链表

Linked List

河南省实验中学信息技术组

2026 年 02 月 03 日

链表

- 链表就是一种支持在任意位置快速插入或删除元素，但是只能按照顺依次访问元素的线性表。
- 链表根据实现和功能一般分为：
  - 单向链表 (STL `forward_list`)
  - 双向链表 (STL `list`)
  - 循环链表
- 链表在实现时可以用指针或者数组来实现。
- 信息学竞赛中为了方便，一般设计双向链表。

双向链表

- 结点中需要设计 l, r 分别指向前驱和后继结点，另外设计额外两个结点 $head$ 和 $tail$ 表示链表头结点和尾结点。
- 在信息学竞赛中，链表的结点数不会超过某一个值，例如数据个数 n，所以令头结点 $head = 0$，尾结点 $tail = n + 1$，删除结点时不回收空白结点。

```
struct Node
{
    int x;  // 元素的值
    int l, r; // 前驱和后继
} a[N];

int head, tail, m;  // m 表示目前链表长度

void init()
{
    m = 0;
    head = 0, tail = n + 1;
    a[head].r = tail, a[tail].l = head;
}
```

双向链表

- 插入结点：在结点 p 后插入元素 x。

```
void insert(int p, int x)
{
    int q = ++m;
    a[q].x = x;
    a[a[p].r].l = q, a[q].r = a[p].r;
    a[q].l = p, a[p].r = q;
}
```

- 删除结点：删除结点 p。

```
void remove(int p)
{
    a[a[p].l].r = a[p].r;
    a[a[p].r].l = a[p].l;
}
```

STL list

- 在 STL 库中，定义了双向链表list容器，支持在任意位置 O(1) 的时间复杂度插入或删除元素，但是不支持数组表示法和随机访问，使用时需要引用头文件#include<list>。
- 定义一个数据元素均为整数的链表的方式为list<int> l;
- 使用list容器的方式主要使用内置函数。

函数	功能
l.empty()	判断列表是否为空
l.size()	返回链表大小
l.front()	链表头部元素
l.back()	链表尾部元素
l.push_front(x)	链表头部插入元素
l.pop_front()	删除链表头部元素
l.push_back(x)	链表尾部插入元素
l.pop_back()	删除链表尾部元素
l.remove(x)	删除所有值为 x 的结点
l.clear()	清空链表

STL list

- list容器的很多操作都是基于迭代器的。

函数	功能
l.insert(it, x)	在 it(迭代器) 前插入元素 x
l.earse(it)	删除 it 位置的结点
l.sort()	排序
l.unique()	删除重复元素 (需要先排序)
l.merge(l2)	合并链表 (都要有序)
l.reverse()	链表翻转
l.splice(it,l2)	将链表 l2 接到位置 it 后
l.splice(it,l2,f,e)	将链表 l2 的 $[f, e)$ 位置的元素接到 it 后

STL list

- 遍历链表所有元素。

```cpp
// for(auto it = l.begin(); it != l.end(); ++it)   // 需要 C++11 以上
for(list<int>::iterator it = l.begin(); it != l.end(); ++it)
{
    cout << *it << " ";      // 链表内数据类型为简单变量
    cout << *it.val << " "; // 如果链表内存储的是结构体
    cout << it->val << " "; // 如果链表内存储的是结构体
}
```

- 最好不要在遍历链表的过程中删除链表的元素，如果需要删除需要做额外处理。

```cpp
for(auto it = l.begin(); it != l.end(); ++it)   // 需要 C++11 以上
{
    if(ok) l.earse(it);  // 错误，已经删除了 it，自然无法 ++it
}

for(auto it = l.begin(); it != l.end();)   // 需要 C++11 以上
{
    if(ok) l.earse(it++);  // 自己维护迭代器 it
    else it++;
}
```

【例】邻值查找

【题目描述】
给定一个长度为 n 的序列 A，A 中的数各不相同。对于 A 中的每一个数 A_i，求：

$$\min_{1\le j<i} |A_i - A_j|$$

以及令上式取到最小值的 j（记为 P_i）。若最小值点不唯一，则选择使 A_j 较小的那个。
【输入格式】
第 1 行一个正整数 $n(n \le 10^5)$。
第 2 行 n 个整数 $A_1, A_2, \cdots, A_n(|A_i| \le 10^9)$。
【输出格式】
$n-1$ 行，每行两个空格隔开的整数。分别表示当 i 取 $2, 3, \cdots, n$ 时，对应的 $\min_{1\le j<i} |A_i - A_j|$ 和 P_i。

【例】邻值查找

【样例输入】

```
10
4 5 6 1 2 3 7 8 9 10
```

【样例输出】

```
1 1
1 2
3 1
1 4
1 5
1 3
1 7
1 8
1 9
```

【例】邻值查找

- 将序列 a 从小到大排序，然后一次串成一个链表。
- 在排序的同时，建立一个数组 b，b_i 表示原始序列中 a_i 处在链表中的哪个位置。
- 因为链表有序，所以在链表中 b_n 结点的 l 和 r 分别指的是 a_n 的前驱和后继(如果存在)，也是离 a_n 最近的两个元素，通过比较两个元素，就能求出与 a_n 最接近的值。
- 然后，删除 b_n 结点，然后考虑 b_{n-1} 的前驱和后继，依此类推可以求出与每个 a_i 最接近的值。
- 时间复杂度：$O(N \log N + N)$。

【例】邻值查找

```
for (int i = 1; i <= n; ++i) scanf("%d", &a[i].x), a[i].id = i;
sort(a + 1, a + n + 1);
a[0].r = 1; a[n + 1].l = n; // 建立链表
for (int i = 1; i <= n; ++i)
{
    b[a[i].id] = i;
    a[i].l = i - 1;
    a[i].r = i + 1;
}
for(int i = n; i >= 2; --i)
{
    int p = b[i];
    int L = a[p].l, R = a[p].r;  // 最接近的元素位置
    ans[i] = 2e9 + 10, P[i] = n + 1;
    if(R != n + 1) ans[i] = a[R].x - a[p].x, P[i] = a[R].id;
    if(L != 0 && a[p].x - a[L].x <= ans[i]) ans[i] = a[p].x - a[L].x, P[i] = a[L].id;
    del(p);
}
```

- 能否用 STL list 实现?

【例】动态中位数

【题目描述】
依次读入一个整数序列 A，每当已经读入的整数个数为奇数时，输出已读入的整数构成的序列的中位数。
【输入格式】
第 1 行一个正整数 $n(n \leq 50000$ 且 n 一定是奇数 $)$。
第 2 行 n 个整数 $A_1, A_2, \cdots, A_n$。
【输出格式】
输出有 $\frac{n+1}{2}$ 行，每行一个整数表示中位数。

【样例输入】

```
9
5 8 9 2 4 1 3 6 7
```

【样例输出】

```
5
8
5
4
5
```

【例】动态中位数

- 先将整个序列 a 读入，然后从小到大排序后插入链表。
- 在排序的同时，建立一个数组 b，b_i 表示原始序列中 a_i 处在链表中的哪个位置。
- 那么当第 n 个数输入时，中位数必然是链表中第 $p = \frac{n+1}{2}$ 个元素。
- 那么现在从链表中删除第 n 和第 $n-1$ 个数，中位数必然与 p 相邻，设第 n 和第 $n-1$ 的在链表中的位置为 c_1, c_2：
  - 如果 $c_1 \le p$ 且 $c_2 \le p$，那么删除 c_1, c_2 后，p 应当变为它的后继；
  - 如果 $c_1 \ge p$ 且 $c_2 \ge p$，那么删除 c_1, c_2 后，p 应当变为它的前驱。
- 后续依此类推，每次删除两个数字。
- 时间复杂度：$\mathrm{O}(N \log N + N)$。

【例】动态中位数

```
for (int i = 1; i <= n; ++i) scanf("%d", &a[i].x), a[i].id = i;
sort(a + 1, a + n + 1);
a[0].r = 1; a[n + 1].l = n;
for (int i = 1; i <= n; ++i)
{
    b[a[i].id] = i;
    a[i].l = i - 1;
    a[i].r = i + 1;
}
int p = (n >> 1) + 1;
for (int i = n; i >= 3; i -= 2)
{
    ans[i] = a[p].x;
    int c1 = b[i], c2 = b[i - 1];
    if(c1 <= p && c2 <= p) p = a[p].r;
    else if(c1 >= p && c2 >= p) p = a[p].l;
    del(c1), del(c2);
}
ans[1] = a[b[1]].x;
```

- 能否用 STL list 实现?

【例】小熊的果篮

【题目描述】
小熊的水果店里摆放着一排 n 个水果。每个水果只可能是苹果或桔子，从左到右依次用正整数 $1, 2, 3, \cdots, n$ 编号。连续排在一起的同一种水果称为一个“块”。小熊要把这一排水果挑到若干个果篮里，具体方法是：每次都把每一个“块”中最左边的水果同时挑出，组成一个果篮。重复这一操作，直至水果用完。注意，每次挑完一个果篮后，“块”可能会发生变化。比如两个苹果“块”之间的唯一桔子被挑走后，两个苹果“块”就变成了一个“块”。请帮小熊计算每个果篮里包含的水果。

【输入格式】
输入的第一行包含一个正整数 n，表示水果的数量。
输入的第二行包含 n 个空格分隔的整数，其中第 i 个数表示编号为 i 的水果的种类，1 代表苹果，0 代表桔子。

【输出格式】
第 i 行表示第 i 次挑出的水果组成的果篮。从小到大排序输出该果篮中所有水果的编号，每两个编号之间用一个空格分隔。

【例】小熊的果篮

【样例 1 输入】

```
12
1 1 0 0 1 1 1 0 1 1 0 0
```

【样例 1 输出】

```
1 3 5 8 9 11
2 4 6 12
7
10
```

【样例 1 解释】

所有水果一开始的情况是 $[1,1,0,0,1,1,1,0,1,1,0,0]$，一共有 6 个块。
在第一次挑水果组成果篮的过程中，编号为 $1,3,5,8,9,11$ 的水果被挑了出来。
之后剩下的水果是 $[1,0,1,1,1,0]$，一共 4 个块。
在第二次挑水果组成果篮的过程中，编号为 $2,4,6,12$ 的水果被挑了出来。
之后剩下的水果是 $[1,1]$，只有 1 个块。
在第三次挑水果组成果篮的过程中，编号为 7 的水果被挑了出来。
最后剩下的水果是 $[1]$，只有 1 个块。
在第四次挑水果组成果篮的过程中，编号为 10 的水果被挑了出来。

【例】小熊的果篮

【样例 2 输入】

```
20
1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 0 0 0 0
```

【样例 2 输出】

```
1 5 8 11 13 14 15 17
2 6 9 12 16 18
3 7 10 19
4 20
```

【数据范围与约定】

对于 10% 的数据，$n \leq 5$。

对于 30% 的数据，$n \leq 1000$。

对于 70% 的数据，$n \leq 50000$。

对于 100% 的数据，$1 \leq n \leq 2 \times 10^5$。

【例】小熊的果篮

- 将水果分块，每次只需要挑出块最左侧的水果即可，如果出现两个块水果相同就合并。
  - 首先，所有水果自己形成一个块。
  - 扫描所有的块，对于第 i 块，如果第 $i+1$ 个块与其元素相同，那么将第 $i+1$ 个块连接在第 i 个块后，删除第 $i+1$ 个块。
  - 扫描合并后所有的块，然后删除块第一个元素，如果删除后块为空，删除对应的块。
- 如何实现快速的水果块合并？双向链表。
- 每个块内的水果利用双向链表存储，所有的块也用双向链表存储。
- 时间复杂度：$\mathrm{O}(N)$。

【例】小熊的果篮

```
// 手动实现
struct Node
{
    int x;
    int l, r;
} b[N], c[3 * N];  // 大链表 小链表
// head[0] 是大链表 head[1~n] 小链表
int head[N], tail[N];
head[0] = 0, tail[0] = n + 1;
b[head[0]].r = 1, b[tail[0]].l = n;
for(int i = 1; i <= n; ++i) b[i].l = i - 1, b[i].r = i + 1;
for(int i = 1; i <= n; ++i)
{
    head[i] = i + n, tail[i] = i + 2 * n;
    c[i] = {i, head[i], tail[i]};
    c[head[i]].r = i, c[tail[i]].l = i;
    b[i].x = i;    // 第 i 个链表
}
```

【例】小熊的果篮

```
while(n)
{
    for(int x = b[head[0]].r; x != tail[0];)   // 将相同的块合并
    {
        int y = b[x].r;
        if(y == tail[0]) break;
        int bx = c[head[x]].r, by = c[head[y]].r;
        if(a[c[bx].x] == a[c[by].x]) merge(c, x, y), del(b, y); // 前后两块相同
        else x = b[x].r;
    }
    for(int x = b[head[0]].r; x != tail[0]; x = b[x].r)
    {
        int bx = c[head[x]].r;
        printf("%d ", c[bx].x);
        del(c, bx);    // 删除块头
        --n;
        if(c[head[x]].r == tail[x]) del(b, x); // 链表空 删除之
    }
    puts("");
}
```

【例】小熊的果篮

```
// STL list
list<list<int>> l;
for(int i = 1; i <= n; ++i) l.push_back({i});
while(n)
{
    for(auto x = l.begin(); x != l.end(); )    // 合并相同块
    {
        auto y = next(x);
        if(y == l.end()) break;
        if(a[x->front()] == a[y->front()])
        {
            x->splice(x->end(), *y);
            l.erase(y);
        } else ++x;
    }
    for(auto x = l.begin(); x != l.end();) // 取出水果
    {
        printf("%d ", x->front());
        x->pop_front(); --n;
        if(x->size() == 0) l.erase(x++);
        else x++;
    }
    puts("");
}
```

练习

- 邻值查找 (COGS 2853)
- 动态中位数 (COGS 3415)
- 小熊的果篮 [CSP-J 2021]COGS 3618)
- 内存分配 [NOI 1999](COGS 284)
- 数据备份 [CTSC/APIO 2007](COGS 2884)
- 种树 [国家集训队 2011](COGS 1862)
- 舞蹈课 (COGS 3296)