

## Lesson3

### 哈希表

#### 作用

- 把一个较大的数据空间映射到较小的数据空间（例如： $0 \sim 10^9 \rightarrow 0 \sim 10^5$ ）。
- 离散化是特殊的哈希方式。
- 平均时间复杂为 $O(1)$

#### 哈希函数的写法

- 取模： $x \bmod b$ （ $b$ 一般取为质数，且里2的整次幂尽可能远）

导致冲突的存在

#### 存储结构（处理冲突方式的不同）

- **开放寻址法**
  - 一维数组存储，不使用链表，经验上开到题目数据范围的2-3倍且为质数，从而降低冲突概率
  - 操作：添加，查找，删除

```
1  static int N = 200003, nul = 0x3f3f3f3f;
2
3  static int n;
4  static int[] h = new int[N];
5
6
7  //当x存在时返回x存储的位置，不存在时返回应该存储的位置
8  static int find(int x) {
9      int k = (x % N+N) % N;
10
11      while (h[k] != nul && h[k] != x) {
12          k++;
13          if (k == N) k = 0;
14      }
15
16      return k;
17  }
18
19
20 public static void main(String[] args) throws Exception {
21     // for (int i=200000; true; i++) {
22     //     int flag = 1;
23     //     for (int j=2; j<=i/j; j++)
24     //         if (i % j == 0) {
25     //             flag = 0;
26     //             break;
27     //         }
28     //     if (flag == 1) {
29     //         out.println(i);
30     //         break;
31     //     }
32     // }
```

```

33     Arrays.fill(h, null);
34     ins.nextToken(); n = (int)ins.nval;
35
36     while (n-- > 0) {
37         ins.nextToken(); String op = (String)ins.sval;
38         ins.nextToken(); int x = (int)ins.nval;
39
40         if (op.equals("I")) h[find(x)] = x;
41         else
42             if (h[find(x)] == x) out.println("Yes");
43             else out.println("No");
44     }
45
46     out.flush();
47 }

```

- 拉链（单链表）法

- 期望算法，平均情况下链的长度为1，所以时间复杂度为 $O(1)$
- 操作：添加，查找（算法题意义下），删除（通过bool数组打删除标记，不是真正删除）

```

1  static int N = 100003;
2
3  static int n;
4  static int[] h = new int[N], e = new int[N], ne = new int[N];
5  static int idx;
6
7  static void insert(int x) {
8      int k = (x % N+N) % N;
9
10     e[idx] = x; ne[idx] = h[k]; h[k] = idx++;
11 }
12
13 static boolean find(int x) {
14     int k = (x % N+N) % N;
15
16     for (int i=h[k]; i!=-1; i=ne[i])
17         if (e[i] == x) return true;
18     return false;
19 }
20
21 public static void main(String[] args) throws Exception {
22     // for (int i=100000; true; i++) {
23     //     int flag = 1;
24     //     for (int j=2; j<=i/j; j++) {
25     //         if (i % j == 0) {
26     //             flag = 0;
27     //             break;
28     //         }
29     //     }
30
31     //     if (flag == 1) {
32     //         out.println(i);
33     //         break;

```

```

34     //    }
35     // }
36     Arrays.fill(h, -1);
37     ins.nextToken(); n = (int)ins.nval;
38
39     while (n-- > 0) {
40         ins.nextToken(); String op = (String)ins.sval;
41         ins.nextToken(); int x = (int)ins.nval;
42
43         if (op.equals("I")) insert(x);
44         else
45             if (find(x)) out.println("Yes");
46             else out.println("No");
47     }
48
49     out.flush();
50 }

```

### 字符串哈希 (字符串前缀哈希法)

字符串看成是P进制数, "ABCD" = (1234)<sub>p</sub> = (1[实际使用ASCII码] \* p<sup>3</sup> + 2 \* p<sup>2</sup> + 3 \* p<sup>1</sup> + 4 \* p<sup>0</sup>) mod Q (取模映射到0~Q-1)

#### 注意事项

- 不能映射到0, 例如A->0, 则AA->0, AA...->0, 一般情况下映射空间从1开始
- 字符串哈希, 假定不存在冲突, 当p=131或者13331, Q=2<sup>64</sup>时, 在一般情况下(99.99%)不会出现冲突
- 左边是高位, 右边是低位
- 使用unsigned long long 存储所有哈希值, 溢出则相当于 mod 2<sup>64</sup>。

#### 用处:

- 利用前缀哈希算出任意一个子串的哈希值  
例如: 通过  $h[R] - h[L-1] * p^{(R-1)-(L-2)} = h[R] - h[L-1] * p^{(R-L+1)} = h[L-R]$
- 快速判断两个字符串是否相等

预处理:  $h[i] = h[i-1]*p + str[i]$  (下标需从1开始)

```

1  static int N = 100010, P = 131;
2
3  static int n, m;
4  //Java中无unsigned long long类型, 此处用long类型替代, 范围是-2^63~2^63-1, 即Q
   从2^64变为2^63。
5  static long[] h = new long[N], p = new long[N];
6
7  static long get(int l, int r) {
8      return h[r]-h[l-1]*p[r-l+1];
9  }
10
11 public static void main(String[] args) throws Exception {
12     String[] ss = inb.readLine().split(" ");
13     n = Integer.parseInt(ss[0]); m = Integer.parseInt(ss[1]);
14     String str = " "+inb.readLine(); //不能用ins.sval, 字符串长度超过缓存
15

```

```

16 //预处理h和p数组
17 p[0] = 1;
18 for (int i=1; i<=n; i++) {
19     p[i] = p[i-1]*P;
20     h[i] = h[i-1]*P+str.charAt(i);
21 }
22
23 while (m-- > 0) {
24     String[] tp = inb.readLine().split(" ");
25     int l1 = Integer.parseInt(tp[0]);
26     int r1 = Integer.parseInt(tp[1]);
27     int l2 = Integer.parseInt(tp[2]);
28     int r2 = Integer.parseInt(tp[3]);
29
30     if (get(l1, r1) == get(l2, r2)) out.println("Yes");
31     else out.println("No");
32 }
33
34 out.flush();
35 }

```

## STL

系统为某一程序分配空间时，所需时间与空间大小无关，与申请次数有关

- pair<int, int> 存储一个二元组
  - first: 第一个元素
  - second: 第二个元素
  - 支持比较运算，以first为第一关键字，以second为第二关键字（字典序）
- vector
  - 变长数组，倍增的思想，
  - size() 返回元素个数
  - clear() 清空
  - empty() 返回是否为空（queue无清空函数）
  - front() / back()
  - push\_back() / pop\_back()
  - begin() / end() 迭代器
  - []
  - erase ()
  - 支持比较运算，按字典序
- string
  - 字符串，substr(), c\_str()
  - size()/length() 字符串长度
  - empty ()
  - clear ()
- queue
  - 队列
  - push(), 队尾插入一个元素
  - front(), 返回队头元素
  - pop(), 弹出队头元素

- back () 返回队尾元素
  - size ()
  - empty ()
  - 没有clear () 函数
- priority\_queue
  - 优先队列 (堆), 默认是大根堆
  - push () 向堆中插入一个元素
  - top () 返回堆顶元素
  - pop () 弹出堆顶元素
  - 小根堆: priority\_queue<int, vector, greater> p;
- stack
  - 栈
  - push () 向栈顶插入一个元素
  - top () 返回栈顶元素
  - pop () 弹出栈顶元素
  - 无clear ()
- deque (效率较低, 比一般数组满上几倍)
  - 双端队列 (加强版vector)
  - size, empty (), clear
  - front, back,
  - push\_back, pop\_back,
  - push\_front, pop\_front
  - [] 随机寻址
  - begin/end
- set, map, multiset, multimap, 基于平衡二叉树实现 (红黑树), 动态维护有序序列
  - set/multiset
    - insert() 插入一个数  $O(1)$
    - size (), empty (), clear ()
    - begin() / end() ++, -- 返回前驱和后继, 时间复杂度是 $O(\log n)$
    - find () 查找一个数, 不存在返回end迭代器
    - count () 返回某一个数的个数 (set 0 或者 1, multiset 有几个返回几个)
    - erase ()
      - (1) 输入一个数x, 删除所有x  $O(k + \log n)$  k是x的个数
      - (2) 输入一个迭代器, 删除这个迭代器
    - **lower\_bound() / upper\_bound()**
      - lower\_bound(x) 返回大于等于x的最小的数的迭代器, 不存在返回end()
      - upper\_bound(x) 返回大于x的最小的数的迭代器
  - map/multimap
    - insert() 插入的数是一个pair
    - erase() 输入的参数是pair或者迭代器
    - find ()
    - [] 随机索引 (时间复杂度是 $O(\log n)$ )
    - **lower\_bound() / upper\_bound()**
- unordered\_set, unordered\_map, unordered\_multiset, unordered\_multimap 哈希表
  - 和上面类似, 增删改查的时间复杂度是 $O(1)$ ,
  - 缺点: 不支持lower\_bound() / upper\_bound(), 不支持迭代器的 ++, --
- bitset 压位

- `bitset<100000> bs;`
- `~, &, |, ^, >>, <<, ==, !=, []`, `count()`返回有多少个1, `any/none()`
- `any ()` 判断是否至少有一个1
- `none ()` 判断是否全为0
- `set()` 把所有位置成1
- `set(k, v)` 把第k位变成v
- `reset ()` 把所有位变成0
- `flip ()` 等价于~
- `flip (k)` 把第k位取反
- list

```
1 #include <iostream>
2 #include <cstring>
3 #include <algorithm>
4 #include <cstdio>
5 #include <vector>
6
7 using namespace std;
8
9 int main(void) {
10     vector<int> a(10, 3); //长度为10的vector,每个数都是3
11 }
```