Review:

上一节课讲了5种string基本操作：

1. **Removal**
   - 1.1. **remove some particular chars from a string.**
   - 1.2. **remove all leading/trailing/duplicated empty spaces from a string**
2. **De-duplication** **aaaabbbb_ccc ->ab_c**
3. **Substring → strstr**
4. **Reversal (swap)** **e.g. I love yahoo -> yahoo love I**
5. **Replacement** **e.g. replace empty space " " with "20%"**

4. String Reversal

Q4 (String Reversal)

(4.1) `apple → elppa`

(4.3) I love yahoo -> yahoo love I

sf

W1 W2 W3

I love yahoo

i j

```
Step1: swap every single word(two pointers) →    I evol  oohay
Step2: swap the whole sentence      →  yahoo love i
```

(4.4) abcd ef → ef abcd shift the whole string to the right-hand side by k = 2 positions.

w1 w2 w2 w1

```
for (int i = 0; i < k; i++) {
        for () {

        }
}
```

Discussion

1. The idea for “I LOVE YAHOO” can be combined to form more complex problem. e.g., if we have empty/leading/trailing spaces in the input. ____I_ _ _ LOVE_YAHOO
2. The idea can be **extended** to other problems as well.
    a. E.g., **String (array) shifting by X chars to the right:**
    “ab cdef” shift to the left by two steps → “cdef ab”
    Step 1: 两个区间各自reverse: ab cdef → ba fedc
    Step 2: 整个word reverse: ba fedc → cdef ab

5. Char Replacement

Example: “student” → “stuXXt” (pattern 1: den → pattern2: XX)

Solution:

input = s t u X X t t

s

f →

Step1: find the start index of the pattern1 to be replaced

Step2: replace it!

Time = O(n)

5. Char Replacement

Example: "student" → "stuXXt" (den → XX)

s1 s2

de stu XXX nt

What if we do not know the size relationship between s1 and s2?

Case1: if s1.length() >= s2.length()

Step1: find every single occurrence of s1 in the original string, and just replace s1 with s2, until we are done

Case2: if s1.length() < s2.length()

"studentdet" → "stuXXXntXXXt" (de → XXX)

How many extra spaces should we get???

Step1: count how many times s1 show up in the original string. for example, two times

Step2: 2 x (s2.size - s1.size)

studentdet_ _

34 78

Step3:

```
s t u X  X X n t X X X t
      f
      s
```

```
s: all letters to the right-hand of s are processed area
f: current index to tmove   ←
```

Advanced Topic

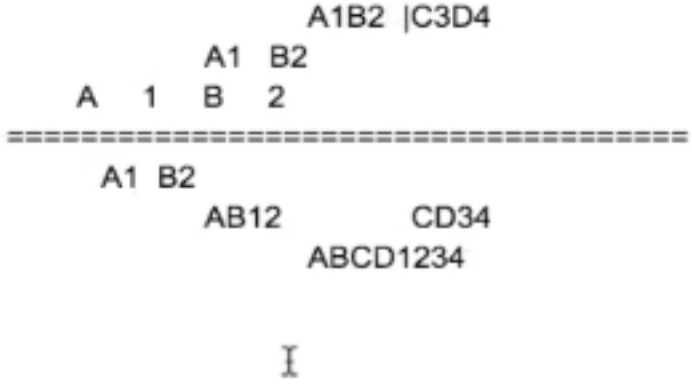
1. Shuffling e.g. ABCD1234 -> A1B2C3D4
2. Permutation (use DFS)
3. Decoding/encoding aaaabcc → a4b1c2 (Run Length Encoding)
4. Sliding windows using slow/fast pointers
    4.1. Longest substring that contains only unique chars **abcd**a

1. String Shuffling

1.1 First direction: "A1B2C3D4E5" → "ABCDE12345"

1. String Shuffling

1.1 First direction: **“A1B2C3D4E5” → “ABCDE12345”**



关键问题：注意 n/2 == 7 == 奇数情况

```
index =  0 1 2 3 4 5 6 | 7 8 9 0 1 2 3
         A B C D E F G | 1 2 3 4 5 6 7
               lm        m     rm
        C1[0-2] C2[3-6]  C3[7-9]   C4[10-13]
size = 14
mid = left + size/2 = 7
leftmid = left  + ¼ * size = 3
rightmid = left + ¼ * size  = 10
```

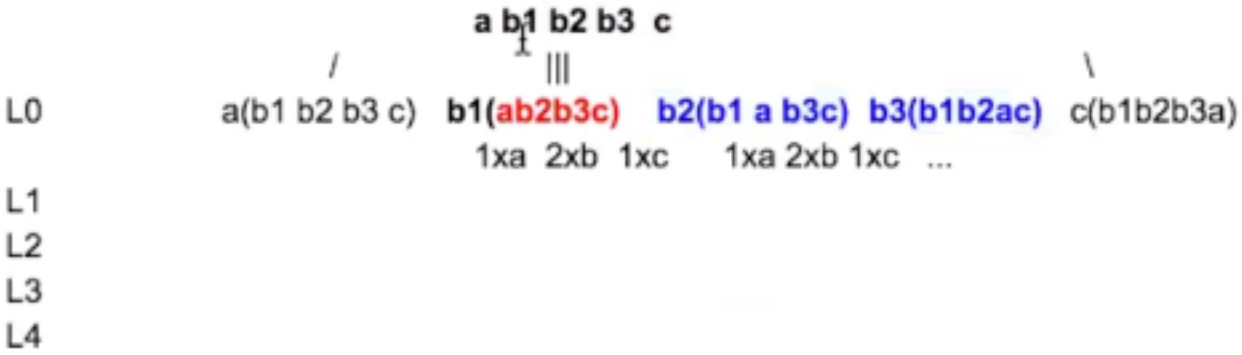
```
00 void convert(char a[], int left, int right) {
01     if (right - left <= 1)
02         return;
03     int size = right - left + 1;// how many elements in the section
04     int mid      = left + size/2;
05     int leftmid  = left + size/4;
06     int rightmid = left + size * 3/4;

07     Reverse(a, leftmid, mid-1);      // I love yahoo trick is here!!
08     Reverse(a, mid, rightmid-1);
09     Reverse(a, leftmid, rightmid-1);  // DE123 -> 123DE

10     convert(a, left, left + 2 * (leftmid-left) -1);      // chunk 1+3
11     convert(a, left+2*(leftmid-left),  right);      // chunk 2+4
12 }
```

Q2.2 duplicate letters in input string.

E.g. input string **"a b1 b2 b3 c"**



```
// level is the current level to try
public void permutation(char[] c, int index  == 0) {
      if (index == c.length) {  // base case
            printArray(c);
            return;
      }
      HashSet<Character> st = new HashSet<Character>();
      for (int i = index; i < c.length; i++) {
            if (!st.contains(c[i])) {
                  st.add(c[i]);
                  swap(c, index, i);
                  permutation(c, index + 1);
                  swap(c, index, i);
            }
      }
}
```

3. String En/Decoding "aaaabccaaaaa" → "a4b1c2a5" **Restriction**: in-place

www.yahoo.com
www.yahoo.com
www.bing.com
www.bing.com
www.bing.com
www.cnn.com
www.yahoo.com
…

bing:	**<1m, A>**
yahoo:0.8m	<0.8, B>
cnn.0.005m	<0.005, C>

Run Length Encoding
a a a a b b b c c c c c e

www.yahoo.com 2;www.bing.com 3; www.cnn.com 1;
B2;A3;C1;....

Similar to Char replacement (be careful about two cases: become longer / shorter).

Step1: deal with the cases where the adjacent occurrence of the letters >= 2, which will make the original string shorter. **In the meantime**, we need to record the number of times, the occurrence of the letter == 1

Step2: deal with the cases where the adjacent occurrence of the letters ==1 (from Right to Left, **this case is similar to that in Question 5.Case2**)

Step3: finalize the string by calling resize().

Key point of this problem: we use a hashMap to store the information between the slow and fast index **[window information]**

max_ = f - s

Solution:

Use two pointers slow(s) and fast(f)

slow: the begin index of the a solution

fast: the current index

In the meantime, we are using a **hash_table**

<**key** = letter, **value** = frequency> to record the frequency of all letters between [s, f]

When to move **fast** pointer?

- When there is no value > 1 in the hash_table

When to move **slow** pointer and when to stop?

- When there is a value > 1 in the hash_table, say letter 'D', and for each letter that the slow pointer is pointing to, we decrement its value before do slow++;
- When to stop: when <D, value = 1>, we stop slow

When to update the final solution?

- When do fast++, we check whether (fast - slow + 1) > global_max
- if so, update global_max.

index	0	1	2	3	4	5	6	7	
S[N]	B	**D1**	E	F	G	A	**D2**	E	

Q4.2 Find all anagrams of a substring S2 in a long string S1

```
string s2 = "aabc"                          size = k

        s1 = "zzzz cde bcaa b cyywwwww"   size = n
             aabb

             ____
             s  f
```

Method1: use two hashMaps

Step1: HashMap to store s2

```
<a,2>              <a, 2>
<b,1>              <b, 2>
<c,1>

============================
<a,2> size=1           <b,2> size=1
```

Step2: n times sliding, and each time we slide the window, we use O(k) time to check whether hashMap1 == heshMap2

Total = O(n * k)

Method2: use 1 hashMaps O(n)

Step1: 1 HashMap to store s2

<a,0>

<b,0>

<c,0> 0 = the number of types of letters to match

print out the substring between [s, f] because

Q4.3 Given a 0-1 array, you can flip at most k '0's to '1's. Please find the longest subarray that consists of all '1's.

0 → 1

0101010100111110000010101010

change k =4 zero to one, such that the contiguous 1s are longest

11111111110000011110000100111111111

Solution:

It is actually a sliding window problem. The window can contain **AT MOST k zeros** in the window.

Q4.3 Given a 0-1 array, you can flip at most k '0's to '1's. Please find the longest subarray that consists of all '1's.

0 → 1

k = 4

q 1010101001111100000101010 1

s

f→