Manacher and AC-Automaton

雷宇辰

CUGACM2018

2018 年 10 月 27 日

Section 1

Manacher (Manacher 1975)

Manacher 算法要解决的是寻找最长回文子串的问题。一种比较暴力的方式是枚举每个字符或间隔，然后像两边扩展以求出以当前位置为中心的最长回文串，其同时也是整个字符串的一个回文子串。很明显这样做的时间复杂度为 $O(n^2)$

为了避免枚举字符或间隔带来的问题，我们可以在每个间隔（包括字符串）的首尾增加一个字符串中不可能出现的字符，比如 #，* 一类的。这样我们便能够通过枚举新字符串中的每个字符来代替原来的枚举字符与空格了。

我们假设 str 是原数组。

```
int len = strlen(buf), n = 0;
str[n++] = '!', str[n++] = '#'
for (int i = 0; i < len; i++)
    str[n++] = buf[i], str[n++] = '#';
str[n++] = '#', str[n++] = '?'
```

这样我们就可以将一个长度为 n 的字符串扩展成一个长度为 $2n + 3$ 的字符串。而且我们在开头和结尾各加上了一个不同的字符，这样就可以不用判断边界条件。

我们把一个回文串中最左或最右位置的字符与其对称轴的距离称为回文半径。我们定义一个回文半径数组 R，用 R[i] 表示以第 i 个字符为对称轴的回文串的回文半径。比如:

```
i    :    0 1 2 3 4 5 6 7 8
char:    # a # b # a #
R    :    1 2 1 4 1 2 1
char:    # a # b # b # a #
R    :    1 2 1 2 5 2 1 2 1
```

于是我们发现，R[i] 刚好就是原来那个字符串中以 i 为中心的最长回文子串的长度 $+1$。现在我们已经将问题简化到如何高效的求出 R 了。我们引入一个变量 mx 来表示当前所有回文子串的最右边是什么，顺便记录下这个位置所对应的字符串中点的位置 p，然后开始分类讨论。

具体的做法

情况 1：i 在 mx 的左边

首先我们可以找到 i 关于 p 所对应的位置 j，由于原来的计算，我们已经求出了 R[j]。当 R[j] 不是那么长的时候，即 i + R[j] <= mx 的时候，R[i] 至少跟 R[j] 相同；而当 i + R[j] > mx 的时候，由于我们没有计算 mx 以右的信息，所以 R[i] 至少跟 mx - i 相同。

情况 2：i 不在 mx 的左边

这时候也是由于我们没有 mx 以右的信息，所以 R[i] = 1。

接下来我们就可以以当前的 R[i] 为基础，向两边扩展回文串。最后更新 mx 与 p。整个过程就完成了。

示例代码

```
for (int i = 1; i < n; i++)
{
    R[i] = mx > i ? min(R[2 * p - i], mx - i) : 1;
    while (str[i + R[i] == str[i - R[i]]]) R[i]++;
    if (R[i] + i > mx) mx = i + R[p = i];
}
```

可以看到它非常的简短。

Section 2

Aho-Corasick Automaton (Aho and Corasick 1975)

前言

上次讨论的内容应该是 Trie 和 KMP，而这两个知识点正是 AC 自动机的基础。
我们知道 `next[]` 是 KMP 的核心部分，表示的是在失配时应该跳转到模式串的哪个位置进行下一次匹配。
AC 自动机要解决的是多模式串匹配问题，模仿 KMP 的思想，AC 自动机是这样做的：将各个模式串都插入到同一个 Trie 中，而 `next[]` 在 AC 自动机中的对应物便是 `fail` 指针，表示当失配时应该跳转到哪个节点继续匹配。

具体做法

首先我们知道 fail 指针的是失配时的跳转位置，于是我们需要将一个节点的 fail 设置成当前路径在这棵 Trie 中的最长后缀的节点。为了达成这一目的，我们可以使用 bfs。其中我加入了一个 `virt` 节点，它是 root 的 fail，而且它的所有儿子都是 root，这样可以避免特判，我讨厌特判。
从下一页的代码中我们可以看到，我们对于每一个子节点的 fail 的计算遵循如下原则：

1. 先从当前节点沿着 fail 向上跳，直到某一个节点的有当前的子节点。
2.
   - 若当前子节点不为空，即将这个节点的 fail 设为上一步中找到的节点，并将其入队列。
   - 若当前子节点为空，即将这个子节点设为上一步中找到的节点（不然走到 nullptr 又要特判了）。
3. 重复过程直到队列空。

这样当你匹配的时候，只需要对着待匹配串不停的进行 trans 转移，而不用考虑什么时候显式的使用 fail 指针。

示例实现

```
struct node
{
    node *trans[26], *fail;
    int cnt;
} nodes[N], virt;
void buildFail()
{
    int h = 0, t = 0;
    root->fail = &virt;
    que[t++] = root;
    while (h < t)
    {
        node *cur = que[h++];
        for (int i = 0; i < 26; i++)
        {
            node *f = cur->fail;
            while (f->trans[i] == 0) f = f->fail;
            f = f->trans[i];
            if (cur->trans[i])
                (que[t++] = cur->trans[i])->fail = f;
            else
                cur->trans[i] = f;
        }
    }
}
```

Section 3

Appendix

Examples

Manacher

POJ3974 - Palindrome

kuangbin 专题十六

Aho-Corasick Automaton

HDU2222 - Keywords Search

kuangbin 专题十七

References

Aho, Alfred V, and Margaret J Corasick. 1975. “Efficient String Matching: An Aid to Bibliographic Search.” *Communications of the ACM* 18 (6). ACM: 333–40.

Manacher, Glenn. 1975. “A New Linear-Time‘On-Line’Algorithm for Finding the Smallest Initial Palindrome of a String.” *Journal of the ACM (JACM)* 22 (3). ACM: 346–51.