allowed; when a client puts a key-value pair into a table already containing that key (and an associated value), the new value replaces the old one

What is a Symbol Table?

Conventions:

- No duplicate keys are allowed; when a client puts a key-value pair into a table already containing that key (and an associated value), the new value replaces the old one
- Keys/values must not be `null`

What is a Symbol Table?

Conventions:

- No duplicate keys are allowed; when a client puts a key-value pair into a table already containing that key (and an associated value), the new value replaces the old one
- Keys/values must not be `null`
- Deleting a key involves removing the key (and the associated value) from the table immediately

API

API

BasicST<Key, Value>

`boolean isEmpty()`	returns `true` if this symbol table is empty, and `false` otherwise
`int size()`	returns the number of key-value pairs in this symbol table
`void put(Key key, Value value)`	inserts the `key` and `value` pair into this symbol table
`Value get(Key key)`	returns the value associated with `key` in this symbol table, or `null`
`boolean contains(Key key)`	returns `true` if this symbol table contains `key`, and `false` otherwise
`void delete(Key key)`	deletes `key` and the associated value from this symbol table
`Iterable<Key> keys()`	returns all the keys in this symbol table

API

API

OrderedST<Key extends Comparable<Key>, Value>

`boolean isEmpty()`	returns `true` if this symbol table is empty, and `false` otherwise
`int size()`	returns the number of key-value pairs in this symbol table
`void put(Key key, Value value)`	inserts the `key` and `value` pair into this symbol table
`Value get(Key key)`	returns the value associated with `key` in this symbol table, or `null`
`boolean contains(Key key)`	returns `true` if this symbol table contains `key`, and `false` otherwise
`void delete(Key key)`	deletes `key` and the associated value from this symbol table
`Iterable<Key> keys()`	returns all the keys in this symbol table in sorted order
`Key min()`	returns the smallest key in this symbol table
`Key max()`	returns the largest key in this symbol table
`void deleteMin()`	deletes the smallest key and the associated value from this symbol table
`void deleteMax()`	deletes the largest key and the associated value from this symbol table
`Key floor(Key key)`	returns the largest key in this symbol table that is smaller than or equal to `key`
`Key ceiling(Key key)`	returns the smallest key in this symbol table that is greater than or equal to `key`
`int rank(Key key)`	returns the number of keys in this symbol table that are strictly smaller than `key`
`Key select(int k)`	returns the key in this symbol table with the rank `k`
`int size(Key lo, Key hi)`	returns the number of keys in this symbol table that are in the interval `[lo, hi]`
`Iterable<Key> keys(Key lo, Key hi)`	returns the keys in this symbol table that are in the interval `[lo, hi]` in sorted order

API

API

Program: `FrequencyCounter.java`

API

Program: `FrequencyCounter.java`

- Command-line input: *minLen* (int)

API

Program: `FrequencyCounter.java`

- Command-line input: *minLen* (int)
- Standard input: sequence of words

API

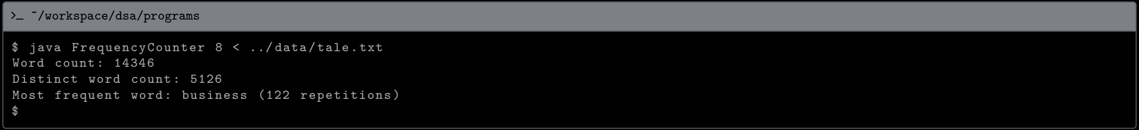
Program: `FrequencyCounter.java`

- Command-line input: *minLen* (int)
- Standard input: sequence of words
- Standard output: for the words that are at least as long as *minLen*, the total word count, the number of distinct words, and the most frequent word

API

Program: `FrequencyCounter.java`

- Command-line input: *minLen* (int)
- Standard input: sequence of words
- Standard output: for the words that are at least as long as *minLen*, the total word count, the number of distinct words, and the most frequent word



```
$ java FrequencyCounter 8 < ../data/tale.txt
Word count: 14346
Distinct word count: 5126
Most frequent word: business (122 repetitions)
$
```

API

API

</> FrequencyCounter.java

```java
import dsa.SeparateChainingHashST;
import stdlib.StdIn;
import stdlib.StdOut;

public class FrequencyCounter {
    public static void main(String[] args) {
        SeparateChainingHashST<String, Integer> st = new SeparateChainingHashST<>();
        int minLen = Integer.parseInt(args[0]);
        int distinct = 0, words = 0;
        while (!StdIn.isEmpty()) {
            String key = StdIn.readString();
            if (key.length() < minLen) {
                continue;
            }
            words++;
            if (st.contains(key)) {
                st.put(key, st.get(key) + 1);
            } else {
                st.put(key, 1);
                distinct++;
            }
        }
        int maxFreq = 0;
        String maxFreqWord = "";
        for (String word : st.keys()) {
            if (st.get(word) > maxFreq) {
                maxFreq = st.get(word);
                maxFreqWord = word;
            }
        }
        StdOut.println("Word count: " + words);
        StdOut.println("Distinct word count: " + distinct);
        StdOut.printf("Most frequent word: %s (%d repetitions)\n", maxFreqWord, maxFreq);
    }
}
```