

Info 2

Laboratory 12

Prof. Weber-Wulff

21.01.2020



Chung-Fan Tsai

(s0571269@htw-berlin.de)

Muhammad Safarov

(s0570690@htw-berlin.de)

Pavel Tsvyatkov

(s0559632@htw-berlin.de)

Index

Laboratory Report	1
Lab Exercise	2
Reflection	14
Appendix: Code	15

Lab Exercise

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 were happy with our solution from Exercise 11 so we continued on with it.

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.

```
/* if s2 is a permutation of s1
public boolean isSubset(String s1,String s2) { //ppp
    int size1=s1.length();
    int size2=s2.length();

    //if more letters, return false
    if(size2>size1) return false;

    String s1Norm=normalize(s1);
    String s2Norm=normalize(s2);

    //if same amount of letters, check normalize
    if(size2==size1) return s1Norm.equals(s2Norm);

    //if s2 has less letters, check using contains
    for(int i=0;i<s2.length();i++) { //watch out for how many char eg. keynote nono or none
        if(s1Norm.contains(s2Norm.substring(i,i+1))) {
            int index = s1Norm.indexOf(s2Norm.substring(i,i+1));
            if(index==0)
                s1Norm = s1Norm.substring(1);
            else if(index>0 && index<s1Norm.length()-1)
                s1Norm = s1Norm.substring(0,index)+s1Norm.substring(index+1);
            else if(index==s1Norm.length()-1)
                s1Norm = s1Norm.substring(0,index);

        }else return false;
    }

    return true;
}
```

We have already solved this problem in the previous lab, so we just improved our method.

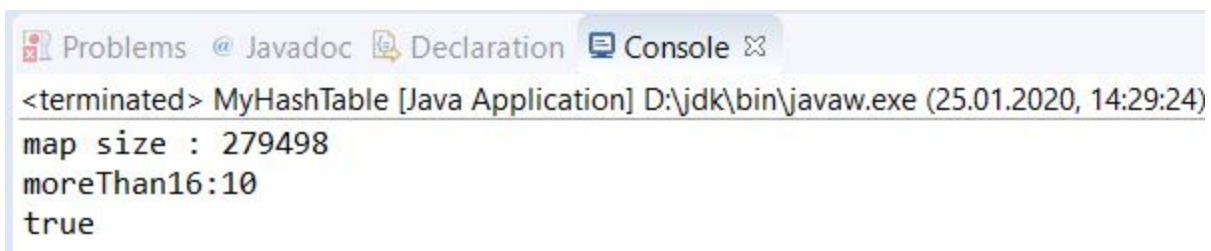
Our method receives 2 Strings as a parameter and the method will be checking if they are permutations of each other.

First thing that we want to check if the length of both strings are equal or not. If the length of one of the Strings is greater or less than the length of the other String, it's obviously not a permutation and method will return false.

If the lengths of the Strings are the same, the method will normalize both of them and check them again if they are permutations of each other using a substring() method of a class String.

```
}  
public static void main(String args[]) {  
    MyHashTable<Integer, String> map = new MyHashTable<>(75011, "D://Eclipse//Projekte//Lab11 only//src//words-279k.txt");  
    System.out.println(map.isSubset("blate", "table"));  
}
```

Now we test if 2 strings ("blate" and "table") are permutations.



```
<terminated> MyHashTable [Java Application] D:\jdk\bin\javaw.exe (25.01.2020, 14:29:24)  
map size : 279498  
moreThan16:10  
true
```

As you can see, the method returns true.

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

In our Scrabble Cheater class we created the method `generateRandom`, which takes an `int` (`numOfLetters` we want) as a parameter and returns a `String`.

In the method's body, we have a `String` local variable called `temp` assigned to the empty string. After that, we loop `numOfLetters` times and on each iteration we are going to **grab** a letter from our **bag**, that we called `scrabble_letters`, (which stores the letters available in English Scrabble) and add it to `temp`. In the end we return the `String`, containing the randomly generated letters.

```
37 //Generates a random selection of letters
38 public String generateRandom(int numOfLetters) {
39     String temp = "";
40     for(int i=0;i<numOfLetters;i++) {
41         temp+=scrabble_letters.grab();
42     }
43     return temp;
44 }
```

We wrote the following in our main and ran it several times to see how it works.

```
41 System.out.print("The random String is: ");
42 System.out.print(scrabble.generateRandom(7));
```

```
The random String is: wuattse
```

```
The random String is: oadkutr
```

```
The random String is: ohigeia
```

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.**
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.**

When we were working on calculating sets, some of us worked on subsets. But there is a big difference between that subset and this subset since with this subset, the order of the letters/ elements doesn't matter because we will normalize the word anyways. So we tried using our algorithm first using the following logic. We first start thinking about how human brains solve this problem. When we encounter a string abcd, how do we find substrings:

Substring with 4 letters:abcd

Substring with 3 letters: abc, abd, acd, bcd

Substring with 2 letters:ab, ac, ad, bc, bd, cd

If we don't care about the order of the letters, we basically traverse from `string.charAt(0 to strlength-substringLength)` in a loop, and then call the same function with less string. For example, we first have a for loop from 2(letters) to string length(letters), then we call the function in the loop, with the original string and substring length as params, so we get different lengths of substrings separately. When we want 2 letter substrings with "abc", in first layer/ first call, we loop from a to b (string length - substringlength). When we are at a, if the substring length we want is longer than 1 letter, meaning there are different combinations, we call the function again with "bc" and `substringlength - 1 (2 - 1 = 1)` as param, then in second layer, we loop from b to c, and since now we are only looking for one character, so we add ab to our list of substrings and ac as well when we finish the loop, then we go back up one layer, now we move to b, and since b now only has one option to form a two letter substring, we add bc to our list.

But then we realized that we probably need to practice implementing and using Bag with this exercise, so we scratched that solution and start implementing a MyBag ADT. We started out implementing the methods Bag may need:

```
public boolean add(String c);

public boolean remove(String s);

public String grab();

public boolean contains(String s);

public boolean isEmpty();

public int size();

public String toString();
```

We then created a class `MyBagWithLinkedList` that implements `MyBag` ADT. We chose to use list to implement bag because we find a lot of the listed methods above can easily be done with existing methods of list.

Since Bag allows duplicates, we don't even need to check if a string exists before adding, so we can simply reuse list methods:

```
public boolean add(String c) {  
    return list.add(c);  
}  
  
public boolean remove(String s) {  
    return list.remove(s);  
}
```

...and we implement `isEmpty`, `contains`, `size`, and `toString` also using list methods.

With `grab` is a bit different. As we need to randomly pick one element from the bag and then remove it from the bag so we don't "draw" the same element twice:

```
public String grab() {  
    int size = size();  
    int r = (int)(Math.random()*size);  
    String temp =list.get(r);  
    list.remove(r);  
    return temp;  
}
```

Afterwards, we implement the constructor so that when we instantiate a Bag with the input string(the letters we got when playing scrabble), each letter is added individually:

```
public MyBagWithLinkedList(String str) {  
    s_input = normalize(str);  
    for(char c : s_input.toCharArray()) {  
        add(""+c);  
    }  
}
```

```

    }

    substrList=new LinkedList<LinkedList<String>>();

}

```

Then we divide and conquer. If the input string has 7 letters, we use a loop to get 2,3,4...7 letter substrings separately. Our logic is, we can use combination equation to know if we want to pick k elements from n elements while order doesn't matter, there are C_n, k combinations, so we can just keep drawing and storing a "new combo"(one that we haven't drawn and stored yet) until all combinations are drawn and stored:

```

public LinkedList<String> getSubstringsOfLength(int k){
    int size = size();

    LinkedList<String> sub = new LinkedList<String>();

    //from picking k elements from n
    int combi_num = getHowManyCombiThereIs(k);

    int contains=0;
    while(sub.size()<combi_num && contains<100) {
        String temp="";
        MyBagWithLinkedList bag_temp = new MyBagWithLinkedList(s_input);
        for(int i=0;i<k;i++) {
            temp+=bag_temp.grab();
        }
        temp = normalize(temp);
        if(!sub.contains(temp)) {
            sub.add(temp);
        }else contains++;
    }
    return sub;
}

```

But then we realized calculating combinations isn't as simple as we thought as we are not using n different elements to pick from, we may have uncertain amount of same elements, like JAVA has 2 A's. In this case, we will have less combinations, but how do we know exactly how many?

```

public int getHowManyCombiThereIs(int k) {

    //if k = 5, 5 same, 4 same, 3 same, 2 same, all diff
    //if k = 4, 4 same, 3 same, 2 same , all diff
    //if k =3, 3 same, 2 same, all diff
    //if k =2, 2 same, all diff
    charCount();
    if(charCount[2]==0) return C(charCount[1],k);
}

```

So in the method to get the amount of combinations, we first call `charCount()` (to count characters) then check if there's no char appearing twice, we simply use $C(n,k)$.

And our `charCount()` works like this: we first normalize (to lower case and sort) the string, then we iterate through and count how many times a certain character appears in the string:

```
public void charCount() {
    for(int i=0; i<charCount.length;i++) {
        charCount[i]=0;
    }
    for(int i = 0 ; i < s_input.length() ;) {
        char c = s_input.charAt(i);
        int count = s_input.length() - s_input.replaceAll(""+s_input.charAt(i), "").length();
        i+=count; //aaabc
        for(int j=1; j<charCount.length && j<=count;j++) {
            charCount[j]++;
        }
    }
}
```

So if we get "JAVAABB", we'd get:

`charCount[1]` (char that appear at least once): 4 (J,V,A,B)

`charCount[2]` (char that appear at least twice): 2 (A,B)

`charCount[3]` (char that appear at least thrice): 1 (A)

With the help of this, we can then calculate the combinations with maybe some of the elements in k the same, so we first assume, all k of elements are the same, and see if `charCount[k]` is not 0, let's say if we pick 2 letters from "JAVVA", the combination calculation for 2 letter substrings works like this:

Case 1: 2 letters are the same : pick 1 from `charCount[2]` (A,V) = $C(2,1)=2$: AA,VV

Case 2: 1 letter the same, meaning all different:

$C(\text{charCount}[1],2)=C(3(J,V,A),2)=3$: JA,JV,VA

So for "JAVVA", if we want 2 letter substrings, we get $2+3=5$ combinations instead of $C(5,2)=10$.

And this is how the code looks like:


```

//5:(5 same, 4 same, 3 same (3 same 2 diff, 3 same 2 same), 2 same (2 s 3 dif, 2 s 2s 1)
for(int i=k;i>=1;i--) { // how many same
    //choose k, k has i same char
    if(i>1 && k-i>0) {
        int c1=0;
        for(int y=1;i-y>=1;y++) { //i-1...=1
            c1 += C(charCount[i],1)*C(charCount[i-y]-1,k-i);
            if(i*2<=k) {
                c1 += C(charCount[i],2)*C(charCount[i-y]-1,k-i*2);
            }
        }
        combi+=c1;
    } else { //i==k || i==1
        int c2 = C(charCount[i],i==k?1:k);
        combi+=c2;
    }
}

```

But just in case we miss anything ('cause if we do, we'll forever keep grabbing new combo from the bag, and the program will not stop), we still use a variable to control the grabbing process.

And this is what we get from "JAVA":

```

Start cheating: java
Bag elements: [a, a, j, v]
Substrings of java :
av aj jv aa
aaj ajv aav
aajv
***** java can spell *****
----- 2 letters -----
ja aa
----- 3 letters -----
ava
----- 4 letters -----
java

```

Note that we normalize the substrings so that we can hash them and find the location that stores those permutations.

After having the substrings, we check if these combinations of different lengths and set of letters can form an actual word by using get to see if they exist in our dictionary, if get doesn't return null, we add it to our list of substrings that form actual words and then print them out from short to long.

```

public void getValidWordsFromSubstrings(String str) {
    findSubstringsWithBag(str); // substringList
    if(!substringList.isEmpty()) {
        System.out.println("***** "+str+" can spell "+" *****");
        for(LinkedList<String> ithLengthSubstring:substringList) {
            if(!ithLengthSubstring.isEmpty()) {
                boolean dividerPrinted=false;
                int items=0;
                for(String s : ithLengthSubstring) {
                    LinkedList<String> perm;
                    perm=findPermutation(s);
                    if(!perm.isEmpty()) {
                        if(!dividerPrinted) {
                            System.out.println("----- "+s.length()+" letters -----");
                            dividerPrinted=true;
                        }
                        for(String s_p: perm) {
                            if(items!=0 && items%7==0)
                                System.out.println();
                            System.out.print(s_p+" ");
                            items++;
                        }
                    }
                }
            }
        }
    }
}

```

When we output the words we did find in the dictionary, we also find that we don't know a lot of them, like the example of the valid words we can spell with "dfaeafe":

```

***** dfaeafe can spell *****
----- 2 letters -----
ea ae da ad ed de aa
ee fe ef fa
----- 3 letters -----
faa eff fae fee dae aff fed
def fad dee
----- 4 letters -----
fade deaf feed daff
----- 5 letters -----
effed

```

In order to confirm that these words actually exist, and to create a more useful Scrabble Cheater that helps you learn new words as well, we figured if we give the user a chance to learn the word(spelling and definition) at the same time, it's more likely that next time the user can come up with that word one their own.

And since we found a word text from Collins dictionary with definitions that look like this in a text file:

```

1 AA (Hawaiian) a volcanic rock consisting of angular blocks of lava wit
2 AAH an interjection expressing surprise [interj] / to exclaim in surpri
3 AAHED AAH, to exclaim in surprise [v]
4 AAHING AAH, to exclaim in surprise [v]
5 AAHS AAH, to exclaim in surprise [v]
6 AAL (Hindi) the Indian mulberry tree, aka noni, also AL [n -S]
7 AALII (Hawaiian) a tropical tree [n -S]
8 AALIIS AALII, (Hawaiian) a tropical tree [n]
9 AALS AAL, (Hindi) the Indian mulberry tree, aka noni, also AL [n]

```

We then just need to use string methods to divide the word and its definition so that we can store them in different variables, this allows us to still find the word.

Then we decide to make using it easier, so we allow users to directly input strings from console, and then look up definitions as well.

When we run Scrabble_Cheater class, we see this in console:

```

Press Enter to randomly generate 7 letters
Start cheating:

```

You can simply press enter to let the program randomly assigns 7 letters for you or type it in and get the following result:

```

Press Enter to randomly generate 7 letters
Start cheating: abbqqdd
Bag elements: [a, b, b, d, d, q, q]
Substrings of abbqqdd :
bd dq bq qq ab aq bb ad dd
abd aqq bdq adq bdd add abb abq bbd bqq bbq ddq dqq
abbq abdd abqq abbd bddq abdq bdqq ddqq adqq bbdq bbdd addq bbqq
abddq bbdqq bddqq abdqq abbdq bbddq abbqq addqq abbdd
bbddqq abddqq abbddq abbdqq
abddqq
***** abbqqdd can spell *****
----- 2 letters -----
ba ab da ad |
----- 3 letters -----
dab bad dad add abb

```

```

- You Entered dictionary -
Get definition:

```

Then you automatically enter our dictionary, so you can look up words that you can spell:

```
- You Entered dictionary -
Get definition: abb
Def: a woof yarn [n -S]
- Type "q" to leave dictionary -
Get definition: dab
Def: to touch lightly [v DABBED, DABBING, DABS]
- Type "q" to leave dictionary -
Get definition: ba
Def: the soul, in ancient Egyptian religion [n -S]
- Type "q" to leave dictionary -
Get definition:
```

If you want to leave the dictionary to go back to Scrabble Cheater, just type "q" and "enter".

6. (For the bored) Generate your random selection of characters depending on the Scrabble distribution for English

We found the rule of Scrabble and instantiate a Scrabble Bag using the correct amount of each letter (and with 2 blanks symbolized by "-"):

English Scrabble letter distribution
(Number of tiles across, point values down)

	x1	x2	x3	x4	x6	x8	x9	x12
0		[blank]						
1				LSU	NRT	O	AI	E
2			G	D				
3		BCMP						
4		FHVWY						
5	K							
8	JX							
10	QZ							

```
public Scrabble_Cheater(){  
    //instantiate MyBag of scrabble letters for english  
    //1 k,j,x,q,z , 2 b,c,m,p,f,h,v,w,y, 3g  
    //4 l,s,u,d,6 n,r,t,8 o,9 a,i,12 e,2 blank  
    scrabble_letters = new MyBagWithLinkedList("--k,j,x,q,z,b,c,m,p,f,h,v,w,y,b,c"  
        + "l,p,f,h,v,w,y,g,g,g,l,s,u,d,l,s,u,d,l,s,u,d,n,r,t,n,r,t,n,r,t,n,r,t,o,o,o,o,o,o,a,i,a,i,a,i,a,i,a,i,e,e,e,e,e,e,e,e,e,e");  
}
```

We included blank in our Scrabble bag and iterate through a-z when the substring combination has blank in it, for example, if we got 2 letter substrings with a- (- as blank), we add aa,ab,ac,ad...ay,az to the two letter combinations.

```

public void matchSubsetWithBlank(String s, Set permSet) {
    //System.out.println("yes and :"+s.indexOf("-"));
    LinkedList<String> possibilities = new LinkedList<String>();

    char base = 'a';
    int count = s.length() - s.replaceAll("-", "").length();

    while((int)base<=(int)'z') {
        possibilities.add(s.replace("-", (""+base)));
        base = (char) (base+1);
        //System.out.println("base:"+base);

        if(count==2) {
            char base2 = 'a';
            while((int)base2<=(int)'z') {
                String temp = possibilities.getLast();
                temp = temp.replace("-", (""+base2));
                possibilities.set(possibilities.size()-1,temp);
                base2++;
            }
        }
    }
}

```

So when we press enter at "Start Cheating:", we can even get "-" blank letters:

```

Start cheating:
Bag elements: [-, a, a, n, o, p, z]
Substrings of anopaz- :
-p oz -a pz az ap aa -n op an ao np no
aaz -np aan anp apz -nz -op npz -ao -an
-aap -anz -aaz aaop anpz -anp aapz aopz -a
aanpz -aaoz -anpz aanop -aano -aanz -aopz -
-aanop -anopz -aanoz aanopz -aaopz -aanpz
-aanopz

```

We calculate the substrings the same way as shown in task 4 and 5, but before checking if the word can be found in the dictionary, we check if blanks exist in the string, generate for "a-" : aa,ab,ac...,ay,az. Then we check if their permutations exist in dictionary:

```

Start cheating: a-
Bag elements: [-, a]
Substrings of a- :
-a
***** a- can spell *****
----- 2 letters -----
aa ba ab da ad ea ae
fa ag ha ah ai ja ka
la al ma am na an pa
ar as ta at aw ax ya
ay za
- You Entered dictionary -
Get definition:

```

As a result, we got the above 2 letter words we can create with "-" blank and "a".

Reflection

Fan:

Bag is quite an interesting data type as it imitates a real "bag". We hit a bit of a hurdle when we were figuring out how many combinations there are when we get k elements from n elements, with uncertain amount of duplicated elements inside n . But this is where time traveling back to high school math helps. If we have more time, we would like to count the score of each valid combination and sort them using the score so that you know which is the best option you got for the letters you have. But this is when I start to think, with my little knowledge of scrabble, if we want to create a Scrabble bot to play against human players, besides the fact that they would know all the words, maybe it's also like playing Bridge, an experienced player(or all-knowing bot) would estimate the amount of each characters are still in the bag, and if saving some letters for when you get a higher scored combination is better than just picking the highest scored word you can spell with the current hand. Kind of like if we implement a chess bot, we can't just calculate the valid positions we can move our piece too, and also not just aiming to take one piece from the opponent, but instead think 9 steps ahead and simulate possible situations, which is very complicated but could be very fun, to implement a playable scrabble game, with GUI and all, that is.

Pavel:

While working on this exercise I learned how we can implement our own Bag using a linked list. Having a Bag ADT let us easily add new functionality when we had ideas or when we wanted to change something and adapt it after. I saw how to generate different letter substrings and output only those that contain letters from the generated String. I think it was a great idea to have the option to get the definition of a specific word, since there were so many words we didn't know about.

Muhammad: Before this lab i had only theoretical knowledges about a Bag and because of the lack of time i never had a chance to really implement it and see how it works. Implementing our own back and coming up with a new ideas for an ADT, discussing it with in a team was very fun. Even this case with binomialkoeffizient was confusing us, but after your explanation it was finally clear why we have to use a bag. Looking for a new challenges in the next semester Professor.

Code

MyBag

```
package lab12_scrabble_cheater;

import java.util.LinkedList;

public interface MyBag {

    public String normalize(String s);
    public boolean add(String c);
    public boolean remove(String s);
    public String grab();
    public boolean contains(String s);
    public boolean isEmpty();
    public int size();
    public String toString();
    public LinkedList<String> getSubstringsOfLength(int k);
    public int getHowManyCombiThereIs(int k);
    public int fac(int n) ;
    public void charCount();
}
```

MyBagWithLinkedList

```
package lab12_scrabble_cheater;

import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

public class MyBagWithLinkedList implements MyBag{

    String s_input;
    // store letters
```

```

LinkedList<String> list = new LinkedList<String>();
//store substrings
static LinkedList<LinkedList<String>> substrList;
int[] charCount = new int[13]; //1-12
public MyBagWithLinkedList(String str) {
    s_input = normalize(str);
    for(char c : s_input.toCharArray()) {
        add(""+c);
    }
    substrList=new LinkedList<LinkedList<String>>();
}
public void charCount() {
    for(int i=0; i<charCount.length;i++) {
        charCount[i]=0;
    }
    for(int i = 0 ; i < s_input.length() ;) {
        char c = s_input.charAt(i);
        int count = s_input.length() -
s_input.replaceAll(""+s_input.charAt(i), "").length();
        i+=count; //aaabc
        for(int j=1; j<charCount.length && j<=count;j++) {
            charCount[j]++;
        }
    }
}
public String normalize(String s) {
    s=s.toLowerCase();
    char temp[] = s.toCharArray();
    Arrays.sort(temp);
    return new String(temp);
}
public boolean add(String c) {
    return list.add(c);
}

```



```

}

public boolean remove(String s) {
    return list.remove(s);
}

public String grab() {
    int size = size();
    int r = (int)(Math.random()*size);
    String temp =list.get(r);
    list.remove(r);
    return temp;
}

public boolean contains(String s) {
    return list.contains(s);
}

public boolean isEmpty() {
    return list.isEmpty();
}

public int size() {
    int size = list.size();
    return size;
}

public String toString() {
    String all = "Bag elements: ";
    if(!isEmpty()) {
        all+=list.toString();
        return all;
    }
    return "Bag is empty";
}

public LinkedList<String> getSubstringsOfLength(int k){
    int size = size();
    LinkedList<String> sub = new LinkedList<String>();
    //from picking k elements from n

```

```

int combi_num = getHowManyCombiThereIs(k);
int contains=0;
while(sub.size()<combi_num && contains<100) {
    String temp="";
    MyBagWithLinkedList bag_temp = new MyBagWithLinkedList(s_input);
    for(int i=0;i<k;i++) {
        temp+=bag_temp.grab();
    }
    temp = normalize(temp);
    if(!sub.contains(temp)) {
        sub.add(temp);
    }else contains++;
}
return sub;
}

//normally from n diff elements pick k is n!/(k!*(n-k)!)
//but since we may use same elements
public int getHowManyCombiThereIs(int k) {
    //if k = 5, 5 same, 4 same, 3 same, 2 same, all diff
    //if k = 4, 4 same, 3 same, 2 same , all diff
    //if k =3, 3 same, 2 same, all diff
    //if k =2, 2 same, all diff
    charCount();
    int combi=0;
    int str_length = s_input.length();
    if(str_length<k) return 0;
    //if no repetitive elements
    if(charCount[2]==0) return C(charCount[1],k);
    if(k==str_length) return 1;
    //5:(5 same, 4 same, 3 same (3 same 2 diff, 3 same 2 same), 2 same (2
s 3 dif, 2 s 2s 1)
    for(int i=k;i>=1;i--) { // how many same
        //choose k, k has i same char

```

```

        if(i>1 && k-i>0) {
            int c1=0;
            for(int y=1;i-y>=1;y++) { //i-1...=1
                c1 += C(charCount[i],1)*C(charCount[i-y]-1,k-i);
                if(i*2<=k) {
                    c1 += C(charCount[i],2)*C(charCount[i-y]-1,k-i*2);
                }
            }
            combi+=c1;
        }else{//i==k || i==1
            int c2 = C(charCount[i],i==k?1:k);
            combi+=c2;
        }
    }
    return combi;
}

public static void main(String args[]) {
    String str= "parsley";
    MyBag bag = new MyBagWithLinkedList(str);
    System.out.println(bag.toString());
    int size = bag.size();
    System.out.println("Substrings of "+str+" :");
    for(int i =2; i<=size;i++) { //get 2,3...,size lengths of substrings
        LinkedList<String> substringsOfCertainLength=
bag.getSubstringsOfLength(i);
        substrList.add(substringsOfCertainLength);
        for(int j=0;j<substringsOfCertainLength.size();j++) {
            System.out.println(substringsOfCertainLength.get(j));

        }
    }
    bag.charCount();
}

```

```

    }

    public int C(int n, int m) {
        if(n==0) return 0;
        return fac(n)/(fac(m)*fac(n-m));
    }

    public int fac(int n) {
        if(n<=1) return 1;
        return fac(n-1)*n;
    }
}

```

MyHashTable

```

package lab12_scrabble_cheater;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.TreeSet;
import javax.swing.text.html.HTMLDocument.Iterator;

// A node of chains
class HashNode
{
    int key;
    String value;
    String def;
    // Reference to next node

```

```

    HashNode next;
    // Constructor
    public HashNode(int hashedKey, String keyStr, String def)
    {
        this.key = hashedKey;
        this.value = keyStr;
        this.def = def;
    }
}

// Class to represent entire hash table
class MyHashTable
{
    // bucketArray is used to store array of chains
    private HashNode[] bucketArray;
    // Current capacity of array list
    private static int numBuckets;
    //substrings of a given string
    LinkedList<LinkedList<String>> substringList;
    Map<String,Integer> points;
    // Current size of array list
    private int size;
    int moreThan16=0;
    int moreThan100=0;
    // Constructor (Initializes capacity, size and
    // empty chains.
    public MyHashTable(int size, String filelocation)
    {
        bucketArray = new HashNode[size];
        numBuckets = size;
        // Create empty chains
        for (int i = 0; i < numBuckets; i++)
            bucketArray[i]=null;

        readAndAdd(filelocation);
    }
}

```

```

        initializePoints();
        System.out.println("num of entry : "+size());
        System.out.println("map size : "+numBuckets);
        getSizesOfChains();
        System.out.println("moreThan16:"+moreThan16);
    }

    public int size() { return size; }
    public boolean isEmpty() { return size() == 0; }
    // This implements hash function to find index
    // for a key
    protected int getBucketIndex(String keyStr)
    {
        int hashCode = hashCode(normalize(keyStr));
        int index = hashCode % numBuckets;
        return index;
    }
    public int[] getSizesOfChains() {
        int[] sizes = new int[numBuckets];
        int i=0;
        for(HashNode n: bucketArray) {
            sizes[i] = getSizeOfSingleChain(i);
            i++;
        }
        return sizes;
    }
    public int getSizeOfSingleChain(int index) {
        HashNode current = bucketArray[index];
        int i =0;
        while(current!=null) {
            current = current.next;
            i++;
        }
        //System.out.println(index+": "+i);
    }

```

```

    if(i>16) {
        // System.out.println("over16: "+i);
        moreThan16++;
    }
    if(i>100) moreThan100++;
    return i;
}

// Method to remove a given key
public String remove(String key) {
    // Apply hash function to find index for given key
    int bucketIndex = getBucketIndex(key);
    // Get head of chain
    HashNode head = bucketArray[bucketIndex];
    // Search for key in its chain
    HashNode prev = null;
    while (head != null)
    {
        // If Key found
        if (head.value.equals(key))
            break;
        // Else keep moving in chain
        prev = head;
        head = head.next;
    }
    // If key was not there
    if (head == null)
        return null;
    // Reduce size
    size--;
    // Remove key
    if (prev != null)
        prev.next = head.next;
    else

```

```

        bucketArray[bucketIndex]=head.next;
    return head.value;
}
// Returns value for a key
public String get(String key)
{
    // Find head of chain for given key
    int bucketIndex = getBucketIndex(key);
    HashNode head = bucketArray[bucketIndex];
    // Search key in chain
    while (head != null)
    {
        if (head.value.toLowerCase().equals(key.toLowerCase()))
            return head.value;
        head = head.next;
    }
    // If key not found
    return null;
}
public String getDef(String key) {
    // Find head of chain for given key
    int bucketIndex = getBucketIndex(key);
    HashNode head = bucketArray[bucketIndex];
    // Search key in chain
    while (head != null)
    {
        if (head.value.toLowerCase().equals(key.toLowerCase()))
            return head.def;
        head = head.next;
    }
    // If key not found
    return "can't find "+key;
}

```



```

// Adds a key value pair to hash
public void add(String keyStr)
{
    boolean breakLine=false;
    String def="";
    char base = 'A';
    //check current capital, next >= 1 space, then
    for(int i=0;i<keyStr.length()-1;i++) {
        int cur = keyStr.charAt(i)-base;
        int next = keyStr.charAt(i+1)-base;
        if((cur<26 && cur>=0) && (next>26||next<0)) {
            def=keyStr.substring(i+1);
            keyStr=keyStr.substring(0,i+1);
            keyStr=keyStr.replaceAll("\\s", "");
            breakLine=true;
            break;
        }
    }
    if(!breakLine) {
        return;
    }
    int hashedKey = hashKey(normalize(keyStr));
    // Find head of chain for given key
    int bucketIndex = getBucketIndex(keyStr.toLowerCase());
    HashNode head = bucketArray[bucketIndex];
    // Check if key is already present
    while (head != null)
    {
        if (head.value.equals(keyStr))
        { //already exists
            return;
        }
        head = head.next;
    }
}

```

```

    }
    // Insert key in chain
    size++;
    head = bucketArray[bucketIndex];
def);    HashNode newNode = new HashNode(hashKey, keyStr.toLowerCase(),
        newNode.next = head;
        bucketArray[bucketIndex]=newNode;
    }
    // Driver method to test Map class
    public LinkedList<String> getSubset(String s) {///gggp
        LinkedList<LinkedList<String>> list = new
LinkedList<LinkedList<String>>();
        LinkedList<String> list_formatted = new LinkedList<String>();
        //s="0123456";
        int size = s.length();

        list.add(new LinkedList<String>());
        agreg=0;
        for(int i=1; i<=s.length();i++) {
            list =recur(0,i,1, s,list);
            list.add(new LinkedList<String>());
        }
        int n=(int) Math.pow(2,s.length());
        for(int i=0;i<list.size();i++) {
            String temp="";
            for(int j=0;j<list.get(i).size();j++) {
                temp +=list.get(i).get(j);
            }
            if(temp.length()>=1)
                list_formatted.add(temp);
        }
        return list_formatted;
    }
}

```

```

    int agreg = 0;

    public LinkedList<LinkedList<String>> recur(int prev, int i, int j,
String s, LinkedList<LinkedList<String>> list) {

        for(int k=prev; k<s.length()-(i-j); k++) {
            list.getLast().add(""+s.charAt(k));
            if(i-j>0)
                recur(k+1, i, j+1, s, list);
            else if(i-j==0) {
                LinkedList<String> temp = list.getLast();
                list.add(new LinkedList<String>());
                if(k+1<(s.length()-(i-j)) && temp.size()-3>=0) {
                    // System.out.println("*");
                    for(int y=temp.size()-3; y<temp.size()-1; y++) {
                        list.getLast().add(temp.get(y));
                    }
                }
            }
        }

        if(prev==0 && j==1 && i>1) {
            for(int t=0; t<list.size(); t++) {
                if(list.get(t).size()==0) {
                    list.remove(t);
                    t--;
                }
            }

            for(; agreg<list.size(); agreg++) {
                int diff = i - list.get(agreg).size();
                if(diff!=0 && agreg>s.length()) {
                    if(agreg-1>=0) {
                        LinkedList<String> temp = list.get(agreg-1);
                        for(int e=diff; e>0; e--) {
                            if(e-1>=0)
                                list.get(agreg).addFirst(temp.get(e-1));
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
}
}
}
return list;
}

public int fac(int n) {
    if (n > 12) throw new IllegalArgumentException(n + " is out of
range");
    return (1 > n) ? 1 : n * fac(n - 1);
}

public LinkedList<String> findPermutation(String s){
    LinkedList<String> validWords = new LinkedList<String>();
    LinkedList<String> substring= new LinkedList<String>();

    if(s.contains("-")) {
        for(int i=0;i<26;i++) {
            String temp=s.replaceFirst("-", ""+(char)('a'+i));
            if(temp.contains("-")) {
                for(int t=0;t<26;t++) {
                    String temp2=temp.replaceFirst("-", ""+(char)('a'+t));
                    substring.add(temp2);
                }
            }else {
                substring.add(temp);
            }
        }
    }else { substring.add(s);}

    for(int i=0;i<substring.size();i++) {
        HashNode head = bucketArray[getBucketIndex(substring.get(i))];
        while(head!=null) {
            if(normalize(substring.get(i)).equals(normalize(head.value))) {

```

```

        validWords.add(head.value);
    }
    head = head.next;
}
}
return validWords;
}

public void findSubstringsWithBag(String str){
    substringList=new LinkedList<LinkedList<String>>();
    MyBag bag = new MyBagWithLinkedList(str);
    System.out.println(bag.toString());
    int size = bag.size();
    System.out.println("Substrings of "+str+" :");
    for(int i =2; i<=size;i++) { //get 2,3...,size lengths of substrings
        LinkedList<String> substringsOfCertainLength=
bag.getSubstringsOfLength(i);
        substringList.add(substringsOfCertainLength);
        for(int j=0;j<substringsOfCertainLength.size();j++) {
            System.out.print(substringsOfCertainLength.get(j)+" ");
        }
        System.out.println();
    }
}

public void matchSubsetWithBlank(String s, Set permSet) {
LinkedList<String> possibilities = new LinkedList<String>();
    char base = 'a';
    int count = s.length() - s.replaceAll("-", "").length();
    while((int)base<=(int)'z') {
        possibilities.add(s.replace("-", (""+base)));
        base = (char) (base+1);
        if(count==2) {
            char base2 = 'a';
            while((int)base2<=(int)'z') {
                String temp = possibilities.getLast();

```

```

        temp = temp.replace("-", (" "+base2));
        possibilities.set(possibilities.size()-1,temp);
        base++;
    }
}
}

HashNode head;

    for(String p: possibilities) {
        head = bucketArray[getBucketIndex(p)];
        while(head!=null) {
if(normalize(p).equals(normalize(head.value))) permSet.add(head.value);
            head = head.next;
        }
    }
}

public boolean isPrime(int n) {
    return true;
}

public LinkedList<String> getWordsFromSameBucket(String s) {
    LinkedList<String> list = new LinkedList<String>();
    HashNode head = bucketArray[getBucketIndex(s)];
    while(head!=null) {
        System.out.print(head.value+" ");
        head=head.next;
    }
    return list;
}

public String normalize(String s) {
    s=s.toLowerCase();
    char temp[] = s.toCharArray();
    Arrays.sort(temp);
    return new String(temp);
}

```

```

public int keyToIndex(int key) {
    return key%numBuckets;
}

public void readAndAdd(String fileLocation)
{
    try{
        File file=new File(fileLocation);    //creates a new file instance
        FileReader fr=new FileReader(file);    //reads the file
        character input stream
        BufferedReader br=new BufferedReader(fr); //creates a buffering
        for(String line="";(line=br.readLine())!=null;){
            add(line);
        }
        fr.close();    //closes the stream and release the resources
    }catch(IOException e){
        e.printStackTrace();
    }
}

public void initializePoints() {
    points= new HashMap<String,Integer>();

    points.put("q",10);points.put("z",10);
    points.put("j",8);points.put("x",8); points.put("k",5);

    points.put("f",4);points.put("h",4);points.put("v",4);points.put("w",4);po
    ints.put("y",4);
    points.put("b",3);points.put("c",3);points.put("m",3);points.put("p",3);

    points.put("g",2);points.put("d",2);

    points.put("l",1); points.put("s",1); points.put("u",1);
    points.put("n",1); points.put("r",1); points.put("t",1);
    points.put("o",1); points.put("a",1); points.put("i",1);
    points.put("e",1);
}

public void lookUpWords() {
    System.out.println("- You Entered dictionary -");
    System.out.print("Get definition: ");
    Scanner in = new Scanner(System.in);
    String s = in.nextLine();

```

```

while(!s.equals("q")) {
    String def = this.getDef(s);
    for(int i=0;i<def.length();i++) {
        if((int)def.charAt(i)==9) {
            if(i+1<def.length())
                def=def.substring(i+1);
            break;
        }
    }
    System.out.println("Def: "+def);
    System.out.println("- Type \"q\" to leave dictionary -");
    System.out.print("Get definition: ");
    s = in.nextLine();
}
}

public void getValidWordsFromSubstrings(String str) {
    findSubstringsWithBag(str);//substringList
    if(!substringList.isEmpty()) {
        System.out.println("***** "+str+" can spell "+" *****");
        for(LinkedList<String> ithLengthSubstring:substringList) {
            if(!ithLengthSubstring.isEmpty()) {
                boolean dividerPrinted=false;
                int items=0;
                for(String s : ithLengthSubstring) {
                    LinkedList<String> perm;
                    perm=findPermutation(s);
                    if(!perm.isEmpty()) {
                        if(!dividerPrinted) {
                            System.out.println("----- "+s.length()+" letters
-----");
                            dividerPrinted=true;
                        }
                        for(String s_p: perm) {

```



```

//instantiate MyBag of scrabble letters for english
//1 k j x q z , 2 b c m p f h v w y, 3 g
//4 l s u d, 6 n r t, 8 o, 9 a i, 12 e, 2 blank

scrabble_letters = new MyBagWithLinkedList("--k j x q z b c m p f h v w y b c"
+
"m p f h v w y g g g l s u d l s u d l s u d l s u d n r t n r t n r t n r t n r t o o o o o o o a i a i a i a i a i a i e e e
e e e e e e e e");

}

public static void main(String[] args) { //30637 , 39989, 25163

    Scrabble_Cheater cheater = new Scrabble_Cheater();

    MyHashTable map = new
    MyHashTable(57787, "src/lab12_scrabble_cheater/wordsList_collins2019_def.tx
t");

    while(true) {
        System.out.println("Press Enter to randomly generate 7 letters");
        System.out.print("Start cheating: ");
        Scanner in = new Scanner(System.in);
        String s = in.nextLine();
        if(s.isEmpty()) {
            s = cheater.generateRandom(7);
        }
        map.getValidWordsFromSubstrings(s);
        map.lookupWords();
    }
}

public String generateRandom(int numOfLetters) {
    String temp = "";
    for(int i=0; i<numOfLetters; i++) {
        temp += scrabble_letters.grab();
    }
    return temp;
}
}

```