极客大学算法训练营
第二十课
字符串

覃超

Sophon Tech 创始人，前 Facebook 工程师

字符串基础知识

字符串

- Python:

  x = ‘abbc’
  x = “abbc”

- Java:

  String x = “abbc”;

- C++:

  string x(“abbc”);

string immutable：https://lemire.me/blog/2017/07/07/are-your-strings-immutable/

遍历字符串

- Python:

```
for ch in “abbc”:
  print(ch)
```

- Java:

```
String x = “abbc”;
for (int i = 0; i < x.size(); ++i) {
  char ch = x.charAt(i);
}
for ch in x.toCharArray() {
  System.out.println(ch);
}
```

- C++:

```
string x(“abbc”);
for (int i = 0; i < s1.length(); i++) {
  cout << x[i];
}
```

字符串比较

Java:

String x = “abb”;
String y = “abb”;

x == y —-> false

x.equals(y) —-> true
x.equalsIgnoreCase(y) —-> true

字符串相关算法

基础问题

1. https://leetcode-cn.com/problems/to-lower-case/
2. https://leetcode-cn.com/problems/length-of-last-word/
3. https://leetcode-cn.com/problems/jewels-and-stones/
4. https://leetcode-cn.com/problems/first-unique-character-in-a-string/
5. https://leetcode-cn.com/problems/string-to-integer-atoi/

Atoi

```java
public int myAtoi(String str) {
    int index = 0, sign = 1, total = 0;
    //1. Empty string
    if(str.length() == 0) return 0;

    //2. Remove Spaces
    while(str.charAt(index) == ' ' && index < str.length())
        index ++;

    //3. Handle signs
    if(str.charAt(index) == '+' || str.charAt(index) == '-'){
        sign = str.charAt(index) == '+' ? 1 : -1;
        index ++;
    }

    //4. Convert number and avoid overflow
    while(index < str.length()){
        int digit = str.charAt(index) - '0';
        if(digit < 0 || digit > 9) break;

        //check if total will be overflow after 10 times and add digit
        if(Integer.MAX_VALUE/10 < total ||
            Integer.MAX_VALUE/10 == total && Integer.MAX_VALUE %10 < digit)
            return sign == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;

        total = 10 * total + digit;
        index ++;
    }
    return total * sign;
}
```

Atoi

```
class Solution(object):

    def myAtoi(self, s):

        if len(s) == 0 : return 0
        ls = list(s.strip())

        sign = -1 if ls[0] == '-' else 1

        if ls[0] in ['-','+'] : del ls[0]

        ret, i = 0, 0

        while i < len(ls) and ls[i].isdigit() :
            ret = ret*10 + ord(ls[i]) - ord('0')
            i += 1

        return max(-2**31, min(sign * ret,2**31-1))
```

字符串操作问题

1. https://leetcode-cn.com/problems/longest-common-prefix/description/

2. https://leetcode-cn.com/problems/reverse-string
   https://leetcode-cn.com/problems/reverse-string-ii/

3. https://leetcode-cn.com/problems/reverse-words-in-a-string/
   https://leetcode-cn.com/problems/reverse-words-in-a-string-iii/

4. https://leetcode-cn.com/problems/reverse-only-letters/

Anagram异位词问题 - Homework

1. https://leetcode-cn.com/problems/valid-anagram/

2. https://leetcode-cn.com/problems/group-anagrams/

3. https://leetcode-cn.com/problems/find-all-anagrams-in-a-string/

Palindrome 回文串问题

1. https://leetcode-cn.com/problems/valid-palindrome/
2. https://leetcode-cn.com/problems/valid-palindrome-ii/
3. https://leetcode-cn.com/problems/longest-palindromic-substring/

高级字符串算法

最长子串、子序列

1. Longest common sequence（最长子序列）
   https://leetcode-cn.com/problems/longest-common-subsequence/

   dp[i][j] = dp[i-1][j-1] + 1 (if s1[i-1] == s2[j-1])
   else dp[i][j] = max(dp[i-1][j], dp[i][j-1])

2. Longest common substring (最长子串）
   dp[i][j] = dp[i-1][j-1] + 1 (if s1[i-1] == s2[j-1])
   else dp[i][j] = 0

3. Edit distance（编辑距离）
   https://leetcode-cn.com/problems/edit-distance/

https://leetcode-cn.com/problems/longest-palindromic-substring/

1. 暴力 O(n^3)

2. 中间向两边扩张法 O(n^2)

3. 动态规划

首先定义 P(i, j):

$$P(i, j) = \begin{cases} \text{true} & s[i, j] \text{ 是回文串} \\ \text{false} & s[i, j] \text{ 不是a回文串} \end{cases}$$

接下来

P(i, j) = (P(i+1, j-1) && S[i] == S[j])

字符串 + 递归 or DP

https://leetcode-cn.com/problems/regular-expression-matching/

重点：
https://leetcode-cn.com/problems/regular-expression-matching/solution/ji-yu-guan-fang-ti-jie-gen-xiang-xi-de-jiang-jie-b/

https://leetcode-cn.com/problems/wildcard-matching/

https://leetcode-cn.com/problems/distinct-subsequences/

1. 暴力递归

2. 动态规划

dp[i][j] 代表 T 前 i 字符串可以由 s 前 j 字符串组成最多个数。

所以动态方程：

当 S[j] == T[i], dp[i][j] = dp[i-1][j-1] + dp[i][j-1]

当 S[j] != T[i], dp[i][j] = dp[i][j-1]

字符串匹配算法

字符串匹配算法

1. 暴力法（brute force） - O(mn)

2. Rabin-Karp 算法

3. KMP 算法

- 课后了解：
  Boyer-Moore 算法：https://www.ruanyifeng.com/blog/2013/05/boyer-moore_string_search_algorithm.html
  Sunday 算法：https://blog.csdn.net/u012505432/article/details/52210975

暴力法

```
public static int forceSearch(String txt, String pat) {
    int M = txt.length();
    int N = pat.length();

    for (int i = 0; i <= M - N; i++) {
        int j;
        for (j = 0; j < N; j++) {
            if (txt.charAt(i + j) != pat.charAt(j))
                break;
        }
        if (j == N) {
            return i;
        }
        // 更加聪明?
        // 1. 预先判断— hash(txt.substring(i, M)) == hash(pat)
        // 2. KMP
    }
    return -1;
}
```

Rabin-Karp 算法

在朴素算法中，我们需要挨个比较所有字符，才知道目标字符串中是否包含子串。那么， 是否有别的方法可以用来判断目标字符串是否包含子串呢?

答案是肯定的，确实存在一种更快的方法。为了避免挨个字符对目标字符串和子串进行比较， 我们可以尝试一次性判断两者是否相等。因此，我们需要一个好的哈希函数（hash function）。 通过哈希函数，我们可以算出子串的哈希值，然后将它和目标字符串中的子串的哈希值进行比较。 这个新方法在速度上比暴力法有显著提升。

Rabin-Karp 算法

Rabin-Karp 算法的思想：

1. 假设子串的长度为 M (pat)，目标字符串的长度为 N (txt)
2. 计算子串的 hash 值 hash_pat
3. 计算目标字符串txt中每个长度为 M 的子串的 hash 值（共需要计算 N-M+1 次）
4. 比较 hash 值：如果 hash 值不同，字符串必然不匹配; 如果 hash 值相同，还需要使用朴素算法再次判断

Rabin-Karp

```
public final static int D = 256;
public final static int Q = 9997;

static int RabinKarpSerach(String txt, String pat) {
    int M = pat.length();
    int N = txt.length();
    int i, j;
    int patHash = 0, txtHash = 0;

    for (i = 0; i < M; i++) {
        patHash = (D * patHash + pat.charAt(i)) % Q;
        txtHash = (D * txtHash + txt.charAt(i)) % Q;
    }

    int highestPow = 1;  // pow(256, M-1)
    for (i = 0; i < M - 1; i++)
        highestPow = (highestPow * D) % Q;

    for (i = 0; i <= N - M; i++) { // 枚举起点
        if (patHash == txtHash) {
            for (j = 0; j < M; j++) {
                if (txt.charAt(i + j) != pat.charAt(j))
                    break;
            }
            if (j == M)
                return i;
        }
        if (i < N - M) {
            txtHash = (D * (txtHash - txt.charAt(i) * highestPow) + txt.charAt(i + M)) % Q;
            if (txtHash < 0)
                txtHash += Q;
        }
    }

    return -1;
}
```

KMP 算法

KMP算法（Knuth-Morris-Pratt）的思想就是，当子串与目标字符串不匹配时，其实你已经知道了前面已经匹配成功那 一部分的字符（包括子串与目标字符串）。以阮一峰的文章为例，当空格与 D 不匹配时，你其实 知道前面六个字符是“ABCDAB”。KMP 算法的想法是，设法利用这个已知信息，不要把“搜索位置” 移回已经比较过的位置，继续把它向后移，这样就提高了效率。

https://www.bilibili.com/video/av11866460?from=search&seid=17425875345653862171

http://www.ruanyifeng.com/blog/2013/05/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm.html

Homework

1. https://leetcode-cn.com/problems/first-unique-character-in-a-string/
2. https://leetcode-cn.com/problems/string-to-integer-atoi/
3. https://leetcode-cn.com/problems/reverse-string-ii/
   https://leetcode-cn.com/problems/reverse-words-in-a-string/
   https://leetcode-cn.com/problems/reverse-words-in-a-string-iii/
4. https://leetcode-cn.com/problems/reverse-only-letters/
5. https://leetcode-cn.com/problems/find-all-anagrams-in-a-string/
6. https://leetcode-cn.com/problems/longest-palindromic-substring/
   https://leetcode-cn.com/problems/isomorphic-strings/
   https://leetcode-cn.com/problems/valid-palindrome-ii/
7. https://leetcode-cn.com/problems/wildcard-matching
8. https://leetcode-cn.com/problems/longest-valid-parentheses
9. https://leetcode-cn.com/problems/distinct-subsequences/

THANKS!