Data Structures and Algorithms
Spring 2020
Assignment 1

IT University Innopolis

Contents

1 About this homework

1.1 Coding part

For practical (coding) part you have to provide you program as a single Java class file that contains all necessary definitions and a `main` method.

The program should read from standard input and write to standard output unless mentioned otherwise.

1.1.1 Preliminary tests

For some coding parts a CodeForces or CodeTest system will be used for preliminary tests. You will have to specify a submission number in those systems as a proof that your solution is correct. Only correct solutions will be assessed further by TAs.

1.1.2 Code style and documentation

The source code should contain adequate internal documentation in the form of comments. Internal documentation should explain how the code has been tested, e.g. the various scenarios addressed in the test cases.

Do not forget to include your name in the internal documentation.

All important exceptions should be handled properly.

Bonus points might be awarded for elegant solutions, great documentation and the coverage of test cases. However, these bonus points will only be able to cancel the effects of penalties.

1.2 Theoretical part

Solutions to theoretical exercises have to be elaborate and clear. Write all solutions in a single document and make it a PDF for submitting.

Do not forget to include your name in the document.

1.3 Submission

You are required to submit your solutions to all parts via Moodle.

For this assignment you will need to submit:

- Java class file(s) for the coding parts;
- link(s) to accepted submission(s) in CodeForces as a proof of correctness;
- a PDF file with solutions for theoretical parts.

Submit files as a single archive with your name and group number. E.g. `BS18-00_Ivanov_Ivan.zip`.

2 Coding part

2.1 Implementing List ADT

Implement the List ADT using both Dynamic-Array and Doubly-Linked-List implementations. Implementations should be generic and support storing elements of arbitrary types.

Note: you can declare List ADT in Java as either an interface or an abstract class.

The List ADT should support the following operations:

1. `isEmpty()` — check if the list is empty;
2. `add(i, e)` — add element `e` at position `i`;
3. `addFirst(e)` — add element `e` to the start of the list;
4. `addLast(e)` — add element `e` to the end of the list;
5. `delete(e)` — delete element `e` if it exists in the list;
6. `delete(i)` — delete element at position `i`;
7. `deleteFirst()` — remove the first element from the list;
8. `deleteLast()` — remove the last element from the list;
9. `set(i, e)` — replace element at position `i` with new element `e`;
10. `get(i)` — retrieve element at position `i`.

Both implementations should provide clear documentation for every method, including time complexity using big-O or Θ notation.

For doubly-linked list implementation you can choose to use sentinel or not.

2.2 Balanced parentheses

Implement a program that reads a sequence of tokens and determines whether parenthesis (`()`), brackets (`[]`) and braces (`{}`) are properly balanced.

If input sequence has properly balanced delimiters then just output

```
Input is properly balanced.
```

Otherwise your program should specify the problematic location (line and column numbers) in the input and what symbol was expected. For example for this input

```
function random () {
```

```
  return 4]
}
```

Your program should output this message:

```
Error in line 2, column 11: expected '}', but got ']'.
```

In this exercise you are allowed to reuse Stack implementation from labs or implement a new one from scratch.

2.3 Phonebook

Write a program that manages a phonebook. Input for your program is a series of commands. Every command can either update the phonebook or query it:

- `ADD <Contact name>,<phone>` — add a phone number `<phone>` to contact `<Contact name>`; the contact should be created if not exists; examples:
  - `ADD Ivan Ivanov,+79991234567`
  - `ADD Bro,89990123456`
  - `ADD Ivan Ivanov,+71234567890`
- `DELETE <Contact name>` — delete entire contact from the phonebook; examples:
  - `DELETE Ivan Ivanov`
- `DELETE <Contact name>,<phone>` — delete a specific phone number from a contact; if a contact does not exist or it does not have specified phone number — do nothing; examples:
  - `DELETE Ivan Ivanov,+79991234567`
- `FIND <Contact name>` — lookup contact info; this is the only command that has output:
  - if contact is not found (or has no associated phone numbers) output `No contact info found for <Contact name>`, e.g.:
    ```
    FIND Petr Petrov
    No contact info found for Petr Petrov
    ```
  - otherwise output all phone numbers associated with the contact, e.g.:
    ```
    FIND Ivan Ivanov
    Found 2 phone numbers for Ivan Ivanov: +71234567890 +79991234567
    ```

3 Theoretical part

3.1 Big O notation

Prove or disprove the following statements using the definition of big O notation:

1. $\frac{n^2}{3} - 3n = O(n^2)$
2. $k_1\, n^2 + k_2\, n + k_3 = O(n^2)$
3. $3^n = O(2^n)$
4. $0.001\, n \log n - 2000\, n + 6 = O(n \log n)$

3.2 Hashing

Consider a hash table of size 7 with hash function $h(k) = k \,\mathbf{mod}\, 7$. Draw the table that results after inserting, in the given order, the following values: $19, 26, 13, 48, 17$ for each of the scenarios below:

1. When collisions are handled by **separate chaining**.
2. When collisions are handled by **linear probing**.
3. When collisions are handled by **double hashing** using the secondary hash function $h'(k) = 5 - (k \,\mathbf{mod}\, 5)$.