Trie 树与 AC 自动机

字典树共享前缀 + 0-1 Trie 异或极值 + AC 自动机多模式一次扫描

HGNU-ACM 实验室算法组

配套：字符 Trie 与二进制 0-1 Trie + BFS 建 `fail` 与 Trie 图 + 4 道经典真题逐题精讲





本章目录

字典树（Trie）与二进制 0-1 Trie

AC 自动机：Trie 树 + KMP 失配指针

第 1 节学习字典树 Trie 如何共享前缀省内存，以及二进制 0-1 Trie 贪心求最大异或；第 2 节学习如何把 KMP 的 π 搬到 Trie 树变成 AC 自动机。

本章目录（续）

例题精讲（一）：0-1 Trie 经典实战

例题精讲（二）：AC 自动机经典实战

常见错误、检查清单与练习安排

第 3、4 节逐题精讲《The XOR Largest Pair》《Codechef REBXOR》《Keywords Search》《Censoring》，第 5 节总结易错对照、检查清单与练习安排。

生活引入：查英汉词典为什么不用从第一页翻到最后一页？

承接上一章：从单串匹配到海量词库

上一章我们用 KMP 在长文章里找 **1** 个模式串，时间是线性 $O(n+m)$。但如果词库里一下子有十万个单词，甚至是一堆数字的二进制位，怎么存才又省内存又查得快？

- 翻英汉词典找 `apple` 时，你先翻到 `a` 区，再找 `p`，再找第二个 `p`——相同开头的单词被收束在一起。
- 计算机里把这种“按字符一层层分叉”的树形结构叫做字典树（**Trie** 树）。
- 比如 `cat` 和 `car`，前两个字母 `c-a` 完全一样，在树上只存一份，到第 3 个字母才分叉成 `t` 和 `r`！

本页要点

字典树（Trie）的核心思想：用树枝共享公共前缀，查一个长度为 $|S|$ 的单词只需走 $|S|$ 步！

为什么需要 Trie？暴力存字符串与对比的痛点

- 假设词库有 $N = 10^5$ 个单词，总长度 10^6，接下来有 $Q = 10^5$ 次询问：“以 `pre` 开头的单词有多少个？”
- 暴力做法：把 10^5 个单词存在数组里，每次询问遍历所有单词比前缀，总操作次数 $10^5 \times 10^5 = 10^{10}$ 次，必 **TLE**！
- 直接哈希的局限：字符串哈希能查“整个词一模一样”，但遇到“求某前缀出现次数”或“二进制异或最大”就不好直接贪心了。
- **Trie** 树的优势：查询、插入一个长度 L 的串都只需要 $O(L)$ 时间，完全和词库里有多少万个单词无关！

本页要点

Trie 树的查询时间只取决于当前单词有多长 $O(|S|)$，和词库总单词数 N 无关！

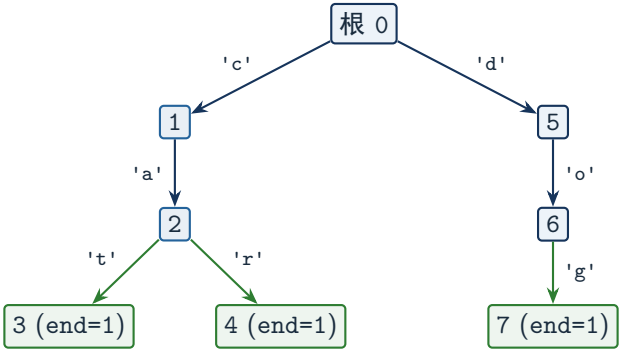
新术语大白话：节点、字符边与结尾标记 end_cnt

- 根节点 0：代表空字符串 ""，所有单词都从 0 号点出发。
- 字符边 tr[u][c]：从节点 u 沿着字母 c 走到下一个节点编号（如果没有路就是 0）。字母写在边上，节点编号本身没有字母含义！
- 结尾标记 end_cnt[u]：记录有多少个完整单词恰好停在节点 u。
- 为什么一定要有 end_cnt[u]？因为如果你插入了 "cat"，树上就已经有了 c -> a 这条路；这时别人问“词库里有 "ca" 吗?”，必须看节点 a 的 end_cnt 是不是 > 0！

本页要点

Trie 树边上放字符，节点上放计数：必须靠 end_cnt[u] > 0 区分“只是前缀”还是“完整单词”。

独占图解：把 cat，car，dog 放进 Trie 树



本页要点

cat 与 car 共享了 0->1->2（前缀 "ca"），只在节点 2 才分出 't' 和 'r' 两条边。

手算跟踪表：依次插入 cat，car，ca 并查询 ca 与 cap

假设新节点编号从 1 开始递增（idx 初始为 0）：

步骤	操作	经过节点路径	新建节点	结果说明
1	插入 cat	0->1->2->3	1，2，3	end_cnt[3] = 1
2	插入 car	0->1->2->4	仅新建 4	0->1->2 直接复用，end_cnt[4] = 1
3	插入 ca	0->1->2	无新节点	在节点 2 标记 end_cnt[2] = 1
4	查询 ca	0->1->2	—	查 end_cnt[2] == 1 $\Longrightarrow$ 存在！
5	查询 cap	0->1->2->?	—	终止：tr[2]['p'-'a'] == 0，无路可走 $\Longrightarrow$ 不存在

本页要点

插入时遇到 tr[u][c] == 0 就分配新编号；查询时一旦遇到 tr[u][c] == 0 立刻返回失败！

模板代码（1/2）：字符 Trie 的结构与插入

```
struct CharTrie {
    vector<array<int, 26>> tr{{}};              // 0 号点为根，26 个小写字母边
    vector<int> end_cnt{0};                     // 每个节点结尾的单词个数
    void insert(const string &s) {
        int u = 0;
        for (char ch : s) {
            int c = ch - 'a';
            if (!tr[u][c]) { tr[u][c] = (int)tr.size(); tr.emplace_back(); end_cnt.
    push_back(0); }
            u = tr[u][c];
        }
        ++end_cnt[u];                           // 在末尾节点打上单词结束标记
    }
```

本页要点

用 vector<array<int, 26>> 动态扩容，再也不用担心静态数组开小 **RE**。

模板代码（2/2）：字符 Trie 的完整词查询

```
    int query(const string &s) const {          // 返回 s 作为完整词出现的次数
        int u = 0;
        for (char ch : s) {
            u = tr[u][ch - 'a'];
            if (!u) return 0;                   // 没路可走，直接返回不存在
        }
        return end_cnt[u];
    }
};
```

本页要点

如果要统计“以 s 为前缀的单词有多少个”，只需在 insert 循环里顺便累加 pass[u]。

为什么这样写？节点上界与 end_cnt 的必要性

- 为什么静态数组要开 $\sum |S_i| + 5$ 而不是 N？
  - N 是单词个数，但每个单词最多 50 个字母；最坏所有单词没有公共前缀，每个字母都要占一个节点！
  - 所以节点总数上限是所有单词长度之和 $\sum |S_i| + 1$。
- 为什么 query 里判断 `if (!u) return 0` 就够了？
  - 因为新分配的节点编号 ≥ 1，只有“不存在这条边”时 `tr[u][c]` 才是 `0`。一旦走到 `0`，说明后面没路了，可以直接返回 0。

本页要点

记住：Trie 树空间复杂度是 $O(\sum |S_i| \times |\Sigma|)$（其中 $|\Sigma| = 26$ 是字符集大小）。

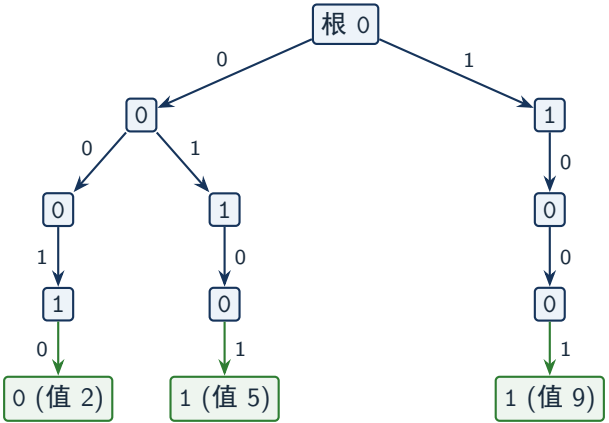
从字符到二进制：什么是 0-1 Trie？

- 如果把字典树的字符集从 26 个字母换成只有 `0` 和 `1` 两个字符，它就成了一棵只有左右两个分叉的二叉树——这就是 **0-1 Trie**！
- 怎么把一个非负整数 x（比如 $0 \le x < 2^{31}$）放进 0-1 Trie？
- 把 x 看成从第 30 位（最高位）到第 0 位（最低位）一共 31 位的二进制字符串，不够 31 位的前面补 `0`！
- 例如 5 的 4 位二进制是 `0101`，就沿着根走 `0 -> 1 -> 0 -> 1`！

本页要点

0-1 Trie 把每个整数固定拆成 31 位（`b = 30..0`）的 `0/1` 串插入二叉树，所有数在树上的深度完全一样！

独占图解：把 2(0010)，5(0101)，9(1001) 放进 4 位 0-1 Trie



本页要点

每个数都补齐前导零从最高位往下插，所有叶子都在同一层！

贪心核心：为什么查询 x 时高位能走相反位（c ^ 1）就一定要走？

- 异或运算的规则：相同得 0，不同得 1（0^1 = 1，1^0 = 1）。
- 想让 $x \oplus y$ 最大，就是要让结果的高位尽可能多出 1！
- 假设当前看第 b 位，x 的这一位是 c（0 或 1）：
  - 如果树上存在相反方向 ch[u][c ^ 1] != 0，走过去这一位就能赚到 2^b，把 res |= (1 << b)；
  - 如果不存在相反方向，只能被迫走相同方向 ch[u][c]，这一位贡献 0。
- 为什么高位贪心一定是对的？因为 $2^b > 2^{b-1} + 2^{b-2} + \cdots + 2^0 = 2^b - 1$！高位多一个 1，稳赢后面所有低位之和！

本页要点

0-1 Trie 求最大异或的口诀：从最高位 30 往下试，能走相反比特 c ^ 1 就走，没有才走 c！

0-1 Trie 手算跟踪表：在 $\{2,5,9\}$ 中查询与 7 (0111) 异或的最大值

已知树里有 2(0010)，5(0101)，9(1001)，现在拿 $x=7$（二进制 0111）从高到低匹配：

比特位 b	x 的位 c	期望相反位 c^1	树上有路吗？	实际走向	累加 res 值
3（权重 8）	0	1	有（通往 9）	走 1	res = 0 + 8 = 8
2（权重 4）	1	0	有（9 是 0）	走 0	res = 8 + 4 = 12
1（权重 2）	1	0	有（9 是 0）	走 0	res = 12 + 2 = 14
0（权重 1）	1	0	无（只有 1）	被迫走 1	终止：返回 res = 14（$7 \oplus 9 = 14$）

本页要点

每走一层就把所有不满足高位最优的分支剪掉，只用 31 步就从几万个数里挑出最优搭档！

0-1 Trie 正确性论证（高位优先贪心不变量）

步骤	高位贪心不变量论证
初始化	进入第 30 位前，节点 $u = 0$ 代表树中所有数的前缀集（非空），已得高位前缀 res $= 0$。
保持	若当前子树中存在使第 b 位异或为 1 的分支，因 $2^b > \sum_{k=0}^{b-1} 2^k$，任何选该分支的数所得异或和必严格大于不选该分支的所有数；故只保留该子树不丢失全局最大值。
终止	当 b 减到 0 并完成选择后，到达的叶子对应某个真实插入过的数 y，返回的 res 即为最大 $x \oplus y$。

本页要点

只要树非空（已经至少插入过 1 个数），从根到叶子的路径必然不会断掉。

模板代码（1/2）：0-1 Trie 结构与插入（code/trie_xor.cpp）

```cpp
struct BinaryTrie {
    vector<array<int, 2>> ch{{0, 0}};           // 只有 0 和 1 两个儿子
    void insert(int x) {
        int u = 0;
        for (int b = 30; b >= 0; --b) {         // 从第 30 位到第 0 位依次插入
            int c = (x >> b) & 1;
            if (!ch[u][c]) { ch[u][c] = (int)ch.size(); ch.push_back({0, 0}); }
            u = ch[u][c];
        }
    }
```

本页要点

每个整数固定占 31 层；如果题目数据达到 `long long`，循环改从 `b = 60` 开始。

模板代码（2/2）：0-1 Trie 查询最大异或（code/trie_xor.cpp）

```
    int max_xor(int x) const {
        int u = 0, res = 0;
        for (int b = 30; b >= 0; --b) {          // 必须从最高位 30 往下试！
            int c = (x >> b) & 1;
            if (ch[u][c ^ 1]) { res |= (1 << b); u = ch[u][c ^ 1]; }
            else u = ch[u][c];
        }
        return res;
    }
};
```

本页要点

若用 long long，记得把 (1 << b) 改成 (1LL << b) 防移位溢出！

上机验证与反事实追问

3 组上机测试数据（输入数组，输出两数最大异或值）

1. 普通情况：`[2, 9, 5, 7, 0]` $\to 9 \oplus 7 = 14$（对应 `code/trie_xor.cpp`）。
2. 最小边界：`[5]`（只有 1 个数）$\to$ 不进入查询直接输出 `best = 0`；或 `[3, 3]` $\to 3 \oplus 3 = 0$。
3. 特殊大数：`[0, 1073741824]`（即 0 与 2^{30}）$\to$ 测最高位 $b = 30$ 是否能正确异或出 2^{30}。

反事实追问

如果把 `max_xor` 里的循环方向写反，写成从低位到高位 `for (int b = 0; b <= 30; ++b)` 会怎样？

$\Longrightarrow$ 既和 `insert` 的建树顺序（从高位建）完全错位，又违背高位优先贪心，结果彻底错误（**WA**）！

本页要点

记住：建树与查询必须方向一致——永远从最高位向第 0 位走！

Trie 与 0-1 Trie 高频易错对照

错误写法	正确写法	触发的评测结果
静态数组大小只开单词数 N	开总字符数 $\sum \lvert S_i \rvert$ 或用 `vector`	节点不够数组越界 $\rightarrow$ **RE**
0-1 Trie 还没插数就先查	先插或从第 2 个数开始查（前缀异或先插 0）	在空树里一直走 0 节点 $\rightarrow$ **WA**
忘了 `end_cnt[u]` 直接判非空	必须检查 `end_cnt[u] > 0`	把前缀误判成完整单词 $\rightarrow$ **WA**
`long long` 用 `1 << b`	用 `1LL << b`	当 $b \geq 31$ 时移位溢出 $\rightarrow$ **WA**

本页要点

做区间最大异或和时（结合前缀异或 $s_r \oplus s_{l-1}$），千万别忘了在最开始先往树里插入一个 0！

问题升级：一次要在一篇文章里查一万个敏感词怎么办？

- 给你 $n = 10^4$ 个敏感词（总长 5×10^5）和一篇长文章 T（长 10^6），问文章里出现了多少个敏感词？
- 如果拿上一章的 **KMP** 跑一万遍：每次都要把长 10^6 的文章从头扫一遍！
- 总操作次数：$10^4 \times 10^6 = 10^{10}$ 次，妥妥 **TLE**！
- **AC** 自动机（**Aho-Corasick**）的绝妙想法：把所有敏感词先塞进同一棵 **Trie** 树，再给每个节点配上类似 KMP 的失配指针 `fail`——这样整篇文章只从头到尾扫一遍，就能同时匹配所有敏感词！

本页要点

一句话理解 AC 自动机：**Trie** 树 + **KMP** 的 `fail` 失配指针，实现多模式串一次线性扫描。

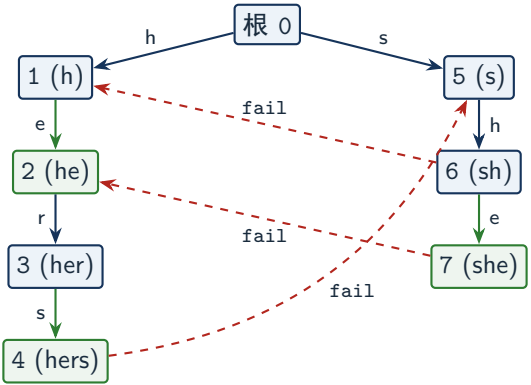
新术语大白话：fail[u] 到底指向谁？

- 在 KMP 里，pi[j-1] 表示当前前缀的最长相等真后缀。
- 在 AC 自动机里，每个节点 u 代表从根到 u 拼出来的一个字符串 S_u。
- fail[u] 的大白话定义：在整棵 Trie 树里，找一个最长的、同时也是 S_u 真后缀的节点！如果一个都没找到，就指回根节点 0。
- 比如树里有 "she" 和 "he"：当文章匹配到节点 "she" 后面失配了，发现 "he" 正好是 "she" 的真后缀，于是顺着 fail 指针跳到 "he" 的结尾继续匹配！

本页要点

fail[u] 指的就是“当前串的最长后缀在 Trie 树上对应的那个节点”。

独占图解（1/2）：he，she，hers 建出的 Trie 与 fail 指针



本页要点

看图：sh 的最长真后缀是 h（点 1），she 的是 he（点 2），hers 的是 s（点 5）！

BFS 按层构建 fail 的核心公式

怎么算节点 v = tr[u][c] 的失配指针 fail[v]？

- 用广度优先搜索（BFS）一层一层往下算，所以算到父亲 u 时，fail[u] 早就已经算好了！
- 看父亲的失配节点 fail[u] 有没有 c 这条边：

核心公式

```
fail[tr[u][c]] = tr[fail[u]][c];
```

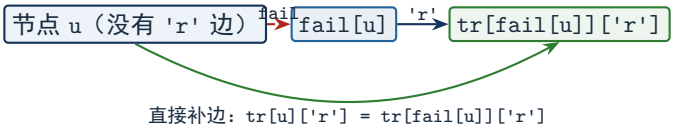
- 意思是：把“父亲的最长后缀”后面也接一个字符 c，那它就是“儿子的最长后缀”！
- 如果 fail[u] 没有 c 儿子怎么办？下面一页的 **Trie** 图压缩直接把它铺平！

本页要点

记住一句话：儿子的失配点，等于“父亲失配点的同一个字母儿子”！

独占图解（2/2）：Trie 图路径压缩——直接铺平失配边

如果当前节点 u 根本没有字母 c 的儿子（tr[u][c] == 0），原本匹配时要沿着 fail 一直跳到有 c 儿子为止。



- 在 build() 时顺手加上：else tr[u][c] = tr[fail[u]][c];
- 把缺失的树边全部补成“传送门”，整棵树变成一张确定性自动机（**Trie** 图）！主串匹配时每读一个字符只需一步 u = tr[u][c]，彻底省去 while 跳 fail！

本页要点

Trie 图路径压缩把“边跳边找”变成“一步直达”，代码更短、速度更快！

手算跟踪表：对 he，she，hers 跑 build() 的 BFS 队列

初始第 1 层只有 1(h) 和 5(s)，它们的 fail 默认都是 0，直接入队：

出队节点 u	已有 fail[u]	扫描到的实儿子	动作与入队结果
初始入队	—	1(h), 5(s)	队列：[1, 5]
$u=1$ (h)	0	e -> 2	fail[2] = tr[0]['e'] = 0; 入队 2
$u=5$ (s)	0	h -> 6	fail[6] = tr[0]['h'] = 1; 入队 6
$u=2$ (he)	0	r -> 3	fail[3] = tr[0]['r'] = 0; 入队 3
$u=6$ (sh)	1	e -> 7	fail[7] = tr[1]['e'] = 2; 入队 7
$u=3$ (her)	0	s -> 4	fail[4] = tr[0]['s'] = 5; 入队 4
$u=7,4$	2，5	无实儿子	补全虚边后队列为空，终止！

本页要点

看 $u=6$（sh）：因为 fail[6]=1，所以儿子 7(she) 直接继承 tr[1]['e']=2(he)，一步算对！

千万别踩的坑：为什么第 1 层绝对不能把根 0 入队？

- 很多初学者偷懒写 `q.push(0)`，以为能少写一个 `for` 循环，结果直接死循环或全错！
- 推演一下如果把 0 入队会发生什么：
  1. 弹出 $u = 0$，此时 `fail[0] = 0`；
  2. 检查儿子 $c =$ `'h'`，发现 `tr[0]['h'] = 1` 存在；
  3. 执行公式：`fail[1] = tr[fail[0]]['h'] = tr[0]['h'] = 1`!!
  4. 节点 1 的失配指针竟指向了它自己！以后顺着 `fail` 跳就会陷入死循环 **TLE**！

正确初始化

只把根节点 0 真实存在的儿子 `if (tr[0][c]) q.push(tr[0][c])`；压入队列，第 1 层节点的 `fail` 保持为 0！

本页要点

AC 自动机 `build()` 的铁律：绝对不能把根 0 放进队列，必须先把第 1 层实儿子入队！

正确性论证：为什么 BFS 顺序一定能算出正确的 fail？

步骤	BFS 构建 fail 不变量论证
初始化	第 1 层节点（单字母）没有任何非空真后缀，其 fail 必为 0；队列初始恰包含全部第 1 层节点。
保持	每次从队头取深度为 d 的节点 u：因为 fail[u] 对应真后缀，长度严格 $< d$，故 fail[u] 必已出队并完成了 Trie 图补全，因此 tr[fail[u]][c] 已经是最终转移目标（u 的最长后缀接 c）。
终止	当队列清空时，所有深度节点全部处理完毕，全图每个 tr[u][c] 均为合法状态。

本页要点

“真后缀长度严格更短”＋“BFS 按深度递增出队”共同保证了 fail[u] 永远先算完！

模板代码（1/2）：AC 自动机结构与插入（code/ac_automaton.cpp）

```
struct AC {
    vector<array<int, 26>> tr{{}};                  // Trie 图转移表
    vector<int> fail{0}, end_cnt{0};                // 失配指针与结尾词计数
    void insert(const string &s) {
        int u = 0;
        for (char ch : s) {
            int c = ch - 'a';
            if (!tr[u][c]) { tr[u][c] = (int)tr.size(); tr.emplace_back(); fail.
    push_back(0); end_cnt.push_back(0); }
            u = tr[u][c];
        }
        ++end_cnt[u];
    }
```

本页要点

与 code/ac_automaton.cpp 一致：每新开一个节点，同步给 fail 和 end_cnt 补一个 0。

模板代码（2/2）：BFS 建 fail 与 Trie 图（code/ac_automaton.cpp）

```
    void build() {
        queue<int> q;
        for (int c = 0; c < 26; ++c) if (tr[0][c]) q.push(tr[0][c]); // 仅第 1 层实
    儿子入队
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int c = 0; c < 26; ++c)
                if (tr[u][c]) { fail[tr[u][c]] = tr[fail[u]][c]; q.push(tr[u][c]);
    }
                else tr[u][c] = tr[fail[u]][c]; // 路径压缩补成 Trie 图!
        }
    }
};
```

本页要点

与 code/ac_automaton.cpp 一致：所有模式串全部 insert 完之后，再调用一次 build()！

主串匹配的两大题型：统计“出现过几种”vs “各出现几次”

- 建好 AC 自动机后，拿着主串 T 从根出发，每读一个字符就走一步 `u = tr[u][c]`。
- 注意：停在节点 `u` 时，不仅 `u` 本身可能是一个敏感词，沿着 `fail` 往回跳的那些后缀节点（如 she 跳到 he）也同时匹配成功了！
- 题型一：问文章里出现过多少“种”词？（如《Keywords Search》）
  - 沿着 `fail` 累加 `end_cnt[p]` 后，立刻把 `end_cnt[p] = -1` 封死！以后再跳到 `-1` 就直接 `break`，这样每个节点最多被跳到一次，总时间 $O(|T| + \sum |P|)$。
- 题型二：问“每个敏感词分别出现几次”？
  - 不能封死成 `-1` 了！但直接每步暴力跳 `fail` 会超时，怎么办？见下一页！

本页要点

只问“有没有出现过”可以置 `-1` 去重；要统计“每个词出现总次数”不能置 `-1`，必须用拓扑倒推！

为什么统计频次不能每步顺着 fail 暴跳？

- 假设词库是 $k = 1000$ 个嵌套词：`"a"`, `"aa"`, `"aaa"`, ..., `"aaa...a"`（最长 1000）。
- 文章 T 是 10^5 个 `'a'` 组成的 `"aaaaa..."`。
- 如果文章每读一个 `'a'`，都写 `for (int p = u; p; p = fail[p]) ++cnt[p];`：
- 每次都要从深度 1000 沿着 `fail` 一路跳回 `0`，一共跳 $10^5 \times 1000 = 10^8$ 次（若 $k = 10^4$ 就是 10^9 次），直接退化成 $O(|T| \cdot \sum |P|)$ **TLE**！

怎么破？

扫文章时只做一件事：在当前停靠点 `++ans[u]` 打标记！等整篇文章扫完，再按从深到浅的倒序把 `ans[u]` 一次性累加给 `ans[fail[u]]`！

本页要点

把“每步沿 `fail` 向上跳”改成“先原地打标记，最后顺 `fail` 树自底向上汇总”，复杂度降回严格线性！

线性优化：BFS 逆序汇总（拓扑倒推）

- 怎么知道哪个节点深、哪个节点浅？根本不用再写拓扑排序！
- 在 `build()` 时把每次出队的节点顺手存进一个数组 `order.push_back(u)`，它天然就是按深度从浅到深排好的（也是 `fail` 树上的拓扑序）！
- 整篇文章扫完后，倒着遍历 `order`：

三行汇总代码

```
for (int i = (int)order.size() - 1; i >= 0; --i)
  int u = order[i]; ans[fail[u]] += ans[u];
```

- 每个节点只被加一次，总时间 $O(\sum |P| + |T|)$，彻底告别 **TLE**！

本页要点

记录 `build()` 的 BFS 出队序 `order`，扫完主串后逆序 `ans[fail[u]] += ans[u]`，即可在 $O(1)$ 每节点内完成汇总！

上机验证与反事实追问

3 组上机测试数据（词库 + 文章，输出匹配到的总词数/频次）

1. 普通情况：词库 {"he", "she", "hers"}，文章 "ushers" → 命中 she，he，hers 共 3 个。
2. 嵌套重叠：词库 {"a", "aa", "aaa"}，文章 "aaaa" → a 出现 4 次、aa 3 次、aaa 2 次（验证拓扑汇总不漏不重）。
3. 无匹配：词库 {"abc", "def"}，文章 "xyz" → 输出 0。

反事实追问

如果写 `build()` 时忘了先写 `for (int c = 0; c < 26; ++c) if (tr[0][c]) q.push(tr[0][c]);`，而是写成 `q.push(0);` 会怎样？

$\Longrightarrow$ 第 1 层节点的 `fail` 会指向自己，主串查询或统计时只要沿着 `fail` 跳就会无限死循环 **TLE**！

本页要点

用 "a", "aa", "aaa" 配 "aaaa" 测一下 AC 自动机，能同时验出 `fail` 链和拓扑汇总是否正确。

AC 自动机高频易错对照

错误写法	正确写法	触发的评测结果
`build()` 把根 0 入队	只把 `tr[0][c]` 非空儿子入队	儿子 `fail` 指回自身死循环 → **TLE**
扫文章时每步暴力跳 `fail`	查种类置 `-1`；查次数用 BFS 逆序汇总	全 `'a'` 嵌套数据退化 $O(N^2)$ → **TLE**
多组数据没清空 `tr`，`fail`	每组数据新建一个 `AC` 结构体	上一组模式串残留污染答案 → **WA**
边插词边调用 `build()`	所有词全部 `insert` 完再 `build()`	补上的虚边干扰后续插入 → **WA**

本页要点

一定等所有模式串全都 `insert` 完再调用一次 `build()`，多组数据记得重新构造 `AC`！

例 1 | The XOR Largest Pair [基础]：先读懂要求

大白话题意

给你 N 个非负整数 $A_1, A_2, \ldots, A_N$（$1 \le N \le 10^5,\ 0 \le A_i < 2^{31}$），请你从里面挑出两个数做按位异或（$\oplus$），问能算出的最大异或值是多少？

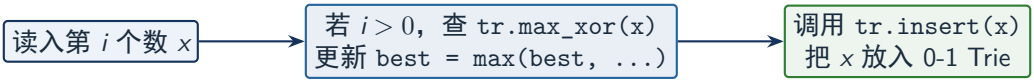
要素	本题具体要求
输入	第一行 N；第二行 N 个整数 A_i。
处理	找出所有数对 (A_i, A_j) 中 $A_i \oplus A_j$ 的最大值。
输出	一个整数表示答案。样例 `2 9 5 7 0` $\rightarrow$ 输出 `14`（$9 \oplus 7 = 14$）。

本页要点

$N \le 10^5$ 时，两重循环枚举所有数对需要 10^{10} 次运算必超时，必须降到 $O(31N)$！

例 1 | The XOR Largest Pair：怎么想到的？

- 暴力枚举 (i, j) 是 $O(N^2)$，必 **TLE**。
- 固定右边的数 A_i，问题变成：“在前面的数 $A_1 \dots A_{i-1}$ 里，找一个和 A_i 异或最大的数”！
- 这正是 **0-1 Trie** 的拿手好戏：一边读入 x，先去树里查 `tr.max_xor(x)` 更新答案，再把 x 插进树里！



本页要点

“先查询再插入”可以保证 x 只和前面已经出现过的数配对，不会自己和自己异或（虽然自己异或自己是 0 也不影响最大值）。

例 1 | The XOR Largest Pair：样例手推跟踪表

按顺序处理样例 [2，9，5，7，0]（用 4 位二进制展示，best 初值为 0）：

下标 i	x（二进制）	先查 max_xor(x)	最佳搭档	更新后 best	随后动作
0	2 (0010)	跳过（树为空）	—	0	插入 2
1	9 (1001)	$9 \oplus 2 = 11$	2	11	插入 9
2	5 (0101)	$5 \oplus 9 = 12$	9	12	插入 5
3	7 (0111)	$7 \oplus 9 = 14$	9	**14**	插入 7
4	0 (0000)	$0 \oplus 9 = 9$	9	14	插入 0，终止输出 14

本页要点

看 $i = 3$：拿 7（0111）在 $\{2, 5, 9\}$ 里查，第 3 位优先选相反比特 1 走向 9（1001），一举算出 14！

例 1 | The XOR Largest Pair：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 初始化 `BinaryTrie tr`	仅有根 0，`best = 0`	$N = 1$ 时 `best` 是 0 吗？
2. 循环读入 x	当前整数 x	比特位上界 30 够不够 $2^{31} - 1$？
3. 若 $i > 0$ 先查最大值	`best = max(best, tr.max_xor(x))`	有没有在空树时就去查？
4. 再把 x 插入树	树中新增一条 31 层路径	31 位全插进去了吗？

相对基础模板改了哪里？

直接复用第 1 节的 `BinaryTrie` 结构体，主函数只需 10 行循环“边查边插”即可。

本页要点

为什么 $b = 30$ 够用？因为 $A_i < 2^{31}$，最高有效位就是第 30 位（2^{30}），第 31 位符号位恒为 0！

例 1 | The XOR Largest Pair：核心代码（code/trie_xor.cpp）

```cpp
// BinaryTrie::insert(x) 与 BinaryTrie::max_xor(x) 见第 1 节模板
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n; if (!(cin >> n)) return 0;
    BinaryTrie tr; int best = 0;                // best 初值为 0
    for (int i = 0, x; i < n; ++i) {
        cin >> x;
        if (i > 0) best = max(best, tr.max_xor(x)); // 先查前面已插入的数
        tr.insert(x);                           // 再把自己插进 0-1 Trie
    }
    cout << best << '\n';
}
```

本页要点

与 code/trie_xor.cpp 一致：时间复杂度 $O(31N)$，空间最多 $31N$ 个节点。

例 1 ｜ The XOR Largest Pair：易错点与 3 组自测边界

常见错误对照

- 第 **1** 个数 `i == 0` 也去查空树：树里还没有任何分支，虽然不会越界，但会走到根本不存在的 0 节点算出错误值。
- 静态数组节点数只开 100005：每个数最多新建 31 个节点，总节点数最多 3.1×10^6！开小必 **RE**（用 `vector` 最稳）。

3 组自测边界

1. 最小 $N = 1$：`[42]` → 输出 0；　2. 全相等：`[7, 7, 7]` → 输出 0；　3. 极大值：`[0, 2147483647]` → 输出 2147483647。

本页要点

记住：一个数插进 0-1 Trie 最多占 31 个节点，静态数组要开 $32 \times N$ 以上！

例 2 | Codechef REBXOR（1473）[提高]：先读懂要求

大白话题意

给定长为 N（$N \le 4 \times 10^5$）的数组 A，请你选出两段不相交的连续子段 $[l_1, r_1]$ 和 $[l_2, r_2]$（满足 $1 \le l_1 \le r_1 < l_2 \le r_2 \le N$），使两段各自异或和的相加之和最大！

要素	本题具体要求
输入	第一行 N；第二行 N 个整数 $A_1 \dots A_N$（$0 \le A_i < 2^{31}$）。
处理	求 $\max\left((A_{l_1} \oplus \cdots \oplus A_{r_1}) + (A_{l_2} \oplus \cdots \oplus A_{r_2})\right)$。
输出	样例 1 2 3 1 2 $\rightarrow$ 选 $[1,2]$（$1 \oplus 2 = 3$）和 $[4,5]$（$1 \oplus 2 = 3$），之和为 6。

本页要点

两段子段必须“严格不重叠”（$r_1 < l_2$），而且最后是把两段的结果加起来！

例 2 | Codechef REBXOR：怎么想到的？

- 第一招（前缀异或）：令 $s[i] = A_1 \oplus \cdots \oplus A_i$（$s[0] = 0$），那么任意一段 $[l, r]$ 的异或和等于 $s[r] \oplus s[l-1]$！
- 第二招（枚举分界线）：两段不能相交，中间一定有一条分界线 i：左段全在 $[1..i]$ 内，右段全在 $[i+1..N]$ 内！
- 于是只要预处理：
  - $L[i]$：前 i 个数里能选出的最大单段异或和（从左到右用 0-1 Trie 扫描）；
  - $R[i]$：后一段 $[i..N]$ 里能选出的最大单段异或和（从右到左用 0-1 Trie 扫描）；
- 最后枚举分界点 $i \in [1, N-1]$，取 $\max(L[i] + R[i+1])$！

本页要点

“两段不相交子段最值”经典套路：预处理前缀最优 $L[i]$ 与后缀最优 $R[i]$，再枚举分界点求和！

例 2 | Codechef REBXOR：样例手推跟踪表

模拟样例 `A = [1, 2, 3, 1, 2]`：

位置 i	$A[i]$	前缀异或 $s[i]$	前缀最优 $L[i]$	后缀最优 $R[i]$	枚举分界 $L[i]+R[i+1]$
1	1	1	1（$[1,1]$）	3（$[1,2]$）	$L[1]+R[2]=1+3=4$
2	2	3	3（$[1,2]$）	3（$[4,5]$）	$L[2]+R[3]=3+3=\mathbf{6}$
3	3	0	3（不变）	3（$[4,5]$）	$L[3]+R[4]=3+3=\mathbf{6}$
4	1	1	3（不变）	3（$[4,5]$）	$L[4]+R[5]=3+2=5$
5	2	3	3（不变）	2（$[5,5]$）	终止：最大和为 **6**

本页要点

注意 $L[i]$ 记的是“$[1..i]$ 内的最大子段”，所以 $L[i]=\max(L[i-1],\text{tr.max_xor}(s[i]))$，要和前一位取 max！

例 2 | Codechef REBXOR：核心步骤表与关键细节

按顺序计算	纸上得到	核对问题
1. 清空树，先插 0	树含前缀 `s[0] = 0`	漏插 0 会发生什么？
2. 从左到右求 $L[1..N]$	前缀最大单段异或	更新 $L[i]$ 时跟 $L[i-1]$ 取 max 了吗？
3. 清空树，再插 0	准备算后缀异或	两遍之间有没有清空 0-1 Trie？
4. 从右到左求 $R[1..N]$	后缀最大单段异或	枚举 $i = 1..N-1$ 求 $L[i] + R[i+1]$

为什么要先往 0-1 Trie 里插入 0？

因为子段 $[1..r]$ 的异或和等于 $s[r] \oplus s[0] = s[r] \oplus 0$！如果树里没有先插 0，从第 1 个元素开始的子段就永远选不到了！

本页要点

用前缀异或 + 0-1 Trie 求最大子段异或和时，必须先插入 0！

例 2 | Codechef REBXOR：核心代码片段

```
BinaryTrie tr; tr.insert(0);                          // 必须先插入前缀和 0!
for (int i = 1, cur = 0; i <= n; ++i) {
    cur ^= a[i]; L[i] = max(L[i - 1], tr.max_xor(cur)); tr.insert(cur);
}
tr.reset(); tr.insert(0);                             // 清空树，准备从右往左求 R[i]
for (int i = n, cur = 0; i >= 1; --i) {
    cur ^= a[i]; R[i] = max(R[i + 1], tr.max_xor(cur)); tr.insert(cur);
}
long long ans = 0;
for (int i = 1; i < n; ++i) ans = max(ans, 0LL + L[i] + R[i + 1]);
```

本页要点

注意 $N = 4 \times 10^5$，两遍共插入 8×10^5 个数，记得写 `tr.reset()` 把节点清空重用省内存！

例 2 | Codechef REBXOR：易错点与 3 组自测边界

常见错误对照

- 漏插 0：遇到整个最优子段就是 $[1..r]$ 的情况会算出错误答案（**WA**）。
- 忘记写 `L[i] = max(L[i-1], ...)`：把 $L[i]$ 写成“必须以 i 结尾”而非“前 i 个里的最大”（**WA**）。
- 用 `int` 算 $L[i]+R[i+1]$ 溢出：每段最大接近 $2^{31}-1$，两段相加会超过 $2^{31}-1$！最终答案 `ans` 必须开 `long long`！

3 组自测边界

1. 最小 $N=2$：`[3, 5]` $\to 3+5=8$；　2. 溢出测试：`[2147483647, 2147483647]` $\to 4294967294$（必开 `long long`）；　3. 全零：`[0, 0, 0]` $\to 0$。

本页要点

两段 31 位整数相加可能达到 $2^{32}-2$，超过 `int` 上限，求和变量 `ans` 一定要开 `long long`！

例 3 | Keywords Search [基础]：先读懂要求

大白话题意

有多组数据。每组给你 n（$\le 10^4$）个小写英文字母单词（每个长 ≤ 50）和一篇长为 m（$\le 10^6$）的文章，问：这 n 个单词里，有多少个在文章中出现过？（同一个单词出现多次只算 1 次；若输入了两个相同的单词且都出现，算 2 次）。

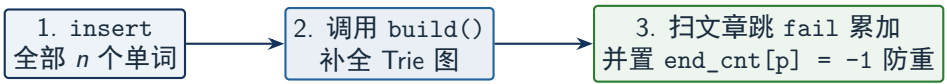
要素	本题具体要求
输入	第一行 T；每组先 n，再 n 个单词，最后一行文章串。
处理	多模式串匹配去重计数。
输出	样例单词 `she`，`he`，`say`，`shr`，`her`，文章 `yasherhs` $\rightarrow$ 出现 `she`，`he`，`her`，输出 3。

本页要点

这正是 AC 自动机的“入门第一题”：所有单词建树 + `build()` + 扫一遍文章收割答案。

例 3 | Keywords Search：怎么想到的？

- $n \le 10^4,\ m \le 10^6$，暴力对每个单词跑 KMP 是 $O(nm)$ 超时，必须把 n 个单词放进同一棵 **AC** 自动机！
- 扫文章走到节点 `u` 时，沿着 `fail` 往上跳：`for (int p = u; p && end_cnt[p] != -1; p = fail[p])`：
  - 把 `end_cnt[p]` 加进答案，然后立刻把 `end_cnt[p] = -1`！
  - 下次再有别的路径通过 `fail` 跳到 `p` 时，看到 `-1` 就直接 `break`，这样每个节点最多只会被跳到一次！



本页要点

用 `end_cnt[p] = -1` 封死已统计过的节点，既解决了“出现多次只算 1 次”，又把时间压到了严格线性 $O(m + \sum |P|)$！

例 3 | Keywords Search：样例手推跟踪表

单词：she，he，say，shr，her，文章：yasherhs（只列出关键命中步骤）：

位置	字符	当前节点	沿着 fail 收割到的词	累计 ans
$0,1$	y，a	0	无	0
$2,3$	s，h	sh	sh 与 h 的 end_cnt 为 0	0
4	'e'	she	收 she$(+1)$，跳 fail 到 he$(+1)$ 置 -1	**2**
5	'r'	her	经虚边直达 her！收 her$(+1)$ 置 -1	**3**
$6,7$	h，s	s	无新词，终止输出 3	3

本页要点

看第 5 步：she 节点并没有 'r' 儿子，但 Trie 图补全的虚边把它直接传送到了 her！

例 3 | Keywords Search：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 新建 `AC ac`	清空的自动机	多组数据时重新创建了吗？
2. 读 n 个词并 `insert`	建好 Trie，记下 `end_cnt`	重复输入同一词用 `++end_cnt` 了吗？
3. 调用 `ac.build()`	`fail` 与 Trie 图就绪	有没有只把第 1 层实儿子入队？
4. 扫文章跑 `query(T)`	沿 `fail` 累加并置 `-1`	循环条件写了 `end_cnt[p] != -1` 吗？

相对基础模板（code/ac_automaton.cpp）改了哪里？

直接复用 `code/ac_automaton.cpp` 的 `AC` 结构体，只需补一个 8 行的 `query` 成员函数！

本页要点

若词库输入两个一样的 `"he"`，`++end_cnt[u]` 会记为 2，命中一次加 2，符合题意。

例 3 | Keywords Search：核心代码（在 AC 中补全 query）

```
int query(const string &text) {                    // 接在 struct AC 内部
    int u = 0, ans = 0;
    for (char ch : text) {
        u = tr[u][ch - 'a'];                       // Trie 图一步直达，无需 while 跳
    fail
        for (int p = u; p && end_cnt[p] != -1; p = fail[p]) {
            ans += end_cnt[p]; end_cnt[p] = -1; // 封死标记，防止后续重复统计！
        }
    }
    return ans;
}
```

本页要点

如果题目问“出现过多少个不同单词”，只要把 `++end_cnt[u]` 改成 `end_cnt[u] = 1` 即可！

例 3 | Keywords Search：易错点与 3 组自测边界

常见错误对照

- 多组数据没清空 AC：把上一组的词留到下一组一起匹配（**WA**）。
- 用 `if (end_cnt[p])` 代替 `for` 沿 `fail` 跳：走到 she 时忘了跳去统计它的后缀 he（**WA**）！
- 封死条件写成 `p && end_cnt[p] > 0`：如果中间经过一个 `end_cnt == 0` 的普通节点就提前停了，会漏掉更上面的后缀词！必须写成 `end_cnt[p] != -1`。

3 组自测边界

1. 后缀嵌套：词库 {"he", "she"}，文章 "she" $\to 2$；　2. 重复词：词库 {"a", "a"}，文章 "aa" $\to 2$；　3. 无命中："abc" in "xyz" $\to 0$。

本页要点

最隐蔽的坑：`for` 条件必须写 `end_cnt[p] != -1`，不能写 `end_cnt[p] > 0`，否则遇到非结尾节点就断链了！

例 4 | Censoring（一本通 1470）[提高]：先读懂要求

大白话题意

给你一个长串 S（$|S| \le 10^5$）和 n 个屏蔽词 $t_1 \dots t_n$（$\sum |t_i| \le 10^5$）。从头扫描 S，每当遇到一个屏蔽词就把它立刻删掉；删掉后左右两半拼在一起可能新产生屏蔽词，继续删，直到没有屏蔽词为止。输出最终剩下的 S！（题目保证没有一个屏蔽词是另一个的子串）。

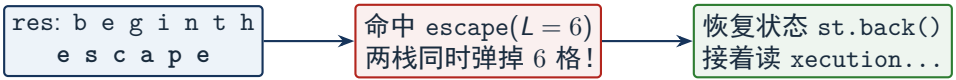
要素	本题具体要求
输入	第 1 行 S；第 2 行 n；接下来 n 行屏蔽词 t_i（均小写字母）。
处理	边扫描边删除屏蔽词，删后自动拼接继续匹配。
输出	样例 `begintheescapexecutionatthebreakofdawn`，删 `escape`，`execution` $\to$ `beginthatthebreakofdawn`。

本页要点

如果每次删掉就从头重新扫一遍，最坏删除 10^4 次就是 $O(N^2)$ 超时，必须用栈接着匹配！

例 4 | Censoring：怎么想到的？

- 删掉一段长为 L 的屏蔽词后，前面没被删掉的字符其实还在原位，我们只需要回到“这 L 个字符刚进来之前的那个自动机状态”就行！
- 所以维护两个栈（直接用 `string res` 和 `vector<int> st`）：
  - `res` 存当前保留的字符；`st` 同步记录每步走到的 AC 自动机节点编号（初始先压入 `st = {0}`）。
  - 每读入字符 `c`，从当前栈顶状态 `u = tr[st.back()][c-'a']` 走一步，把 `c` 和 `u` 压栈。
  - 如果节点 `u` 恰好是某个长为 L 的屏蔽词结尾（记作 `len[u] > 0`），就把 `res` 和 `st` 同时弹出末尾 L 格！



本页要点

因为每个字符最多入栈一次、出栈一次，总时间严格 $O(|S| + \sum |t_i|)$！

例 4 | Censoring：样例手推跟踪表

用更直观的短串演示删词拼接：主串 $S =$ "aaba"，屏蔽词 "ab"（$L = 2$）：

读入字符	res 栈	状态栈 st	命中 len[u]？	动作说明
初始	""	[0]	—	预先放入根状态 0
'a'	"a"	[0, 1]	0	走到 a
'a'	"aa"	[0, 1, 1]	0	经虚边仍停在 a(1)
'b'	"aab"	[0, 1, 1, 2]	**2**（"ab"）	弹掉最后 2 格！res="a", st=[0, 1]
'a'	"aa"	[0, 1, 1]	0	从 st.back()=1 继续走，终止输出 "aa"

本页要点

如果原串是 "aabb"、屏蔽词 "ab"：删掉中间的 "ab" 后，剩下的 'a' 和末尾的 'b' 拼成新 "ab" 也会立刻被栈顶弹掉，最终输出空串！

例 4 | Censoring：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. insert 记词长	len[u] = s.size()	原题无子串包含，不用沿 fail 传
2. 初始化 st = {0}	栈底哨兵状态 0	删空栈时 st.back() 能读到 0 吗？
3. 走一步双双入栈	res, st 同步压入	是从 st.back() 转移吗？
4. 命中 len[u] 弹栈	循环 len[u] 次 pop_back()	两个栈都一起弹了吗？

相对基础模板改了哪里？

1. end_cnt[u] 改为记录词长 len[u]；　2. 从状态栈顶 st.back() 转移并同步压栈。

本页要点

st 初始放入哨兵 0 极其重要：就算字符全删光，st.back() 依然是 0，不会空栈崩溃！

例 4 | Censoring：核心代码

```
// 建树 insert 时记录 len[u] = (int)s.size(); 并调用 build()
string res; vector<int> st{0};                    // 哨兵 0 留在栈底防空栈
for (char ch : S) {
    int u = tr[st.back()][ch - 'a'];              // 从当前栈顶状态走一步
    res.push_back(ch); st.push_back(u);
    if (len[u] > 0) {                             // 发现屏蔽词结尾，回退 len[u] 格
        for (int k = 0; k < len[u]; ++k) { res.pop_back(); st.pop_back(); }
    }
}
cout << res << '\n';
```

本页要点

代码极其简洁！res 既是字符栈也是最终要输出的答案字符串。

例 4 | Censoring：易错点与 3 组自测边界

常见错误对照

- 弹栈后把状态设成 0 而非取 `st.back()`：删完屏蔽词后丢掉了前面的前缀状态（**WA**）。
- 忘了在 `st` 初始压入 0：当开头字符被删光时，下一轮读 `st.back()` 触发空栈越界 **RE**。
- 只弹出 `res` 忘了弹出 `st`：两个栈长度错位，后续状态全乱（**WA**）。

3 组自测边界

1. 连锁反应：`"aabb"` 删 `"ab"` → 输出空串；　2. 无屏蔽词：`"abc"` 删 `"xyz"` → `"abc"`；　3. 连续删：`"abab"` 删 `"ab"` → 输出空串。

本页要点

记住“双栈同步进退 + 栈底哨兵 0”，AC 自动机删词类问题迎刃而解！

Trie 树与 AC 自动机高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
`build()` 把根 0 入队	写 `q.push(0)` 直接进 `while`	会把第 1 层儿子的 `fail` 指回自己死循环 **TLE**！必须只把 `tr[0][c]` 入队
暴力沿着 `fail` 跳到底	统计频次时每步 `while(p) ... p=fail[p]`	遇到嵌套串退化成 $O(N^2)$ **TLE**，改用 BFS 逆序拓扑倒推
Trie 节点数组开小	只开单词个数 N 当数组上限	节点上限是所有串长度总和 $\sum \lvert S_i \rvert$，用动态 `vector` 最稳
区间异或漏插 0	前缀异或 0-1 Trie 没先插入 0	漏掉从下标 1 开始的子段 **WA**
混淆“前缀”与“完整词”	走到非零节点就认为查到单词	必须检查节点结束标志 `end_cnt[u] > 0`

本页要点

“`build` 只入第 1 层，频次汇总靠逆序，0-1 Trie 先插 0，节点上限按总长”。

评测结果定位指南（CE / WA / RE / TLE）

评测结果	本章高频排查顺序
CE	检查 `array<int, 26>`、`emplace_back()` 是否在 C++17 下编译。
WA	1. 多组数据是否新建了 AC 结构体；2. 前缀异或是否漏插 0；3. REBXOR 两段相加是否开了 `long long`；4. 统计种类时 `for` 条件是否误写了 `end_cnt[p] > 0`。
RE	1. 静态 Trie 节点数是否没开到 $\sum \lvert S_i \rvert$（或 0-1 Trie 没开 $32N$）；2. Censoring 状态栈 `st` 初始是否漏放哨兵 0；3. 字符转下标是否越界（非小写字母）。
TLE	1. `build()` 是否误把根 0 压入队列导致死循环；2. 统计出现次数是否暴力跳 `fail` 没用拓扑倒推。

本页要点

遇到 **TLE** 先查 `q.push(0)` 和暴力跳 `fail`；遇到 **RE** 先查节点数组大小和栈底哨兵。

提交前检查清单（六必查）

1. AC 自动机 `build()` 时，是不是只把根节点 0 真实存在的非空儿子入队，而没有把 0 入队？
2. 是否等所有模式串全部 `insert` 完之后，才调用一次 `build()` 补全 `fail` 与 Trie 图？
3. 节点数组规模是否按所有串总长 $\sum |S_i|$（或 0-1 Trie 的 $32N$）开够，或直接使用 `vector`？
4. 0-1 Trie 求最大异或时，比特位是不是从高位 30（`long long` 用 60）倒序向 0 贪心？
5. 涉及子段异或最大（如 REBXOR）时，是否在查第 1 个前缀之前先把 0 塞进 0-1 Trie？
6. 统计每个模式串各自出现次数时，有没有用 BFS 队列顺序的逆序一次性把贡献传给 `fail[u]`？

最后一步

对 AC 自动机手造 a，aa，aaa 嵌套词匹配 "aaaa"，确认不超时不漏计后再提交！

本页要点

六条自查全过，Trie 与 AC 自动机代码就几乎没有硬伤。

本章全部练习与完成标准

题目（一本通题号）	难度	核心思路提要
The XOR Largest Pair	[基础]	边查 max_xor 边插 0-1 Trie，从第 30 位贪心反选
Keywords Search（HDU 2222）	[基础]	标准 AC 自动机，沿 fail 累加并置 -1 去重
Codechef REBXOR（1473）	[提高]	前缀异或 + 0-1 Trie 正反扫描求 $L[i], R[i]$ 枚举分界
Censoring（1470）	[提高]	AC 自动机配合字符栈与节点栈，命中后同步弹出 L 格
背单词（1477）	[提高]	单词反转建 Trie 构造后缀树，贪心按子树大小填词

完成标准

能默写 0-1 Trie 与 AC 自动机 build()；说清为何不把 0 入队及拓扑汇总；独立 AC 上述 5 题。

本页要点

先把 code/trie_xor.cpp 与 code/ac_automaton.cpp 默写一遍，再挑战提高题！

课后安排与下一篇预告

- 复习：合上讲义，再画一遍 "he"，"she"，"hers" 的 Trie 树与每层节点的 fail 指针指向。
- 练习：独立复写 code/ 下两份源码，尝试给 ac_automaton.cpp 加上主串查询计数函数。
- 下一篇预告：恭喜完成全部《07_ 字符串算法》篇！下一篇正式进入全套最后一篇《08_ 数学基础》第 1 章《快速幂、矩阵与线性递推》——看矩阵乘法怎样把 $O(n)$ 递推一口气降到 $O(\log n)$！

本章一句话

Trie 把共同前缀收束成一棵树，AC 自动机借 BFS 失配边让多串匹配一遍扫完文章绝不回头。

谢谢！