Practical 1

COS212

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science

Deadline: 03/03/2023 at 23:59

Marks: 160

1 General instructions:

- This assignment should be completed individually; no group effort is allowed.
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of Java's built-in data structures. Doing so will result in a mark of zero. You may only make use of native arrays where applicable. If you require additional data structures, you must implement them yourself.
- If your code does not compile, you will be awarded a zero mark. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- All submissions will be checked for plagiarism.
- Read the entire assignment before you start coding.
- You will be afforded five upload opportunities.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.** Also, note that the OOP principle of code reuse does not mean you should copy and adapt code to suit your solution.

3 Outcomes

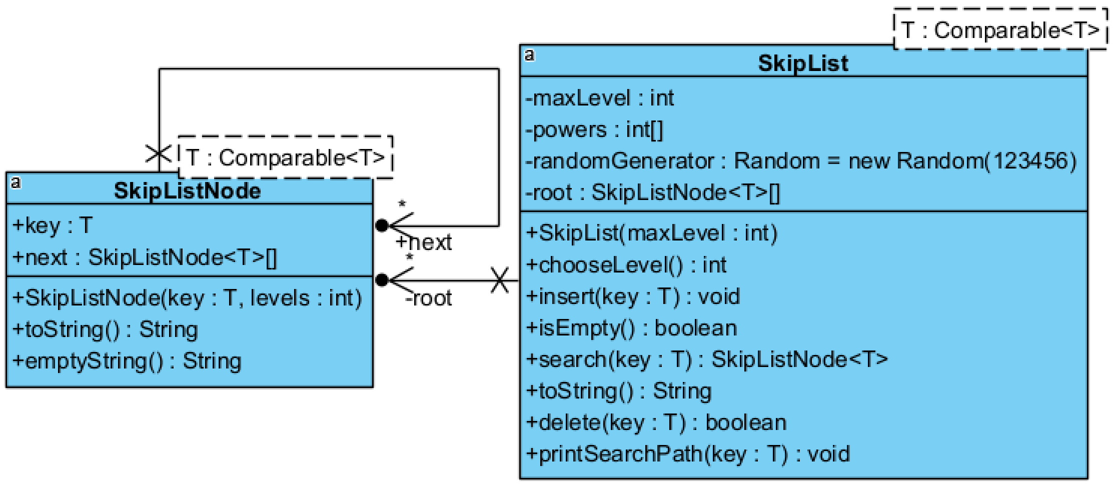
Upon completing this assignment, you will have implemented the following:

- A skip list using Java's generics.
- Functions to print out the structure of the data structure, which can be used to debug the structure and make sure the links are correct.

4 Background

Skip lists are data structures designed to address one of the biggest flaws of linked lists. This is because searching is inefficient since it always has to be done sequentially. To address this, skip lists have multiple links, and the upper layers have fewer nodes connected, allowing a faster search since nodes can be skipped. This still has some drawbacks since a random number generator is used to decide the node level on insertion, which might lead to inefficient searches, but in the general case, it is more efficient than normally linked lists. Skip lists must always maintain a sorted order used in the searching algorithm, and in the normal version, duplicate values are not allowed. This assessment has modified this slightly and allows duplicate values, as explained in the later sections.

5 Tasks



T : Comparable<T>
a
SkipListNode
+key : T
+next : SkipListNode<T>[]
+SkipListNode(key : T, levels : int)
+toString() : String
+emptyString() : String
*
+next
*
-root
T : Comparable<T>
a
SkipList
-maxLevel : int
-powers : int[]
-randomGenerator : Random = new Random(123456)
-root : SkipListNode<T>[]
+SkipList(maxLevel : int)
+chooseLevel() : int
+insert(key : T) : void
+isEmpty() : boolean
+search(key : T) : SkipListNode<T>
+toString() : String
+delete(key : T) : boolean
+printSearchPath(key : T) : void

Figure 1: UML diagram

5.1 SkipListNode<T>

This is a generic class used as a node in the skip list. The generic variable T extends the Comparable interface, which allows the use of compareTo() and is used to keep the list sorted.

- Members
  - key:T
    - This is the value of the node.
  - next:SkipListNode<T>[]
    - This is the array used as the next links for this node.
    - *Note: The length of this array is not stored but can be retrieved using the default Java member "length".*
- Functions
  - SkipListNode(key:T,levels:int)
    - This is the constructor for the node class.
    - Set the key variable to the passed-in parameter.
    - Initialise the next array to the size passed in with all the links set to null.
  - toString():String
    - This is a string representation of the node. The toString function is called on the key variable and is enclosed in square brackets. You may assume the key will never be null.
    - Ex if the key is the integer 2:

```
[2]
```

- emptyString():String
  - This function is used when this node's level is not the max level.
  - This should be a string consisting only of '-'. The length should be the same as the toString function.
  - This function will be used in the toString function of SkipList<T> to help with printing the list.
  - Ex if the key is the integer 2:

```
---
```

5.2 SkipList<T>

This is a generic class used as a skip list. The generic variable T extends the Comparable interface, which allows the use of compareTo() and is used to keep the list sorted.

- Members
  - maxLevel:int
    - This is the maximum level of the skip list. Note that due to randomisation, it is possible for there to be no nodes in the list with this level.
  - powers:int[]
    - This is an array of size maxLevel. It is used in the chooseLevel() function to decide the number of levels in new nodes.
    - Ex: In a list with maxLevel of 5, the powers array will look as follows:

```
[1,17,25,29,31]
```

  - randomGenerator:Random
    - This is used to generate the random numbers needed for choosing the number of levels for new nodes.
    - The generator uses the seed *123456*. This is done for marking purposes, ensuring that running the same input multiple times will always give the same output. This variable is already initialised and can be used with the nextInt function to generate the next number.
  - root:SkipListNode<T>[]
    - This is an array of nodes used as the root of the list. Since this is the root, the size of the array is maxLevel.

- Functions
  - SkipList(maxLevel:int)
    - This is the constructor for a skip list.
    - First, set the maxLevel variable to the passed-in parameter.
    - Initialise the root variable to an array of size maxLevel. The links should start as null.
    - The powers array should also be initialised to match the rule described in the lectures.
    - *You are allowed to use Math.pow()*
  - chooseLevel():int
    - This function uses the same algorithm as the one explained in the lectures.
    - A number between 1 and $2^{maxLevel}$ is generated. This number is then compared to the entries in the powers array. This level is returned as soon as a number larger than the random number is encountered.
  - insert(key:T):void
    - This function is used to insert a new value in the list.
    - The chooseLevel function should be called to determine the level of the newNode. **Very important: Don't call the chooseLevel function more than once since this will change the random numbers and result in different outputs to the memo.**
    - Unlike normal skip lists, this list can include duplicate values. The new node should be inserted at the end when inserting duplicate values. *Note that this does not mean the end of the skip list but the end of the duplicate values. The list should still always be sorted. If there are duplicate values, the duplicate values are then sorted by insertion order.*
  - isEmpty():boolean
    - If the list is empty, return true; otherwise, return false.
  - search(key:T):SkipListNode<T>
    - Searches through the list and returns the node with the same value as the passed-in key. If the key is not found, return null.
    - Figure 2 shows three examples of the search function. The search values are 1,2, and 3 respectively.

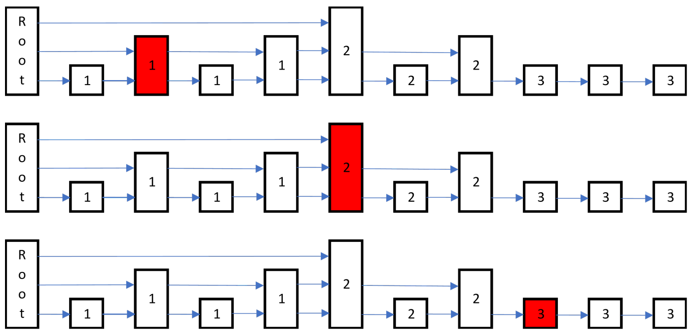


Figure 2: Searching with duplicate values example. The returned node is highlighted in red.

- toString():String
  - This is a String representation of the list. All of the links are shown, making it easier to debug and clear which links are incorrect.
  - An example of a skip list with maxLevel 3 is shown below.

```
[Lvl 2]------>[4]------>[8]->[10]
[Lvl 1]------>[4]------>[8]->[10]------------------->[18]
[Lvl 0]->[2]->[4]->[6]->[8]->[10]->[12]->[14]->[16]->[18]
```

  - There are multiple ways to get this output. As such, you can use any algorithm to achieve this output. Note that character-by-character comparison is used in marking, so make sure every character matches the output of the given main.
  - Pseudocode is given in Section 8, although it is recommended to try and figure this out for yourself before looking at the pseudocode since it is an essential skill to be able to debug the data structures in this course. Use the example above and the main combined with the output given to see what is needed for this function.
- delete(key:T):boolean
  - This function removes a node with the value of the passed-in parameter.
  - Since duplicate values are allowed deleting a node deletes the node, which is returned by the search function.
- printSearchPath(key:T):void
  - This function prints out the path that the search function uses.
  - Every node visited is printed inside square brackets, which are then concatenated together.
  - If the key is not in the list, don't print anything.

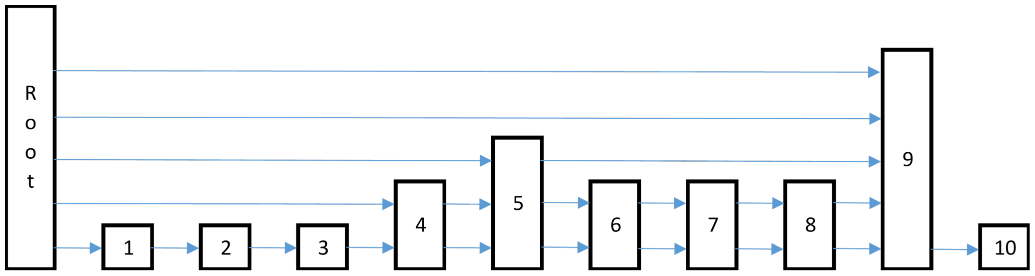


Figure 3: Example list for printSearchPath

  - Using Figure 3, printing the path to 7 gives the following:

```
[9][9][5][9][6][7]
```

  - Using Figure 3, printing the path to 1 gives the following:

```
[9][9][5][4][1]
```

6 Submission instructions

Do not submit any custom function classes.
Your code should be able to be compiled and run with the following commands:
javac *.java
java Main

Once you are satisfied that everything is working, you must create an archive of all your Java code into one archive called uXXXXXXXX.{tar.gz/tar/zip} where X is your student number. Submit your code for marking under the appropriate link before the deadline. **Make sure your archive consists of only java files and no folders or subfolders.**

Please ensure that you thoroughly test your code before submitting it. Just because your code runs with local testing does not mean that your code works for every situation. **DO NOT USE FITCH FORK AS A DEBUGGER**

7 Submission checklist

- SkipListNode.java
- SkipList.java

8 Psuedocode

```
Create a String array called levels with a size of maxLevel.
Initialise each string to [LvlX], with X being the current level.

while(Loop through the bottom layer of the skip list with a variable called ptr){
    for(Loop through the levels array with a variable called i){
        if(i is less than the level of ptr){
            Append "->" and ptr.toString to levels[i]
        }else{
            Append "--" and ptr.emptyString to levels[i]
        }
    }
}

Remove the excess "-" at the end of each string to ensure that each string ends
   with a "]"

Start from the end of the levels array and loop to the start of the array. Split
   the Strings using a new line but do not add one after the last line.
```