# 2-SAT & 后缀三连

王翰坤

哈尔滨工业大学

2020 年 9 月 3 日

# 目录

# 2-SAT 问题

- 给出一组 $n$ 个布尔变量 $x$ 以及 $m$ 个条件，求一组合法的解
- 每个条件都形如“$x_i$ 为真（假）或者 $x_j$ 为真（假）
- 尝试从图论角度建模、研究
- 提示：可以用拆点法

# 2-SAT 问题

- 对每个变量 $x_i$ 都拆出两个点 $2i-1$ 和 $2i$，分别代表 $x_i$ 为假和 $x_i$ 为真
- 最后，我们在 $2i-1$ 和 $2i$ 中选择一个节点标记，表示我们给 $x_i$ 的赋值
- 连边的原则是：连有向边，表示“推导出”
- 再逐个考虑每个没有赋值的变量，假定其为真，沿着我们建好的图遍历（推导）即可。若出现矛盾，则说明此 2-SAT 问题无解
- 但这个算法实践效率比较低，请思考用 SCC 相关知识优化

# 2-SAT 问题

- 一个 SCC 中，要么全满足，要么全部不满足
- 缩成 DAG 后整体赋值
- 一个巧妙的赋值方法：$x_i = true$ 等价于 $2i-1$ 所在的 SCC 的拓扑序在 $2i$ 之后
- 如果 $2i-1$ 和 $2i$ 在同一个 SCC 内，则问题无解

# 例题 I

- $n$ 个人分为 A、B、C 三组。只有年龄大于等于 $x$（$x$ 已知）的人才能分配到 A 组；只有年龄严格小于 $x$ 的人才能分配到 B 组；C 组无限制。再给出 $m$ 对相互讨厌的关系，相互讨厌的人不能分到同一组。给出每个人的年龄，找出一个合法的分配方案（或输出无解信息）
- $n, m \leq 10^5$

# 例题 I / UVa 1391

- 注意到每个人只有两种选择：做任务 A/B（$x_i = \text{true}$）或做任务 C（$x_i = \text{false}$）
- 思考如何处理讨厌关系
- 分两种情形
  - 一方大龄，一方小龄：$x_i$ 和 $x_j$ 不能同为 false，可表达为 $x_i \vee x_j$ 同为大龄人或小龄人：$x_i$ 和 $x_j$ 必须不同（异或值为 true），请思考如何表达
    - 用两个子句表达：$x_i \vee x_j$、$\overline{x_i} \vee \overline{x_j}$
- 即转化为一个 2-SAT 实例

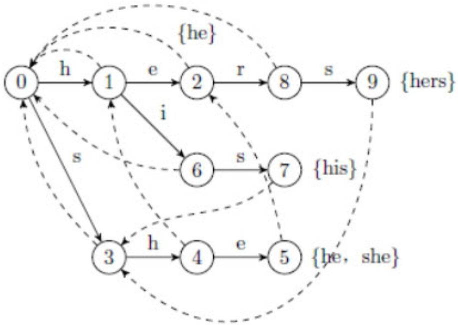
# SAT 问题

- 实际上，2-SAT 问题是 SAT 问题的一个特例
- SAT 问题不再限制一个布尔子句内的变元数量
- SAT 问题是历史上第一个被发现的 NP-Complete 问题，是一切 NPC 问题的老祖宗
- 现在，能归约到 SAT 问题上的经典 NPC 问题已经多达几百个
- 2-SAT 多项式可解是巧合吗？n-SAT 问题多大程度上是可解的？
- 事实上，不受限制的 3-SAT 问题就已经是 NP 的了
- 有兴趣的同学可以去学习计算理论的相关内容

# 目录

# 回顾 AC 自动机

- AC 自动机用于解决多模板的匹配问题
- 即已知多个模板串，查询有多少个模板串在另输入的文本串中出现

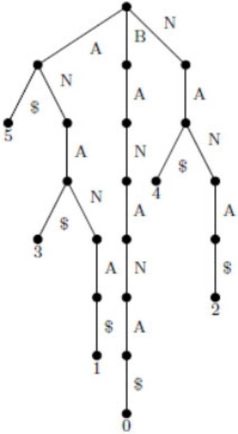


# 后缀 Trie

- 如果无法事先知道模板串，就需要先处理文本串 $S$
- 先在 $S$ 的末尾添加一个特殊字符 \$（$S$ 中未出现），以区别所有的后缀
- 再把所有后缀插入到 Trie 中

# 后缀 Trie

- 下图为 BANANA$ 对应的后缀 Trie



# 后缀 Trie

- 有了后缀 Trie，要查找某个长度为 $m$ 模板串 $P$ 是否在文本串 $S$ 中出现，只需顺着 Trie 走即可
- 时间复杂度为 $O(m)$
- 实际应用中，为提高效率，会把后缀 Trie 中的冗余链合并到一起，形成*后缀树*
- 实现比较复杂，算法竞赛一般不涉及

# 后缀数组

- 后缀数组是后缀树的替代品，好写且效率高
- 先给出一些定义和记号：
  - 字符串后缀：从字符串末尾的某个位置开始到其末尾的子串（包括空串及原串）
  - 后缀数组：将某字符串的所有后缀按字典序排序得到的数组
  - 用 $S[i...]$ 表示从位置 $i$ 开始的后缀
  - 用 $S[i,k]$ 表示从位置 $i$ 开始，长度为 $k$ 的子串（如果不足，就到末尾为止）
  - 用 $sa[i]$ 表示字典序排在第 $i$ 位的后缀的起始位置

# 后缀数组

- 用 $S[i...]$：从位置 $i$ 开始的后缀
- 用 $S[i,k]$：从位置 $i$ 开始，长度为 $k$ 的子串
- $sa[i]$：字典序排在第 $i$ 位的后缀的起始位置

"abracadabra"对应的后缀数组sa

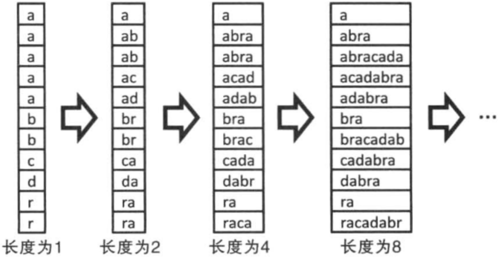
| i | sa[i] | S[sa[i]...] | i | sa[i] | S[sa[i]...] |
|---|---|---|---|---|---|
| 0 | 11 | （空字符串） | 6 | 8 | bra |
| 1 | 10 | a | 7 | 1 | bracadabra |
| 2 | 7 | abra | 8 | 4 | cadabra |
| 3 | 0 | abracadabra | 9 | 6 | dabra |
| 4 | 3 | acadabra | 10 | 9 | ra |
| 5 | 5 | adabra | 11 | 2 | racadabra |

- 如何高效地给后缀排序?
- 设字符串长度为 $n$，则暴力 $O(n^2 \log n)$

# 后缀数组

## 构造

- 倍增思想
- 先计算从每个位置开始的，长度为 1 的子串的顺序，再利用这个结果计算所有长度为 2 的子串的顺序
- 然后利用长度为 2 的子串顺序计算长度为 4 的子串顺序，依次不断倍增，直到长度大于等于 $n$




计算abracadabra的后缀数组的过程

- 如何用 $k$ 长度的子串的顺序对 $2k$ 长度的子串排序?
- 核心思想：相当于对二元组排序
- 记 $rank_k(i)$ 为 $S[i,k]$ 在 $k$ 长度的子串中的排名
- 要计算长度为 $2k$ 的子串的顺序，只要对两个 $rank$ 组成的数对进行排序即可
- 换句话说，就是通过对形如

$$(rank_k(i), rank_k(i+k))$$

的数对的比较，代替对形如 $S[i,2k]$ 的字符串的比较

# 后缀数组
## 构造

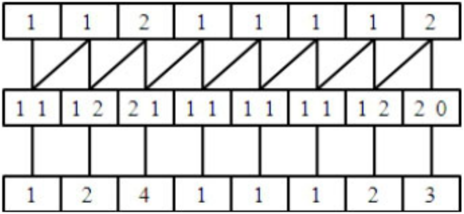
- 以 aabaaaab$ 为例，说明 SA 构造过程

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *a* | *a* | *b* | *a* | *a* | *a* | *a* | *b* |
| 1 | 1 | 2 | 1 | 1 | 1 | 1 | 2 |

# 后缀数组

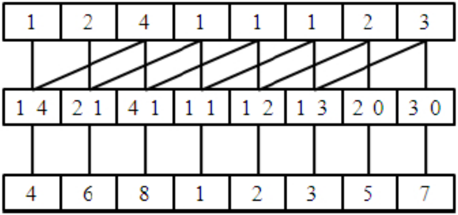
## 构造

- 从 $k = 1$ 到 $k = 2$

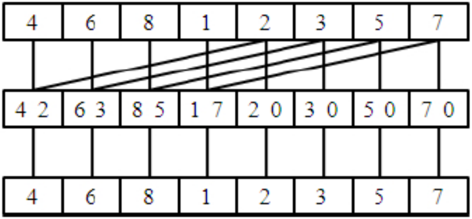


# 后缀数组
## 构造

- 从 $k=2$ 到 $k=4$



- 从 $k = 4$ 到 $k = 8$



- 时间复杂度?
- 相当于 $\log n$ 次快速排序，因此复杂度为 $O(n\log^2 n)$
- 注意到字符数最多 $n$ 种，可以考虑空间换时间
- 用基数排序代替快速排序，优化掉一个 $\log$
- 还有一些线性算法，最常用的是 DC3，可以准备一下模板

- 如何利用后缀数组实现字符串匹配?
- 二分搜索即可
- 时间复杂度为 $O(m \log n)$

```
bool contain(string S, int *sa, string T) {
  int a = 0, b = S.length();
  while (b - a > 1) {
    int c = (a + b) / 2;
    // 比较S从位置sa[c]开始长度为|T|的子串与T
    if (S.compare(sa[c], T.length(), T) < 0) a = c;
    else b = c;
  }
  return S.compare(sa[b], T.length(), T) == 0;
}
```

# 例题 II

- 给定 $n$ 个数字组成的序列 $a_1, a_2, \ldots, a_n$，其中 $a_1$ 比其他数字都大
- 现在要把这个序列分成三段，并将每段分别反转，求能得到的字典序最小的序列是什么
- 要求每段均不能为空，$n \leq 2 \times 10^5$

**输入**

```
N = 5
A = {10, 1, 2, 3, 4}
```

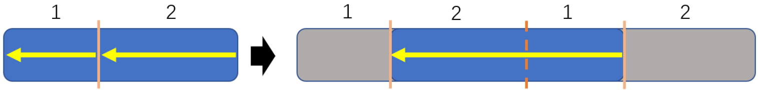
**输出**

```
1 10 2 4 3（分成{10, 1}, {2}, {3, 4}三段）
```

# 例题 II / POJ 3581

## Solution

- 因为 $a_1$ 是最大的，所以两次切割是相互独立的
- 第一个分割点相当于求反转后最小的前缀
- 这对应着原串反转后最小的后缀
- 然后将剩余部分 $S$ 再分成两段，但这次切割后两段不再独立
- 将序列分割成两段再分别反转得到的序列，可以看作两个原序列 $S$ 拼接得到的新序列 $SS$ 中的某个子串的反转
- 因此，只要计算反转后的 $SS$ 的后缀数组，再从中选取字典序最小的即可



# 后缀数组
## 高度数组

- 经常遇到「查询两个后缀的最长公共前缀」的问题
- 最长公共前缀，Longest Common Prefix (LCP)
- 引入高度数组 lcp：存储后缀数组中相邻两个后缀的 LCP 的长度
- $\mathsf{lcp}[i]$ 是后缀 $S[\mathsf{sa}[i]\ldots]$ 与 $S[\mathsf{sa}[i+1]\ldots]$ 的 LCP 长度

# 后缀数组

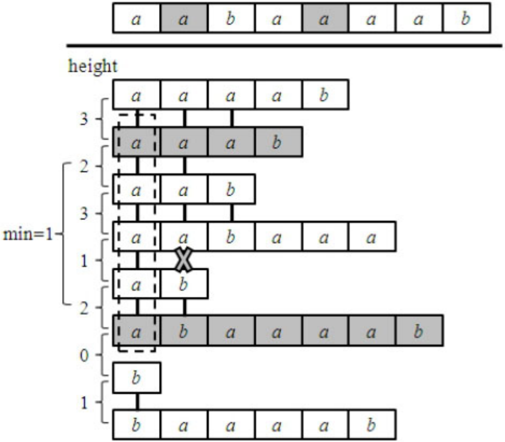
## 高度数组

- 如何利用 lcp 数组快速计算两个后缀的 lcp?
- 设辅助数组 rank 为 sa 的逆：$\mathsf{rank}[i]$ 表示后缀 $i$ 在 sa 数组中的下标（即后缀 $i$ 的排名）
- 对于两个后缀 $j$ 和 $k$，不妨设 $\mathsf{rank}[j] < \mathsf{rank}[k]$
- 则 $j$ 和 $k$ 的 LCP 就是

$$\min\{\mathsf{lcp}[\mathsf{rank}[j]], \mathsf{lcp}[\mathsf{rank}[j]+2], \ldots, \mathsf{lcp}[\mathsf{rank}[k]-1]\}$$

# 后缀数组

## 高度数组



a a b a a a a b
height
a a a a b
3
a a a b
2
a a b
3
min=1
a a b a a a
1
a b
2
a b a a a a b
0
b
1
b a a a a b

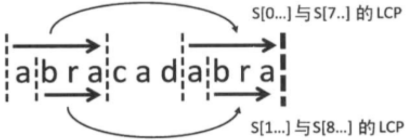
- 如何快速地求出 lcp 数组?

abracadabra所对应的后缀数组sa和高度数组lcp

| i | sa[i] | lcp[i] | S[sa[i]...] | i | sa[i] | lcp[i] | S[sa[i]...] |
|---|---|---|---|---|---|---|---|
| 0 | 11 | 0 | （空字符串） | 6 | 8 | 3 | bra |
| 1 | 10 | 1 | a | 7 | 1 | 0 | bracadabra |
| 2 | 7 | 4 | abra | 8 | 4 | 0 | cadabra |
| 3 | 0 | 1 | abracadabra | 9 | 6 | 0 | dabra |
| 4 | 3 | 1 | acadabra | 10 | 9 | 2 | ra |
| 5 | 5 | 0 | adabra | 11 | 2 | – | racadabra |

- 从位置 0 的后缀到位置 $n-1$ 的后缀，从前往后依次计算后缀 $S[i\dots]$ 和后缀 $S[sa[\text{rank}[i]-1]]\dots]$（即后缀数组中的前一个后缀）的 LCP，记为高度 $h_i$（即 $\text{lcp}[\text{rank}[i]]$）
- 重要性质：若已经求得位置 $i$ 的高度 $h_i$，则

$$h_{i+1} \geq h_i - 1$$



# 后缀数组

## 高度数组的求法

```
int rank[MAX_N + 1];

// 传入字符串S和对应的后缀数组sa，计算高度数组lcp
void construct_lcp(string S, int *sa, int *lcp) {
  int n = S.length();
  for (int i = 0; i <= n; i++) rank[sa[i]] = i;

  int h = 0;
  lcp[0] = 0;
  for (int i = 0; i < n; i++) {
    // 计算字符串中从位置i开始的后缀及其在后缀数组中的前一个后缀的LCP
    int j = sa[rank[i] - 1];

    // 将h先减去首字母的1长度，在保持前缀相同前提下不断增加
    if (h > 0) h--;
    for (; j + h < n && i + h < n; h++) {
      if (S[j + h] != S[i + h]) break;
    }

    lcp[rank[i] - 1] = h;
  }
}
```

- 复杂度如何?
- 高度最多增加 $n$ 次，所以是 $O(n)$
- 另一理解：区间 $[i, i + h_i)$ 的两个端点都始终不会左移
- 在 lcp 数组上做 RMQ，就能在 $O(1)$ 时间内求得任意两个后缀的 LCP 长度了

# 例题 III

- $T$ 组数据，每次给定一个长度为 $n$ 的只包含小写字母的串 $S$
- $q$ 个查询 $(l, r, k)$，输出子串 $S_l S_{l+1} \ldots S_r$ 第 $k$ 次出现的起始位置
- $T \leq 20$，$n, q \leq 10^5$

**Sample Input**

```
2
12 6
aaabaabaaaab
3 3 4
2 3 2
7 8 3
3 4 2
1 4 2
8 12 1
1 1
a
1 1 1
```

**Sample Output**

```
5
2
-1
6
9
8
1
```

- 首先建出后缀数组 sa
- 由 sa 性质知，以 $S_l S_{l+1} \dots S_r$ 开头的后缀一定处在连续区间
- 利用 lcp 数组上的 RMQ 二分出这个区间
- 问题就转化成求 sa 上的区间第 $k$ 大了
- 经典问题，用主席树/划分树等求解

## 温馨提示

- 大家测试模板的时候，请到UOJ 的后缀数组模板提交
- 如果只是洛谷过了可能会出事

# 目录

# 后缀树

- 重要的字符串数据结构
- 但算法竞赛中几乎不使用
- 给出一些有用的参考资料，有兴趣的同学可以自行学习
  - 后缀树
  - 后缀树系列一: 概念以及实现原理（the Ukkonen algorithm）
  - 后缀树系列二: 线性时间内构建后缀树（包含代码实现）
  - Ukkonen's Algorithm 构造后缀树实录
  - Ukkonen 算法动画
  - 算法导论：后缀树

# 目录

# 后缀自动机

- 后缀自动机（Suffix Automaton，SAM）是一个功能全面、强大的字符串数据结构
- 举个例子，以下的字符串问题都可以在线性时间内通过 SAM 解决:
  - 在一个字符串中搜索另一个字符串的所有出现位置
  - 计算给定的字符串中有多少个不同的子串
- 直观上，字符串的 SAM 可以理解为给定字符串的*所有子串*的压缩形式
- 对于一个长度为 $n$ 的字符串，其空间复杂度仅为 $O(n)$

- 字符串 $s$ 的后缀自动机 SAM 是一个接受 $s$ 的所有后缀的最小 DFA

换句话说：

- SAM 是一张有向无环图。结点被称作 **状态** ，边被称作状态间的 **转移** 。
- 图存在一个源点 $t_0$ ，称作 **初始状态** ，其它各结点均可从 $t_0$ 出发到达。
- 每个 **转移** 都标有一些字母。从一个结点出发的所有转移均 **不同** 。
- 存在一个或多个 **终止状态** 。如果我们从初始状态 $t_0$ 出发，最终转移到了一个终止状态，则路径上的所有转移连接起来一定是字符串 $s$ 的一个后缀。 $s$ 的每个后缀均可用一条从 $t_0$ 到某个终止状态的路径构成。
- 在所有满足上述条件的自动机中，SAM 的结点数是最少的。

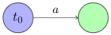
- 字符串 $s$ 的 SAM 完整且恰好包含了 $s$ 的所有子串的信息
- 从初始状态到接受状态的所有路径与 $s$ 的所有子串一一对应
- 到达某个状态的路径可能不止一条，因此我们说一个状态对应一些字符串的集合，这个集合的每个串对应这些路径

# 后缀自动机

## 例子

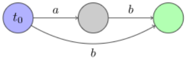
对于字符串 $s = \varnothing$：


对于字符串 $s = \mathbf{a}$：




对于字符串 $s = \mathbf{aa}$：

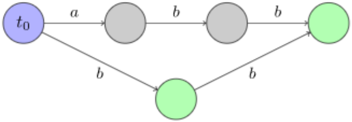

对于字符串 $s = \mathbf{ab}$：



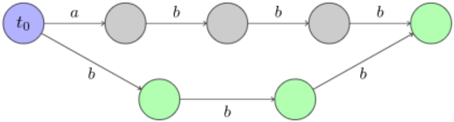
# 后缀自动机

## 例子

对于字符串 $s = \mathbf{abb}$：




对于字符串 $s = \mathbf{abbb}$：



- 推荐大家到 OI wiki 上进一步学习
- OI wiki - 后缀自动机
- 到这里看代码实现后缀自动机详解
- 还有陈立杰的课件（已发到群中）

# EOF

- 谢谢大家 $\cdot\omega\cdot$