Generating all permutations of a given string

What is an elegant way to find all the permutations of a string. E.g. `ba`, would be `ba` and `ab`, but what about `abcdefgh`? Is there any example Java implementation?

java algorithm

edited Jun 6 '13 at 13:57
Mirzhan Irkegulov
8,567 ●2 ●53 ●106

asked Nov 21 '10 at 20:11
dotnetdev
22k ●71 ●199 ●330

protected by Community ♦ May 5 '14 at 18:45

This question is protected to prevent "thanks!", "me too!", or spam answers by new users. To answer it, you must have earned at least 10 reputation on this site (the association bonus does not count).

2 There are lots of answers here: stackoverflow.com/questions/361/… – Marek Sapota Nov 21 '10 at 20:25

7 There is an assumption need to be mentioned. The characters are unique. For example, for a String "aaaa" there is just one answer. To have a more general answer, you can save the strings in a set to avoid duplication – Afshin Moazami Dec 8 '12 at 18:12

1 Is repetition of characters allowed, or is repetition of characters not allowed? Can a single string have multiple occurrences of the same character? – Anderson Green May 22 '13 at 19:35

1 Read the theory (or if, like me, you're lazy, go to en.wikipedia.org/wiki/Permutation) and implement a real algorithm. Basically you can generate a sequence of orderings of elements (that fact that it's a string is irrelevant) and walk through the orderings until you get back to the start. Steer clear of anything that involves recursion or string manipulations. – CurtainDog May 15 '14 at 1:43

|

41 Answers

1 2 next

```
public static void permutation(String str) {
    permutation("", str);
}

private static void permutation(String prefix, String str) {
    int n = str.length();
    if (n == 0) System.out.println(prefix);
    else {
        for (int i = 0; i < n; i++)
            permutation(prefix + str.charAt(i), str.substring(0, i) +
str.substring(i+1, n));
    }
}
```

(via Introduction to Programming in Java)

edited Jan 21 '16 at 19:49
Eric Leschinski
65.7k ●28 ●271 ●225

answered Nov 21 '10 at 20:59
SuperJulietta
5,093 ●1 ●8 ●11

56 Solution seems to be coming from here introcs.cs.princeton.edu/java/23recursion/… – cyber-monk Aug 8 '12 at 16:05

36 That's not rocket science, I came up with pretty much the same answer. Minor tweak: instead of recursing until `n==0`, you can stop a level earlier at `n==1` and print out `prefix + str`. – jpatokal Oct 23 '12 at 10:26

6 "what is the time and space complexity of this?" without some sort of partial answer caching any algorithm that outputs permutation is o(n!) because the result set to the permutation question is factorial to the input. – jeremyjjbrown Jan 2 '13 at 2:26

5 Elegant, yes. But a solution that converts to a char array and swaps to generate the permutations will require much less copying and generate much less garbage. Also this algorithm fails to take repeated characters into account. – Gene May 25 '13 at 19:52

18 @AfshinMoazami I think str.substring(i+1, n) can be replaced by str.substring(i+1). Using str.substring(i) will cause java.lang.StackOverflowError. – Ayusman Aug 7 '13 at 7:25

|

Use recursion.

- Try each of the letters in turn as the first letter and then find all the permutations of the remaining letters using a recursive call.
- The base case is when the input is an empty string the only permutation is the empty string.

edited Nov 21 '10 at 20:46

answered Nov 21 '10 at 20:14

Mark Byers
514k ● 107 ● 1231 ● 1278

1 How can you add a return type to the permute method? compiler cannot determine the return type of this method at every iteration, even though it is a String type obviously. – user1712095 Sep 3 '14 at 2:13

Here is my solution that is based on the idea of the book "Cracking the Coding Interview" (P54):

```java
/**
 * List permutation of a string
 *
 * @param s the input string
 * @return  the list of permutation
 */
public static ArrayList<String> permutation(String s) {
    // The result
    ArrayList<String> res = new ArrayList<String>();
    // If input string's length is 1, return {s}
    if (s.length() == 1) {
        res.add(s);
    } else if (s.length() > 1) {
        int lastIndex = s.length() - 1;
        // Find out the last character
        String last = s.substring(lastIndex);
        // Rest of the string
        String rest = s.substring(0, lastIndex);
        // Perform permutation on the rest string and
        // merge with the last character
        res = merge(permutation(rest), last);
    }
    return res;
}

/**
 * @param list a result of permutation, e.g. {"ab", "ba"}
 * @param c    the last character
 * @return     a merged new list, e.g. {"cab", "acb" ... }
 */
public static ArrayList<String> merge(ArrayList<String> list, String c) {
    ArrayList<String> res = new ArrayList<String>();
    // Loop through all the string in the list
    for (String s : list) {
        // For each string, insert the last character to all possible postions
        // and add them to the new list
        for (int i = 0; i <= s.length(); ++i) {
            String ps = new StringBuffer(s).insert(i, c).toString();
            res.add(ps);
        }
    }
    return res;
}
```

Running output of string "abcd":

- Step 1: Merge [a] and b: [ba, ab]
- Step 2: Merge [ba, ab] and c: [cba, bca, bac, cab, acb, abc]
- Step 3: Merge [cba, bca, bac, cab, acb, abc] and d: [dcba, cdba, cbda, cbad, dbca, bdca, bcda, bcad, dbac, bdac, badc, bacd, dcab, cdab, cadb, cabd, dacb, adcb, acdb, acbd, dabc, adbc, abdc, abcd]

edited Oct 27 '13 at 22:59

answered Oct 27 '13 at 22:49

jean.timex
904 ● 9 ● 9

Page (71) in Cracking the Coding Interview Book, 6th Edition. :) – KarimIhab Nov 28 '15 at 12:23

1 Is this really a good solution? It relies on storing the results in a list, therefore for a short input string it grows out of control. – Androrider Oct 30 '16 at 23:32

Of all the solutions given here and in other forums, i liked Mark Byers the most. That description actually made me think and code it myself. Too bad i cannot voteup his solution as i am newbie. Anyways here is my implementation of his description

```java
public class PermTest {

public static void main(String[] args) throws Exception {
    String str = "abcdef";
    StringBuffer strBuf = new StringBuffer(str);
    doPerm(strBuf,str.length());

}

private static void doPerm(StringBuffer str, int index){

    if(index <= 0)
        System.out.println(str);
    else { //recursively solve this by placing all other chars at current first pos
        doPerm(str, index-1);
        int currPos = str.length()-index;
        for (int i = currPos+1; i < str.length(); i++) {//start swapping all other
chars with current first char
            swap(str,currPos, i);
            doPerm(str, index-1);
            swap(str,i, currPos);//restore back my string buffer
        }
    }
}

private  static void swap(StringBuffer str, int pos1, int pos2){
    char t1 = str.charAt(pos1);
    str.setCharAt(pos1, str.charAt(pos2));
    str.setCharAt(pos2, t1);
}
}
```

I prefer this solution ahead of the first one in this thread because this solution uses StringBuffer.I wouldn't say my solution doesn't create any temporary string (it actually does in system.out.println where the toString() of StringBuffer is called). But i just feel this is better than the first solution where too many string literals are created . May be some performance guy out there can evalute this in terms of 'memory' (for 'time' it already lags due to that extra 'swap')

edited Nov 20 '13 at 6:11

answered Jun 24 '12 at 19:20
srikanth yaradla
925 ● 8 ● 12

A very basic solution in Java is to use recursion + Set (to avoid repetitions) if you want to store and return the solution strings :

```
public static Set<String> generatePerm(String input)
{
    Set<String> set = new HashSet<String>();
    if (input == "")
        return set;

    Character a = input.charAt(0);

    if (input.length() > 1)
    {
        input = input.substring(1);

        Set<String> permSet = generatePerm(input);

        for (String x : permSet)
        {
            for (int i = 0; i <= x.length(); i++)
            {
                set.add(x.substring(0, i) + a + x.substring(i));
            }
        }
    }
    else
    {
        set.add(a + "");
    }
    return set;
}
```

answered Dec 16 '13 at 15:04
devastator
241 ● 3 ● 4

1 What is the time complexity of this alogrithm?? – ashisahu Sep 2 '16 at 7:14

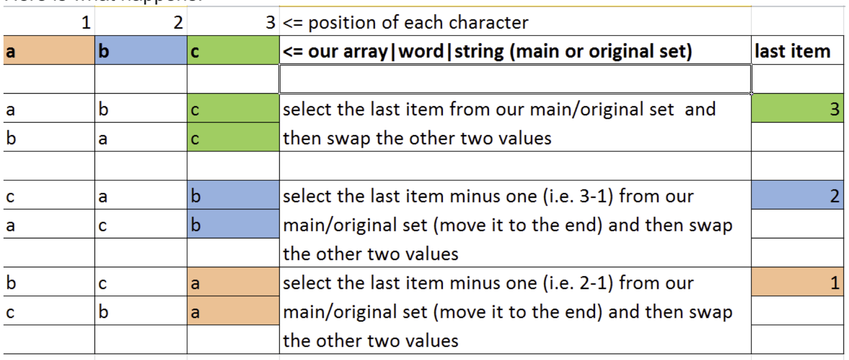
|

All the previous contributors have done a great job explaining and providing the code. I thought I should share this approach too because it might help someone too. The solution is based on (heaps' algorithm)

Couple of things:

1. Notice the last item which is depicted in the excel is just for helping you better visualize the logic. So, the actual values in the last column would be 2,1,0 (if we were to run the code because we are dealing with arrays and arrays start with 0).
2. The swapping algorithm happens based on even or odd values of current position. It's very self explanatory if you look at where the swap method is getting called.You can see what's going on.

Here is what happens:



1	2	3	<= position of each character	
a	**b**	**c**	**<= our array|word|string (main or original set)**	**last item**
a	b	c	select the last item from our main/original set and	3
b	a	c	then swap the other two values	
c	a	b	select the last item minus one (i.e. 3-1) from our	2
a	c	b	main/original set (move it to the end) and then swap	
			the other two values	
b	c	a	select the last item minus one (i.e. 2-1) from our	1
c	b	a	main/original set (move it to the end) and then swap	
			the other two values	

```java
public static void main(String[] args) {

        String ourword = "abc";
        String[] ourArray = ourword.split("");
        permute(ourArray, ourArray.length);

    }

    private static void swap(String[] ourarray, int right, int left) {
        String temp = ourarray[right];
        ourarray[right] = ourarray[left];
        ourarray[left] = temp;
    }

    public static void permute(String[] ourArray, int currentPosition) {
        if (currentPosition == 1) {
            System.out.println(Arrays.toString(ourArray));
        } else {
            for (int i = 0; i < currentPosition; i++) {
                // subtract one from the last position (here is where you are
                // selecting the the next last item
                permute(ourArray, currentPosition - 1);

                // if it's odd position
                if (currentPosition % 2 == 1) {
                    swap(ourArray, 0, currentPosition - 1);
                } else {
                    swap(ourArray, i, currentPosition - 1);
                }
            }
        }
    }
```

answered Feb 1 '15 at 1:41

CPU 100

6,651 ● 1 ● 39 ● 45

This one is without recursion

```java
public static void permute(String s) {
    if(null==s || s.isEmpty()) {
        return;
    }

    // List containing words formed in each iteration
    List<String> strings = new LinkedList<String>();
    strings.add(String.valueOf(s.charAt(0))); // add the first element to the list

     // Temp list that holds the set of strings for
     //  appending the current character to all position in each word in the
original list
    List<String> tempList = new LinkedList<String>();

    for(int i=1; i< s.length(); i++) {

        for(int j=0; j<strings.size(); j++) {
            tempList.addAll(merge(s.charAt(i), strings.get(j)));
                        }
        strings.removeAll(strings);
        strings.addAll(tempList);

        tempList.removeAll(tempList);

    }

    for(int i=0; i<strings.size(); i++) {
        System.out.println(strings.get(i));
    }
}

/**
 * helper method that appends the given character at each position in the given
string
 * and returns a set of such modified strings
 * - set removes duplicates if any(in case a character is repeated)
 */
private static Set<String> merge(Character c,  String s) {
    if(s==null || s.isEmpty()) {
        return null;
    }

    int len = s.length();
    StringBuilder sb = new StringBuilder();
    Set<String> list = new HashSet<String>();

    for(int i=0; i<= len; i++) {
        sb = new StringBuilder();
        sb.append(s.substring(0, i) + c + s.substring(i, len));
        list.add(sb.toString());
    }

    return list;
}
```

edited Mar 23 '14 at 5:01
Jorge Israel Peña
19.8k ● 13 ● 66 ● 107

answered Nov 4 '13 at 21:39
Jeya
117 ● 1 ● 2

|

Let's use input `abc` as an example.

Start off with just the last element (`c`) in a set (`["c"]`), then add the second last element (`b`) to its front, end and every possible positions in the middle, making it `["bc", "cb"]` and then in the same manner it will add the next element from the back (`a`) to each string in the set making it:

```
"a" + "bc" = ["abc", "bac", "bca"]  and  "a" + "cb" = ["acb" ,"cab", "cba"]
```

Thus entire permutation:

```
["abc", "bac", "bca","acb" ,"cab", "cba"]
```

Code:

```
public class Test
{
    static Set<String> permutations;
    static Set<String> result = new HashSet<String>();

    public static Set<String> permutation(String string) {
        permutations = new HashSet<String>();

        int n = string.length();
        for (int i = n - 1; i >= 0; i--)
        {
            shuffle(string.charAt(i));
        }
        return permutations;
    }

    private static void shuffle(char c) {
        if (permutations.size() == 0) {
            permutations.add(String.valueOf(c));
        } else {
            Iterator<String> it = permutations.iterator();
            for (int i = 0; i < permutations.size(); i++) {

                String temp1;
                for (; it.hasNext();) {
                    temp1 = it.next();
                    for (int k = 0; k < temp1.length() + 1; k += 1) {
                        StringBuilder sb = new StringBuilder(temp1);

                        sb.insert(k, c);

                        result.add(sb.toString());
                    }
                }
            }
            permutations = result;
            //'result' has to be refreshed so that in next run it doesn't contain
stale values.
            result = new HashSet<String>();
        }
    }

    public static void main(String[] args) {
        Set<String> result = permutation("abc");

        System.out.println("\nThere are total of " + result.size() + "
permutations:");
        Iterator<String> it = result.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

edited Jun 8 '14 at 15:11
Dukeling
40.7k ● 9 ● 51 ● 96

answered May 25 '13 at 19:30
Vihaan Verma
3,962 ● 10 ● 58 ● 95

Well here is an elegant, non-recursive, O(n!) solution:

```java
public static StringBuilder[] permutations(String s) {
        if (s.length() == 0)
            return null;
        int length = fact(s.length());
        StringBuilder[] sb = new StringBuilder[length];
        for (int i = 0; i < length; i++) {
            sb[i] = new StringBuilder();
        }
        for (int i = 0; i < s.length(); i++) {
            char ch = s.charAt(i);
            int times = length / (i + 1);
            for (int j = 0; j < times; j++) {
                for (int k = 0; k < length / times; k++) {
                    sb[j * length / times + k].insert(k, ch);
                }
            }
        }
        return sb;
    }
```

answered Jun 27 '15 at 8:36

Adilli Adil

600 ● 9 ● 20

One of the simple solution could be just keep swapping the characters recursively using two pointers.

```java
public static void main(String[] args)
{
    String str="abcdefgh";
    perm(str);
}
public static void perm(String str)
{  char[] char_arr=str.toCharArray();
    helper(char_arr,0);
}
public static void helper(char[] char_arr, int i)
{
    if(i==char_arr.length-1)
    {
        // print the shuffled string
            String str="";
            for(int j=0; j<char_arr.length; j++)
            {
                str=str+char_arr[j];
            }
            System.out.println(str);
    }
    else
    {
    for(int j=i; j<char_arr.length; j++)
    {
        char tmp = char_arr[i];
        char_arr[i] = char_arr[j];
        char_arr[j] = tmp;
        helper(char_arr,i+1);
        char tmp1 = char_arr[i];
        char_arr[i] = char_arr[j];
        char_arr[j] = tmp1;
    }
}
}
```

answered Jul 28 '15 at 17:48

Khushi

322 ● 3 ● 13

|

Use recursion.

when the input is an empty string the only permutation is an empty string.Try for each of the letters in the string by making it as the first letter and then find all the permutations of the remaining letters using a recursive call.

```java
import java.util.ArrayList;
import java.util.List;

class Permutation {
    private static List<String> permutation(String prefix, String str) {
        List<String> permutations = new ArrayList<>();
        int n = str.length();
        if (n == 0) {
            permutations.add(prefix);
        } else {
            for (int i = 0; i < n; i++) {
                permutations.addAll(permutation(prefix + str.charAt(i),
str.substring(i + 1, n) + str.substring(0, i)));
            }
        }
        return permutations;
    }
    public static void main(String[] args) {
        List<String> perms = permutation("", "abcd");

        String[] array = new String[perms.size()];
        for (int i = 0; i < perms.size(); i++) {
            array[i] = perms.get(i);
        }

        int x = array.length;

        for (final String anArray : array) {
            System.out.println(anArray);
        }
    }
}
```

edited Mar 19 '15 at 22:58
Jared Burrows
33k ● 16 ● 106 ● 138

answered Mar 7 '15 at 15:46
coder101
369 ● 4 ● 16

---

this worked for me..

```
import java.util.Arrays;

public class StringPermutations{
    public static void main(String args[]) {
        String inputString = "ABC";
        permute(inputString.toCharArray(), 0, inputString.length()-1);
    }

    public static void permute(char[] ary, int startIndex, int endIndex) {
        if(startIndex == endIndex){
            System.out.println(String.valueOf(ary));
        }else{
            for(int i=startIndex;i<=endIndex;i++) {
                 swap(ary, startIndex, i );
                 permute(ary, startIndex+1, endIndex);
                 swap(ary, startIndex, i );
            }
        }
    }

    public static void swap(char[] ary, int x, int y) {
        char temp = ary[x];
        ary[x] = ary[y];
        ary[y] = temp;
    }
}
```

edited Apr 12 '16 at 11:19
Tunaki
74.1k ● 21 ● 143 ● 188

answered Apr 12 '16 at 11:17
Arun Kumar Mudraboyina
537 ● 7 ● 24

---

python implementation

```
def getPermutation(s, prefix=''):
        if len(s) == 0:
                print prefix
        for i in range(len(s)):
                getPermutation(s[0:i]+s[i+1:len(s)],prefix+s[i] )



getPermutation('abcd','')
```

answered Aug 8 '16 at 6:56
riteshkasat
398 ● 2 ● 13

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;
public class hello {
    public static void main(String[] args) throws IOException {
        hello h = new hello();
        h.printcomp();
    }
      int fact=1;
    public void factrec(int a,int k){
        if(a>=k)
        {fact=fact*k;
        k++;
        factrec(a,k);
        }
        else
        {System.out.println("The string  will have "+fact+" permutations");
        }
        }
    public void printcomp(){
        String str;
```

```java
        int k;
        Scanner in = new Scanner(System.in);
        System.out.println("enter the string whose permutations has to b found");
        str=in.next();
        k=str.length();
        factrec(k,1);
        String[] arr =new String[fact];
        char[] array = str.toCharArray();
        while(p<fact)
        printcomprec(k,array,arr);
            // if incase u need array containing all the permutation use this
            //for(int d=0;d<fact;d++)
        //System.out.println(arr[d]);
    }
    int y=1;
    int p = 0;
    int g=1;
    int z = 0;
    public void printcomprec(int k,char array[],String arr[]){
        for (int l = 0; l < k; l++) {
            for (int b=0;b<k-1;b++){
            for (int i=1; i<k-g; i++) {
                char temp;
                String stri = "";
                temp = array[i];
                array[i] = array[i + g];
                array[i + g] = temp;
                for (int j = 0; j < k; j++)
                    stri += array[j];
                arr[z] = stri;
                System.out.println(arr[z] + "   " + p++);
                z++;
            }
            }
            char temp;
            temp=array[0];
            array[0]=array[y];
            array[y]=temp;
            if (y >= k-1)
                y=y-(k-1);
            else
                y++;
        }
        if (g >= k-1)
            g=1;
        else
            g++;
    }

}
```

edited Aug 2 '13 at 12:25

answered Aug 2 '13 at 12:08

Antony Johnson
51 ● 2

```java
/** Returns an array list containing all
 * permutations of the characters in s. */
public static ArrayList<String> permute(String s) {
    ArrayList<String> perms = new ArrayList<>();
    int slen = s.length();
    if (slen > 0) {
        // Add the first character from s to the perms array list.
        perms.add(Character.toString(s.charAt(0)));

        // Repeat for all additional characters in s.
        for (int i = 1;  i < slen;  ++i) {

            // Get the next character from s.
            char c = s.charAt(i);

            // For each of the strings currently in perms do the following:
            int size = perms.size();
            for (int j = 0;  j < size;  ++j) {

                // 1. remove the string
                String p = perms.remove(0);
                int plen = p.length();

                // 2. Add plen + 1 new strings to perms.  Each new string
                //    consists of the removed string with the character c
                //    inserted into it at a unique location.
                for (int k = 0;  k <= plen;  ++k) {
                    perms.add(p.substring(0, k) + c + p.substring(k));
                }
            }
        }
    }
    return perms;
}
```

answered Sep 26 '13 at 2:10

Barzee
318 ● 1 ● 4 ● 18

Here is a straightforward minimalist recursive solution in Java:

```
public static ArrayList<String> permutations(String s) {
    ArrayList<String> out = new ArrayList<String>();
    if (s.length() == 1) {
        out.add(s);
        return out;
    }
    char first = s.charAt(0);
    String rest = s.substring(1);
    for (String permutation : permutations(rest)) {
        out.addAll(insertAtAllPositions(first, permutation));
    }
    return out;
}
public static ArrayList<String> insertAtAllPositions(char ch, String s) {
    ArrayList<String> out = new ArrayList<String>();
    for (int i = 0; i <= s.length(); ++i) {
        String inserted = s.substring(0, i) + ch + s.substring(i);
        out.add(inserted);
    }
    return out;
}
```

answered Nov 13 '13 at 22:26

Jay Taylor
7,005 ● 5 ● 37 ● 58

This is what I did through basic understanding of Permutations and Recursive function calling. Takes a bit of time but it's done independently.

```
public class LexicographicPermutations {

public static void main(String[] args) {
    // TODO Auto-generated method stub
    String s="abc";
    List<String>combinations=new ArrayList<String>();
    combinations=permutations(s);
    Collections.sort(combinations);
    System.out.println(combinations);
}

private static List<String> permutations(String s) {
    // TODO Auto-generated method stub
    List<String>combinations=new ArrayList<String>();
    if(s.length()==1){
        combinations.add(s);
    }
    else{
        for(int i=0;i<s.length();i++){
            List<String>temp=permutations(s.substring(0, i)+s.substring(i+1));
            for (String string : temp) {
                combinations.add(s.charAt(i)+string);
            }
        }
    }
    return combinations;
}}
```

which generates **Output** as `[abc, acb, bac, bca, cab, cba]`.

Basic logic behind it is

For each character, consider it as 1st character & find the combinations of remaining characters. e.g. `[abc](Combination of abc)->`.

1. `a->[bc](a x Combination of (bc))->{abc,acb}`
2. `b->[ac](b x Combination of (ac))->{bac,bca}`
3. `c->[ab](c x Combination of (ab))->{cab,cba}`

And then recursively calling each `[bc]`, `[ac]` & `[ab]` independently.

answered Sep 8 '16 at 12:29

Shabbir Essaji
79 ● 8

|

```
//Rotate and create words beginning with all letter possible and push to stack 1

//Read from stack1 and for each word create words with other letters at the next
location by rotation and so on

/*  eg : man

    1. push1 - man, anm, nma
    2. pop1 - nma ,  push2 - nam,nma
       pop1 - anm ,  push2 - amn,anm
       pop1 - man ,  push2 - mna,man
*/

public class StringPermute {

    static String str;
    static String word;
    static int top1 = -1;
    static int top2 = -1;
    static String[] stringArray1;
    static String[] stringArray2;
    static int strlength = 0;

    public static void main(String[] args) throws IOException {
        System.out.println("Enter String : ");
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader bfr = new BufferedReader(isr);
        str = bfr.readLine();
        word = str;
        strlength = str.length();
        int n = 1;
        for (int i = 1; i <= strlength; i++) {
            n = n * i;
        }
        stringArray1 = new String[n];
        stringArray2 = new String[n];
        push(word, 1);
        doPermute();
        display();
    }

    public static void push(String word, int x) {
        if (x == 1)
            stringArray1[++top1] = word;
        else
            stringArray2[++top2] = word;
    }

    public static String pop(int x) {
        if (x == 1)
            return stringArray1[top1--];
        else
            return stringArray2[top2--];
    }

    public static void doPermute() {

        for (int j = strlength; j >= 2; j--)
            popper(j);

    }

    public static void popper(int length) {
        // pop from stack1 , rotate each word n times and push to stack 2
        if (top1 > -1) {
            while (top1 > -1) {
                word = pop(1);
                for (int j = 0; j < length; j++) {
                    rotate(length);
                    push(word, 2);
                }
            }
        }
        // pop from stack2 , rotate each word n times w.r.t position and push to
        // stack 1
        else {
            while (top2 > -1) {
                word = pop(2);
                for (int j = 0; j < length; j++) {
                    rotate(length);
                    push(word, 1);
                }
            }
        }

    }

    public static void rotate(int position) {
        char[] charstring = new char[100];
        for (int j = 0; j < word.length(); j++)
            charstring[j] = word.charAt(j);

        int startpos = strlength - position;
        char temp = charstring[startpos];
        for (int i = startpos; i < strlength - 1; i++) {
            charstring[i] = charstring[i + 1];
        }
        charstring[strlength - 1] = temp;
        word = new String(charstring).trim();
    }

    public static void display() {
        int top;
        if (top1 > -1) {
            while (top1 > -1)
                System.out.println(stringArray1[top1--]);
        } else {
            while (top2 > -1)
                System.out.println(stringArray2[top2--]);
        }
    }
}
```

edited Jan 23 '14 at 1:39
Andrew Schulman
2,791 ● 1 ● 12 ● 18

answered Jan 23 '14 at 1:17
Nimmi Chettakarical Varkey
139 ● 4 ● 18

We can use factorial to find how many strings started with particular letter.

Example: take the input `abcd`. `(3!) == 6` strings will start with every letter of `abcd`.

```
static public int facts(int x){
    int sum = 1;
    for (int i = 1; i < x; i++) {
        sum *= (i+1);
    }
    return sum;
}

public static void permutation(String str) {
    char[] str2 = str.toCharArray();
    int n = str2.length;
    int permutation = 0;
    if (n == 1) {
        System.out.println(str2[0]);
    } else if (n == 2) {
        System.out.println(str2[0] + "" + str2[1]);
        System.out.println(str2[1] + "" + str2[0]);
    } else {
        for (int i = 0; i < n; i++) {
            if (true) {
                char[] str3 = str.toCharArray();
                char temp = str3[i];
                str3[i] = str3[0];
                str3[0] = temp;
                str2 = str3;
            }

            for (int j = 1, count = 0; count < facts(n-1); j++, count++) {
                if (j != n-1) {
                    char temp1 = str2[j+1];
                    str2[j+1] = str2[j];
                    str2[j] = temp1;
                } else {
                    char temp1 = str2[n-1];
                    str2[n-1] = str2[1];
                    str2[1] = temp1;
                    j = 1;
                } // end of else block
                permutation++;
                System.out.print("permutation " + permutation + " is   -> ");
                for (int k = 0; k < n; k++) {
                    System.out.print(str2[k]);
                } // end of loop k
                System.out.println();
            } // end of loop j
        } // end of loop i
    }
}
```

edited Jun 8 '14 at 15:51
Dukeling
40.7k ●9 ●51 ●96

answered May 25 '13 at 13:02
user1685821
19 ●1

Here is a java implementation:

```
/* All Permutations of a String */

import java.util.*;
import java.lang.*;
import java.io.*;

/* Complexity O(n*n!) */
class Ideone
{
     public static ArrayList<String> strPerm(String str, ArrayList<String> list)
     {
        int len = str.length();
        if(len==1){
            list.add(str);
            return list;
        }

        list = strPerm(str.substring(0,len-1),list);
        int ls = list.size();
        char ap = str.charAt(len-1);
        for(int i=0;i<ls;i++){
            String temp = list.get(i);
            int tl = temp.length();
            for(int j=0;j<=tl;j++){
                list.add(temp.substring(0,j)+ap+temp.substring(j,tl));
            }
        }

        while(true){
            String temp = list.get(0);
            if(temp.length()<len)
                list.remove(temp);
            else
                break;
        }

        return list;
```

```
    }

    public static void main (String[] args) throws java.lang.Exception
    {
        String str = "abc";
        ArrayList<String> list = new ArrayList<>();

        list = strPerm(str,list);
        System.out.println("Total Permutations : "+list.size());
        for(int i=0;i<list.size();i++)
            System.out.println(list.get(i));

    }
}
```

http://ideone.com/nWPb3k

answered Nov 3 '14 at 12:27

mav3n
640 ● 1 ● 5 ● 17

Recursion is not necessary, even you can calculate **any permutation directly**, this solution uses generics to permute any array.

Here is a good information about this algorihtm.

For **C#** developers here is more useful implementation.

```
public static void main(String[] args) {
    String word = "12345";

    Character[] array = ArrayUtils.toObject(word.toCharArray());
    long[] factorials = Permutation.getFactorials(array.length + 1);

    for (long i = 0; i < factorials[array.length]; i++) {
        Character[] permutation = Permutation.<Character>getPermutation(i, array,
factorials);
        printPermutation(permutation);
    }
}

private static void printPermutation(Character[] permutation) {
    for (int i = 0; i < permutation.length; i++) {
        System.out.print(permutation[i]);
    }
    System.out.println();
}
```

This algorithm has **O(N)** *time* and *space* complexity to calculate each **permutation**.

```
public class Permutation {
    public static <T> T[] getPermutation(long permutationNumber, T[] array, long[]
factorials) {
        int[] sequence = generateSequence(permutationNumber, array.length - 1,
factorials);
        T[] permutation = generatePermutation(array, sequence);

        return permutation;
    }

    public static <T> T[] generatePermutation(T[] array, int[] sequence) {
        T[] clone = array.clone();

        for (int i = 0; i < clone.length - 1; i++) {
            swap(clone, i, i + sequence[i]);
        }

        return clone;
    }

    private static int[] generateSequence(long permutationNumber, int size, long[]
factorials) {
        int[] sequence = new int[size];

        for (int j = 0; j < sequence.length; j++) {
            long factorial = factorials[sequence.length - j];
            sequence[j] = (int) (permutationNumber / factorial);
            permutationNumber = (int) (permutationNumber % factorial);
        }

        return sequence;
    }

    private static <T> void swap(T[] array, int i, int j) {
        T t = array[i];
        array[i] = array[j];
        array[j] = t;
    }

    public static long[] getFactorials(int length) {
        long[] factorials = new long[length];
        long factor = 1;

        for (int i = 0; i < length; i++) {
            factor *= i <= 1 ? 1 : i;
            factorials[i] = factor;
        }
        return factorials;
    }
}
```

edited May 23 at 10:31
Community ♦
1 ● 1

answered Oct 1 '15 at 21:03
Najera
2,247 ● 3 ● 17 ● 34

Java implementation without recursion

```
public Set<String> permutate(String s){
    Queue<String> permutations = new LinkedList<String>();
    Set<String> v = new HashSet<String>();
    permutations.add(s);

    while(permutations.size()!=0){
        String str = permutations.poll();
        if(!v.contains(str)){
            v.add(str);
            for(int i = 0;i<str.length();i++){
                String c = String.valueOf(str.charAt(i));
                permutations.add(str.substring(i+1) + c +  str.substring(0,i));
            }
        }
    }
    return v;
}
```

answered Jul 24 at 14:37
Hadi Elmougy
75 ● 6

//insert each character into an arraylist

```
static ArrayList al = new ArrayList();

private static void findPermutation (String str){
    for (int k = 0; k < str.length(); k++) {
        addOneChar(str.charAt(k));
    }
}

//insert one char into ArrayList
private static void addOneChar(char ch){
    String lastPerStr;
    String tempStr;
    ArrayList locAl = new ArrayList();
    for (int i = 0; i < al.size(); i ++ ){
        lastPerStr = al.get(i).toString();
        //System.out.println("lastPerStr: " + lastPerStr);
        for (int j = 0; j <= lastPerStr.length(); j++) {
            tempStr = lastPerStr.substring(0,j) + ch +
                    lastPerStr.substring(j, lastPerStr.length());
            locAl.add(tempStr);
            //System.out.println("tempStr: " + tempStr);
        }
    }
    if(al.isEmpty()){
        al.add(ch);
    } else {
        al.clear();
        al = locAl;
    }
}

private static void printArrayList(ArrayList al){
    for (int i = 0; i < al.size(); i++) {
        System.out.print(al.get(i) + "  ");
    }
}
```

answered Jul 6 '13 at 0:29
David Lee
19

|

Improved Code for the same

```
    static String permutationStr[];
    static int indexStr = 0;

    static int factorial (int i) {
        if (i == 1)
            return 1;
        else
            return i * factorial(i-1);
    }

    public static void permutation(String str) {
        char strArr[] = str.toLowerCase().toCharArray();
        java.util.Arrays.sort(strArr);
        int count = 1, dr = 1;
        for (int i = 0; i < strArr.length-1; i++){
            if ( strArr[i] == strArr[i+1]) {
                count++;
            } else {
                dr *= factorial(count);
                count = 1;
            }
        }
        dr *= factorial(count);

        count = factorial(strArr.length) / dr;

        permutationStr = new String[count];

        permutation("", str);

        for (String oneStr : permutationStr){
            System.out.println(oneStr);
        }
    }

    private static void permutation(String prefix, String str) {
        int n = str.length();
        if (n == 0) {
            for (int i = 0; i < indexStr; i++){
                if(permutationStr[i].equals(prefix))
                    return;
            }
            permutationStr[indexStr++] = prefix;
        } else {
            for (int i = 0; i < n; i++) {
                permutation(prefix + str.charAt(i), str.substring(0, i) +
str.substring(i + 1, n));
            }
        }
    }
```

answered Jul 13 '13 at 1:49

dayitv89
2,234 ● 2 ● 20 ● 40

|

```
/*
     * eg: abc =>{a,bc},{b,ac},{c,ab}
     * =>{ca,b},{cb,a}
     * =>cba,cab
     * =>{ba,c},{bc,a}
     * =>bca,bac
     * =>{ab,c},{ac,b}
     * =>acb,abc
     */
    public void nonRecpermute(String prefix, String word)
    {
        String[] currentstr ={prefix,word};
        Stack<String[]> stack = new Stack<String[]>();
        stack.add(currentstr);
        while(!stack.isEmpty())
        {
            currentstr = stack.pop();
            String currentPrefix = currentstr[0];
            String currentWord = currentstr[1];
            if(currentWord.equals(""))
            {
                System.out.println("Word ="+currentPrefix);
            }
            for(int i=0;i<currentWord.length();i++)
            {
                String[] newstr = new String[2];
                newstr[0]=currentPrefix + String.valueOf(currentWord.charAt(i));
                newstr[1] = currentWord.substring(0, i);
                if(i<currentWord.length()-1)
                {
                    newstr[1] = newstr[1]+currentWord.substring(i+1);
                }
                stack.push(newstr);
            }

        }

    }
```

answered Apr 21 '14 at 20:30

Nimmi Chettakarical Varkey
139 ● 4 ● 18

This can be done iteratively by simply inserting each letter of the string in turn in all locations of the previous partial results.

We start with `[A]`, which with `B` becomes `[BA, AB]`, and with `C`, `[CBA, BCA, BAC, CAB, etc]`.

The running time would be `O(n!)`, which, for the test case `ABCD`, is `1 x 2 x 3 x 4`.

In the above product, the `1` is for `A`, the `2` is for `B`, etc.

Dart sample:

```
void main() {

  String insertAt(String a, String b, int index)
  {
    return a.substring(0, index) + b + a.substring(index);
  }

  List<String> Permute(String word) {

    var letters = word.split('');

    var p_list = [ letters.first ];

    for (var c in letters.sublist(1)) {

      var new_list = [ ];

      for (var p in p_list)
        for (int i = 0; i <= p.length; i++)
          new_list.add(insertAt(p, c, i));

      p_list = new_list;
    }

    return p_list;
  }

  print(Permute("ABCD"));

}
```

edited Jun 8 '14 at 15:04
Dukeling
40.7k ●9 ●51 ●96

answered Jan 24 '14 at 7:53
Troy Dawson
48 ●5

Here are two c# versions (just for reference): 1. Prints all permuations 2. returns all permutations

Basic gist of the algorithm is (probably below code is more intuitive - nevertheless, here is some explanation of what below code does): - from the current index to for the rest of the collection, swap the element at current index - get the permutations for the remaining elements from next index recursively - restore the order, by re-swapping

Note: the above recursive function will be invoked from the start index.

```
private void PrintAllPermutations(int[] a, int index, ref int count)
        {
            if (index == (a.Length - 1))
            {
                count++;
                var s = string.Format("{0}: {1}", count, string.Join(",", a));
                Debug.WriteLine(s);
            }
            for (int i = index; i < a.Length; i++)
            {
                Utilities.swap(ref a[i], ref a[index]);
                this.PrintAllPermutations(a, index + 1, ref count);
                Utilities.swap(ref a[i], ref a[index]);
            }
        }
        private int PrintAllPermutations(int[] a)
        {
            a.ThrowIfNull("a");
            int count = 0;
            this.PrintAllPermutations(a, index:0, count: ref count);
            return count;
        }
```

version 2 (same as above - but returns the permutations in lieu of printing)

```
private int[][] GetAllPermutations(int[] a, int index)
        {
            List<int[]> permutations = new List<int[]>();
            if (index == (a.Length - 1))
            {
                permutations.Add(a.ToArray());
            }

            for (int i = index; i < a.Length; i++)
            {
                Utilities.swap(ref a[i], ref a[index]);
                var r = this.GetAllPermutations(a, index + 1);
                permutations.AddRange(r);
                Utilities.swap(ref a[i], ref a[index]);
            }
            return permutations.ToArray();
        }
        private int[][] GetAllPermutations(int[] p)
        {
            p.ThrowIfNull("p");
            return this.GetAllPermutations(p, 0);
        }
```

Unit Tests

```
[TestMethod]
        public void PermutationsTests()
        {
            List<int> input = new List<int>();
            int[] output = { 0, 1, 2, 6, 24, 120 };
            for (int i = 0; i <= 5; i++)
            {
                if (i != 0)
                {
                    input.Add(i);
                }

Debug.WriteLine("================PrintAllPermutations===================");
                int count = this.PrintAllPermutations(input.ToArray());
                Assert.IsTrue(count == output[i]);

Debug.WriteLine("=====================GetAllPermutations=================");
                var r = this.GetAllPermutations(input.ToArray());
                Assert.IsTrue(count == r.Length);
                for (int j = 1; j <= r.Length;j++ )
                {
                    string s = string.Format("{0}: {1}", j,
                        string.Join(",", r[j - 1]));
                    Debug.WriteLine(s);
                }
                Debug.WriteLine("No.OfElements: {0}, TotalPerms: {1}", i, count);
            }
        }
```

answered Jul 30 '14 at 5:30

Dreamer
2,015 ● 2 ● 17 ● 38

Another simple way is to loop through the string, pick the character that is not used yet and put it to a buffer, continue the loop till the buffer size equals to the string length. I like this back tracking solution better because:

1. Easy to understand
2. Easy to avoid duplication
3. The output is sorted

Here is the java code:

```
List<String> permute(String str) {
  if (str == null) {
    return null;
  }

  char[] chars = str.toCharArray();
  boolean[] used = new boolean[chars.length];

  List<String> res = new ArrayList<String>();
  StringBuilder sb = new StringBuilder();

  Arrays.sort(chars);

  helper(chars, used, sb, res);

  return res;
}

void helper(char[] chars, boolean[] used, StringBuilder sb, List<String> res) {
  if (sb.length() == chars.length) {
    res.add(sb.toString());
    return;
  }

  for (int i = 0; i < chars.length; i++) {
    // avoid duplicates
    if (i > 0 && chars[i] == chars[i - 1] && !used[i - 1]) {
      continue;
    }

    // pick the character that has not used yet
    if (!used[i]) {
      used[i] = true;
      sb.append(chars[i]);

      helper(chars, used, sb, res);

      // back tracking
      sb.deleteCharAt(sb.length() - 1);
      used[i] = false;
    }
  }
}
```

Input str: 1231

Output list: {1123, 1132, 1213, 1231, 1312, 1321, 2113, 2131, 2311, 3112, 3121, 3211}

Noticed that the output is sorted, and there is no duplicate result.

edited Jun 10 '15 at 23:31

answered Jun 10 '15 at 23:14

jean.timex
904 ● 9 ● 9

This is a C solution:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>


char* addLetter(char* string, char *c) {
    char* result = malloc(sizeof(string) + 2);
    strcpy(result, string);
    strncat(result, c, 1);
    return result;
}

char* removeLetter(char* string, char *c) {
    char* result = malloc(sizeof(string));
    int j = 0;
    for (int i = 0; i < strlen(string); i++) {
        if (string[i] != *c) {
            result[j++] = string[i];
        }
    }
    result[j] = '\0';

    return result;
}

void makeAnagram(char *anagram, char *letters) {

    if (*letters == '\0') {
        printf("%s\n", anagram);
        return;
    }

    char *c = letters;
    while (*c != '\0') {
        makeAnagram(addLetter(anagram, c),
                    removeLetter(letters, c));
        c++;
    }

}

int main() {

    makeAnagram("", "computer");

    return 0;
}
```

answered Jan 26 '16 at 17:16

etayluz
5,021 ● 2 ● 37 ● 56

My implementation based on Mark Byers's description above:

```
    static Set<String> permutations(String str){
        if (str.isEmpty()){
            return Collections.singleton(str);
        }else{
            Set <String> set = new HashSet<>();
            for (int i=0; i<str.length(); i++)
                for (String s : permutations(str.substring(0, i) +
str.substring(i+1)))
                    set.add(str.charAt(i) + s);
            return set;
        }
    }
```

edited May 23 '16 at 10:33

answered May 23 '16 at 10:10

Grisha Weintraub
5,878 ● 1 ● 13 ● 35

---

1 2 next

Answer This Question