字符串进阶：AC 自动机

Ebola

Institute of Mathematics,
Zhejiang University.

Jan, 2024

1. 基础回顾
2. AC 自动机

1. 基础回顾
2. AC 自动机

字典树（Trie）

字典树把所有的字符串存储在一棵树中，可以方便地查询所有的前缀。

我们来回顾一下字典树模板题。

字典树（Trie）

插入操作

```
int mapping(char c){
    if(c>='A' && c<='Z') return c-'A';
    else return c-'a'+26;
}
void insert(char s[]){
    int n = strlen(s+1);
    int cur = 0;
    sz[0]++;
    for(int i = 1; i <= n; i++){
        int j = mapping(s[i]);
        if(ch[cur][j]==0){
            ch[cur][j] = tot;
            tot++;
        }
        cur = ch[cur][j];
        sz[cur]++;
    }
}
```

字典树（Trie）

查询操作

```
int query(char s[]){
    int cur = 0;
    int n = strlen(s+1);
    for(int i = 1; i <= n; i++){
        int j = mapping(s[i]);
        if(ch[cur][j]==0) return 0;
        cur = ch[cur][j];
    }
    return sz[cur];
}
```

最大异或和问题

最大异或和问题是 Trie 的一个经典应用。

给定 $n\ (\leq 10^5)$ 个数，所有数均不超过 $2^{31}-1$. 从中选两个数，使它们异或起来最大。

最大异或和问题

把所有的数都转化为 31 位二进制数，高位不足则补零，然后把二进制数当成字符串插入进 Trie 中。

最大异或和问题

把所有的数都转化为 31 位二进制数，高位不足则补零，然后把二进制数当成字符串插入进 Trie 中。

现在，我们枚举 $x = a_i\ (i = 1, ..., n)$，来找一个数 a_j，使它和 x 异或起来最大。

最大异或和问题

把所有的数都转化为 31 位二进制数，高位不足则补零，然后把二进制数当成字符串插入进 Trie 中。

现在，我们枚举 $x = a_i\ (i = 1, ..., n)$，来找一个数 a_j，使它和 x 异或起来最大。

从高位到低位贪心，尽可能让异或和的高位为 1。例如：如果 x 最高位是 0，那么我们希望选出的 a_j 最高位是 1，这样异或起来最高位才会是 1，因此我们第一步从 Trie 的根节点往 1 的方向走。

最大异或和问题

总之，如果 x 的第 k 位是 x_k，那么这一步就尽量往 x_k xor 1 方向走，除非 Trie 不存在对应的分支，此时不得不往 x_k 走。最后代码像这样：

```
int query(int x){
    int cur = 0;
    const int n = 31;
    for(int i = 1; i <= n; i++){
        int j = (x >> (31-i)) & 1;
        if(ch[cur][j^1]==0) cur = ch[cur][j];
        else cur = ch[cur][j^1];
    }
    return val[cur];
}
```

第 k 大异或和问题

如果想对于给定的 x，找到一个 a_i，使 $x \oplus a_i$ 是 $x \oplus a_1, \ldots, x \oplus a_n$ 中第 k 大的数，应该如何写？

第 k 大异或和问题

如果想对于给定的 x，找到一个 a_i，使 $x \oplus a_i$ 是 $x \oplus a_1, \ldots, x \oplus a_n$ 中第 k 大的数，应该如何写？

```
int query(int x, int k)
{
    int o=1;
    int res=0;
    for(int i=31;i>=0;i--)
    {
        int j=(x>>i)&1;
        if(sz[ch[o][j^1]]>=k) o=ch[o][j^1],res|=1u<<i;
        else k-=sz[ch[o][j^1]],o=ch[o][j];
    }
    return res;
}
```

[十二省联考 2019] 异或粽子

给定 $n\ (\le 5 \times 10^5)$ 个数 $a_1, ..., a_n$，选一个区间 $[l, r]$，将 $a_l, ..., a_r$ 全部异或起来，得到这个区间的权值。求权值前 $m\ (\le 2 \times 10^5)$ 大的区间权值之和。

[十二省联考 2019] 异或粽子

令 $b_i = a_1 \oplus \ldots \oplus a_i$，那么 $a_l \oplus \ldots \oplus a_r = b_{l-1} \oplus b_r$，转化为两个数的异或，可以用 Trie 解决。

[十二省联考 2019] 异或粽子

令 $b_i = a_1 \oplus \ldots \oplus a_i$，那么 $a_l \oplus \ldots \oplus a_r = b_{l-1} \oplus b_r$，转化为两个数的异或，可以用 Trie 解决。

具体地，先把 $b_0, \ldots, b_n$ 的二进制插入 Trie 中。接下来对每个 b_i 找到一个 b_j 使它和 b_i 异或起来最大。

[十二省联考 2019] 异或粽子

令 $b_i = a_1 \oplus \ldots \oplus a_i$，那么 $a_l \oplus \ldots \oplus a_r = b_{l-1} \oplus b_r$，转化为两个数的异或，可以用 Trie 解决。

具体地，先把 $b_0, \ldots, b_n$ 的二进制插入 Trie 中。接下来对每个 b_i 找到一个 b_j 使它和 b_i 异或起来最大。

将这些值存进一个优先队列中。每次从优先队列取出最大的异或和，弹出，如果它是 b_i 和其它数的异或和中第 k 大的，就找到 b_i 和其它数的异或和中第 $k+1$ 大的加入优先队列。不断重复，直到弹出的数达到 $2m$ 个为止，最后答案要除以 2.（为什么？）

基本概念

AC 自动机 $=$ 字典树 $+$ fail 数组

对所有模式串建立字典树，根节点为 0 号。

基本概念

AC 自动机 = 字典树 + fail 数组

对所有模式串建立字典树，根节点为 0 号。

设 p 是字典树上的一个节点，从根节点到 p 的路径上所有的字母拼起来一定是某个模式串的前缀，我们记这个前缀为 S_p，我们说它是 p 代表的字符串

基本概念

AC 自动机 = 字典树 + fail 数组

对所有模式串建立字典树，根节点为 0 号。

设 p 是字典树上的一个节点，从根节点到 p 的路径上所有的字母拼起来一定是某个模式串的前缀，我们记这个前缀为 S_p，我们说它是 p 代表的字符串

`fail[p]` 是字典树的一个节点，而且它代表的前缀也是 S_p 的后缀，并且是最长的那个。

基本概念

AC 自动机 = 字典树 + fail 数组

对所有模式串建立字典树，根节点为 0 号。

设 p 是字典树上的一个节点，从根节点到 p 的路径上所有的字母拼起来一定是某个模式串的前缀，我们记这个前缀为 S_p，我们说它是 p 代表的字符串

`fail[p]` 是字典树的一个节点，而且它代表的前缀也是 S_p 的后缀，并且是最长的那个。用数学语言就是：

$$\mathcal{F}(p) = \{q \mid q\text{是节点，满足}q \neq p\text{，且}S_q\text{是}S_p\text{的后缀}\}$$

$$fail[p] \in \mathcal{F}(p), \quad \text{且 } |S_{fail[p]}| = \max_{q \in \mathcal{F}(p)} |S_q|$$

基本概念

AC 自动机 = 字典树 + fail 数组

对所有模式串建立字典树，根节点为 0 号。

设 p 是字典树上的一个节点，从根节点到 p 的路径上所有的字母拼起来一定是某个模式串的前缀，我们记这个前缀为 S_p，我们说它是 p 代表的字符串

`fail[p]` 是字典树的一个节点，而且它代表的前缀也是 S_p 的后缀，并且是最长的那个。用数学语言就是：

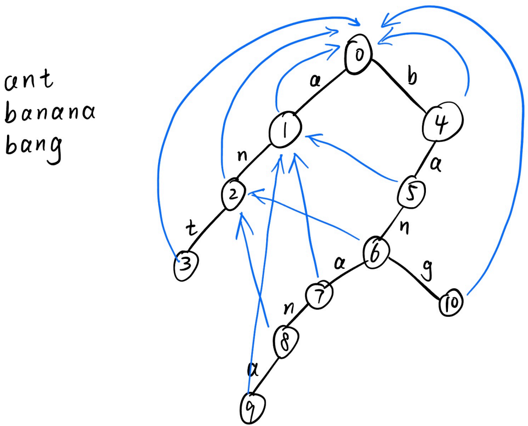
$$\mathcal{F}(p) = \{q \mid q\text{是节点，满足}q \neq p\text{，且}S_q\text{是}S_p\text{的后缀}\}$$

$$fail[p] \in \mathcal{F}(p), \quad \text{且 } |S_{fail[p]}| = \max_{q \in \mathcal{F}(p)} |S_q|$$

特别地，如果 $\mathcal{F}(p) = \varnothing$，那么 $fail[p] = 0$。

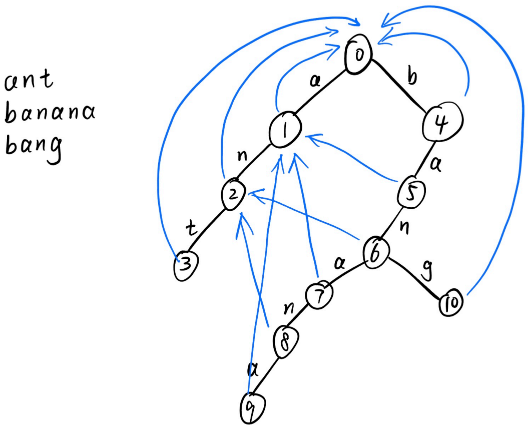
fail 树

如果把每个节点 p 向 $fail[p]$ 连一条蓝色的边，我们会发现，所有蓝色的边构成一棵树，其中 p 的父亲是 $fail[p]$.



fail 树

如果把每个节点 p 向 $fail[p]$ 连一条蓝色的边，我们会发现，所有蓝色的边构成一棵树，其中 p 的父亲是 $fail[p]$.



求 fail

现在，我们来看如何求 `fail` 数组。注意，必须先把所有模式串的字典树建立好，才能开始求 `fail`。

求 fail

现在，我们来看如何求 `fail` 数组。注意，必须先把所有模式串的字典树建立好，才能开始求 `fail`。

我们用字典树的 BFS 序来求 `fail`，现在遍历到节点 `u`，我们来枚举它的下一个节点 `v=ch[u][c]`。

求 fail

现在，我们来看如何求 `fail` 数组。注意，必须先把所有模式串的字典树建立好，才能开始求 `fail`。

我们用字典树的 BFS 序来求 `fail`，现在遍历到节点 `u`，我们来枚举它的下一个节点 `v=ch[u][c]`。

试想，如果 `fail[v]` 不指向根，那么 `fail[v]=q`，其中 `q=ch[f][c]`（一定存在这样的 f）。这个 `f` 在哪里找？

求 fail

现在，我们来看如何求 `fail` 数组。注意，必须先把所有模式串的字典树建立好，才能开始求 `fail`。

我们用字典树的 BFS 序来求 `fail`，现在遍历到节点 `u`，我们来枚举它的下一个节点 `v=ch[u][c]`。

试想，如果 `fail[v]` 不指向根，那么 `fail[v]=q`，其中 `q=ch[f][c]`（一定存在这样的 f）。这个 `f` 在哪里找？

显然，$f \in \mathcal{F}(u)$，因此只要从 u 出发，沿着 fail 树往上跳，如果发现某个 `f` 满足 `ch[f][c]`$\neq 0$ 就立刻停止，并令 `fail[v]=ch[f][c]`。

求 fail

```
void getfail(){
    queue<int> q;
    for(int c = 0; c < 26; c++)
        if(ch[0][c]) q.push(ch[0][c]);
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(int c = 0; c < 26; c++){
            int v = ch[u][c];
            if(!v) continue;
            int f = fail[u];
            while(f && !ch[f][c]) f = fail[f];
            fail[v] = ch[f][c];
            q.push(v);
        }
    }
}
```

它是 $O(n)$ 的，为什么？

[P3808] AC 自动机（简单版）

给定 n 个模式串 s_i 和一个文本串 t，求有多少个不同的模式串在文本串里出现过。
两个模式串不同当且仅当他们 **编号**不同。

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

现在，我们在字典树上从根节点出发，顺着文本串 s 一步一步走，即 `u=0`，`u=ch[u][s[1]]`，`u=ch[u][s[2]]`，……（我们先假设不会碰到走不下去的情况）

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

现在，我们在字典树上从根节点出发，顺着文本串 s 一步一步走，即 `u=0`，`u=ch[u][s[1]]`，`u=ch[u][s[2]]`，⋯⋯（我们先假设不会碰到走不下去的情况）

可以知道，在这个过程中，S_u 都在 s 中出现过；S_p $(p \in \mathcal{F}(u))$ 也都在 s 中出现过；除此之外不会有其它的出现过。

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

现在，我们在字典树上从根节点出发，顺着文本串 s 一步一步走，即 u=0，u=ch[u][s[1]]，u=ch[u][s[2]]，⋯⋯（我们先假设不会碰到走不下去的情况）

可以知道，在这个过程中，S_u 都在 s 中出现过；S_p $(p \in \mathcal{F}(u))$ 也都在 s 中出现过；除此之外不会有其它的出现过。

这就产生了一种做法：令 appear[u] 表示 S_u 是否在 s 中出现过，u 每走一步，就把 appear[u] 标记成 true；然后令 f 从 u 出发，沿着 fail 一直跳到根，把途径的 appear[f] 都标记成 true.

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

现在，我们在字典树上从根节点出发，顺着文本串 s 一步一步走，即 `u=0`，`u=ch[u][s[1]]`，`u=ch[u][s[2]]`，……（我们先假设不会碰到走不下去的情况）

可以知道，在这个过程中，S_u 都在 s 中出现过；S_p $(p \in \mathcal{F}(u))$ 也都在 s 中出现过；除此之外不会有其它的出现过。

这就产生了一种做法：令 `appear[u]` 表示 S_u 是否在 s 中出现过，u 每走一步，就把 `appear[u]` 标记成 `true`；然后令 f 从 u 出发，沿着 fail 一直跳到根，把途径的 `appear[f]` 都标记成 `true`.

当然，在 f 往上跳的过程中，如果发现 `appear[f]` 已经被标记成了 `true`，那么可以立即终止循环，因为之后的肯定早就被标记过了。这样可以保证 fail 树上的每个节点只被标记一次，从而保证复杂度线性。

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

现在，我们在字典树上从根节点出发，顺着文本串 s 一步一步走，即 `u=0`，`u=ch[u][s[1]]`，`u=ch[u][s[2]]`，⋯⋯（我们先假设不会碰到走不下去的情况）

可以知道，在这个过程中，S_u 都在 s 中出现过；S_p $(p \in \mathcal{F}(u))$ 也都在 s 中出现过；除此之外不会有其它的出现过。

这就产生了一种做法：令 `appear[u]` 表示 S_u 是否在 s 中出现过，u 每走一步，就把 `appear[u]` 标记成 `true`；然后令 f 从 u 出发，沿着 fail 一直跳到根，把途径的 `appear[f]` 都标记成 `true`.

当然，在 f 往上跳的过程中，如果发现 `appear[f]` 已经被标记成了 `true`，那么可以立即终止循环，因为之后的肯定早就被标记过了。这样可以保证 fail 树上的每个节点只被标记一次，从而保证复杂度线性。

现在，如果 u 在走的过程中发现走不下去，怎么办？

[P3808] AC 自动机（简单版）

首先构建模式串的字典树，并求出 fail 数组。

现在，我们在字典树上从根节点出发，顺着文本串 s 一步一步走，即 `u=0`，`u=ch[u][s[1]]`，`u=ch[u][s[2]]`，⋯⋯（我们先假设不会碰到走不下去的情况）

可以知道，在这个过程中，S_u 都在 s 中出现过；S_p $(p \in \mathcal{F}(u))$ 也都在 s 中出现过；除此之外不会有其它的出现过。

这就产生了一种做法：令 `appear[u]` 表示 S_u 是否在 s 中出现过，u 每走一步，就把 `appear[u]` 标记成 `true`；然后令 f 从 u 出发，沿着 fail 一直跳到根，把途径的 `appear[f]` 都标记成 `true`.

当然，在 f 往上跳的过程中，如果发现 `appear[f]` 已经被标记成了 `true`，那么可以立即终止循环，因为之后的肯定早就被标记过了。这样可以保证 fail 树上的每个节点只被标记一次，从而保证复杂度线性。

现在，如果 u 在走的过程中发现走不下去，怎么办？

沿着 fail 往上跳，直到 `ch[u][s[i]]`$\neq 0$ 为止。（请思考：为什么这样能保证不重不漏地标记在 s 中出现过的所有 S_p）

[P3808] AC 自动机（简单版）

代码像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        while(u && !ch[u][c]) u = fail[u];
        u = ch[u][c];
        appear[u] = true;
        int f = fail[u];
        while(f && !appear[f])
            appear[f] = true, f = fail[f];
    }
}
```

[P3808] AC 自动机（简单版）

代码像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        while(u && !ch[u][c]) u = fail[u];
        u = ch[u][c];
        appear[u] = true;
        int f = fail[u];
        while(f && !appear[f])
            appear[f] = true, f = fail[f];
    }
}
```

在字典树 insert 的过程中，记录一下每一个模式串 t_i 对应的节点是哪个，记作 idx[i]，然后根据 appear[idx[i]] 来统计答案。

求 fail（路径压缩版）

我们可以在求 fail 的时候顺便压缩路径。

求 fail（路径压缩版）

我们可以在求 fail 的时候顺便压缩路径。

具体而言，因为我们知道在 traval 的过程中，如果发现 ch[u][c]=0，我们会从 u 出发沿着 fail 往上跳，直到找到 ch[f][c]=q$\neq 0$ 的位置为止。我们不妨在求 fail 的时候就把这一步做好，一步到位令 ch[u][c]=q.

求 fail（路径压缩版）

我们可以在求 fail 的时候顺便压缩路径。

具体而言，因为我们知道在 traval 的过程中，如果发现 ch[u][c]=0，我们会从 u 出发沿着 fail 往上跳，直到找到 ch[f][c]=q$\neq 0$ 的位置为止。我们不妨在求 fail 的时候就把这一步做好，一步到位令 ch[u][c]=q.

```
void getfail(){
    queue<int> q;
    for(int c = 0; c < 26; c++)
        if(ch[0][c]) q.push(ch[0][c]);
    while(!q.empty()){
        int u = q.front(); q.pop();
        int f = fail[u];
        for(int c = 0; c < 26; c++){
            int& v = ch[u][c];
            if(!v) v = ch[f][c];
            else fail[v] = ch[f][c], q.push(v);
        }
    }
}
```

搜索（路径压缩版）

有了路径压缩，我们在 travel 的过程中就不需要跳 fail 了。（当然，标记答案的时候还是要跳 fail）

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        u = ch[u][c];
        appear[u] = true;
        int f = fail[u];
        while(f && !appear[f])
            appear[f] = true, f = fail[f];
    }
}
```

[P5357] 【模板】AC 自动机

给定 n 个模式串 s_i 和一个文本串 t，求每个模式串在文本串里出现的次数。

[P5357] 【模板】AC 自动机

我们回顾之前的 travel 代码，很容易会想把 appear 数组改成 cnt 数组，用于统计出现次数，像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        u = ch[u][c];
        cnt[u]++;
        int f = fail[u];
        while(f) cnt[f]++, f = fail[f];
    }
}
```

[P5357] 【模板】AC 自动机

我们回顾之前的 travel 代码，很容易会想把 appear 数组改成 cnt 数组，用于统计出现次数，像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        u = ch[u][c];
        cnt[u]++;
        int f = fail[u];
        while(f) cnt[f]++, f = fail[f];
    }
}
```

但是，回顾我们之前说的，我们是如何保证 travel 的复杂度是线性的？

[P5357] 【模板】AC 自动机

我们回顾之前的 travel 代码，很容易会想把 appear 数组改成 cnt 数组，用于统计出现次数，像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        u = ch[u][c];
        cnt[u]++;
        int f = fail[u];
        while(f) cnt[f]++, f = fail[f];
    }
}
```

但是，回顾我们之前说的，我们是如何保证 travel 的复杂度是线性的？

必须要保证每个节点只被标记一次。可是这里我们不能保证，怎么办？

[P5357] 【模板】AC 自动机

还记得 fail 构成一棵树吗？我们可以在 travel 的时候只令 cnt[u]++，最后来一次 dfs，把每个节点 u 的子树里面的 cnt 全加起来。最后代码像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        u = ch[u][c];
        cnt[u]++;
    }
}

void dfs(int u){
    for(int v : g[u])
        dfs(v), cnt[u] += cnt[v];
}
```

[P5357] 【模板】AC 自动机

还记得 fail 构成一棵树吗？我们可以在 travel 的时候只令 cnt[u]++，最后来一次 dfs，把每个节点 u 的子树里面的 cnt 全加起来。最后代码像这样：

```
void traval(char s[]){
    int n = strlen(s+1), u = 0;
    for(int i = 1; i <= n; i++){
        int c = s[i] - 'a';
        u = ch[u][c];
        cnt[u]++;
    }
}

void dfs(int u){
    for(int v : g[u])
        dfs(v), cnt[u] += cnt[v];
}
```

当然，在调用完 travel 之后，要把 fail 树像存图那样存起来，然后再调用 dfs。

Thank You