Data Structures and Algorithms
Spring 2019
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.

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; - submission number or a link to submission in CodeForces as a proof of correctness; - a PDF file with solutions to theoretical parts.

Submit files as a single archive with your name and group number. E.g. `BS18-01_Ivanov_Ivan.zip`.

1.4 Plagiarism

Plagiarism will not be tolerated and a plagiarised code will be heavily penalised for all parties involved. Remember that you learn nothing when you copy someone else's work which defeats the purpose of the exercise!

2 Working with Lists

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 implement 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`.

2.2 Sorting Lists

Implement a sorting algorithm on your List ADT. You are required to implement either

- insertion sorting algorithm from Cormen's book (and lecture);
- selection sorting algorithm as discussed during labs.

Note: sorting algorithm should work whenever there is a way to compare elements of a list.

2.3 Tournament Rankings

Write a program that uses your List ADT and sorting algorithm to produce a ranked (i.e. sorted) list of teams in a series of football tournaments. The program should read the tournament name, team names, and games played, and it should output the tournament standings.

A team wins a game if it scores more goals than its opponent, and loses if it scores fewer goals. Both teams tie if they score the same number of goals. A team earns 3 points for each win, 1 point for each tie, and 0 points for each loss.

Teams are ranked according to these rules (in this order):

- most points earned;
- most wins;
- most goal difference (i.e. goals scored – goals against);
- case-insensitive lexicographic order.

Bonus Question: if you were only interested in top K teams, which sorting algorithm can be easily modified to provide that kind of information?

2.3.1 Input

The first line of input will be an integer N in a line alone ($0 < N \leq 1000$). Then follow N tournament descriptions, each beginning with a tournament name. These names can be any combination of at most 100 letters, digits, spaces, etc., on a single line. The next line will contain a number T ($1 < T \leq 30$), which stands for the number of teams participating in this tournament. Then follow T lines, each containing one team name. Team names consist of at most 30 characters and may contain any character with ASCII code greater than or equal to 32 (space), except for `"#"` and `":"` characters. Following the team names, there will be a non-negative integer G ($0 \leq G \leq 1000$) on a single line which stands for the number of games already played in this tournament. G lines then follow with the results of games played in the format:

```
<team_name_1>#<goals1>:<goals2>#<team_name_2>
```

For instance, `Team A#3:1#Team B` means that in a game between «Team A» and «Team B», «Team A» scored 3 goals and «Team B» scored 1. All goals will be non-negative integers less than 20.

You may assume that all team names in game results will have appeared in the team names section, and that no team will play against itself.

Sample input:

```
2
World Cup 1998 - Group A
4
Brazil
Norway
Morocco
Scotland
6
Brazil#2:1#Scotland
Norway#2:2#Morocco
Scotland#1:1#Norway
Brazil#3:0#Morocco
Morocco#3:0#Scotland
Brazil#1:2#Norway
Some strange tournament
5
Team A
Team B
Team C
Team D
Team E
5
Team A#1:1#Team B
Team A#2:2#Team C
Team A#0:0#Team D
Team E#2:1#Team C
Team E#1:2#Team D
```

2.3.2 Output

For each tournament, you must output the tournament name followed by standings for that tournament. The tournament name should have its own line and then T lines with standings should follow. The output format for each line is shown below:

`[a]) [Team_name] [b]p, [c]g ([d]-[e]-[f]), [g]gd ([h]-[i])`

where

- `[a]` is team rank,
- `[b]` is the total points earned,
- `[c]` is the number of games played,
- `[d]` is wins,
- `[e]` is ties,
- `[f]` is losses,
- `[g]` is goal difference,
- `[h]` is goals scored and
- `[i]` is goals against.

There must be a single blank space between fields and a single blank line between tournaments.

For example, the correct output for the sample input above would be:

```
World Cup 1998 - Group A
1) Brazil 6p, 3g (2-0-1), 3gd (6-3)
2) Norway 5p, 3g (1-2-0), 1gd (5-4)
3) Morocco 4p, 3g (1-1-1), 0gd (5-5)
4) Scotland 1p, 3g (0-1-2), -4gd (2-6)

Some strange tournament
1) Team D 4p, 2g (1-1-0), 1gd (2-1)
2) Team E 3p, 2g (1-0-1), 0gd (3-3)
3) Team A 3p, 3g (0-3-0), 0gd (3-3)
4) Team B 1p, 1g (0-1-0), 0gd (1-1)
5) Team C 1p, 2g (0-1-1), -1gd (3-4)
```

3 Asymptotic Notation

3.1 Array containment

Consider the problem of determining if every value in an array A is also in an array B (it is possible that B has values that are not in A), where A and B are two arrays of size n with integer values. The pseudocode for solving this problem is shown as follows:

```
    isContainedArray(A, B)
1.      for i ← 0 to n-1 do
2.          contained ← FALSE
3.          for j ← 0 to n-1 do
4.              if A[i] = B[j] then contained ← TRUE
5.          if not contained then return FALSE
6.      return TRUE
```

Answer the following questions about the `isContainedArray` algorithm:

a. Show that the worst-case runtime of the algorithm is $O(n^2)$. For full grade, you must justify your calculation with references to the algorithm (the number of lines will help you).

b. Suppose $A = \langle 1, 1, 1, \ldots, 1 \rangle$. Describe an array B which guarantees that the algorithm has runtime $\Omega(n^2)$. Support your answer.

c. Is the worst-case runtime of the algorithm $\Theta(n^2)$? Support your answer.