Decompress String I (Medium)

Given a string in compressed form, decompress it to the original string. The adjacent repeated characters in the original string are compressed to have the character followed by the number of repeated occurrences. If the character does not have any adjacent repeated occurrences, it is not compressed.

Assumptions

- The string is not null
- The characters used in the original string are guaranteed to be ‘a ’- ‘z’
- There are no adjacent repeated characters with length > 9

Examples

- “acb2c4” →“ acbbcccc”

```
void decompressString(string& s){
    int d = 0;
    int sLen = s.length();
    // compute the length of new string
    for(int i = 0; i < sLen; i++){
        if(isdigit(s[i]))
            d += s[i] - '2';
    }
    int slow = sLen + d - 1, fast = sLen - 1;
    // modify the length of string if necessary
    s.resize(slow+1);
    while(fast >= 0){
        int num = 0;
        // when we encounter a digit, store the num
        if(isdigit(s[fast])) {
            num = s[fast] - '0';
            fast--;
            while(num > 0){
                s[slow--] = s[fast];
                num--;
            }
            // move to next element (next digit or character)
            fast--;
        }
        else
            s[slow--] = s[fast--];
    }
}
```

Time Complexity: O(n), since we traverse the string twice.

//Ok, great!

Encode Space (Easy)

In URL encoding, whenever we see an space " ", we need to replace it with "20%". Provide a method that performs this encoding for a given string.

Examples

- "google/q?flo wer market" → "google/q?flo20%wer20%market"

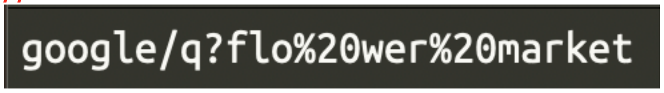
Corner Cases

- If the given string is null, we do not need to do anything.

```
void encodeSpace(string &s){
    int sLen = s.length();
    int d = 0;
    // compute length of new string
    for(int i = 0; i < sLen; i++){
        if(s[i] == ' ')
            d += 2;
    }
    int fast = sLen - 1, slow = sLen + d - 1;
    // modify the length of string if necessary
    s.resize(slow+1);
    while(fast >= 0){
        if(s[fast] == ' '){
            s[slow--] = '%';
            s[slow--] = '0';
            s[slow--] = '2';
            //s[slow--] = '%';
            fast--;
        }
        else
            s[slow--] = s[fast--];
    }
}
```

Time Complexity: O(n), since we traverse the string twice.

//Your result is



```
google/q?flo%20wer%20market
```

//So the order is not correct.

String Replace (Hard)

Given an original string input, and two strings S and T, replace all occurrences of S in input with T.

Assumptions

- input, S and T are not null, S is not empty string

Examples

- input = "appledogapple", S = "apple", T = "cat", input becomes "catdogcat"

- input = "dodododo", S = "dod", T = "a", input becomes “aoao"

```
// Rabin-Karp method to find all substring (HW9)
#define h 26
int substrSearch(string txt, string pat){
    vector<int> result;
    int patLen = pat.length();
    int txtLen = txt.length();
    int t = 0, p = 0, q = 1;

    // compute pow(h, patLen - 1)
    for(int i = 0; i < patLen - 1; i++)
        q = q * h;
    // compute the hash value of txt and pat window
    for(int i = 0; i < pat.length(); i++){
        t =  t * h + (txt[i] - 'a');
        p = p * h + (pat[i] - 'a');
    }
    // move the window from left to right and compute new hash value for txt
window
    for(int i = 0; i <= txtLen - patLen; i++){
        if(t == p)
            return i;
        t = (t - (txt[i] - 'a') * q) * h + (txt[i + patLen] - 'a');
    }
    return -1;
}

// if length of s >= length of t, traverse from left to right
void longToShort(string& input, string s, string t, int index){
    if(index == -1)
        return;
    int sLen = s.length();
    int tLen = t.length();
    int slow = index, fast = index + sLen;
    for(int i = 0; i < tLen; i++){
        input[slow++] = t[i];
    }
    while(fast < input.length()){
        input[slow++] = input[fast++];
    }
    input.resize(slow);
}

// if length of s < length of t, traverse from right to left
void shortToLong(string& input, string s, string t, int index){
    if(index == -1)
        return;
    int fast = input.length() - 1, slow = input.length() + (t.length() -
s.length()) - 1;
    input.resize(slow+1);
    int rIndex = index + s.length() - 1;
    while(fast > rIndex){
        input[slow--] = input[fast--];
    }
    for(int i = t.length() - 1; i >= 0; i--){
        input[slow--] = t[i];
    }
    return;
}

void stringReplace(string& input, string s, string t){
    int index = substrSearch(input, s);
    if(index == -1)
        return;
    int sLen = s.length();
    int tLen = t.length();
    while(index != -1) {
        if(sLen >= tLen){
            longToShort(input, s, t, index);
            index = substrSearch(input, s);
        }else{
            shortToLong(input, s, t, index);
            index = substrSearch(input, s);
        }
    }
    return;
}
```

Time Complexity: O(nm), m is the number of occurrences of S in input.

//Ok, great! The idea of this problem seems like automata and you can learn it by yourself.

Compress String (Hard)

Given a string, replace adjacent, repeated characters with the character followed by the number of repeated occurrences. If the character does not has any adjacent, repeated occurrences, it is not changed.

Assumptions

- The string is not null
- The characters used in the original string are guaranteed to be ‘a ’- ‘z’

Examples

- “abbcccdeee” →“ ab2c3de3”

```
void compressString(string& s){
    int slow = 0;
    for(int i = 0; i < s.length(); i++){
        int count = 1;
        while(i < s.length() - 1 && s[i] == s[i+1]){
            i++;
            count++;
        }
        s[slow++] = s[i];
        if(count > 1) {
            s[slow++] = char(count + '0');
        }
    }
    s.resize(slow);
    return;
}
```

Time Complexity: O(n), since we traverse the string once.

//What if the string is “aaaaaaaaaaaaaaaaaa”. The problem don’t guarantee the length <= 9. Please submit a new version next homework.