字符串从入门到入门

ShanLunjiaJian

上海交通大学

2024-07-30

说在前面

据说很多同学没有什么字符串水平，所以今天的课会比较入门，希望大家都能入门，谢谢朋友们!!!

Outline

最小表示

manacher

dfa

- dfa 是什么
- 笛卡尔积
- dfa 最小化

kmp

- kmp
- acam

最小表示

最小表示

给串 s，找到它所有循环移位中字典序最小的那个。

最小表示

Algorithm

爆力比较。如果 i 开头比 j 开头更优，且在第 k 位才更优，那么对于 $t \leq k$，$i+t$ 开头也比 $j+t$ 开头更优。

最小表示

Algorithm

爆力比较。如果 i 开头比 j 开头更优，且在第 k 位才更优，那么对于 $t \leq k$，$i+t$ 开头也比 $j+t$ 开头更优。
跳过不优的即可。复杂度 $O(n)$。

manacher

manacher

给串 s，求出所有回文半径。

manacher

Algorithm

维护覆盖了当前位置且右端点最右的回文串，寻找和当前位置对称的点。

manacher

Algorithm

维护覆盖了当前位置且右端点最右的回文串，寻找和当前位置对称的点。
如果信息不够，爆力向右扩展。

manacher

Algorithm

维护覆盖了当前位置且右端点最右的回文串，寻找和当前位置对称的点。
如果信息不够，爆力向右扩展。
每次扩展，这个右端点最右的回文串的右端点都会向右一步，所以最多扩展 $O(n)$ 次。

dfa

dfa 是什么??

dfa 是什么???

dfa

dfa 是什么??

dfa 是什么???
dfa 是一个有向图，用来识别字符串，dfa 的点被称为状态，每条边上有一个字符，一开始你在状态 $\varnothing$，从左往右扫字符串，对于每个字符沿着对应的边走，然后有一组接受状态，如果最后到达了一个接受状态，那么这个串就在这个 dfa 识别的串的集合中，否则不在。

dfa

dfa 是什么??

dfa 是什么???

dfa 是一个有向图，用来识别字符串，dfa 的点被称为状态，每条边上有一个字符，一开始你在状态 $\varnothing$，从左往右扫字符串，对于每个字符沿着对应的边走，然后有一组接受状态，如果最后到达了一个接受状态，那么这个串就在这个 dfa 识别的串的集合中，否则不在。

dfa 的最大的好处是一个串前面的所有信息都可以被一个结点编号概括。比如我们很容易将 dfa 的一个状态作为 dp 的一维。

dfa

Example

Example

（我来）构造一个识别有奇数个 1 的 01 串的 dfa。

dfa

Example

Example

(我来）构造一个识别有奇数个 1 的 01 串的 dfa。

试看看！ 例题 1.7

你来构造一个识别不含 1 的串的 dfa。

Example

(我来）构造一个识别有奇数个 1 的 01 串的 dfa。

试看看！ 例题 1.7

你来构造一个识别不含 1 的串的 dfa。
你来构造一个识别含子序列 114514 的 dfa。

dfa

笛卡尔积

构造识别有奇数个 1 且含子序列 114514 的串的 dfa??

dfa

笛卡尔积

构造识别有奇数个 1 且含子序列 114514 的串的 dfa??
如果有两个 dfa，我们可以构造一个状态集合为两个 dfa 状态集合笛卡尔积的 dfa，它的每个状态表示" 在两个 dfa 上分别在某个状态"，转移则分别在两个 dfa 上转移。

dfa

笛卡尔积

构造识别有奇数个 1 且含子序列 114514 的串的 dfa??
如果有两个 dfa，我们可以构造一个状态集合为两个 dfa 状态集合笛卡尔积的 dfa，它的每个状态表示" 在两个 dfa 上分别在某个状态"，转移则分别在两个 dfa 上转移。
nfa 是可以有多条字符相同的出边的 dfa，也就是说走到这里时会分成多个分支同时走。最后只要一个分支到达接受状态，这个串就是被接受的。

dfa

笛卡尔积

构造识别有奇数个 1 且含子序列 114514 的串的 dfa??
如果有两个 dfa，我们可以构造一个状态集合为两个 dfa 状态集合笛卡尔积的 dfa，它的每个状态表示" 在两个 dfa 上分别在某个状态"，转移则分别在两个 dfa 上转移。
nfa 是可以有多条字符相同的出边的 dfa，也就是说走到这里时会分成多个分支同时走。最后只要一个分支到达接受状态，这个串就是被接受的。
如果给定 nfa，构造一个等价的 dfa，我们只需要，对于每个状态记录是否有一个分支位于这个状态，来作为这个 dfa 的状态。这样得到一个 2^n 个状态的 dfa，虽然很大但确实是有限的。

dfa

Example

XIX Open Cup, GP of Gomel, J. Ten Ranges

dfa

Example

XIX Open Cup, GP of Gomel, J. Ten Ranges
数 n 以内有多少个数的十进制表示不包含 2, 3, 5, 7, 11, 19, 41, 61, 89, 409, 449, 499, 881, 991, 6469, 6949, 9001, 9049, 9649, 9949, 60649, 666649, 946669, 60000049, 66000049, 66600049 中任何一个作为子序列。
$n \leq 10^{18}$。

dfa

Solution

刚才我们已经说明了如何建立" 子序列自动机"。

dfa

Solution

刚才我们已经说明了如何建立" 子序列自动机"。
现在我们对每个串建立子序列自动机，然后全部笛卡尔积起来。

dfa

Solution

刚才我们已经说明了如何建立" 子序列自动机"。
现在我们对每个串建立子序列自动机，然后全部笛卡尔积起来。
dfs 找到所有可能可达的状态，发现只有约 1.5×10^4 个，数位 dp，可以通过。

dfa

dfa 最小化

如果两个状态的所有出边都相同，那么走到它俩是完全等价的。
我们可以只保留一个，让另一个的入边改为指向这一个。

dfa

dfa 最小化

如果两个状态的所有出边都相同，那么走到它俩是完全等价的。
我们可以只保留一个，让另一个的入边改为指向这一个。
爆力做法就是直接每个状态对所有出边 hash 一下然后合并，直到不能再合并。这显然是 $O(n^2\Sigma)$ 的。

dfa

dfa 最小化

如果两个状态的所有出边都相同，那么走到它俩是完全等价的。我们可以只保留一个，让另一个的入边改为指向这一个。

爆力做法就是直接每个状态对所有出边 hash 一下然后合并，直到不能再合并。这显然是 $O(n^2\Sigma)$ 的。

我们有一个一般更快的做法。一开始让接受状态都合并成一个，拒绝状态都合并成一个，然后分裂若干轮，每次枚举一个等价类，找到它的入边是哪些原 dfa 中的结点贡献的，那么如果一个等价类里面指向这个等价类的情况不相同，就要分裂。直到这一轮 dfa 的状态数没有增大，就停止。这称为 moore 算法。

dfa

dfa 最小化

如果两个状态的所有出边都相同，那么走到它俩是完全等价的。
我们可以只保留一个，让另一个的入边改为指向这一个。
爆力做法就是直接每个状态对所有出边 hash 一下然后合并，直到不能再合并。这显然是 $O(n^2\Sigma)$ 的。
我们有一个一般更快的做法。一开始让接受状态都合并成一个，拒绝状态都合并成一个，然后分裂若干轮，每次枚举一个等价类，找到它的入边是哪些原 dfa 中的结点贡献的，那么如果一个等价类里面指向这个等价类的情况不相同，就要分裂。直到这一轮 dfa 的状态数没有增大，就停止。这称为 moore 算法。
moore 算法期望是 $O(n(\Sigma + \log n))$ 的，但可以卡到 $O(n^2\Sigma)$。

dfa

dfa 最小化

另有一个 hopcroft 算法是最坏 $O(n\Sigma \log n)$ 的。

dfa

dfa 最小化

另有一个 hopcroft 算法是最坏 $O(n\Sigma \log n)$ 的。
正如 spfa 对 bellman-ford 的优化，我们不进行若干轮，而是每次分裂之后都把分裂出的等价类放进一个队列里，每次取出第一个处理。

dfa

dfa 最小化

另有一个 hopcroft 算法是最坏 $O(n\Sigma \log n)$ 的。
正如 spfa 对 bellman-ford 的优化，我们不进行若干轮，而是每次分裂之后都把分裂出的等价类放进一个队列里，每次取出第一个处理。
关键的优化是，我们把一个等价类 x 分裂成两个 y, z 时，如果已经用 x 尝试分裂过了，那么只需要把 y, z 中较小的那个放进队列。

dfa

dfa 最小化

另有一个 hopcroft 算法是最坏 $O(n\Sigma \log n)$ 的。
正如 spfa 对 bellman-ford 的优化，我们不进行若干轮，而是每次分裂之后都把分裂出的等价类放进一个队列里，每次取出第一个处理。
关键的优化是，我们把一个等价类 x 分裂成两个 y, z 时，如果已经用 x 尝试分裂过了，那么只需要把 y, z 中较小的那个放进队列。
这是因为现有的等价类肯定不会被 x 分裂了，每个等价类到 x 的转移情况是相同的，区别就是每个点转移到 y 还是 z，所以只需要加入 y, z 中任何一个。

dfa

dfa 最小化

另有一个 hopcroft 算法是最坏 $O(n\Sigma \log n)$ 的。
正如 spfa 对 bellman-ford 的优化，我们不进行若干轮，而是每次分裂之后都把分裂出的等价类放进一个队列里，每次取出第一个处理。
关键的优化是，我们把一个等价类 x 分裂成两个 y, z 时，如果已经用 x 尝试分裂过了，那么只需要把 y, z 中较小的那个放进队列。
这是因为现有的等价类肯定不会被 x 分裂了，每个等价类到 x 的转移情况是相同的，区别就是每个点转移到 y 还是 z，所以只需要加入 y, z 中任何一个。
每条边最多被处理 log 次。复杂度 $O(n\Sigma \log n)$。不会比 moore 算法表现更差。

dfa

Solution

Review

现在我们对每个串建立子序列自动机，然后全部笛卡尔积起来。dfs 找到所有可能可达的状态，发现只有约 1.5×10^4 个，数位 dp，可以通过。

dfa

Solution

Review

现在我们对每个串建立子序列自动机，然后全部笛卡尔积起来。dfs 找到所有可能可达的状态，发现只有约 1.5×10^4 个，数位 dp，可以通过。

对 dfa 使用最小化吧!!!

dfa

Solution

Review

现在我们对每个串建立子序列自动机，然后全部笛卡尔积起来。
dfs 找到所有可能可达的状态，发现只有约 1.5×10^4 个，数位 dp，可以通过。

对 dfa 使用最小化吧!!!
最小化之后只剩下 19 个状态，可以跑 $n \leq 10^{10^5}$。

dfa

Median Replace Hard

XX Open Cup, GP of Tokyo, J. Median Replace Hard
对于一个长奇数的 01 串，称它是好的，当且仅当可以每次选择相邻三个字符按照给定的规则替换成一个（规则是所有 8 种长 3 的串到 01 的映射），使得最后剩下一个 1。给一个长奇数含 01? 的串，求把? 替换成 0/1 的所有方案中，有多少串是好的。
$n \leq 3 \times 10^5$。

dfa

Median Replace Hard

XX Open Cup, GP of Tokyo, J. Median Replace Hard
对于一个长奇数的 01 串，称它是好的，当且仅当可以每次选择相邻三个字符按照给定的规则替换成一个（规则是所有 8 种长 3 的串到 01 的映射），使得最后剩下一个 1。给一个长奇数含 01? 的串，求把? 替换成 0/1 的所有方案中，有多少串是好的。
$n \leq 3 \times 10^5$。
以下称能使得最后剩下一个 0 的是反好串。

dfa

Solution

像这种题就肯定是要建一个 dfa 并在上面 dp 了。

dfa

Solution

像这种题就肯定是要建一个 dfa 并在上面 dp 了。
我们假设已经建出了一个 dfa 识别长度 $\leq k$ 的好串，那么如何识别长度 $\leq k+2$ 的好串?

dfa

Solution

像这种题就肯定是要建一个 dfa 并在上面 dp 了。
我们假设已经建出了一个 dfa 识别长度 $\leq k$ 的好串，那么如何识别长度 $\leq k+2$ 的好串?
枚举最后一次操作的三个操作数，以及三个数分别从哪个区间上来，把所有可能的情况笛卡尔积求并，一边求一边最小化。

dfa

Solution

像这种题就肯定是要建一个 dfa 并在上面 dp 了。
我们假设已经建出了一个 dfa 识别长度 $\leq k$ 的好串，那么如何识别长度 $\leq k+2$ 的好串?
枚举最后一次操作的三个操作数，以及三个数分别从哪个区间上来，把所有可能的情况笛卡尔积求并，一边求一边最小化。（这个需要我们同时构建识别一个串是不是反好串的 dfa）

dfa

Solution

像这种题就肯定是要建一个 dfa 并在上面 dp 了。
我们假设已经建出了一个 dfa 识别长度 $\leq k$ 的好串，那么如何识别长度 $\leq k+2$ 的好串?
枚举最后一次操作的三个操作数，以及三个数分别从哪个区间上来，把所有可能的情况笛卡尔积求并，一边求一边最小化。（这个需要我们同时构建识别一个串是不是反好串的 dfa）
直到 $k, k+2$ 对应的 dfa 相同就停止。

kmp

kmp

给定串 s，构造 dfa 识别以 s 为后缀的串 t。

kmp

kmp

给定串 s，构造 dfa 识别以 s 为后缀的串 t。
注意到 dfa 识别 t 的过程，其实识别了 t 的每个前缀，所以通过这个 dfa 我们可以找到 t 中 s 的所有出现位置。

kmp

kmp

给定串 s，构造 dfa 识别以 s 为后缀的串 t。
注意到 dfa 识别 t 的过程，其实识别了 t 的每个前缀，所以通过这个 dfa 我们可以找到 t 中 s 的所有出现位置。
一个简单的想法是记录当前长 $|s|$ 的后缀作为 dfa 的状态，转移是显然的。

kmp

kmp

给定串 s，构造 dfa 识别以 s 为后缀的串 t。
注意到 dfa 识别 t 的过程，其实识别了 t 的每个前缀，所以通过这个 dfa 我们可以找到 t 中 s 的所有出现位置。
一个简单的想法是记录当前长 $|s|$ 的后缀作为 dfa 的状态，转移是显然的。这一算法复杂度是指数级的，但对于给定的 s，它构造的确实是一个 dfa。

kmp

Observation

我们希望尽可能简洁地记录当前 t 的末尾的情况。

kmp

Observation

我们希望尽可能简洁地记录当前 t 的末尾的情况。

Lemma

注意到如果两个 t 的" 最长的串 p 满足它是 t 的后缀，也是 s 的前缀" 这个信息 p 相同，那么它们的后续转移同构。

kmp

Observation

我们希望尽可能简洁地记录当前 t 的末尾的情况。

Lemma

注意到如果两个 t 的" 最长的串 p 满足它是 t 的后缀，也是 s 的前缀" 这个信息 p 相同，那么它们的后续转移同构。

Proof

若 p 之前的字符有影响，说明后面有匹配匹配到了 p 之前的字符，说明 p 不是最长的。

kmp

Observation

我们希望尽可能简洁地记录当前 t 的末尾的情况。

Lemma

注意到如果两个 t 的" 最长的串 p 满足它是 t 的后缀，也是 s 的前缀" 这个信息 p 相同，那么它们的后续转移同构。

Proof

若 p 之前的字符有影响，说明后面有匹配匹配到了 p 之前的字符，说明 p 不是最长的。

p 是 s 的前缀，所以只剩下 $n+1$ 种状态。

kmp

Algorithm

Review

p 是最长的串，满足它是 t 的后缀，也是 s 的前缀。

加入一个字符 c 时，考虑 p 如何变化。

kmp

Algorithm

Review

p 是最长的串，满足它是 t 的后缀，也是 s 的前缀。

加入一个字符 c 时，考虑 p 如何变化。

如果 pc 是 s 的前缀最好，否则寻找 qc 既是 t 的后缀，也是 s 的前缀。

kmp

Algorithm

Review

p 是最长的串，满足它是 t 的后缀，也是 s 的前缀。

加入一个字符 c 时，考虑 p 如何变化。

如果 pc 是 s 的前缀最好，否则寻找 qc 既是 t 的后缀，也是 s 的前缀。

那么 qc 是 pc 的后缀，也是 p 的前缀。

kmp

Algorithm

Review

p 是最长的串，满足它是 t 的后缀，也是 s 的前缀。

加入一个字符 c 时，考虑 p 如何变化。

如果 pc 是 s 的前缀最好，否则寻找 qc 既是 t 的后缀，也是 s 的前缀。

那么 qc 是 pc 的后缀，也是 p 的前缀。那么 q 是 p 的后缀，也是 p 的前缀。

kmp

Algorithm

Review

p 是最长的串，满足它是 t 的后缀，也是 s 的前缀。

加入一个字符 c 时，考虑 p 如何变化。

如果 pc 是 s 的前缀最好，否则寻找 qc 既是 t 的后缀，也是 s 的前缀。

那么 qc 是 pc 的后缀，也是 p 的前缀。那么 q 是 p 的后缀，也是 p 的前缀。

我们称 q 是 p 的 border。显然 border 具有传递性。

kmp

Algorithm

Review

p 是最长的串，满足它是 t 的后缀，也是 s 的前缀。

加入一个字符 c 时，考虑 p 如何变化。

如果 pc 是 s 的前缀最好，否则寻找 qc 既是 t 的后缀，也是 s 的前缀。

那么 qc 是 pc 的后缀，也是 p 的前缀。那么 q 是 p 的后缀，也是 p 的前缀。

我们称 q 是 p 的 border。显然 border 具有传递性。

只需求出 s 的每个前缀的最长真 border，就可以从 p 出发枚举 p 的所有 border q，并判断是否有 qc 是 pc 的 border。

kmp

Algorithm

现在问题是如何求每个前缀的最长真 border。

kmp

Algorithm

现在问题是如何求每个前缀的最长真 border。
与上面一样，我们向 s 后加入一个字符 c 时，枚举 s 的最长真 border q 的所有 border q，并判断是否有 qc 是 sc 的 border。

kmp

Algorithm

现在问题是如何求每个前缀的最长真 border。
与上面一样，我们向 s 后加入一个字符 c 时，枚举 s 的最长真 border q 的所有 border q，并判断是否有 qc 是 sc 的 border。
两个算法中每一步的 p 的长度最后会增加最多 1，而每次枚举的 border 变短都会让 p 的长度减小，因此枚举总共不超过 n 次。
复杂度 $O(n)$。

acam

acam

给定 trie，trie 上有一些关键点，构造 dfa 识别以 trie 上某个关键点为后缀的串 t。

acam

Algorithm

类似于 kmp，求出当前 trie 上最深的点满足它是 t 的后缀。