8.字符串算法（差BM、AC自动机）

记在最前

- 处理""

BF算法

- BF/RK/BM图解
- 暴力匹配

```
void BF(string T, string P){
    if(P.size() == 0) return 0;
    int n = T.size();
    int m = P.size();
    int l1 = 0;
    int l2 = 0;
    while(l1 < n){
        if(T[l1] == P[l2]){
            l1++;
            l2++;
            if(l2 == m) return l1 - m;
        }
        else if(T[l1] != P[l2]){
            l1 = l1 - l2 + 1;
            l2 = 0;
        }
    }
    return -1;
}
```

RK算法

- 核心：计算哈希值，冲突之后再验证

```
 bool isMatch(string T, string P, int i)
{
    int m = P.size();
    int it, ip;
```

```
    for(it=i, ip=0; it != m && ip != m; it++, ip++)
        if(T[it] != P[ip]) return false;
    return true;
}

int strStr(string T, string P) {
    int n = T.size();
    int m = P.size();
    if(m > n) return -1;
    if(P.size() == 0) return 0;
    //最高位
    int high = 1;
    int hashT = 0;
    int hashP = 0;
    //模数
    //
    int mod =  661;
    //进制
    int radix = 26;
    for(int i = 0; i < m; i++){
        hashT = (hashT * radix + T[i] - 'a') % mod;
        hashP = (hashP * radix + P[i] - 'a') % mod;
    }
    for(int i = 0; i < m - 1; i++){
        high = (high * radix) % mod;
    }

    for(int i = 0; i <= n - m; i++){
        if(hashT == hashP){
            if(isMatch(T,P,i)) return i;
        }
        if(i < n - m){
            hashT = (radix * (hashT - (T[i] - 'a') * high)  + (T[i+m] - 'a')) %
            if(hashT < 0){
                hashT += mod;
            }
        }
    }
    return -1;
}
```

题目

- 28. 实现 strStr()
- 187. 重复的DNA序列（时间复杂度是多少？因为比哈希表还要慢）

- 接187题，将字符串表示为一个 P 进制的数，P 是一个经验值，P = 131 / 13331/ 131313 不会出现哈希值冲突（概率很小，所以不考虑冲突）？
- RK题解：为什么用int，负数怎么办
- 1044. 最长重复子串(P的问题，26会发生冲突，为了避免冲突用常用质数)

BM算法

- 从后向前
- 好后缀，坏字符
- link1
- link2
- link3

KMP算法

kmp算法（克努特-莫里斯-普拉特操作）

- kmp整体思路

问题：在文本串中找子串

例子：在aabaabaaf中找aabaaf

遍历子串得到next数组（最大相等前后缀长度）

前缀不包括位置i，后缀不包括首字母

文本串中寻找子串存在的位置

根据next数组更新位置

next数组情况分析：

0	1	0	1	2	0	最大相等前后缀长度	j = next[j - 1]
-1	0	1	0	1	2	右移一位	j = next[j]
-1	0	-1	0	1	-1	每个位置-1	j = next[j] + 1

```
1
2 void getNext(vector<int>& next, string x){
3         //i:后缀尾位置，j:前缀尾位置（也是长度）
4         int j = 0;
5         //初始化next数组
6         next.push_back(0);
7
```

```
        for(int i = 1; i < x.length(); i++){
            while(j > 0 && x[i] != x[j] ) j = next[j - 1]; //处理不同情况
            if(x[i] == x[j]) j++; //处理相同情况
            next.push_back(j); //更新next数组
        }
    }
    int strStr(string text, string son) {
        //i:文本串位置，j:子串位置
        vector<int> next;
        getNext(next, son);
        int j = 0;
        for(int i = 0; i < text.size(); i++){
            while(j > 0 && text[i] != sonson[j]) j = next[j - 1]; //处理不同情况
            if(text[i] ==son[j]) j++; //处理相同情况
            ... //根据题目要求操作
        }
        return -1;
    }
```

Sunday算法

- Sunday算法讲解
- 核心：关注的是主串中参加匹配的最末位字符的下一位字符

```
int Sunday(string T, string P) {
    if(P.length() == 0) return 0;
    int n = T.length();
    int m = P.length();
    vector<int> shift(256,m+1); //存字符不存数字，初始化移动量为m+1

    //记录每个字符出现的最后的下标
    for(int i = 0; i < m; i++){
        shift[T[i]] = m - i;
    }

    int s = 0;
    int j;
    while(s <= n - m){
        j = 0;
        while(T[s+j] == P[j]){
            j++;
            if(j >= m) return s;
        }
        s += shift[T[s + m]];
```

```
        //核心：关注的是主串中参加匹配的最末位字符的下一位字符
        //越界但是256防止了访问出错
    }
    return -1;
}
```

题目

- 28. 实现 strStr()

马拉车算法

- 回文串
- 知乎Pecco
- 感觉这个图解的更好

```
//预处理
string preProcess(string s){
    int n = s.length();
    if (n == 0) return "^$";
    string ret = "^";
    for (int i = 0; i < n; i++)
    {
        ret += "#" ;
        ret += s[i];
    }
    ret += "#$";
    return ret;
}
//马拉车算法
string lManacher(string s) {
    s = preProcess(s);
    int max = 0;
    string ans = "";
    int n = s.length();
    vector<int> d(n, 0);
    int l = 0, r =0;//当前回文子串的左右边界
    for(int i = 1; i < n - 1; i++){
        int i_mirror = l + r - i; //对称点
        if(i > r){
            while(i - d[i] >= 0 && i + d[i] < n && s[i - d[i]] == s[i + d[i]])
                d[i]++;
            l = i - d[i] + 1, r = i + d[i] - 1;
        }//已经超出右边界
```

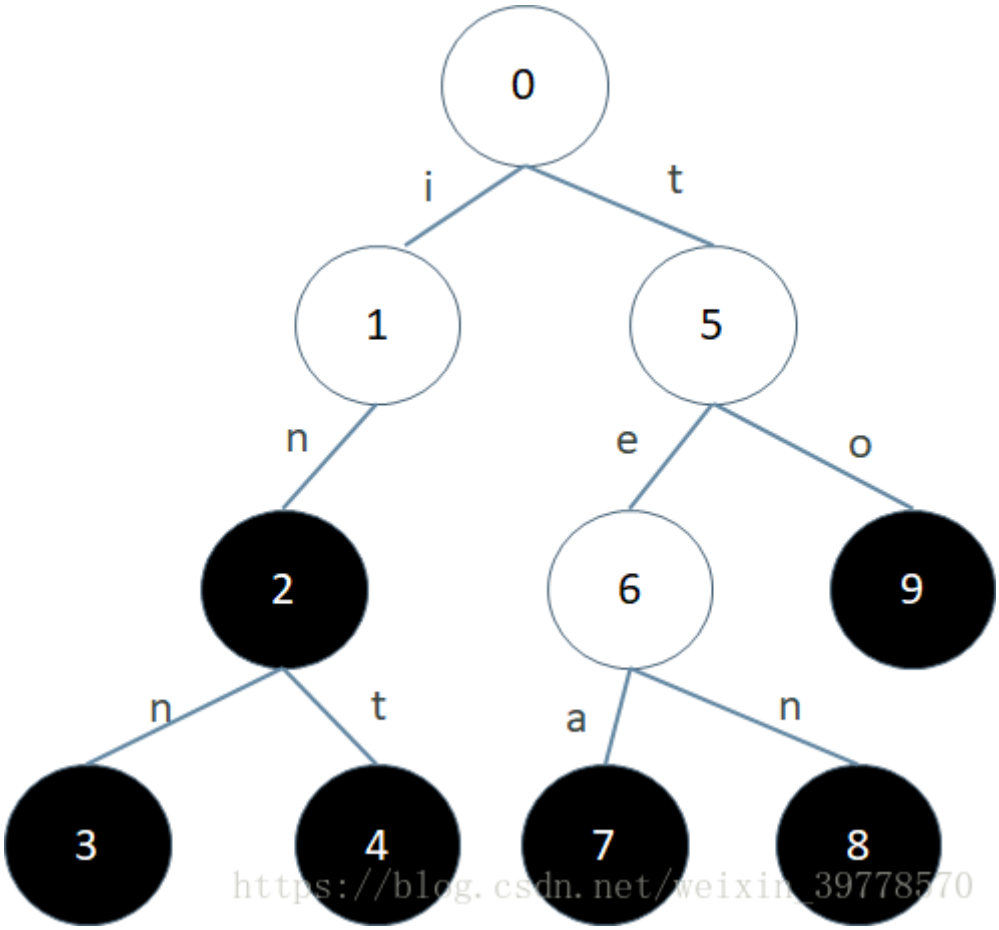
```
        else if(i_mirror - d[i_mirror] + 1 > l){
            d[i] = d[i_mirror];
        }//对称点的子串还在l-r覆盖范围内
        else{
            d[i] = i_mirror - l + 1;
            while (i - d[i] >= 0 && i + d[i] < n && s[i - d[i]] == s[i + d[i]])
                d[i]++;
            l = i - d[i] + 1, r = i + d[i] - 1;
        }//对称点的子串超出范围，只能取部分，后面的再一个一个验证

        if(r - l + 1> max) {
            max = r - l + 1;
            ans = s.substr(l, r - l + 1);
        }
    }

}
```

题目

- 5. 最长回文子串

Trie树

- 快速存储和查找字符串集合的数据结构
- 按照每个字符串顺序进行建树，标记字符串结尾的地方/标记出现次数



字符串集合是{in, inn, int, tea, ten, to}

链表实现

```
1
```

二维数组模拟

```
1  //每个点存的是数字
2
3
4  int son[N][26], cnt[N], idx;
5  // 0号点既是根节点，又是空节点
6  // son[][]存储树中每个节点的子节点
7  // cnt[]存储以每个节点结尾的单词数量
8
9  // 插入
10 void init(){
11     memset(cnt, 0, sizeof(cnt));
12     memset(son, 0, sizeof(son));
13 }
```

```
14 void insert(char *str)
15 {
16     int p = 0;
17     for (int i = 0; str[i]; i ++ )
18     {
19         int u = str[i] - 'a';
20         if (!son[p][u]) son[p][u] = ++ idx;
21         p = son[p][u];
22     }
23     cnt[p] ++ ;
24 }
25
26 // 查询字符串出现的次数
27 int query(char *str)
28 {
29     int p = 0;
30     for (int i = 0; str[i]; i ++ )
31     {
32         int u = str[i] - 'a';
33         if (!son[p][u]) return 0;
34         p = son[p][u];
35     }
36     return cnt[p];
37 }
```

链表实现

题目

- 208. 实现 Trie (前缀树)

AC自动机