**String important question and answers**

### Q1. - Reverse words in a sentence??

### Ans: Problem: Reverse the order of words in a sentence, but keep words themselves unchanged. Words in a sentence are divided by blanks. For instance, the reversed output should be “student. a am I” when the input is “I am a student”.

Ans:**Analysis:**This is a very popular interview question of many companies. It can be solved with two steps: Firstly we reverse all characters in a sentence. If all characters in sentence “I am a student.” are reversed, it becomes “.tneduts a ma I”. Not only the order of words is reversed, but also the order of characters inside each word is reversed. Secondly, we reverse characters in every word. We can get “student. a am I” from the example input string with these two steps.

The key of our solution is to implement a function to reverse a string, which is shown as the Reverse function below:

void Reverse(char \*pBegin, char \*pEnd)

{

    if(pBegin == NULL || pEnd == NULL)

        return;

    while(pBegin < pEnd)

    {

        char temp = \*pBegin;

        \*pBegin = \*pEnd;

        \*pEnd = temp;

        pBegin ++, pEnd --;

    }

}

Now we can reverse the whole sentence and each word based on this Reverse function with the following code:

char\* ReverseSentence(char \*pData)

{

    if(pData == NULL)

        return NULL;

    char \*pBegin = pData;

    char \*pEnd = pData;

    while(\*pEnd != '\0')

        pEnd ++;

    pEnd--;

    // Reverse the whole sentence

    Reverse(pBegin, pEnd);

    // Reverse every word in the sentence

    pBegin = pEnd = pData;

    while(\*pBegin != '\0')

    {

        if(\*pBegin == ' ')

        {

            pBegin ++;

            pEnd ++;

        }

        else if(\*pEnd == ' ' || \*pEnd == '\0')

        {

            Reverse(pBegin, --pEnd);

            pBegin = ++pEnd;

        }

        else

        {

            pEnd ++;

        }

    }

    return pData;

}

Since words are separated by blanks, we can get the beginning position and ending position of each word by scanning blanks. In the second phrase to reverse each word in the sample code above, the pointer pBegin points to the first character of a word, and pEnd points to the last character.

### Q2: First Character Appearing Only Once??

**Problem:** Implement a function to find the first character in a string which only appears once.

For example: It returns ‘b’ when the input is “abaccdeff”.

Ans: **Analysis:** Our native solution for this problem may be scanning the input string from its beginning to end. We compare the current scanned character with every one behind it. If there is no duplication after it, it is a character appearing once. Since it compares each character with O(n) ones behind it, the overall time complexity is O(n2) if there are n characters in a string.

In order to get the numbers of occurrence times of each character in a string, a data container is needed. It is required to get and update the occurrence time of each character in a string, so the data container is used to project a character to a number. Hash tables fulfill this kind of requirement. We can implement a hash table, in which keys are characters and values are their occurrence times in a string.

It is necessary to scan strings twice: When a character is visited, we increase the corresponding occurrence time in the hash table during the first scanning. In second round of scanning, whenever a character is visited we also check its occurrence time in the hash table. The first character with occurrence time 1 is the required output.

Hash tables are complex, and they are not implemented in the C++ standard template library. Therefore, we have to implement one by ourselves.

Characters have 8 bits, so there only 256 variances. We can create an array with 255 numbers, in which indexes are ASCII values of all characters, and numbers are their occurrence times in a string. That is to say, we have a hash table whose size if 256, with ASCII values of characters as keys.

It is time for programming after we get a clear solution. The following are some sample code:

char FirstNotRepeatingChar(char\* pString)

{

    if(pString == NULL)

        return '\0';

    const int tableSize = 256;

    unsigned int hashTable[tableSize];

    for(unsigned int i = 0; i<tableSize; ++ i)

        hashTable[i] = 0;

    char\* pHashKey = pString;

    while(\*(pHashKey) != '\0')

        hashTable[\*(pHashKey++)] ++;

    pHashKey = pString;

    while(\*pHashKey != '\0')

    {

        if(hashTable[\*pHashKey] == 1)

            return \*pHashKey;

        pHashKey++;

    }

    return '\0';

}

In the code above, it costs O(1) time to increase the occurrence time for each character. The time complexity for the first scanning is O(n) if the length of string is n. It takes O(1) time to get the occurrence time for each character, so it costs O(n) time for the second scanning. Therefore, the overall time it costs is O(n).

In the meantime, an array with 256 numbers is created, whose size is 1K. Since the size of array is constant, the space complexity of this algorithm is O(1).

### Q3: Left Rotation of String??

**Problem:** Left rotation of a string is to move some leading characters to its tail. Please implement a function to rotate a string.

For example, if the input string is “abcdefg” and a number 2, the rotated result is “cdefgab”.

**Analysis:** It looks difficult to get rules of left rotation on a string. Fortunately, the 7th problem in this series “[Reverse Words in a Sentence](http://codercareer.blogspot.com/2011/09/no-07-reverse-words-in-sentence.html)” can give us some hints.

If we input a sentence with two words “hello world” for the problem “Reverse Words in a Sentence”, the reversed result should be “world hello”. It is noticeable that the result “world hello” can be viewed as a rotated result of “hello world”. It becomes “world hello” when we move some leading characters of string “hello world” to its ending. Therefore, this problem is quite similar to problem “Reverse Words in a Sentence”.

Let us take a string “abcdefg” as an example. We divide it into two parts: the first part contains the two leading characters “ab”, and the second part contains all other characters “cdefg”. We firstly reverse these two parts separately, and the whole string becomes “bagfedc”. It becomes “cdefgab” if we reverse the whole string, which is the expected result of left rotation with 2.

According to the analysis above, we can see that left rotation of a string can be implemented calling a Reverse function three times to reverse a segment or whole string. The sample code is shown below:

char\* LeftRotateString(char\* pStr, int n)

{

    if(pStr != NULL)

    {

        int nLength = static\_cast<int>(strlen(pStr));

        if(nLength > 0 && n > 0 && n < nLength)

        {

            char\* pFirstStart = pStr;

            char\* pFirstEnd = pStr + n - 1;

            char\* pSecondStart = pStr + n;

            char\* pSecondEnd = pStr + nLength - 1;

            // Reverse the n leading characters

            Reverse(pFirstStart, pFirstEnd);

            // Reverse other characters

            Reverse(pSecondStart, pSecondEnd);

            // Reverse the whole string

            Reverse(pFirstStart, pSecondEnd);

        }

    }

    return pStr;

}

The function Reverse is shown in “[Reverse Words in a Sentence](http://codercareer.blogspot.com/2011/09/no-07-reverse-words-in-sentence.html)”, so we are not going to repeat it here.

### Q4. Minimal Number of Palindromes on a String??

**Problem:** A string can be partitioned into some substrings, such that each substring is a palindrome. For example, there are a few strategies to split the string “abbab” into palindrome substrings, such as: “abba”|”b”, “a”|”b”|”bab” and “a”|”bb”|”a”|”b”.

Given a string *str*, please get the minimal numbers of splits to partition it into palindromes. The minimal number of splits to partition the string “abbab” into a set of palindromes is 1.

**Analysis:** This is a typical problem which can be solved by dynamic programming. We have two strategies to analyze and solve this problem

***Solution 1: Split at any space between two characters***

Given a substring of *str*, starting from the index *i* and ending at the index *j*(denoted as *str*[*i*:*j*]), we define a function *f*(*i*, *j*) to denote the minimal number of splits to partition the substring *str*[*i*:*j*] into a set of palindromes. If the substring is a palindrome itself, we don’t have to split so *f*(*i*, *j*) is 0. If the substring is not a palindrome, the substring is split between two characters *k* and *k*+1. *f*(*i*, *j*)=*f*(*i*,*k*)+*f*(*k*+1, *j*)+1 under such conditions. Therefore, *f*(*i*, *j*) can be defined with the following equation:

http://1.bp.blogspot.com/-WmmwSMuUmK8/USc-stTPIxI/AAAAAAAABb4/rcrMg7T4cek/s1600/Figure+1.PNG

The value of *f*(0, *n*-1) is the value of the minimal number of splits to partition *str* into palindromes, if *n* is the length of *str*.

If the equation is calculated recursively, its complexity grows exponentially with the length *n*. A better choice is to calculate in bottom-up order with a 2D matrix with size *n*×*n*. The following C++ code implements this solution:

int minSplit\_1(const string& str)

{

    int length = str.size();

    int\* split = new int[length \* length];

    for(int i = 0; i < length; ++i)

        split[i \* length + i] = 0;

    for(int i = 1; i < length; ++i)

    {

        for(int j = length - i; j > 0; --j)

        {

            int row = length - i - j;

            int col = row + i;

            if(isPalindrome(str, row, col))

            {

                split[row \* length + col] = 0;

            }

            else

            {

                int min = 0x7FFFFFFF;

                for(int k = row; k < col; ++k)

                {

                    int temp1 = split[row \* length + k];

                    int temp2 = split[(k + 1) \* length + col];

                    if(min > temp1 + temp2 + 1)

                        min = temp1 + temp2 + 1;

                }

                split[row \* length + col] = min;

            }

        }

    }

    int minSplit = split[length - 1];

    delete[] split;

    return minSplit;

}

***Solution 2: Split only before a palindrome***

We split the string *str* with another strategy. Given a substring ending at the index *i*, *str*[0, i], we do not have to split if the substring is a palindrome itself. Otherwise it is split between two characters at index *j* and *j*+1 only if the substring *str*[*j*+1,*i*] is a palindrome. Therefore, an equation*f*(*i*) can be defined as the following:

http://1.bp.blogspot.com/-TvrztCk9Cvw/USc-5ix1EUI/AAAAAAAABcA/qEwq-fhiLeQ/s1600/Figure+2.PNG

The value of *f*(*n*-1) is the value of the minimal number of splits to partition *str* into palindromes, if *n*is the length of *str*.

We could utilize a 1D array to solve this equation in bottom-up order, as listed in the following code:

int minSplit\_2(const string& str)

{

    int length = str.size();

    int\* split = new int[length];

    for(int i = 0; i < length; ++i)

        split[i] = i;

    for(int i = 1; i < length; ++i)

    {

        if(isPalindrome(str, 0, i))

        {

            split[i] = 0;

            continue;

        }

        for(int j = 0; j < i; ++j)

        {

            if(isPalindrome(str, j + 1, i) && split[i] > split[j] + 1)

                split[i] = split[j] + 1;

        }

    }

    int minSplit = split[length - 1];

    delete[] split;

    return minSplit;

}

***Optimization to verify palindromes:***

Usually it costs O(*n*) time to check whether a string with length *n* is a palindrome, and the typical implementation looks like the following code:

bool isPalindrome(const string& str, int begin, int end)

{

    for(int i = begin; i < end - (i - begin); ++i)

    {

        if(str[i] != str[end - (i - begin)])

            return false;

    }

    return true;

}

Both solutions above cost O(*n*3) time. The first solution contains three nesting for-loops. The function isPalindrome is inside two nesting for-loops.

If we could reduce the cost of isPalindrome to O(1), the time complexity of the second solution would be O(*n*2).

### Q5:Translating Numbers to Strings??

**Ans: Question:** Given a number, please translate it to a string, following the rules: 1 is translated to 'a', 2 to 'b', …, 12 to 'l', …, 26 to 'z'. For example, the number 12258 can be translated to "abbeh", "aveh", "abyh", "lbeh" and "lyh", so there are 5 different ways to translate 12258. How to write a function/method to count the different ways to translate a number?

**Analysis:** Let's take the number 12258 as an example to analyze the steps to translate from the beginning character to the ending one. There are two possible first characters in the translated string. One way is to split the number 12258 into 1 and 2258 two parts, and 1 is translated into 'a'. The other way is to split the number 12258 into 12 and 258 two parts, and 12 is translated into 'l'.

When the first one or two digits are translated into the first character, we can continue to translate the remaining digits. Obviously, we could write a recursive function/method to translate.

Let's define a function *f*(*i*) as the count of different ways to translate a number starting from the *ith* digit, *f*(*i*)=*g*(*i*)\**f*(*i*+1)+*h*(*i*, *i*+1)\**f*(*i*+2). The function *g*(i) gets 1 when the *ith* digit is in the range 1 to 9 which can be converted to a character, otherwise it gets 0. The function *h*(*i*, *i*+1) gets 1 the *ith* and (*i*+1)*th* digits are in the range 10 to 26 which can also be converted to a character. A single digit 0 can't be converted to a character, and two digits starting with a 0, such as 01 and 02, can't be converted either.

Even though the problem is analyzed with recursion, recursion is not the best approach because of overlapping sub-problems. For example,  The problem to translate 12258 is split into two sub-problems: one is to translate 1 and 2258, and the other is to translate 12 and 258. In the next step during recursion, the problem to translate 2258 can also split into two sub-problems: one is to translate 2 and 258, and the other is to translate 22 and 58. Notice the sub-problem to translate 258 reoccurs.

Recursion solves problem in the top-down order. We could solve this problem in the bottom-up order, in order to eliminate overlap sub-problems. That's to say, we start to translate the number from the ending digits, and then move from right to left during translation.

The following is the C# code to solve this problem:

public static int GetTranslationCount(int number)

{

    if (number <= 0)

    {

        return 0;

    }

    string numberInString = number.ToString();

    return GetTranslationCount(numberInString);

}

private static int GetTranslationCount(string number)

{

    int length = number.Length;

    int[] counts = new int[length];

    for (int i = length - 1; i >= 0; --i)

    {

        int count = 0;

        if (number[i] >= '1' && number[i] <= '9')

        {

            if (i < length - 1)

            {

                count += counts[i + 1];

            }

            else

            {

                count += 1;

            }

        }

        if (i < length - 1)

        {

            int digit1 = number[i] - '0';

            int digit2 = number[i + 1] - '0';

            int converted = digit1 \* 10 + digit2;

            if (converted >= 10 && converted <= 26)

            {

                if (i < length - 2)

                {

                    count += counts[i + 2];

                }

                else

                {

                    count += 1;

                }

            }

        }

        counts[i] = count;

    }

    return counts[0];

}

In order to simply the code implementation, we first convert the number into a string, and then translate.

**Question6. Implement an algorithm to determine if a string has all unique characters??**

Ans:

|  |
| --- |
| public class UniqueChar { |
|  | /\*\* |
|  | \* @param args |
|  | \*/ |
|  | public static void main(String[] args) { |
|  | String[] strArray ={"abcdef","dfdsafdsf"}; |
|  | for (String str : strArray) |
|  | System.out.println(isUniqueChar1(str)); |
|  | for (String str : strArray) |
|  | System.out.println(isUniqueChar2(str)); |
|  | for (String str : strArray) |
|  | System.out.println(isUniqueChar3(str)); |
|  | } |
|  | //bitmap method |
|  | public static boolean isUniqueChar1(String str){ |
|  | boolean [] set= new boolean[256]; |
|  | for (int i=0; i < 256; i++) |
|  | set[i] = false; |
|  | int len=str.length(); |
|  | for (int j =0; j < len; j++){ |
|  | int val =str.charAt(j); |
|  | if (set[val] == true) |
|  | return false; |
|  | else |
|  | set[val] = true ; |
|  | } |
|  | return true; |
|  | } |
|  | //hash table |
|  | public static boolean isUniqueChar2(String str){ |
|  | Map<Integer,Character> charMap =new HashMap<Integer,Character>(); |
|  | for (int i=0; i < str.length(); i++){ |
|  | char val = str.charAt(i); |
|  | int key = val; |
|  | if ((charMap.containsKey(key)) == true){ |
|  | return false; |
|  | } |
|  | else |
|  | charMap.put(key,val); |
|  | } |
|  | return true; |
|  | } |
|  | //quicksort |
|  | public static boolean isUniqueChar3(String str){ |
|  | char [] data = str.toCharArray(); |
|  | quicksort(data,0,data.length-1); |
|  | for (int i = 1; i < data.length;i++){ |
|  | if (data[i] == data[i - 1]) |
|  | return false; |
|  | } |
|  | return true; |
|  | } |
|  | public static void quicksort(char [] data, int left, int right){ |
|  | if (left < right){ |
|  | int pivotindex = partition(data, left, right); |
|  | quicksort(data,left,pivotindex-1); |
|  | quicksort(data,pivotindex+1,right); |
|  | } |
|  | } |
|  | public static int partition(char[] data, int left, int right){ |
|  | char pivot = data[right]; |
|  | int i = left -1 ; |
|  | char temp; |
|  | for(int j= left; j < right ; j++){ |
|  | if (data[j] <= pivot){ |
|  | i++; |
|  | temp = data [j]; |
|  | data[j]=data[ i]; |
|  | data[i]=temp; |
|  | } |
|  | } |
|  | temp=data[right]; |
|  | data[right] = data[i+1]; |
|  | data[i+1] = temp; |
|  | return i+1; |
|  | } |
|  | } |

**Q7://Write code to reverse a C-Style String. (C-String means that “abcd” is represented as five characters, including the null character.)**

|  |
| --- |
| **Ans:** |
|  | void reverse (char \*pt){ |
|  | char \*end = pt; |
|  | char temp; |
|  | while(true){ |
|  | if (\*end == null) |
|  | break; |
|  | end++; |
|  | } |
|  | --end; |
|  | while(pt<end){ |
|  | temp = \*pt; |
|  | \*pt++ = \*end; |
|  | \*end-- = temp; |
|  | } |
|  | } |

**Q8: \* Design an algorithm and write code to remove the duplicate characters in a string without**

|  |
| --- |
|  |
|  | \* using any additional buffer.  NOTE: One or two additional variables are fine. |
|  | \* An extra copy of the array is not.  **Ans:**   |  | | --- | | public class RemoveDul{ | |  | public static void main(String[] args){ | |  | RemoveDul q = new RemoveDul(); | |  | StringBuffer[] strings ={ | |  | new StringBuffer("abcdddg"), | |  | new StringBuffer("aabbccddeeffgghhiijjkkllmmnn"), | |  | new StringBuffer("abcdefgabcdefg"), | |  | new StringBuffer("aaaaaaa"), | |  | new StringBuffer("t"), | |  | new StringBuffer("ababababababa"), | |  | new StringBuffer("12 32 9 ' e.g e' f"), | |  | new StringBuffer(), | |  | }; | |  | for (StringBuffer str : strings){ | |  | q.DeleteDuplicate1(str); | |  | System.out.println(str); | |  | } | |  | } | |  | public void DeleteDuplicate1(StringBuffer str){ | |  | if (str == null) | |  | return ; | |  | int len = str.length(); | |  | if ((len == 0) || (len ==1)) | |  | return ; | |  | int tail = 0; | |  | for ( int j = 1; j < len; j++){ | |  | int i = tail; | |  | boolean dul = false; | |  | while (i >= 0){ | |  | if (str.charAt(i) == str.charAt(j)){ | |  | dul = true; | |  | break; | |  | } | |  | i--; | |  | } | |  | if (dul == false) | |  | str.setCharAt(++tail,str.charAt(j)); | |  | } | |  | if(tail == len -1) | |  | return ; | |  | str.delete(tail+1, len); | |  | } | |  | }  **Q9: ReplaceSpace between the strings??**  Ans:   |  | | --- | | public class ReplaceSpace { | |  | public static void main(String[] args){ | |  | StringBuffer s = new StringBuffer("Hello Chen Geng"); | |  | int originlen = s.length(); | |  | int spacenum = 0; | |  | for(int i = 0; i < originlen;i++){ | |  | if(s.charAt(i) == ' ') | |  | spacenum++;; | |  | } | |  | for (int j = 0; j < spacenum; j++){ | |  | s.append(' '); | |  | s.append(' '); | |  | } | |  | int newlen = s.length(); | |  | char[]str = s.toString().toCharArray(); | |  | start(str,originlen,newlen); | |  | System.out.println(new String(str)); | |  | } | |  | public static void start(char[]str,int originlen,int newlen){ | |  |  | |  | int k = newlen - 1; | |  | for(int j = originlen - 1; j >=0; j--){ | |  | if (str[j] == ' '){ | |  | str[k] = '%'; | |  | str[--k] = '0'; | |  | str[--k] = '2'; | |  | k--; | |  |  | |  | }else{ | |  | str[k--] = str[j]; | |  |  | |  | } | |  | } | |  |  | |  | } | |  | } | | |

**Q10: Given two strings, s1 and s2, write code to check if s2 is a rotation of s1 using only one**

**call to strstr (eg, “waterbottle” is a rotation of “erbottlewat”).??**

Ans:

|  |
| --- |
| public class Rotation { |
|  | public static void main(String[] args){ |
|  | Rotation r = new Rotation(); |
|  | int[][]matrix1 ={{1,2,3},{4,5,6},{7,8,9}}; |
|  | r.rotate(matrix1, 3); |
|  | for(int i = 0; i< 3;i++){ |
|  | for(int j = 0;j < 3; j++){ |
|  | System.out.print(matrix1[i][j] + " "); |
|  | } |
|  | System.out.println(); |
|  | } |
|  | System.out.println(); |
|  |  |
|  | int[][]matrix2 ={{1,2,3},{4,5,6},{7,8,9}}; |
|  | r.antirotate(matrix2, 3); |
|  | for(int i = 0; i< 3;i++){ |
|  | for(int j = 0;j < 3; j++){ |
|  | System.out.print(matrix2[i][j] + " "); |
|  | } |
|  | System.out.println(); |
|  | } |
|  | } |
|  | public void antirotate(int[][]matrix,int n){ |
|  | for(int layer = 0; layer <n/2; layer ++){ |
|  | int first = layer; |
|  | int last = n - 1- layer; |
|  | for(int i = first; i< last;i ++){ |
|  | int offset = i - first; |
|  | // save top |
|  | int top = matrix[first][i]; |
|  | //top <- right |
|  | matrix[first][i] = matrix[i][last]; |
|  | //right <- bottom |
|  | matrix[i][last] = matrix[last][last - offset]; |
|  | //bottom <-left |
|  | matrix[last][last - offset] = matrix[last - offset][first]; |
|  | //left <- top |
|  | matrix[last - offset][first] = top; |
|  |  |
|  | } |
|  | } |
|  | } |
|  | public void rotate(int[][]matrix,int n){ |
|  | for (int layer = 0; layer < n/2; layer ++){ |
|  | int first = layer; |
|  | int last = n - 1 - layer; |
|  | for (int i = first; i < last; i++){ |
|  | int offset = i - first; |
|  | //save the top |
|  | int top = matrix[first][i]; |
|  | // top <- left |
|  | matrix[first][i] = matrix[last - offset][first]; |
|  | // left <- bottom |
|  | matrix[last - offset][first] = matrix[last][last - offset]; |
|  | //bottom <- right |
|  | matrix[last][last - offset] = matrix[i][last]; |
|  | //right <- top |
|  | matrix[i][last] = top; |
|  |  |
|  | } |
|  | } |
|  | } |
|  | } |

**Q11: Assume you have a method isSubstring which checks if one word is a substring of another.**

**\* Given two strings, s1 and s2, write code to check if s2 is a rotation of s1 using only one**

**\* call to isSubstring (e.g., "waterbottle" is a rotation of "erbottlewat").??**

Ans:

|  |
| --- |
| public class IsRotation { |
|  |  |
|  | public static void main(String[] args) { |
|  |  |
|  | System.out.println(rotation(new StringBuffer("abcdefg"),new StringBuffer("cdefgab"))); |
|  | System.out.println(rotation(new StringBuffer("abcdefg"),new StringBuffer("gfedcba"))); |
|  |  |
|  | } |
|  |  |
|  | public static boolean rotation(StringBuffer s1 , StringBuffer s2){ |
|  | int len1 = s1.length(); |
|  | int len2 = s2.length(); |
|  | if (len1 != len2) |
|  | return false; |
|  | s1.append(s1); |
|  | String a = "anc"; |
|  | String b = "a"; |
|  | a.contains(b); |
|  | return isSubstring(s1,s2); |
|  | } |
|  |  |
|  | public static boolean isSubstring(StringBuffer s1, StringBuffer s2){ |
|  | int len = s2.length(); |
|  | for(int start = 0; start < len; start ++){ |
|  | int j = start; |
|  | for(int i = 0 ;i < len; i++){ |
|  | if (s2.charAt(i) == s1.charAt(j)){ |
|  | if( i == len -1) |
|  | return true; |
|  | j++; |
|  | }else{ |
|  | break; |
|  | } |
|  |  |
|  | } |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | } |

**Q12: write a method to sort an array of strings so that all the anagrams are next to each??**

**Ans:**

|  |
| --- |
| public class  AnagramSort { |
|  | public static void main (String[] args){ |
|  |  |
|  | StringBuffer[] str = { |
|  | new StringBuffer("lhep"), |
|  | new StringBuffer("abc"), |
|  | new StringBuffer("elpH"), |
|  | new StringBuffer("cba"), |
|  | new StringBuffer("help"), |
|  | new StringBuffer("Bca"), |
|  |  |
|  | }; |
|  |  |
|  | SignitureString[] ss = new SignitureString[str.length]; |
|  | for(int i = 0 ; i < str.length; i ++){ |
|  | ss[i] = new SignitureString(str[i]); |
|  | } |
|  | Quicksort<SignitureString> qs = new Quicksort<SignitureString>(); |
|  | qs.sort(ss); |
|  | for(SignitureString sig : ss){ |
|  | System.out.print(sig.getStr() + " "); |
|  | } |
|  |  |
|  | } |
|  | } |