Sets and Maps

Lecturer: Nataliia Stulova
Teaching assistant: Mohammadreza Hazirprasad
Recap on data structure properties

Restrictions on element values:

- usually none:
  - Arrays, Lists, Stacks, Queues
- duplicates allowed:
  - Graphs, General trees*
- absence of duplicates:
  - Binary search tree

Element access:

- **fast** - random (by index)
- **slow** - sequential (by iteration or following element links)

New today: key-value pairs (Maps)

**Continued today:** treating duplicated elements (Sets)
Maps
Map data structure

A data structure composed of \((key, value)\) pairs, such that each possible key appears at most once in the collection:

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>bananas</td>
<td>1</td>
</tr>
<tr>
<td>eggs</td>
<td>12</td>
</tr>
<tr>
<td>lemons</td>
<td>2</td>
</tr>
<tr>
<td>appless</td>
<td>3</td>
</tr>
<tr>
<td>oranges</td>
<td>2</td>
</tr>
</tbody>
</table>

Compare: “regular” array (keys are element indices, unique!)

<table>
<thead>
<tr>
<th>values</th>
<th>indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello</td>
<td>0 1 2 3 4</td>
</tr>
</tbody>
</table>

Other names: an associative array, symbol table, or dictionary
Map examples

- postal indices
- word dictionaries
- software configuration files
- ....

Use whenever you need to represent data as a tuple or simple structure and one of the fields holds unique values and can be used as a key

JSON format:

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "children": [],
  "spouse": null
}
```
Map operations

- add a pair to the collection
- remove a pair from the collection
- modify an existing value
- lookup a value by the key

Q: how to do lookup efficiently?
Map operations

- **add** a pair to the collection
- **remove** a pair from the collection
- **modify** an existing value
- **lookup** a value by the key

**Q:** how to do lookup efficiently?

**option 1:** search tree (previous lecture)

- Keys need to be comparable
Map operations

- add a pair to the collection
- remove a pair from the collection
- modify an existing value
- lookup a value by the key

Q: how to do lookup efficiently?

option 2: hash table
Hash table

**hash function** computes an index (a hash code), into an array of buckets (slots)

since hash function values are comparable - keys themselves do not have to be such!

E.g.: would you order by name, initials...?
java.util.Map

Reference javadoc: Map (Java SE 11 & JDK 11)

A library interface Map<K, V> that provides various useful operations on maps:

- containsKey()
- containsValue()
- get()
- put()
- remove()
- replace()
- keySet()
- values()

Map classes implementing this interface:

- HashMap<K, V>
- TreeMap<K, V>
- Hashtable<K, V>
- LinkedHashMap<K, V>
- ...

pay attention to: iteration order, nullness of keys and values (allowed or not)
Sets
Set data structure

A data structure composed of unique values:

- no ordering of the elements
- no duplication of elements

Use whenever you need to check if a value belongs to a set, or you need to reason about several collections of values: filter duplicates, count unique values....

Compare: “regular” array

<table>
<thead>
<tr>
<th>values</th>
<th>indices</th>
</tr>
</thead>
<tbody>
<tr>
<td>h e l l o</td>
<td>0 1 2 3 4</td>
</tr>
</tbody>
</table>

Respective set: \{h, e, l, o\}
Set operations

- intersection
- union
- symmetric difference
- difference
- complement
Multiset, or Bag

Bag data structure allows duplicates - and stores them as counts.

Uses:

- databases (SQL query results)
- NLP: stop words filtering, text similarity computation

Compare: “regular” array

<table>
<thead>
<tr>
<th>values</th>
<th>h</th>
<th>e</th>
<th>l</th>
<th>l</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>indices</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Respective bag:

\{1:h, 1:e, 2:l, 1:o\}
A library **interface** `Set<E>` that provides various useful operations on sets:

- `contains()`
- `add()`
- `remove()`
- ...

There are classes implementing sets in Java SE

**pay attention to:** iteration order, nullness of keys and values (allowed or not)

For multisets see third-party libraries:

- Apache Commons
- Google Guava
Practice
Exercise 1

associative array - abbreviation expansion

- read a CSV file (5-6 lines)
- in each line first word is a key, second is a value:

| WHO, World Health Organization |
| BBC, British Broadcasting Corporation |

- use 2-3 different Map implementations to store this data
- print out tuples: as you add and as you iterate, compare the order

I/O

- Input: File IO for reading data
- Output: Stream IO to print

Tests

-
Exercise 2

word counts and multisets

- read a sentence, split on whitespace to get words
- add words to a bag
- print 3 words that occur most often

I/O

- Input: manual or System.in
- Output: System.out to print output

Tests

-