Algorithms and Data Structures

B7. ADTs Map and Set

Gabriele Röger

University of Basel

April 24, 2024

Introduction

Reminder: Abstract Data Type

Abstract Data Type

Description of a data type, summarizing the possible data and the possible operations on this data.

- User perspective: How can I use the data type?
- In contrast to data structures, not specifying the concrete representation of the data.

Previous ADTs

What ADTs do we already know?

Previous ADTs

What ADTs do we already know?

- Stacks
- Queues
- Priority Queues

Dynamic Sets

- Mathematical set: unordered collection of distinct objects.
  - Can be finite or infinite.
  - Does not change.

Dynamic Sets

- Mathematical set: unordered collection of distinct objects.
  - Can be finite or infinite.
  - Does not change.
- A dynamic set in computer science is slightly different.
  - Can grow, shrink or otherwise change.
  - Finite.
  - Entries (keys) can sometimes be associated with satellite data.

Dynamic Sets

- Mathematical set: unordered collection of distinct objects.
  - Can be finite or infinite.
  - Does not change.
- A dynamic set in computer science is slightly different.
  - Can grow, shrink or otherwise change.
  - Finite.
  - Entries (keys) can sometimes be associated with satellite data.
- Now: Two ADTs for dynamic sets:
  - Map
  - Set

Map

Map

A map stores (key, value) pairs such that each possible key occurs at most once in the collection. It supports the following operations:

- Insert a given key and value. If the key is already present, update the associated value.
- Remove the entry for a given key.
- Lookup the entry for a given key (or return that there is none).

Also known as associative array, dictionary or symbol table.
Exact names of operations can differ.

Map

A map stores (key, value) pairs such that each possible key occurs at most once in the collection. It supports the following operations:

- Insert a given key and value. If the key is already present, update the associated value.
- Remove the entry for a given key.
- Lookup the entry for a given key (or return that there is none).

Also known as associative array, dictionary or symbol table.
Exact names of operations can differ.

Similar to arrays, but using keys instead of indices.

Question (Slido)

What data structure(s) could you use to implement a map?

Map: Data Structures and Running Times

The following data structures can easily be adapted to implement maps:

data structure	insertion	removal or lookup
	avg./worst	avg./worst
linked list	$O(1)/O(1)$	$O(n)/O(n)$
hash table	$O(1)/O(n)$	$O(1)/O(n)$
binary search tree	$O(\log n)/O(n)$	$O(\log n)/O(n)$
red-black tree	$O(\log n)/O(\log n)$	$O(\log n)/O(\log n)$

Maps in Java and Python

Java:

- Interface `Map`
- For example implemented by `HashMap` (hash table) and `TreeMap` (red-black tree).

```
Map<String, Integer> map = new TreeMap<>();
map.put("a key", 42);;
map.put("another key", 17)
Integer value = map.get("aKey");
```

Maps in Java and Python

Java:

- Interface `Map`
- For example implemented by `HashMap` (hash table) and `TreeMap` (red-black tree).

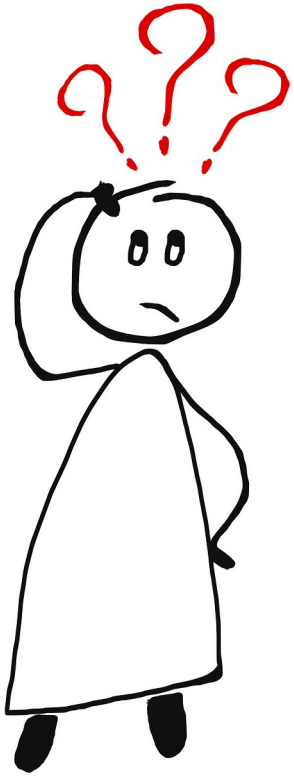
```java
Map<String, Integer> map = new TreeMap<>();
map.put("a key", 42);;
map.put("another key", 17)
Integer value = map.get("aKey");
```

Python:

- Built-in `dict` (hash table)

```python
map = dict()
map["a key"] = 42
map["another key"] = 17
# or alternatively:
# map = {"a key" : 42, "another key" : 17}
value = map["a key"]
```

Questions



Questions?

Set

Set

A set stores keys such that each possible key occurs at most once in the collection. It supports the following operations:

- Insert a given key into a set (if it is not already included).
- Remove the given key from a set.
- Lookup whether a given key is in a set.
- Iteration over all elements of a set in an arbitrary order.

Set

A set stores keys such that each possible key occurs at most once in the collection. It supports the following operations:

- Insert a given key into a set (if it is not already included).
- Remove the given key from a set.
- Lookup whether a given key is in a set.
- Iteration over all elements of a set in an arbitrary order.

In addition, there is often support for the following operators:

- Union of two sets.
- Intersection of two sets.
- Difference of two sets.

Exact names of operations can differ.

Data Structures

We can use the same data structures for sets as for maps.

- Do not store a value with the key.
- Implementation of operators union, intersection can be done based on the core operations or with highly specialized algorithms:
  - E.g., union, intersection and difference possible in $O(m \log(\frac{n}{m} + 1))$ for two red-black trees of sizes m and n (where $m \leq n$).

Sets in Java

Java:

- Interface `Set`
- For example implemented by `HashSet` (based on hash table) and `TreeSet` (based on red-black tree).

```
Set<Integer> nums1 = new HashSet<>();
Set<Integer> nums2 = new HashSet<>();
nums1.add(42);
nums1.add(17);
nums2.add(42);
nums2.add(13);
nums2.add(19);
nums2.remove(13);
nums1.retainAll(nums2); // intersection
if (nums1.contains(42)) {
  System.out.println("Found 42");
}
```

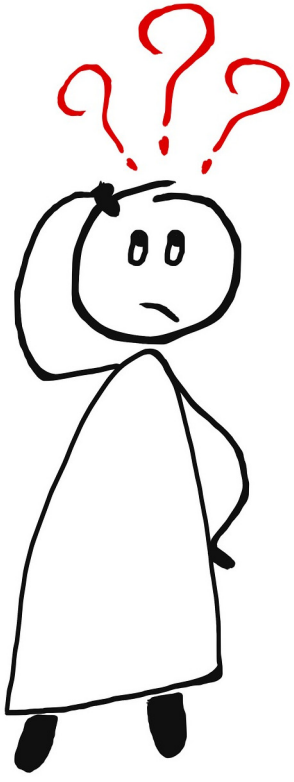
Sets in Python

Python:

- Built-ins `set` and `frozenset` (both based on hash tables; frozen sets are immutable and hashable)

```
s1 = set()
s1.add(42)
s1.add(17)
s2 = {42, 13, 19}
s2.remove(13)
s1 &= s2 # intersection
if 42 in s1:
    print("Found 42")
```

Questions



Questions?

Summary

Summary

- Maps and sets are abstract data types for dynamic sets.
  - Maps map keys to their associated values.
  - Sets only store elements.
  - Both are typically implemented based on hash tables or balanced trees (such as red-black trees).