字符串 一

石丰民

2016年7月13日

字符串匹配问题

KMP算法

最长回文子串问题

Manacher 算法

字典树

字符串匹配问题

- 对于给定的一个文本串T和一个模式串P,求出T中所有的P出现的位置
- T: ATTTATGCGGGGATGCCCCATAT
- P: ATGC
- P在T中出现了2次
- 暴力算法需要$O(nm)$的时间
  - 在T中枚举匹配起点
  - 对于每个起点最坏需要比较$O(m)$次
- 还有更好的方法!

字符串匹配问题

KMP算法

最长回文子串问题

Manacher 算法

字典树

KMP算法

- 能在$\Theta(n)$的时间内求匹配，预处理需要$\Theta(m)$的时间
  - KMP利用模式串P自身的特点来跳过一些不必要的比较，而暴力算法不考虑P自身的特点
- 在暴力算法中：
  - T: 00100001000000
  - P: 00000

  在红色字符处失配了（蓝色表示已匹配的部分）

  在将P右移5次后才成功匹配
  - T: 00100001000000
  - P: 00000

    ⋮

  - T: 00100001000000
  - P: 00000
- 对于只包含0的模式串，有一种优化方法：
  - 在文本串的第i位失配时，文本串指针变为$i+1$，模式串指针变为1，成功匹配后，文本串指针右移一位，模式串指针仍为m，可以用$O(n)$的时间完成匹配
  - KMP算法将这种做法一般化了

next 数组

- KMP算法使用一个辅助数组*next*数组来表示失配后应将失配位与模式串的哪一位比较才可能导致匹配
- $next[i]$ 是小于i的最大的整数j，且满足$P_1P_2\cdots P_{j-1}x$是文本串中最后比较的j个字符，且$P_i \neq P_j, x \neq P_i$，如果不存在这样的j 则令$next[i] = 0$

i	1	2	3	4	5	6	7	8	9	10
P_i	a	b	c	a	b	c	a	c	a	b
$next[i]$	0	1	1	0	1	1	0	5	0	1

- $next[i] = 0$代表要将P右移并越过失配位

$next$数组的计算

- 引入一个辅助数组$f[1 \cdots m]$

  $f[1] = 0$

  $f[i]$是小于i的最大的整数j，且满足$P_1 \cdots P_{j-1} = P_{i-j+1} \cdots P_{i-1}$

- 在$P_i = P_{f[i]}$时，有$f[i+1] = f[i] + 1$

- 在$P_i \neq P_{f[i]}$时，$f[i+1]$可以由$f[i], next[1 \cdots i]$计算得到：

$$
\begin{aligned}
&t = f[i] \\
&\textbf{while } t > 0 \textbf{ and } P_i \neq P_t \\
&\qquad t = next[t] \\
&f[i+1] = t + 1
\end{aligned}
$$

$next$数组的计算

- $next$ 的递推式：

$$next[i] = \begin{cases} f[i] & P_i \neq P_{f[i]} \\ next[f[i]] & P_i = P_{f[i]} \end{cases}$$

- $f[i+1]$由$next[1 \cdots i]$和$f[i]$递推得到，$next[i]$由$next[1 \cdots i-1]$和$f[i]$递推得到，因此在实际程序中不需要开一个f数组，使用一个变量记录最新的$f[i]$即可

计算*next*数组的代码

```
1  void getnext(){
2      next[0] = -1;
3      int i = 0,j = -1;
4      while(i<m){
5          while(j>=0 && p[i]!=p[j]){
6              j = next[j];
7          }
8          j+=1;i+=1;
9          if(p[i]==p[j]){
10             next[i] = next[j];
11         }else{
12             next[i] = j;
13         }
14     }
15 }
```

匹配代码

```
1  void kmp(){
2      int i = 0, j = 0;
3      while(i<n){
4          while(j>=0 && t[i]!=p[j]){
5              j = next[j];
6          }
7          i+=1;j+=1;
8          if(j==m){
9              printf("%d\n",i-m);
10             j = next[m];
11         }
12     }
13 }
```

另一种*next*求法

这种方法不考虑$P_i = P_{f[i]}$时会导致失配的情况，复杂度仍然是$\Theta(n)$，虽然理论上前一种写法更优，但由于多了一些判断语句，实际效率有时这一种更高，这种写法还可以用来求字符串的最小循环节

```
1  void getnext(){
2      next[0] = -1;
3      int i = 0,j = -1;
4      while(i<m){
5          while(j>=0 && p[i]!=p[j]){
6              j = next[j];
7          }
8          j+=1;i+=1;
9          next[i] = j;
10     }
11 }
```

KMP例题: HDOJ 1358 Period

- 题意：给定一个长度为$n(2 \leq n \leq 10^6)$的字符串S，对于S的每个前缀，求一个满足$K > 1$的最大整数K，使得该前缀由一个字符串重复K次组成
- 解法：问题可转化为求每个前缀的最短循环节
- KMP中的$next$数组可以求字符串的最短循环节，长度为i的前缀的最短循环节长 $= i - next[i]$

字符串匹配问题

KMP算法

最长回文子串问题

Manacher 算法

字典树

最长回文子串问题

- 回文串：
  - 正着读和反着读一样的串
  - level, noon 都是回文串
- 最长回文子串：
  - 对于一个给定的字符串，找出它的最长的子串，并且该子串是回文串
  - ababac的最长回文子串是ababa，长度为5
  - 这样的子串可能有很多个
- 暴力解法：
  - 枚举对称中心，并不断向两边延伸子串，直到两端的字符不同
  - 枚举时要分别考虑奇数长度和偶数长度的子串
  - 复杂度为$O(n^2)$
- 也有更好的方法！

字符串匹配问题

KMP算法

最长回文子串问题

Manacher 算法

字典树

Manacher 算法

- Manacher算法能用$O(n)$的时间求最长回文子串
- 预处理
  - 假设给出的字符串为T'，长度为n，要求它的最长回文子串
  - 先在T'的两端和每两个字符间插入一个不会在T'中出现的字符得到T，如

$$ABCBAB \longrightarrow @A@B@C@B@A@B@$$

  - 在插入字符后，原串中奇数长度的回文子串现在是以字母为对称中心的奇数长度的回文子串，原串中偶数长度的回文子串现在是以@为对称中心的奇数长度的回文子串，原串中不是回文子串的子串在插入字符后仍然不是回文串

Manacher 算法

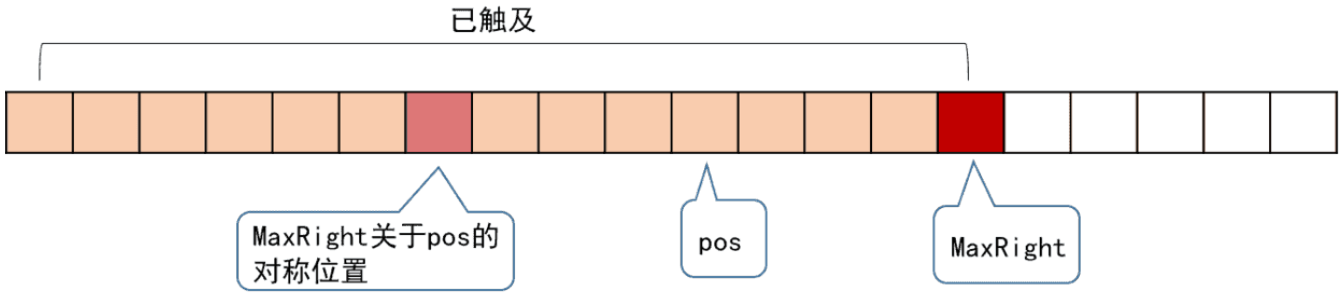
- 定义回文半径数组
  - 定义一个数组 $R[0 \cdots 2n]$ 代表以T中以第T_i为对称中心的最长回文子串的回文半径，即一端到T_i间的字符数，该回文子串的长度则为 $2R[i]-1$，在原串T中的长度为$R[i]-1$

i	0	1	2	3	4	5	6	7	8	9	10	11	12
T_i	@	A	@	B	@	C	@	B	@	A	@	B	@
$R[i]$	1	2	1	2	1	6	1	2	1	4	1	2	1

  - 通过数组R可以确定每一个最长回文子串的长度和位置，现在问题变成了如何高效的计算R

计算 R

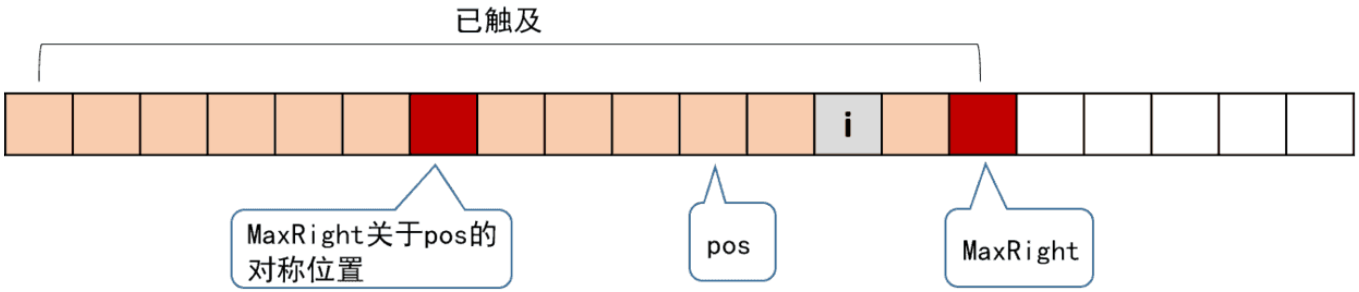
- 引入一个辅助变量*MaxRight*表示当前已确定出的所有回文子串能触及的最右端的字符，pos为其对称中心的位置



- 接下来需要分情况讨论

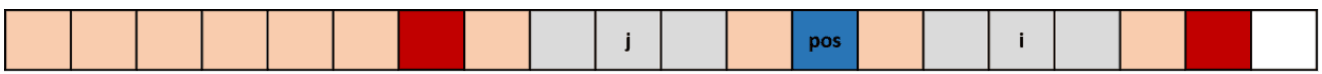
计算 R

情况 1 i在$MaxRight$的左边



利用回文串的对称性，我们可以找到i关于pos的对称位置j，$R[j]$是之前已经求出的，此后根据$R[j]$的大小又要分两种情况

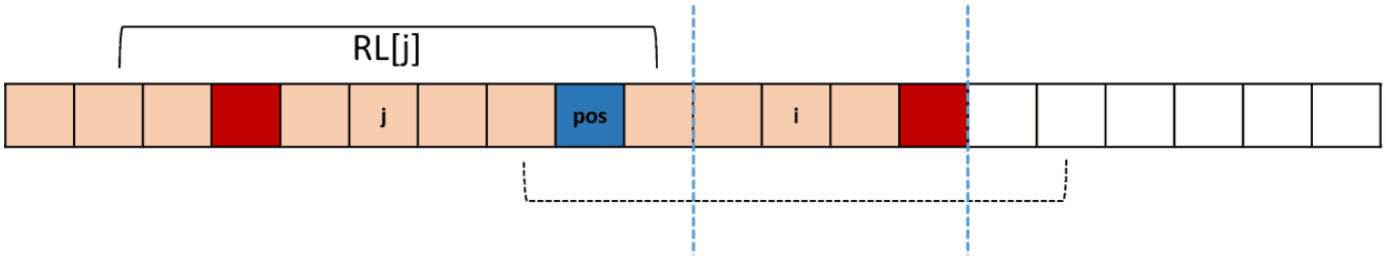
情况 a $R[j] + i - 1 \leq MaxRight$



此时已经确定了以i为中心的回文串的一部分，令$R[i] = R[j]$并继续向两边扩展回文子串

计算 R

情况 b $R[j] + i - 1 > MaxRight$



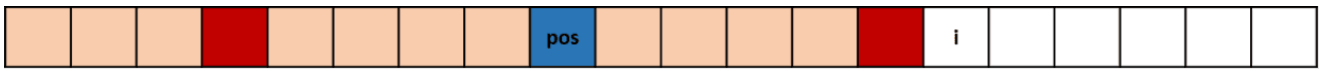
此时只能确定蓝线中间的部分是回文的(左边的蓝线应该再往左一格)，

令$R[i] = MaxRight - i + 1$并继续向两边扩展回文子串

在扩展的同时要更新*MaxRight*和*pos*

计算 R

情况 2 i在pos右边



此时无法通过已有的回文子串扩展，只能令$R[i] = 1$再尝试扩展

核心代码

常数优化：预处理时在串首再添加一个不同的特殊字符，之后扩展回文串时就不需要判断是否越界了

```
1  void Manacher(char *t){
2      int pos=0,MaxRight=0,l=strlen(t);
3      for(int i=1;i<l;i++){
4          if(MaxRight>i)
5              R[i]=min(R[2*pos-i],MaxRight-i+1);
6          else
7              R[i]=1;
8          while(t[i+R[i]]==t[i-R[i]]) R[i]++;
9          if(R[i]+i-1>MaxRight){
10             MaxRight=R[i]+i-1;
11             pos=i;
12         }
13     }
14 }
```

字符串匹配问题

KMP算法

最长回文子串问题

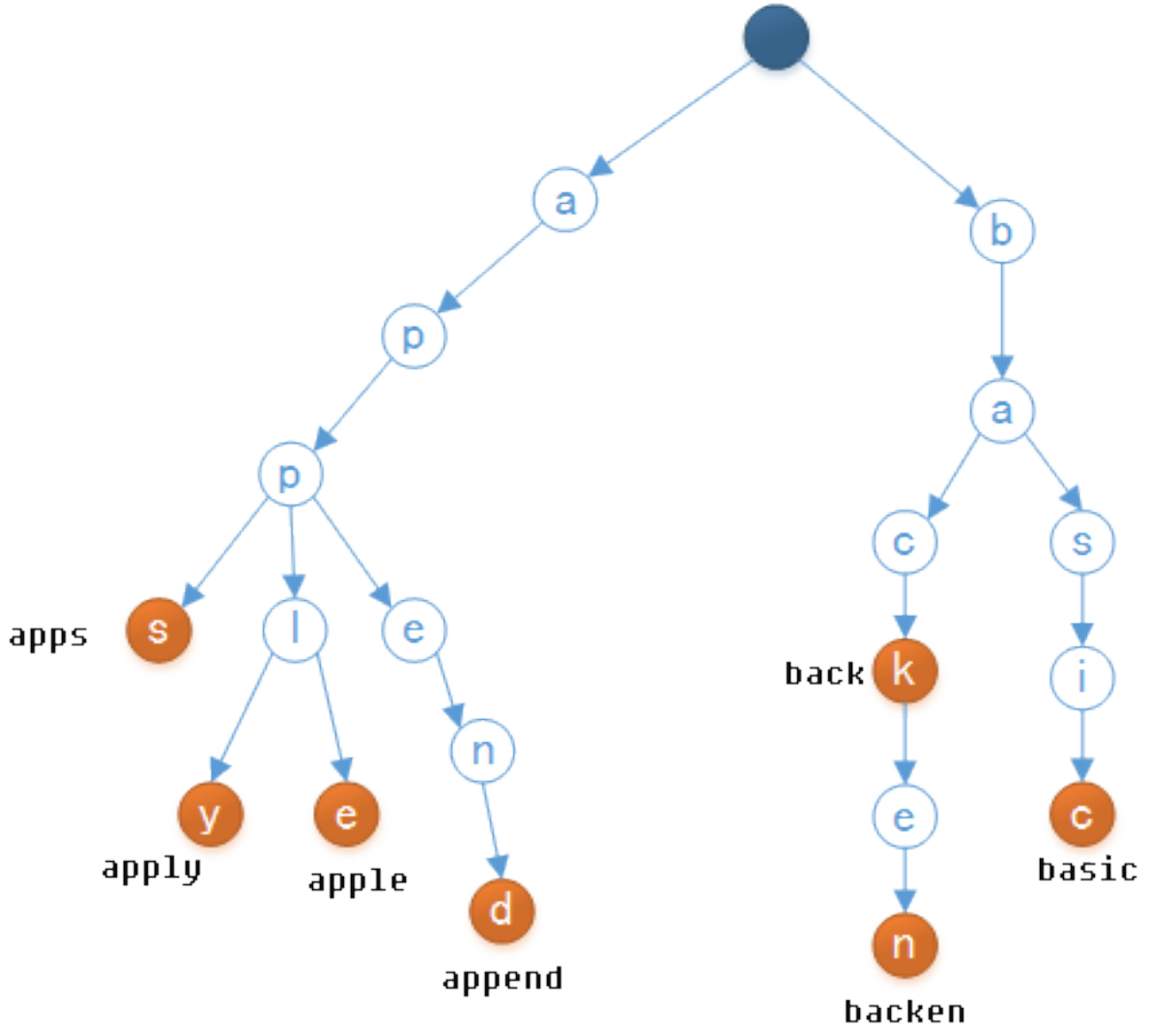
Manacher 算法

字典树

字典树

- 字典树(Trie)是一种有根树，可以用来存储字符串集合，它在每条边上存储一个字符
- 从根节点到某个节点的路径上的字符构成了一个字符串，这个字符串是构成字典树的单词的前缀(也可能是单词本身)
- 节点可以存储一些信息，比如该节点是否是单词节点，单词出现了几次等等
- 两个单词的最长公共前缀(LCP)在字典树中由从根节点到两个对应单词节点的最近公共祖先(LCA)的路径上的字符组成

字典树


a
b
p
a
p
c
s
apps
s
l
e
back
k
i
n
e
c
y
e
basic
apply
apple
d
n
append
backen

节点定义

```
1  const int maxnode = 100001; //最大节点数
2  const int sigma   = 26;     //字符集大小
3  int ch[maxnode][sigma];     //存储节点i同过边j达到的节点的下标
4  int v[maxnode];             //存储节点i上附加的信息
5  int sz;                     //树中结点个数
6
```

插入操作

```
1  void add(char* s,int e){
2      int l=strlen(s),u=0;
3      for(int i=0;i<l;i++){
4          int c = s[i]-'a';
5          if(!ch[u][c]){
6              memset(ch[sz],0,sizeof ch[sz]);
7              ch[u][c]=sz++;
8          }
9          u=ch[u][c];
10     }
11     v[u]+=e;
12 }
```

查询操作

```
int query(char *s){
    int l=strlen(s),u=0;
    for(int i=0;i<l;i++){
        int c = s[i]-'a';
        if(!ch[u][c]) return 0;
        u = ch[u][c];
    }
    return v[u];
}
```