字符串

哈希 & 字典树

邵逸帆 *qωq*

23 电信基地班
兰州大学算法与程序设计集训队

2024 年 7 月 21 日

目录

字符类型

- char 类型
- char[] 类型 (char 数组)
- char* 类型 (char 指针)

字符串字面量

eg. `"Hello, World!"`

- type: char[14] in C, const char[14] in C++
- size: 14

Warning: 考虑这样的代码 `char* str = "Hello, World!";` 那么对于字符串的修改会导致未定义行为 (因为你在修改只读内存)。

我们可以使用字符数组来存储字符串，这很直观。

`std::string` 与 `std::vector<char>` 非常相似，只不过前者提供了更多的字符串操作函数。当然，相较于 `char[]` 更有优势。

std::string

- append() & operator+=: 字符串拼接
- find(): 查找字符串中的某个子串
- substr(): 返回字符串的子串
- operator== & < & > etc.: 字符串比较

诸如 `std::to_string()`,`std::stoi()`,`isdigit()` 等函数有时候会帮助你更优雅地处理字符串。

What is String

字符串是字符的有限序列。对于字符串 s:

- 字符串的长度: 字符串中字符的个数，记为 $|s|$。特别地，空串的长度为 0。
- 子串: $s[i, j]$ 表示从第 i 个字符到第 j 个字符的子串。
- 子序列: $s[i_1, i_2, \cdots, i_k]$ 表示从第 i_1 个字符到第 i_k 个字符的子序列。
- 前缀: $s[1, i]$ 表示从第一个字符到第 i 个字符的前缀。
- 后缀: $s[i, |s|]$ 表示从第 i 个字符到最后一个字符的后缀。
- 回文: s 是回文当且仅当 s 与 s^R 相等，其中 s^R 表示 s 的逆序。
- 字典序: 字符串的字典序是指字符串在字典中的顺序，其中空串是字典序最小的字符串。

字符串的读入

- C style: scanf,getchar,gets(C11 后被弃用)
- C++ style: std::cin,std::getline

对于单一的一个字符，也建议使用字符数组或者 std::string 读入 (因为字符串的读入会跳过空白符)。

输出

- C style: printf,putchar,puts
- C++ style: std::cout,std::endl,std::cerr

std::cerr 是无缓冲的，而 std::cout 是行缓冲的。善用这一特性，对调试有很大帮助。注意：scanf 读入 std::string 时需要提前分配内存。

值得注意的是，在 C++20 中，操作符 `operator>>(istream&, char*)` 被弃用，因为它可能导致缓冲区溢出。

但是，操作符 `operator>>(istream&in, char(&__s)[N])` 仍然是安全的，因为它会自动计算数组的大小。

请不要把数组和退化成的指针搞混。数组暗含了它的大小，而指针没有。

目录

What is Hash

哈希是一种将任意长度的输入通过散列函数变换为固定长度输出的方法。哈希函数的输出通常称为哈希值。

哈希函数

- 一致性: 对于相同的输入，哈希函数应该返回相同的哈希值。
- 高效性: 计算哈希值的时间应该尽可能短。
- 雪崩效应: 输入的微小变化应该导致输出的巨大变化。
- 抗冲突性: 不同的输入应该尽可能产生不同的哈希值。

简单来说，我们让一个非常大的值域映射到一个比较小的值域，尽可能降低冲突的概率，这就是哈希。

简单来说，哈希可以帮助我们快速判断两个数据是否相等。这一思想亦可用于字符串。

我们可以将字符串看做一个 p 进制数，比如说

$$S = s_1 \times p^{n-1} + s_2 \times p^{n-2} + \cdots + s_n \times p^0$$

其中 p 是一个质数，s_i 是字符串的第 i 个字符 (我们让每一个字符对应一个整数即可)。这样我们就可以用一个数来表示一个字符串。这样的方法有很多好处，比如说我们可以用 $O(1)$ 的时间复杂度来判断两个字符串是否相等。

不过，显然这样的方法有很多问题，比如说溢出和冲突问题。我们可以通过取模来解决溢出问题，也就是

$$f(s) = \sum_{i=1}^{len} s[i] \times p^{len-i} \mod M$$

错误率分析

假设 $f(x)$ 将 N 个字符串映射到 M 个位置上，不碰撞的概率为 $\prod_{i=0}^{n-1} \frac{M-i}{M}$
当 $N = 10^6, M = 10^9 + 7$ 时，这个值大概是 6.0×10^{-218}

这样的话几乎一定会冲突。那么要如何解决冲突?

How to avoid conflict

大素数

- 冲突容易发生的原因在于 M 选取的太小了
- 假如我们选取一个 10^{18} 数量级的素数，那么冲突的概率就会小很多
- 问题在于大素数的判断
- 我们可以使用 Miller-Rabin 算法快速判断

双模数

- 如果你不会 Miller-Rabin, 可以使用双模数
- 对两个大质数 (1e9+7,1e9+9) 分别取模，值域扩大到两者之积
- 这时正确率大概是 0.9999995000005012
- 双模数的问题在于常数比较大

How to avoid conflict

trick

- 有时候我们可以取巧, 用溢出代替取模
- 我们可以使用 unsigned long long
- 这样自然溢出即是对 2^{64} 取模
- 但是这样的话, 会有一定的错误率
- 如果出题人没卡数据, 那么这样的方法是可以接受的

思考

- 如何处理多次询问？前缀和！
- 设 $f_i(s) = f(s[1 \cdots i])$，那么我们预处理出 $f_i(s)$ 的值，是不是就可以在 $O(1)$ 的时间内求出 $f(s[l \cdots r])$ 的值了呢？

处理多次询问

- 设 $f_i(s) = f(s[1..i])$，那么我们预处理出 $f_i(s)$ 的值，是不是就可以在 $O(1)$ 的时间内求出 $f(s[l..r])$ 的值了呢？
- $f(s[1..i]) = s[1] \times b^{i-1} + s[2] \times b^{i-2} + \cdots + s[i] \times b^0$
- $f(s[l..r]) = s[l] \times b^{r-l} + s[l+1] \times b^{r-l-1} + \cdots + s[r] \times b^0$
- $f(s[l..r]) = f_r(s) - f_{l-1}(s) \times b^{r-l+1}$

所以只要预处理 b^i 就可以 $O(1)$ 查询了!

template

```
using u = unsigned long long;
u n, h[N], p[N]; char str[N]; u P = 131;
u f(int l, int r){return h[r] - h[l-1] * p[r-l+1];}
void init() {
  p[0] = 1;
  for(int i = 1; i <= n; ++ i) {
    h[i] = h[i - 1] * P + str[i];
    p[i] = p[i - 1] * P;
  }
}
```

给定一个字符串 s 和一个模式串 t，求 t 在 s 中出现的次数。

我们只需要求出 s 和 t 的哈希值，然后在 s 中枚举 t 的起点，然后用 $O(1)$ 的时间复杂度来判断是否匹配即可。

判断回文串

给定一个字符串 s，每次询问一个区间 $[l, r]$，判断 $s[l, r]$ 是否是回文串。

同时求出前缀和与后缀和。后缀和用于求出反串的哈希值，前缀和用于求出 $s[l, r]$ 的哈希值。然后判断两者是否相等即可。

最长回文子串

给定一个字符串 s，求它的最长回文子串。

二分答案做法

- 对于一个回文中心来说，显然存在单调的性质：如果长度为 x 的回文串存在，那么长度为 $x-1$ 的回文串也存在。
- 因此我们可以二分答案，然后判断是否存在长度为 x 的回文串。
- 枚举中心，从两边开始缩小。
- 需要注意的是，如果回文串长度为奇数，那么中心是一个字符，如果回文串长度为偶数，那么中心是两个字符。

$O(N)$ 哈希做法 (非 Manacher 做法)

具体方法就是记 R_i 表示以 i 作为结尾的最长回文的长度，那么答案就是 $\max_{i=1}^{n} R_i$。注意到 $R_i \leq R_{i-1} + 2$，因此我们只需要暴力从 $R_{i-1} + 2$ 开始递减，直到找到第一个回文即可。假设每次开始遍历的区间为 $[l, i]$，显然每一个位置最多只会被 l 扫描两次，因此时间复杂度为 $O(n)$。

目录

字典树，英文名 trie。顾名思义，就是一个像字典一样的树。

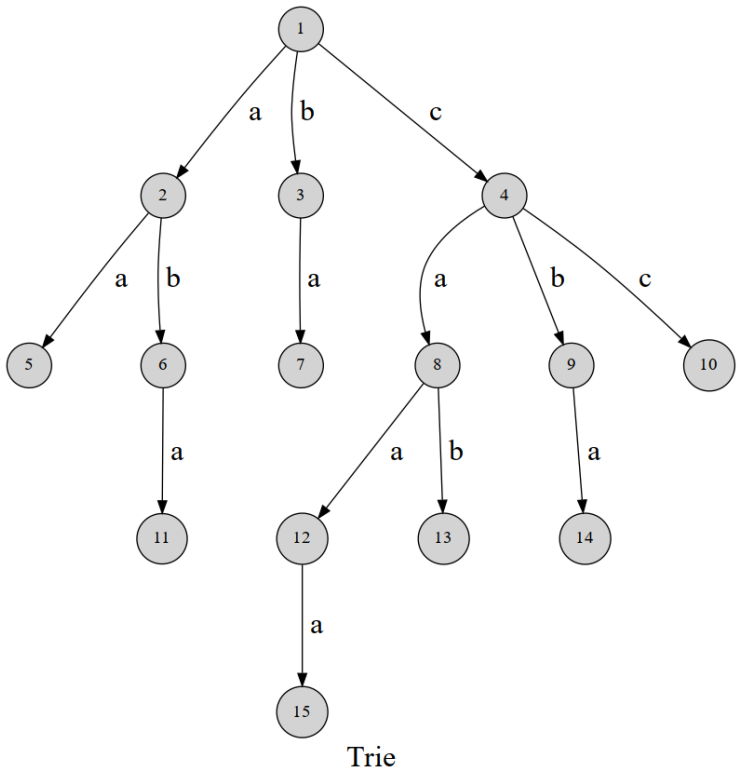

图: 字典树

```
const int M = 1e6 + 10;
// son[i][c]表示第i个节点的第c个儿子 对应字符c的边
// cnt[i]表示以第i个节点结尾的字符串数量
int son[M][26], cnt[M], idx;
void insert(char str[]) { // 插入字符串
  int p = 0;
  for (int i = 0; str[i]; i++){
    int u = str[i] - 'a';
    // 不存在子节点就新建一个
    if (!son[p][u]) son[p][u] = ++ idx;
    p = son[p][u]; // 转移到子节点
  }
  cnt[p] ++; // 以该节点结尾的字符串数量+1
}
```

```
int query(char str[]) { // 查询字符串出现次数
  int p = 0;
  for (int i = 0; str[i]; i++) {
    int u = str[i] - 'a';
    if (!son[p][u]) return 0; // 不存在
    p = son[p][u]; // 转移到子节点
  }
  return cnt[p];
}
```

检索字符串

给定一个字符集合，查询某个字符串是否在集合中出现过。

我们可以将字符串插入到字典树中，然后查询即可。

最大异或值

给定一个长度为 n 的序列，求出序列中两个数的异或值最大的异或值。

思路

我们的想法是维护一个字典树，然后对于每一个数，我们求出这个数可以与字典树中的哪一个数异或得到最大值。最后在这些答案中选取最大值即可。

查询最大异或值

- 维护一颗二进制的字典树。
- 从高位到低位依次贪心地选择最大的数。
- 如果当前位为 1，那么我们就尽量选择 0，否则选择 1。
- 如果当前位为 0，那么我们就尽量选择 1，否则选择 0。

THX 4 Listening! :)