蓝桥杯30天算法冲刺集训

Day-5 字符串Hash

Hash 表

Hash 表又称为散列表，一般由 Hash 函数（散列函数）与链表结构共同实现。与离散化思想类似，当我们要对若干复杂信息进行统计时，可以用 Hash 函数把这些复杂信息映射到一个容易维护的值域内。因为值域变简单、范围变小，有可能造成两个不同的原始信息被 Hash函映射为相同的值，所以我们需要处理这种 **冲突** 情况。有一种称“开散列”的解决方案是，建立一个邻接表结构，以 Hash 函的值域作为表头数组 head，映射后的值相同的原始信息被分到同一类，构成一个链表接在对应的表头之后，链表的节点上可以保存原始信息和一些统计数据。

Hash 表主要包括两个基本操作:

1. 计算 Hash 函数的值。
2. 定位到对应链表中依次遍历、比较。

无论是检查任意一个给定的原始信息在 Hash 表中是否存在，还是更新它在 Hash 表中的统计数据，都需要基于这两个基本操作进行。当 Hash 函数设计较好时，原始信息会被 **比较均匀地** 分配到各个表头之后，从而使每次查找、统计的时间降低到“原始信息总数除以表头数组长度”。若原始信息总数与表头数组长度都是 $O(N)$ 级别且 Hash 函数分散均匀，几乎不产生冲突，那么每次查找、统计的时间复杂度期望为 $O(1)$。

例如，我们要在一个长度为 N 的随机整数序列 A 中统计每个数出现了多少次。可以设计 Hash 函数为 $H(x)=(x \mod P)+1$ 其中 P 是个比较大的质数但不超过 N。显然，这个 Hash 函数把数列 A 分成 P 类，我们可以依次考虑数列中的每个数 $A[i]$，定位到 $head[H(A[i])]$ 这个表头所指向的链表。如果该链表中不包含 $A[i]$ 我们就在表头后插入一个新节点 $A[i]$，并在该节点上记录 $A[i]$ 出了 1 次，否则我们就直接找到已经存在的 $A[i]$ 节点将其出现次加 1。因为整数序列 A 是随机的，所以最终所有的 $A[i]$ 会比较均地分散在各个表头之后，整个算法的时间复杂度可以近

似达到 $O(N)$。

上面的例子是一个非常简单的 Hash 表的直观应用。对于非随机的数列，我们可能需要设计更好的 Hash 函数来保证其时间复度。同样地，如果我们需要维护的是比大整数复杂得多的信息的某些性质(如是否存在、出现次数等)，也可以用 Hash 来解决。字符串就是一种比较一般化的信息，在本节的后半部分，我们将会介绍一个竞赛中极其常用的字符串 Hash 算法。

字符串哈希

概念

有时，我们要对字符串进行查找，这时，可能会用到字符串哈希的技巧。

下面介绍的字符串 Hash 函数把一个任意长度的字符串映射成一个非负整数，并且其冲突概率几乎为 0 。

取一固定值 P，把字符串看作 P 进制数，并分配一个大于 0 的数值，代表每种字符。一般来说，我们分配的数值都远小于 P。例如，对于小写字母构成的字符串，可以令 $a=1, b=2, \ldots, z=26$。取一固定值 M，求出该 P 进制数对 M 的余数作为该字符串的 Hash 值。

一般来说，我们取 $P = 131$ 或 $P = 13331$ 此时 $Hash$ 值产冲突的率极低，只要 Hash 值相同，我们就可以认为原字符串是相等的。通常我们取 $M = 2^{64}$，即直接使用 unsigned long long 类型存储这个 Hash 值，在计算时不处理算术溢出问题，产生

溢出时相当于自动对 2^{64} 取，这样可以避免低效的取模(mod) 运算。

除了在极特殊构造的数据上，上述 Hash 算法很难产生冲突，一般情况下上述 Hash算法完全可以出现在题目的标准解答中。我们还可以多取一些恰当的 P 和 M 值(例如大质数)，多进行几组 Hash 运算，当结果都相同时才认为原字符串相等，就更加难

以构造出使这个 Hash 产生错误的数据。

对字符串的各种操作，都可以直接对 P 进制数进行算术运算反映到 Hash 值上。

- **构造字符串哈希**：如果我们已知字符串 S 的 Hash 为 $H(S)$，那在 S 后添加一个字符 c 构成的新字符串 $S + c$ 的 Hash 值就是 $H(S + c) = (H(S) * P + value[c]) \mod M$，其中乘 P 就相当于 P 进制下的左移运算，$value[c]$ 是我们为 c 选定的代表数值。
- **取子串哈希值**：如果我们已知字符串 S 的 Hash 值为 $H(S)$，字符串 $S + T$的 Hash 值为 $H(S + T)$，那么字符串 T 的 Hash 值 $H(T) = (H(S + T) - H(S) * P^{length(T)}) \mod M$。这就相当于通过 P 进制下在 S 后边补 0 的方式把 S 左移到与 $S + T$ 的左端对齐，然后二者相减就得到了 $H(T)$。

公式推导

例如，$S =$ `abc`，$c =$ `d`，$T =$ `xyz`，则：

S 表示 P 进制数：1 2 3

$H(S) = 1 * P^2 + 2 * P + 3$

$H(S + c) = 1 * P^3 + 2 * P^2 + 3 * P + 4 = H(S) * P + 4$

$S + T$ 表示为 P 进制数：1 2 3 24 25 26

$H(S + T) = 1 * p^5 + 2 * p^4 + 3 * p^3 + 24 * p^2 + 25 * p + 26$

S 在 P 进制下左移 $length(T)$ 位：1 2 3 0 0 0

二者相减就是 T 表示为 P 进制数：24 25 26

$H(T) = H(S + T) - (1 * P^2 + 2 * P + 3) * P^3 = 24 * P^2 + 25 * P + 26$

根据上面两种操作，我们可以通过 $O(N)$ 的时间预处理符所有前缀 Hash 值并在 $O(1)$ 的时间内查询它的任意子串的 Hash 值。

> 问题2：如果两个字符串明明不相等，但他们取模后的结果相等，这个概率有多大?
>
> 回答2：把这个问题换一种问法: 给你 n 个字符串，出现两个字符串明明不相等，他们对应的数字模 P 却相等的概率超过 50% 的可能性有多大? 这就和我们之前学过的生日悖论完全一致了！这里可以直接给出结论——当 $n > \sqrt{P\pi/2}$ 时，概率就会超过50%。也就是说，当 n 很大的时候，比如 n 是 10^6 级别的时候，模数至少要取到 10^{12} 才放心。但是在使用那么大的模数去进行运算的时候，会非常地不方便，所以一般情况下，我们直接利用 `unsigned long long` 的溢出来做取模运算，这样效率还更高。
>
> 当然，我们也可以取两个 10^9 级别的质数来代替取一个 10^{18} 级别的质数，然后分别做哈希，分别比较。只有当两种情况哈希值都相等的情况下，我们才认为原来的字符串相等。
>
> 为什么取两个 10^9 级别的质数就能代替一个 10^{18} 级别的质数呢? 这是因为，我们分别求出了两种情况对应的进制数在模意义下的哈希值，那就可以用中国剩余定理还原成一个 $0 \sim 10^{18}$ 的模意义下的取模后的值。

蓝桥杯真题

最长回文前后缀（蓝桥杯Python2023B组国赛）

本题是一道运用字符串Hash的好题。

首先需要知道的一点是，题目里面告诉我们要找一个前缀后缀能满足拼接起来是一个回文字符串，那么我们找的前缀和后缀长度是不确定的，也就是像是样例里面那样，直接是字符串本身拼接一个长度为1的后缀。

那么我们怎么去简化这个问题呢?

其实我们可以先枚举字符串前缀和后缀相等的长度，例如样例中`aababa`可以看到前后都有一个`a`是相等的，那么我们的最终答案肯定先有两个了，也就是前后的两个`a`字符。那么剩下了两个字符串一个是去掉第一个`a`的前缀字符串`ababa`一个是去掉后缀`a`的后缀字符串`aabab`，我们是不是只要在`ababa`中找到最长回文前缀然后在`aabab`中找到最长回文后缀，然后把两个长度取个最大值再加上最开始找到的首位相等的字符的长度即可。

那么我们怎么去找最长回文前缀/后缀呢?

其实有多种办法，这里用的字符串Hash，我们可以先$O(N)$时间预处理好前缀和后缀的字符串Hash，然后我们可以发现如果要找一个子串他是不是一个回文字符串，那么是不是他在他前缀中的相关位置和在他后缀中的相关位置都是相等的。

也就是如下所示，假设我们要判断`aababa`里面的`bab`是不是回文串

那么我们有两个字符串一个是前缀`aababa`一个是后缀`ababaa`是不是可以看到找到相关位置上，`bab`是相等的

那么我们直接再$O(N)$枚举查找一下就可以了，这个题目最终复杂度是$O(N)$

> [!NOTE]
>
> 下面的示例代码仅用了一个Hash，一般情况下一个Hash也可以AC大多数题目。不过保险起见，还是两个Hash为好，Hash难免发生碰撞，两个Hash也不是就能说万无一失，只是碰撞概率相当之低可以忽略为没有。

```
import sys

P, mod = 131, 1000000007
str = input()
n = len(str) * 2 + 1
s = list(' ' + str + str + ' ')
for i in range(n - 1, 0, -2):
    s[i + 1] = '{'
    s[i] = s[i // 2]
s[1] = '{'

p = [1]
h1 = [0]
h2 = [0]
for i in range(1, n + 1):
    h1.append((h1[i - 1] * P + ord(s[i]) - ord('a')) % mod)
    h2.append((h2[i - 1] * P + ord(s[n - i + 1]) - ord('a')) % mod)
    p.append(p[i - 1] * P % mod)

def get(l, r):
    return (h1[r] - h1[l - 1] * p[r - l + 1] % mod + mod) % mod
```

```
def get2(l, r):
    l, r = n - l + 1, n - r + 1
    return (h2[r] - h2[l - 1] * p[r - l + 1] % mod + mod) % mod

ans = 0
for i in range(1, n + 1):
    l, r = 0, min(i - 1, n - 1)
    while l < r:
        mid = l + r + 1 >> 1
        if get(i - mid, i - 1) != get2(i + mid, i + 1):
            r = mid - 1
        else:
            l = mid
    len = i - l - 1
    if get(1, len) == get2(n, n - len + 1):
        if s[i - l] <= 'z':
            ans = max(ans, l + 1 + len // 2 * 2)
        else:
            ans = max(ans, l + len // 2 * 2)
    len = n - (i + l)
    if get(1, len) == get2(n, n - len + 1):
        if s[i - l] <= 'z':
            ans = max(ans, l + 1 + len // 2 * 2)
        else:
            ans = max(ans, l + len // 2 * 2)
print(ans)
```

碱基（蓝桥杯C/C++2016A组国赛）

可以看到n是最多只有5的，那么我们可以直接枚举m个生物，可以用二进制来表示状态，也就是$00000 \sim 11111$这个区间范围内的二进制进行一个个的枚举，看有没有m个1然后就能表示所有的状态了。

在枚举m之前，我们还需要先把相关的长度为k的子串都在各个生物里面找到，然后再放到对应的生物的map里面，当然再来个Hash也是ok的。

那么我们还是用字符串Hash即可，最后我们需要在枚举各个生物的时候把他们的有的相同的字符子串个数都乘起来。

最终复杂度是$O(N2^NL)$

> [!NOTE]
>
> 下面的Hash用了取MOD的Hash，这也是一种Hash方式，使用上面的定义`unsigned long long`的Hash方式也是可以的。

```
import sys

n, m, k = map(int, input().split())
pw = [0] * 100010
d = [0] * 100010
l = [0] * 100010
r = [[0] * 100010 for _ in range(6)]
len_s = [0] * 6
c = {'A': 1, 'G': 2, 'C': 3, 'T': 4}
mod = 1000000007
ans = 0
```

```
s = ['#'] * 100010
mp = [{} for _ in range(100010)]
dr = {}
wl = 0

def qpow(x, p):
    ans = 1
    while p:
        if p & 1:
            ans = ans * x % mod
        x = x * x % mod
        p >>= 1
    return ans

pw[0] = 1
for i in range(1, 100001):
    pw[i] = pw[i - 1] * 5 % mod
d[100000] = qpow(pw[100000], mod - 2)
for i in range(99999, -1, -1):
    d[i] = d[i + 1] * 5 % mod

for i in range(1, n + 1):
    s[i] = input()
    len_s[i] = len(s[i])
    for j in range(1, len_s[i] + 1):
        r[i][j] = (r[i][j - 1] + c[s[i][j - 1]] * pw[j - 1] % mod) % mod
    for j in range(1, len_s[i] - k + 2):
        qwq = (r[i][j + k - 1] - r[i][j - 1] + mod) % mod * d[j - 1] % mod
        if qwq not in dr:
            dr[qwq] = 1
            wl += 1
            l[wl] = qwq
        mp[i][qwq] = mp[i].get(qwq, 0) + 1

e = 2 ** n
for i in range(1, wl + 1):
    for j in range(e):
        if bin(j).count('1') != m:
            continue
        e_val = 1
        for c in range(1, n + 1):
            if j & (1 << (c - 1)):
                e_val = e_val * mp[c].get(l[i], 0) % mod
        ans = (ans + e_val) % mod

print(ans)
```

习题

小蓝的新游戏

观察数据范围，发现 $n, Q \leq 10^6$。因此正解应是 $O(n+Q)$ 或者 $O(n \log n + Q)$ 的。

每次区间查询必须 $O(1)$ 查询，因此必然预处理，由于异或运算具有 $x \otimes x = 0$ 的特性，因此考虑利用前缀异或处理。

> 问题：对于一个区间 $[l,r]$，“其上每个数字均出现偶数次” 和 “$xo[r] \otimes xo[l-1]=0$” 是否互为充要条件？

记原数组的前缀异或数组为 xo，对于区间 $[l,r]$，如果其上每个数字均出现偶数次，那么必有 $xo[r] \otimes xo[l-1]=0$。但是，我们能否就说 “$xo[r] \otimes xo[l-1]=0$” 可以推出 “区间 $[l,r]$ 上每个数字均出现偶数次” ？

答案是否定的。比如，我们可以构造一个长度为 4 的区间 $[l,r]$，其上数字 a,b,c,d 不是两两成对的（即 $a=b,c=d$ 或者 $a=c,b=d$），且满足 $a \otimes b \otimes c \otimes d=0$，那么就可以使得该区间满足 $xo[r] \otimes xo[l-1]=0$。比如：

```
8 2
7 23 5 21 7 38 5 36
1 4
5 8
```

$7 \otimes 23=16=5 \otimes 21$，$7 \otimes 38=33=5 \otimes 36$此时就会误判。

单纯的前缀异或无法解决本问题的原因：

那怎么办呢？注意到问题中的牌上的数字范围是在 $[1,n]$ 内的，在这种情况下，我们可以轻易构造类似上面的区间元素。为什么说是“轻易构造”呢？——因为异或运算是“不带进位的加法运算”。因此，在取值范围集中在 $[1,n]$ 上时，太容易找到数字满足三个、四个甚至更多个数字满足在这种“不带进位的加法运算”了。

比如，我们随机找一个数字 e，再随机找两个数字 a,c，即可构造 $b=e \otimes a$，$d=e \otimes c$ 即可满足 $a \otimes b \otimes c \otimes d=0$。

基于这个原因，我们甚至可以构造 $5,6,7,...$ 个数字构成区间，并使得区间异或为 0。

如何解决这种问题？

找到问题的症结，那么就成功了一半。此时，我们要么换别的做法，要么优化现有做法，“避免”前缀异或的这个问题。既然这样的构造依赖于异或运算的“半加”性质导致的，那么我们做一个 **哈希**，将原本的 $[1,n]$ 区间上的数字 **映射** 到一个更大的区间上，来破坏其 **半加/半减** 性质。

至于哈希函数的选择，则任意啦，唯一的要求就是映射后的区间尽可能大，来避免出现哈希冲突和异或的冲突。比如：$h(x)=2^x \bmod p$，p 为大质数即可。如果担心出现哈希冲突，多搞几个质数做哈希即可。

```python
mod = 10**9 + 7

def qpow(x, p):
    if x == 0:
        return 1
    t = qpow(x // 2, p)
    return (t * t * (2 if x & 1 else 1)) % p

n, q = map(int, input().split())
a = list(map(int, input().split()))
xo = [0] * (n + 1)

for i in range(1, n + 1):
    a[i - 1] = qpow(a[i - 1], mod)
    xo[i] = xo[i - 1] ^ a[i - 1]

for _ in range(q):
    l, r = map(int, input().split())
```

```
    print("No" if xo[r] ^ xo[l - 1] else "Yes")
```

小蓝与围墙