Lab Report

Exercise 12: Scrabble Cheater Deluxe

Name		Titel des Kurses	Datum
Juri Wiechmann Bartholomäus Berresheim	571085 568624	Prof. Dr. Weber-Wulff Info 2 Group 2	16.01.2020

Index

Introduction

In this weeks lab, we concluded our work on the Scrabble Cheater.

Pre-lab

1.What was a permutation? How can you generate all permutations of the characters in a String? What if some of the letters are the same.

A permutation corresponds to any of the various ways in which a set of things can be ordered.
(https://dictionary.cambridge.org/dictionary/english/permutation).
We can generate them by .

2. Given a selection of *n* pizza toppings - how can you generate a list of all of the different k-toppings where *k*<*n*? Hint: Look at the binomial coefficient. Write a method that takes a String of *n* characters and returns an array of *k*-character Strings that are all characters in the original string.Choose one of your solutions to Exercise 11 from last week (or borrow a working one from someone. Remember to give them credit!).

.

Assignments

1. Choose one of your solutions to Exercise 11 from last week (or borrow a working one from someone. Remember to give them credit!).

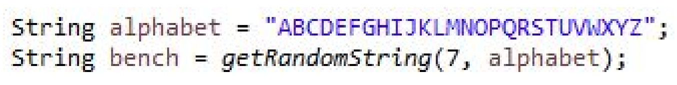
We used our own code from the previous lab.

2. If you didn't do this last week, write a method `bool isPermutation (String a, String b) {...}` that determines if a and b are permutations. Use this in your output from the cheater so that only permutations of the input string are printed out, and not all of the collisions.

We already made a similar method in our previous lab, called permute(), that outputs every permutation from a given input String.

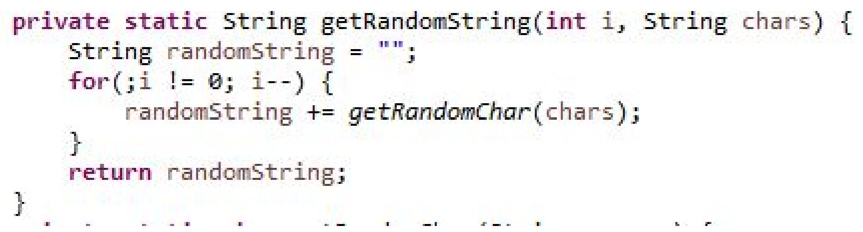
3. Adapt your main method to generate a random selection of seven letters to start the cheater with.

For this task, we first created 2 Strings: “alphabet” that contains every letter of the alphabet, and “bench” that has a calling of the getRandomString() method, that has “7” and “alphabet” as inputs.



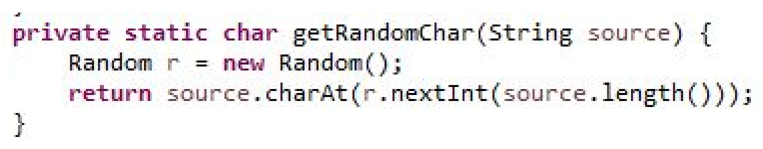
```
String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
String bench = getRandomString(7, alphabet);
```

The getRandomString() method is a new method we added for this task, it takes 2 inputs: an int “i” that gives the wanted size of the output, and a String “chars” that gives us the chars to choose from. In the method, we have a String “randomString” and a for loop that loops for “i” times. In the loop we add a random Character to “randomString”. After which, we return “randomString”.



```java
private static String getRandomString(int i, String chars) {
    String randomString = "";
    for(;i != 0; i--) {
        randomString += getRandomChar(chars);
    }
    return randomString;
}
```

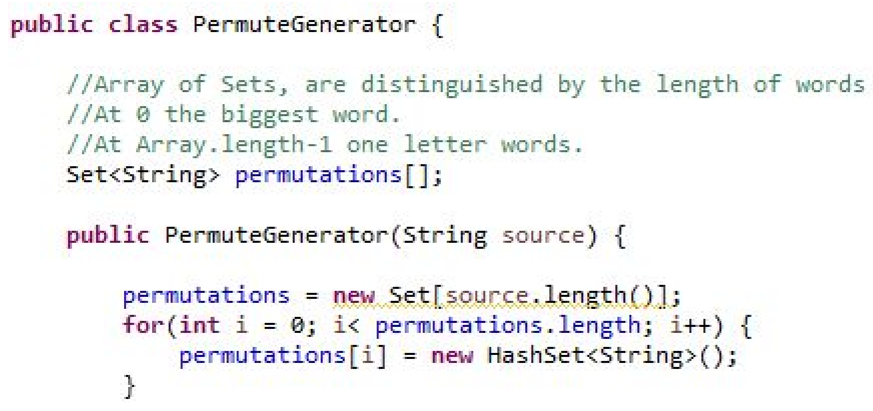
To get the random char, we created a method getRandomChar() that takes a String “source” as input. In it we take a random char from “source” and return it.



```java
private static char getRandomChar(String source) {
    Random r = new Random();
    return source.charAt(r.nextInt(source.length()));
}
```

4. Make a class that upon instantiation with a given String of characters, determines all of the Strings that are substrings in the sense that they only contain letters from the given String, with multiples only up to the number of multiples available. The order of the letters is irrelevant, so this is a bag. For example with 4 letters "JAVA" this would be {"AAJV", "AJV", "AAJ", "AAV", "AA", "AJ", "AV", "JV"}. Don't worry about single letters. Your finger exercise should come in handy here.

For this task, we created a class called PermuteGenerator. In it we created a Set field “permutations” and a constructor that takes a String “source” as input. In the constructor, we first set permutations to a new Set with the size of “source”.length(). After that, we add a new HashSet at each index of “permutations” using a for loop.



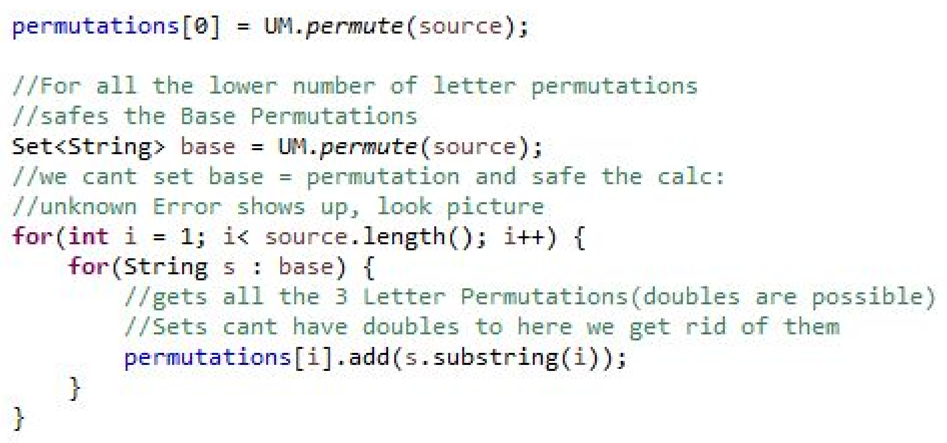
```java
public class PermuteGenerator {

    //Array of Sets, are distinguished by the length of words
    //At 0 the biggest word.
    //At Array.length-1 one letter words.
    Set<String> permutations[];

    public PermuteGenerator(String source) {

        permutations = new Set[source.length()];
        for(int i = 0; i< permutations.length; i++) {
            permutations[i] = new HashSet<String>();
        }
```

After the loop, we add a calling of the permute() method with “source” as input, to “permutations” at the first index. This adds all permutations of the same size as “source” to “permutations”. After that, we have a nested loop, the first one goes on for the length of “source” -1. The inner loop, we add all the smaller permutations to “permutations”.

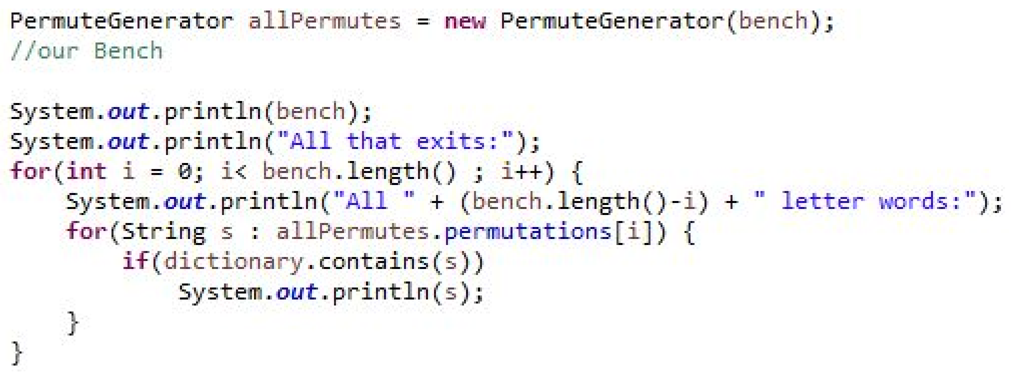


```
permutations[0] = UM.permute(source);

//For all the lower number of letter permutations
//safes the Base Permutations
Set<String> base = UM.permute(source);
//we cant set base = permutation and safe the calc:
//unknown Error shows up, look picture
for(int i = 1; i< source.length(); i++) {
    for(String s : base) {
        //gets all the 3 Letter Permutations(doubles are possible)
        //Sets cant have doubles to here we get rid of them
        permutations[i].add(s.substring(i));
    }
}
```

5.Now set up the Scrabble Cheater DeLuxe: read in 7 letters, split them into collections of 7-, then 6-, then 5-, ... words contained in the input bag of letters. Look up each word in each collection in the corresponding dictionary. If you find something, output it.

To do this, we created an Object of type RandomGenerator “allPermutes” and used “bench” to initiate the constructor. After that, we print out “bench”, followed by a nested loop in which we compare all the permutations we generated using the RandomGenerator, and compare them with the words contained in our dictionary. If one of the permutations is a real word, we print it out.



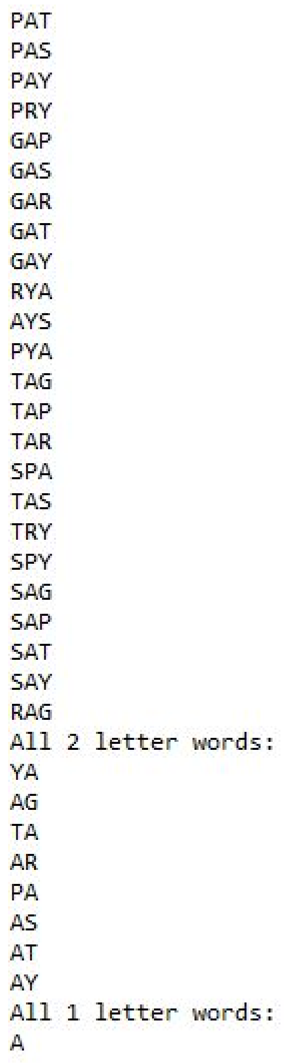
```
PermuteGenerator allPermutes = new PermuteGenerator(bench);
//our Bench

System.out.println(bench);
System.out.println("All that exits:");
for(int i = 0; i< bench.length() ; i++) {
    System.out.println("All " + (bench.length()-i) + " letter words:");
    for(String s : allPermutes.permutations[i]) {
        if(dictionary.contains(s))
            System.out.println(s);
    }
}
```

Now once executed, we get:

```
TableSize: 24593
File loader complete
Amount of empty lists: 11.26%
GAYPRST
All that exits:
All 7 letter words:
All 6 letter words:
PASTRY
All 5 letter words:
ARTSY
RASPY
PASTY
TARPS
SATYR
PARTY
PARTS
PRAYS
PRATS
SPRAY
SPRAT
SPRAG
GRAPY
GRASP
TRAPS
STAGY
GRAYS
PATSY
STRAY
STRAP
TRAYS
All 4 letter words:
SPRY
PRAT
PRAY
GRAY
ARTY
ARTS
GRAT
TRAP
TRAY
YAPS
RAGS
PART
PARS
PYAS
```

```
PATY
PATS
PAST
TAGS
GAPY
RAPS
RAPT
GAPS
GASP
GAST
RYAS
GARS
RATS
RASP
PAYS
GATS
TAPS
TARS
TARP
GAYS
RAYS
GYPS
STAG
STAR
STAY
TSAR
SPAR
SPAY
SPAT
SAGY
All 3 letter words:
RAP
RAT
RAS
APT
RAY
STY
YAG
ARS
ART
YAP
YAS
YAR
ASP
PAR
```



```
PAT
PAS
PAY
PRY
GAP
GAS
GAR
GAT
GAY
RYA
AYS
PYA
TAG
TAP
TAR
SPA
TAS
TRY
SPY
SAG
SAP
SAT
SAY
RAG
All 2 letter words:
YA
AG
TA
AR
PA
AS
AT
AY
All 1 letter words:
A
```

As we can see, the program is working as intended.

Reflections

Juri:

We tried to save as much time as we could to learn for the exams. But even with this time pressure we were happy with our result.

Bartholomäus:

This weeks lab was rather easy, since we had already laid a good ground work in our previous lab.

Code

```java
package ubung12;

public interface HashTable {
	public void resize();
	public void add(String item);
	public boolean contains(String item);
	public void reset();
	public String toString();
	public void print();
}

package ubung12;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Set;
import java.util.TreeSet;

import ubung9.Point;

public interface UM {

	public static String deleteSpaces(String s) {

		return s.replace("\\s+", "");
	}

	public static boolean StringisNumber(String number) {

		if(number.startsWith("."))
			return false;

		char[] cNumber = number.toCharArray();

		int i = 0;
		try {
			if(cNumber[0] == '-' && CharisNumberHEX(cNumber[1]))
				i = 1;
		}
		catch (Exception e) {}

		for (; i < cNumber.length ; i++) {
			if(!(CharisNumberHEX(cNumber[i]) || cNumber[i] == '.'))
				return false;
		}
		return true;
	}
	public static boolean CharisNumberHEX(char number) {
		if( number > 47 && number < 58 ||
			number > 64 && number < 71)
			return true;
		else
			return false;
	}
	public static String setSpacesEachToken(String S) {
		S = deleteSpaces(S);
		char [] S_Array = S.toCharArray();

		//If the input String has 2 dots in a row its wrong Tested here.
		int count = 0;
		for(char E : S_Array) {
			if(E == '.')
				count++;
			else
				count = 0;
			if(count >1)
				return null;
		}

		String token = "";
		S = "";
		for(int i = 0; i< S_Array.length;i++) {
			token = "";

			//negativ numbers
			if(	S_Array[i] == '-') {
				token += S_Array[i];

				try {
					//if its an neg followed by a number
					if(CharisNumberHEX(S_Array[i+1])){
						//If - is the first digit:
						if(i==0) {
							while(CharisNumberHEX(S_Array[i+1]) ||
S_Array[i+1] == '.') {

								i++;
								token += S_Array[i];
							}
						}
						else {
```

```
                                                //if the - is not at the first digit
                                                if(CharisOperator(S_Array[i-1])) {


while(CharisNumberHEX(S_Array[i+1]) || S_Array[i+1] == '.') {
                                                        i++;
                                                        token += S_Array[i];

                                                    }
                                                }
                                            }
                                        }
                                        //if the neg is followed by an (
                                        else if(S_Array[i+1] == '(') {
                                            i++;
                                            token += S_Array[i];
                                        }
                                    }
                                    catch(Exception e) {}
                                    S += token + " ";
                                    continue;
                                }
                                //operatoren
                                if( i > 0  && CharisOperator(S_Array[i])) {
                                    token += S_Array[i];
                                    S += token + " ";
                                    continue;
                                }
                                //positive Zahlen
                                else if(CharisNumberHEX(S_Array[i])){
                                    token = Character.toString(S_Array[i]);
                                    try {
                                        while(CharisNumberHEX(S_Array[i+1]) || S_Array[i+1]
== '.') {
                                            i++;
                                            token += S_Array[i];
                                        }
                                    }
                                    catch(Exception e) {}
                                    S += token + " ";
                                    continue;
                                }
                                else {
                                    //If input is wrong.
                                    return null;
                                }
```

```
		}


		return S.stripTrailing();
	}
	public static boolean CharisToken(char c) {
		if(CharisNumberHEX(c) || CharisOperator(c)) {
			return true;
		}
		else
			return false;
	}
	public static boolean CharisOperator(char c) {
		if( c == '+' ||
			c == '-' ||
			c == 'x' ||
			c == '*' ||
			c == '/' ||
			c == '^' ||
			c == '(' ||
			c == ')' ||
			c == '=')
			return true;
		else
			return false;
	}
	public static String getSep(String input) {
		//if you want too really seaching for an sepperator u need a more complicated
way, this is the bugded version
		if(input.contains("/"))
			return  "/";
		else if(input.contains("-"))
			return  "-";
		else if(input.contains("."))
			return  ".";
		else if(input.contains("_"))
			return  "_";
		return null;
	}
	public static int twoPointsDistance(int x1, int y1, int x2, int y2) {

		return (int) Math.sqrt((y2 - y1) * (y2 - y1) + (x2 - x1) * (x2 - x1));
	}
	public static Point getMidpoint(Point p1, Point p2) {

		return new Point((p1.getX()+p2.getX())/2,(p1.getY()+p2.getY())/2);
	}
	//doesnt work right now
	public static boolean isPrime (BigInteger n) {

		BigInteger THREE = BigInteger.TWO.add(BigInteger.ONE);

		if(0>n.compareTo(BigInteger.TWO))

			return false;

		if(0 == n.compareTo(BigInteger.TWO) || 0 == n.compareTo(THREE))

			return true;

		//https://www.tutorialspoint.com/java/math/biginteger_mod.htm
		boolean first = n.mod(BigInteger.TWO).compareTo(BigInteger.ZERO) == 0;
		boolean second = n.mod(THREE).compareTo(BigInteger.ZERO) == 0;
		if(first || second)
			return false;

		//https://www.geeksforgeeks.org/biginteger-sqrt-method-in-java/
		BigInteger sqrtN = n.sqrt().add(BigInteger.ONE);

		BigInteger SIX = new BigInteger("6");
		BigInteger negativOne = new BigInteger("-1");

		for(BigInteger i = SIX; sqrtN.compareTo(i) != -1; i = i.add(SIX)) {
			first = n.mod(SIX.add(negativOne)).compareTo(BigInteger.ZERO) ==
0;
			second =
n.mod(SIX.add(BigInteger.ONE)).compareTo(BigInteger.ZERO) == 0;
			if(first||second ) {
				return false;
			}
		}
		return true;
	}
	public static boolean isPrime (long n) {
		if (n<0) {
			return false;}
		for (long i =2; i<n; i++){
			if (n%i == 0)
			{
				return false;
			}
		}
```

```
            return true;
        }
        public static BigInteger pow(BigInteger base, BigInteger exponent) {
              BigInteger result = BigInteger.ONE;
              while (exponent.signum() > 0) {
                if (exponent.testBit(0)) result = result.multiply(base);
                base = base.multiply(base);
                exponent = exponent.shiftRight(1);
              }
              return result;
        }
        public static String FileToString(String filePath) throws IOException {
            FileReader fr = new FileReader(filePath);
            BufferedReader br = new BufferedReader(fr);

            String outcome = "";

            while(br.ready()) {
                outcome += br.readLine() + " ";
            }


            System.out.println("File loader complete");
            return outcome;
        }
        //https://stackoverflow.com/questions/9666903/every-combination-of-character-array
        public static Set<String> permute(String chars){
          // Use sets to eliminate semantic duplicates (aab is still aab even if you switch the
two 'a's)
          // Switch to HashSet for better performance
          Set<String> set = new TreeSet<String>();

          // Termination condition: only 1 permutation for a string of length 1
          if (chars.length() == 1){
            set.add(chars);
          }
          else{
            // Give each character a chance to be the first in the permuted string
            for (int i=0; i<chars.length(); i++){
                // Remove the character at index i from the string
                String pre = chars.substring(0, i);
                String post = chars.substring(i+1);
                String remaining = pre+post;

                // Recurse to find all the permutations of the remaining chars
                for (String permutation : permute(remaining)){
```

```
                                        // Concatenate the first character with the permutations of the
remaining chars
                                        set.add(chars.charAt(i) + permutation);
                              }
                    }
          }
          return set;
     }
}

package ubung12;

import java.util.Random;

public class ScrabbleWordFinder {


     public static void main(String[] args) throws Exception {

          //Filling dictionary
          MyHashTable<String> dictionary = new MyHashTable<String>();
          System.out.println("TableSize: " + dictionary.M);
          String filePath = "S:\\\\the real dictionary.txt";
          for(String word : UM.FileToString(filePath).split("\\s+")) {
               dictionary.add(word);
          }
          System.out.println("Amount of empty lists: " +dictionary.getEmptyLists() +
"%");

          String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
          String bench = getRandomString(7, alphabet);

          PermuteGenerator allPermutes = new PermuteGenerator(bench);
          //our Bench

          System.out.println(bench);
          System.out.println("All that exits:");
          for(int i = 0; i< bench.length() ; i++) {
               System.out.println("All " + (bench.length()-i) + " letter words:");
               for(String s : allPermutes.permutations[i]) {
                    if(dictionary.contains(s))
                         System.out.println(s);
               }
          }

          //dictionary.printListsSize();
```

```
	}
	private static String getRandomString(int i, String chars) {
		String randomString = "";
		for(;i != 0; i--) {
			randomString += getRandomChar(chars);
		}
		return randomString;
	}
	private static char getRandomChar(String source) {
		Random r = new Random();
		return source.charAt(r.nextInt(source.length()));
	}
}

package ubung12;

import java.util.HashSet;
import java.util.Set;

public class PermuteGenerator {

	//Array of Sets, are distinguished by the length of words
	//At 0 the biggest word.
	//At Array.length-1 one letter words.
	Set<String> permutations[];

	public PermuteGenerator(String source) {

		permutations = new Set[source.length()];
		for(int i = 0; i< permutations.length; i++) {
			permutations[i] = new HashSet<String>();
		}
		permutations[0] = UM.permute(source);

		//For all the lower number of letter permutations
		//safes the Base Permutations
		Set<String> base = UM.permute(source);
		//we cant set base = permutation and safe the calc:
		//unknown Error shows up, look picture
		for(int i = 1; i< source.length(); i++) {
			for(String s : base) {
				//gets all the 3 Letter Permutations(doubles are possible)
				//Sets cant have doubles to here we get rid of them
				permutations[i].add(s.substring(i));
			}
		}
	}

}

package ubung12;

import java.math.BigInteger;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Set;

public class MyHashTable<Item> implements HashTable {

	int M;
	int listsSize;
	LinkedList<String>[] items;
	int maxSteps;

	public MyHashTable() {
		M = 24593;
		M = getNextPrime();
		listsSize = 64;
		items = new LinkedList[ M];

		for(int i = 0; i<items.length;i++) {
			items[i]= new LinkedList<String>();
		}
	}
	public MyHashTable(int M) {
		this.M = M;
		this.M = getNextPrime();
		items = new LinkedList[M];

		for(int i = 0; i<items.length;i++) {
			items[i]= new LinkedList<String>();
		}
	}
	@Override
	public void resize() {
		System.out.println("resizing");
		MyHashTable<String> newHashTable = new MyHashTable<String>(M*2);

		for(LinkedList<String> List : items) {
			for(String item : List) {
				newHashTable.add(item);
			}
			}

		}
		@Override
		public void add(String item) {
			char[] stringAsChars = item.toUpperCase().toCharArray();

			int i = 0;
			int value = 0;

			BigInteger BigIndex = BigInteger.ZERO;
			BigInteger digitValue;
			for(char c : stringAsChars){
				value = c;
				digitValue = new BigInteger(value+"");
				long multiplicater = (long) Math.pow(20, i);
				BigIndex = BigIndex.add(digitValue.multiply(new
	BigInteger(multiplicater+"")));
				i++;
			}
			BigIndex = BigIndex.mod(new BigInteger(M+""));
			value = BigIndex.intValue();

			//pro
			if(items[value].size()>listsSize-1) {
				value = getNewIndex(value,1);
			}
			items[value].add(item.toUpperCase());

			//resize if we got less then 1/3 free lists
			/*
			if(33>getEmptyLists()) {
				resize();
			}
			*/
		}
		@Override
		public boolean contains(String item) {
			char[] stringAsChars = item.toUpperCase().toCharArray();
			LinkedList<String> MyLinkedList = null;
			int i = 0;
			int value;

			BigInteger BigIndex = BigInteger.ZERO;
			BigInteger digitValue;
			for(char c : stringAsChars){
```

```
                        value = c;
                        digitValue = new BigInteger(value+"");
                        long multiplicater = (long) Math.pow(20, i);
                        BigIndex = BigIndex.add(digitValue.multiply(new
BigInteger(multiplicater+"")));
                        i++;
                }
                BigIndex = BigIndex.mod(new BigInteger(M+""));
                value = BigIndex.intValue();
                MyLinkedList = items[value];

                if(MyLinkedList.contains(item.toUpperCase())) {
                        return true;
                }
                else {
                        if(items[value].size()>listsSize-1) {
                                return findWord(item.toUpperCase(),value,1);
                        }
                        return false;
                }
        }
        @Override
        public void reset() {
                items = new LinkedList[M];
        }
        private int getNextPrime() {
                for(int i = M;true;i++) {
                        if(UM.isPrime((long) i)) {
                                return i;
                        }
                }
        }
        @Override
        public void print() {
                System.out.println(toString());
        }
        @Override
        public String toString() {
                String outcome = "";

                for(LinkedList<String> List : items) {
                        for(String item : List) {
                                outcome = outcome + item + "\n";
                        }
                }
                return outcome;
```

```
}
public ArrayList<String> lookup(String bench) {
	Set<String> permute = UM.permute(bench);
	ArrayList<String> exitsPermute = new ArrayList<String>();
	for(String per : permute) {
		if(contains(per))
			exitsPermute.add(per.toUpperCase());
	}
	return exitsPermute;
}
public void printListsSize() {
	for(LinkedList<String> list : items) {
		if(list.size()>0)
			System.out.println(list.size());
	}
}
private int getNewIndex(int value, int step) {

	int change = (int)Math.pow(step, 2);
	//even numbers change to left
	if(value % 2 == 0) {
		if(items[bordered(value-change)].size()<listsSize){
			return bordered(value-change);
		}
		else {
			return bordered(getNewIndex(value-change, step+1));
		}
	}
	//odd numbers change to right
	else{
		if(items[bordered(value+change)].size()<listsSize){
			return bordered(value+change);
		}
		else {
			return bordered(getNewIndex(value+change, step+1));
		}
	}
}
//gets an int thats round about the array
private int bordered(int index) {
	//already in border
	if(index<items.length && index > -1)
		return index;
	else {
		//if its bigger
		if(index >items.length-1) {
				return bordered(index-items.length);
			}
			//if its lower
			else {
				return bordered(items.length+index);
			}
		}
	}
	private boolean findWord(String word, int value, int step) {
		int change = (int)Math.pow(step, 2);
		//even numbers change to left
		if(value % 2 == 0) {
			value = bordered(value-change);
			if(items[value].contains(word)) {
				return true;
			}
			else {
				if(items[value].size()<listsSize) {
					return false;
				}
				else {
					return findWord(word, value, step+1);
				}
			}
		}
		//odd numbers change to right
		else {
			value = bordered(value+change);
			if(items[value].contains(word)) {
				return true;
			}
			else {
				if(items[value].size()<listsSize) {
					return false;
				}
				else {
					return findWord(word, value, step+1);
				}
			}
		}
	}
	public double getEmptyLists() {
		int i = 0;
		for(LinkedList<String> list : items) {
			if(list.size() == 0)
				i++;
```

```
                }
                double percent= ((double)i/items.length)*100;
                percent = Math.round(percent*100);
                return percent/100;
        }
}
```