

单链表 —— 模板题 AcWing 826. 单链表

```
1 // head存储链表头，e[]存储节点的值，ne[]存储节点的next指针，idx表示当前用到了哪个节点
2 int head, e[N], ne[N], idx;
3
4 // 初始化
5 void init()
6 {
7     head = -1;
8     idx = 0;
9 }
10
11 // 在链表头插入一个数a
12 void insert(int a)
13 {
14     e[idx] = a, ne[idx] = head, head = idx ++ ;
15 }
16
17 // 将头结点删除，需要保证头结点存在
18 void remove()
19 {
20     head = ne[head];
21 }
```

双链表 —— 模板题 AcWing 827. 双链表

```
1 // e[]表示节点的值，l[]表示节点的左指针，r[]表示节点的右指针，idx表示当前用到了哪个节点
2 int e[N], l[N], r[N], idx;
3
4 // 初始化
5 void init()
6 {
7     //0是左端点，1是右端点
8     r[0] = 1, l[1] = 0;
9     idx = 2;
10 }
11
12 // 在节点a的右边插入一个数x
13 void insert(int a, int x)
14 {
15     e[idx] = x;
16     l[idx] = a, r[idx] = r[a];
17     l[r[a]] = idx, r[a] = idx ++ ;
18 }
19
20 // 删除节点a
21 void remove(int a)
22 {
23     l[r[a]] = l[a];
```

```
24     r[l[a]] = r[a];
25 }
```

栈 —— 模板题 AcWing 828. 模拟栈

```
1 // tt表示栈顶
2 int stk[N], tt = 0;
3
4 // 向栈顶插入一个数
5 stk[ ++ tt] = x;
6
7 // 从栈顶弹出一个数
8 tt -- ;
9
10 // 栈顶的值
11 stk[tt];
12
13 // 判断栈是否为空, 如果 tt > 0, 则表示不为空
14 if (tt > 0)
15 {
16
17 }
```

队列 —— 模板题 AcWing 829. 模拟队列

(1) 普通队列:

```
1 // hh 表示队头, tt表示队尾
2 int q[N], hh = 0, tt = -1;
3
4 // 向队尾插入一个数
5 q[ ++ tt] = x;
6
7 // 从队头弹出一个数
8 hh ++ ;
9
10 // 队头的值
11 q[hh];
12
13 // 判断队列是否为空, 如果 hh <= tt, 则表示不为空
14 if (hh <= tt)
15 {
16
17 }
```

(2) 循环队列

```
1 // hh 表示队头, tt表示队尾的后一个位置
2 int q[N], hh = 0, tt = 0;
3
4 // 向队尾插入一个数
5 q[tt ++ ] = x;
6 if (tt == N) tt = 0;
7
8 // 从队头弹出一个数
9 hh ++ ;
10 if (hh == N) hh = 0;
11
12 // 队头的值
13 q[hh];
14
15 // 判断队列是否为空, 如果hh != tt, 则表示不为空
16 if (hh != tt)
17 {
18
19 }
```

单调栈 —— 模板题 AcWing 830. 单调栈

```
1 常见模型: 找出每个数左边离它最近的比它大/小的数
2 int tt = 0;
3 for (int i = 1; i <= n; i ++ )
4 {
5     while (tt && check(stk[tt], i)) tt -- ;
6     stk[ ++ tt] = i;
7 }
```

单调队列 —— 模板题 AcWing 154. 滑动窗口

```
1 常见模型: 找出滑动窗口中的最大值/最小值
2 int hh = 0, tt = -1;
3 for (int i = 0; i < n; i ++ )
4 {
5     while (hh <= tt && check_out(q[hh])) hh ++ ; // 判断队头是否滑出窗口
6     while (hh <= tt && check(q[tt], i)) tt -- ;
7     q[ ++ tt] = i;
8 }
```

KMP —— 模板题 AcWing 831. KMP字符串

```
1 // s[]是长文本, p[]是模式串, n是s的长度, m是p的长度
2 求模式串的Next数组:
3 for (int i = 2, j = 0; i <= m; i ++ )
4 {
5     while (j && p[i] != p[j + 1]) j = ne[j];
6     if (p[i] == p[j + 1]) j ++ ;
7     ne[i] = j;
8 }
9
10 // 匹配
11 for (int i = 1, j = 0; i <= n; i ++ )
12 {
13     while (j && s[i] != p[j + 1]) j = ne[j];
14     if (s[i] == p[j + 1]) j ++ ;
15     if (j == m)
16     {
17         j = ne[j];
18         // 匹配成功后的逻辑
19     }
20 }
```

Trie树 —— 模板题 AcWing 835. Trie字符串统计

```
1 int son[N][26], cnt[N], idx;
2 // 0号点既是根节点, 又是空节点
3 // son[][]存储树中每个节点的子节点
4 // cnt[]存储以每个节点结尾的单词数量
5
6 // 插入一个字符串
7 void insert(char *str)
8 {
9     int p = 0;
10    for (int i = 0; str[i]; i ++ )
11    {
12        int u = str[i] - 'a';
13        if (!son[p][u]) son[p][u] = ++ idx;
14        p = son[p][u];
15    }
16    cnt[p] ++ ;
17 }
18
19 // 查询字符串出现的次数
20 int query(char *str)
21 {
22    int p = 0;
23    for (int i = 0; str[i]; i ++ )
24    {
25        int u = str[i] - 'a';
```

```

26     if (!son[p][u]) return 0;
27     p = son[p][u];
28 }
29 return cnt[p];
30 }

```

并查集 —— 模板题 AcWing 836. 合并集合, AcWing 837. 连通块中点的数量

(1) 朴素并查集:

```

1  int p[N]; //存储每个点的祖宗节点
2
3  // 返回x的祖宗节点
4  int find(int x)
5  {
6      if (p[x] != x) p[x] = find(p[x]);
7      return p[x];
8  }
9
10 // 初始化, 假定节点编号是1~n
11 for (int i = 1; i <= n; i ++ ) p[i] = i;
12
13 // 合并a和b所在的两个集合:
14 p[find(a)] = find(b);

```

(2) 维护size的并查集:

```

1  int p[N], size[N];
2  //p[]存储每个点的祖宗节点, size[]只有祖宗节点的有意义, 表示祖宗节点所在集合中的点的数量
3
4  // 返回x的祖宗节点
5  int find(int x)
6  {
7      if (p[x] != x) p[x] = find(p[x]);
8      return p[x];
9  }
10
11 // 初始化, 假定节点编号是1~n
12 for (int i = 1; i <= n; i ++ )
13 {
14     p[i] = i;
15     size[i] = 1;
16 }
17
18 // 合并a和b所在的两个集合:
19 size[find(b)] += size[find(a)];
20 p[find(a)] = find(b);

```

(3) 维护到祖宗节点距离的并查集:

```
1 int p[N], d[N];
2 //p[] 存储每个点的祖宗节点, d[x] 存储x到p[x]的距离
3
4 // 返回x的祖宗节点
5 int find(int x)
6 {
7     if (p[x] != x)
8     {
9         int u = find(p[x]);
10        d[x] += d[p[x]];
11        p[x] = u;
12    }
13    return p[x];
14 }
15
16 // 初始化, 假定节点编号是1~n
17 for (int i = 1; i <= n; i ++ )
18 {
19     p[i] = i;
20     d[i] = 0;
21 }
22
23 // 合并a和b所在的两个集合:
24 p[find(a)] = find(b);
25 d[find(a)] = distance; // 根据具体问题, 初始化find(a)的偏移量
```

堆 —— 模板题 AcWing 838. 堆排序, AcWing 839. 模拟堆

```
1 // h[N] 存储堆中的值, h[1] 是堆顶, x 的左儿子是 2x, 右儿子是 2x + 1
2 // ph[k] 存储第 k 个插入的点在堆中的位置
3 // hp[k] 存储堆中下标是 k 的点是第几个插入的
4 int h[N], ph[N], hp[N], size;
5
6 // 交换两个点, 及其映射关系
7 void heap_swap(int a, int b)
8 {
9     swap(ph[hp[a]], ph[hp[b]]);
10    swap(hp[a], hp[b]);
11    swap(h[a], h[b]);
12 }
13
14 void down(int u)
15 {
16     int t = u;
17     if (u * 2 <= size && h[u * 2] < h[t]) t = u * 2;
18     if (u * 2 + 1 <= size && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
19     if (u != t)
20     {
```

```

21     heap_swap(u, t);
22     down(t);
23 }
24 }
25
26 void up(int u)
27 {
28     while (u / 2 && h[u] < h[u / 2])
29     {
30         heap_swap(u, u / 2);
31         u >>= 1;
32     }
33 }
34
35 // O(n)建堆
36 for (int i = n / 2; i; i -- ) down(i);

```

一般哈希 —— 模板题 AcWing 840. 模拟散列表

(1) 拉链法

```

1  int h[N], e[N], ne[N], idx;
2
3  // 向哈希表中插入一个数
4  void insert(int x)
5  {
6      int k = (x % N + N) % N;
7      e[idx] = x;
8      ne[idx] = h[k];
9      h[k] = idx ++ ;
10 }
11
12 // 在哈希表中查询某个数是否存在
13 bool find(int x)
14 {
15     int k = (x % N + N) % N;
16     for (int i = h[k]; i != -1; i = ne[i])
17         if (e[i] == x)
18             return true;
19
20     return false;
21 }

```

(2) 开放寻址法

```

1  int h[N];
2
3  // 如果x在哈希表中，返回x的下标；如果x不在哈希表中，返回x应该插入的位置
4  int find(int x)
5  {
6      int t = (x % N + N) % N;
7      while (h[t] != null && h[t] != x)
8      {
9          t ++ ;
10         if (t == N) t = 0;
11     }
12     return t;
13 }

```

字符串哈希 —— 模板题 AcWing 841. 字符串哈希

核心思想：将字符串看成 P 进制数， P 的经验值是131或13331，取这两个值的冲突概率低

小技巧：取模的数用 2^{64} ，这样直接用`unsigned long long`存储，溢出的结果就是取模的结果

```

1  typedef unsigned long long ULL;
2  ULL h[N], p[N]; // h[k]存储字符串前k个字母的哈希值，p[k]存储  $P^k \bmod 2^{64}$ 
3
4  // 初始化
5  p[0] = 1;
6  for (int i = 1; i <= n; i ++ )
7  {
8      h[i] = h[i - 1] * P + str[i];
9      p[i] = p[i - 1] * P;
10 }
11
12 // 计算子串 str[l ~ r] 的哈希值
13 ULL get(int l, int r)
14 {
15     return h[r] - h[l - 1] * p[r - l + 1];
16 }

```

C++ STL简介

```

1  vector, 变长数组, 倍增的思想
2      size() 返回元素个数
3      empty() 返回是否为空
4      clear() 清空
5      front()/back()
6      push_back()/pop_back()
7      begin()/end()
8      []
9      支持比较运算, 按字典序
10

```

```
11 pair<int, int>
12     first, 第一个元素
13     second, 第二个元素
14     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)
15
16 string, 字符串
17     size()/length() 返回字符串长度
18     empty()
19     clear()
20     substr(起始下标, (子串长度)) 返回子串
21     c_str() 返回字符串所在字符数组的起始地址
22
23 queue, 队列
24     size()
25     empty()
26     push() 向队尾插入一个元素
27     front() 返回队头元素
28     back() 返回队尾元素
29     pop() 弹出队头元素
30
31 priority_queue, 优先队列, 默认是大根堆
32     size()
33     empty()
34     push() 插入一个元素
35     top() 返回堆顶元素
36     pop() 弹出堆顶元素
37     定义成小根堆的方式: priority_queue<int, vector<int>, greater<int>> q;
38
39 stack, 栈
40     size()
41     empty()
42     push() 向栈顶插入一个元素
43     top() 返回栈顶元素
44     pop() 弹出栈顶元素
45
46 deque, 双端队列
47     size()
48     empty()
49     clear()
50     front()/back()
51     push_back()/pop_back()
52     push_front()/pop_front()
53     begin()/end()
54     []
55
56 set, map, multiset, multimap, 基于平衡二叉树 (红黑树), 动态维护有序序列
57     size()
58     empty()
59     clear()
60     begin()/end()
61     ++, -- 返回前驱和后继, 时间复杂度  $O(\log n)$ 
62
```

```

63  set/multiset
64  insert()  插入一个数
65  find()   查找一个数
66  count()  返回某一个数的个数
67  erase()
68      (1) 输入是一个数x, 删除所有x   O(k + logn)
69      (2) 输入一个迭代器, 删除这个迭代器
70  lower_bound()/upper_bound()
71      lower_bound(x)  返回大于等于x的最小的数的迭代器
72      upper_bound(x)  返回大于x的最小的数的迭代器
73  map/multimap
74  insert()  插入的数是一个pair
75  erase()   输入的参数是pair或者迭代器
76  find()
77  []  注意multimap不支持此操作。 时间复杂度是 O(logn)
78  lower_bound()/upper_bound()
79
80  unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
81  和上面类似, 增删改查的时间复杂度是 O(1)
82  不支持 lower_bound()/upper_bound(), 迭代器的++, --
83
84  bitset, 压位
85  bitset<10000> s;
86  ~, &, |, ^
87  >>, <<
88  ==, !=
89  []
90
91  count()  返回有多少个1
92
93  any()  判断是否至少有一个1
94  none() 判断是否全为0
95
96  set()  把所有位置成1
97  set(k, v)  将第k位变成v
98  reset() 把所有位变成0
99  flip()  等价于~
100 flip(k) 把第k位取反

```

作者: yxc

链接: <https://www.acwing.com/blog/content/404/>

来源: AcWing

著作权归作者所有。商业转载请联系作者获得授权, 非商业转载请注明出处。