

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  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  const int N = 200003, null = 0x3f3f3f3f;
7
8  int h[N];
9
10 //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
11 int find(int x) {
12     int k = (x%N+N)%N;
13
14     while (h[k] != null && h[k] != x) {
15         k++;
16         if (k == N) k = 0;
17     }
18
19     return k;
20 }
21
22 int main(void) {
23     // for (int i=200000; ; i++) {
24     //     bool flag = true;
25     //     for (int j=2; j*j<=i; j++)
26     //         if (i%j == 0) {
27     //             flag = false;
28     //             break;
29     //         }
30     //     if (flag) {
31     //         cout << i << endl;
32     //         break;
33     //     }
34 }
```

```

33     //     }
34     // }
35
36     int n;
37     scanf("%d", &n);
38
39     memset(h, 0x3f, sizeof h);
40
41     while (n--) {
42         int x;
43         char op[2];
44         scanf("%s%d", op, &x);
45
46         if (*op == 'I') h[find(x)] = x;
47         else {
48             if (h[find(x)] == null) puts("No");
49             else puts("Yes");
50         }
51     }
52
53     return 0;
54 }
55

```

- 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  #include <iostream>
2  #include <cstring>
3
4  using namespace std;
5
6  const int N = 100003;
7
8  int h[N], e[N], ne[N], idx;
9
10 void insert(int x) {
11     int k = (x%N+N)%N;    //哈希函数，使x映射到[0, N-1]的位置
12
13     e[idx] = x, ne[idx] = h[k], h[k] = idx++;
14 }
15
16 bool find(int x) {
17     int k = (x%N+N)%N;
18
19     for (int i=h[k]; i!=-1; i=ne[i])
20         if (e[i] == x)
21             return true;
22
23     return false;
24 }
25

```

```

26  int main(void) {
27
28      //求出离100000最近的质数，冲突概率最小
29      // for (int i=100000; ; i++) {
30      //     bool flag = true;
31      //     for (int j=2; j*j<=i; j++)
32      //         if (i % j == 0) {
33      //             flag = false;
34      //             break;
35      //         }
36
37      //     if (flag) {
38      //         cout << i;
39      //         break;
40      //     }
41      // }
42
43      int n;
44      scanf("%d", &n);
45
46      //初始化h数组，使所有元素为-1
47      memset(h, -1, sizeof h);
48
49      while (n--) {
50          int x;
51          char op[2];
52          scanf("%s%d", op, &x);
53
54          if (*op == 'I') insert(x);
55          else {
56              if (find(x)) puts("Yes");
57              else puts("No");
58          }
59      }
60
61      return 0;
62  }

```

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) c = (1[实际使用ASCII码] * p³ + 2 * p² + 3 * p¹ + 4 * p⁰) 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=2⁶⁴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 2⁶⁴。

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  #include <iostream>
2
3  using namespace std;
4
5  typedef unsigned long long ULL;
6
7  const int N = 100010, P = 131;
8
9  int n, m;
10 char str[N];
11 ULL h[N], p[N];
12
13 ULL get(int l, int r) {
14     return h[r]-h[l-1]*p[r-l+1];
15 }
16
17 int main(void) {
18     scanf("%d%d%s", &n, &m, str+1);
19
20     p[0] = 1;
21     for (int i=1; i<=n; i++) {
22         p[i] = p[i-1]*P;
23         h[i] = h[i-1]*P + str[i];
24     }
25
26     while (m--) {
27         int l1, r1, l2, r2;
28         scanf("%d%d%d%d", &l1, &r1, &l2, &r2);
29
30         if (get(l1, r1) == get(l2, r2)) puts("Yes");
31         else puts("No");
32     }
33
34
35     return 0;
36 }
```

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 (set0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 }

```