

## 05 | 双链表：搜索链表中节点的速度还可以更快吗？

2023-02-22 王健伟 来自北京

《快速上手C++数据结构与算法》



你好，我是王健伟。

上节课，我们学习了单链表的相关操作，我们会用它来对数据进行顺序存储，如果需要频繁增加和删除数据，同样也可以用到单链表。而它也可以衍生出好多种链表结构，双链表（也称双向链表）就是其中一种。

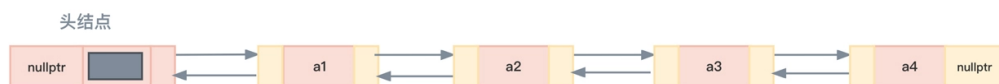
在单链表中，有一个指针域用于指向后继节点。这带来的问题是，如果要寻找单链表中某个已知节点的前趋节点，就会比较繁琐了，我们必须从链表头出发开始寻找，算法的平均情况时间复杂度为  $O(n)$ 。

那要怎么解决这个问题呢？

在单链表的基础上，我们可以增加一个用于指向前趋节点的指针，也称为前趋指针，当然，第一个节点的前趋指针指向 `nullptr`，如果是带头节点的链表，那么就是头节点的前趋指针指向

nullptr。这样，当查找某个节点的前趋节点就会非常容易，查找算法的时间复杂度也会从  $O(n)$  变为  $O(1)$ 。

这种增加了前趋指针的链表，被称为双链表。如果画得形象一点，双链表（带头节点）数据存储的描述图应该如图 9 所示：



极客时间

图9 带头节点的双链表数据存储描述图

双链表的很多操作和单链表相同，比如元素获取、求长度、判断是否为空、链表释放等操作，因为这些操作并不需要用到前趋指针。而有一些常用操作双链表与单链表不同，下面，我还是使用带头结点的代码实现方式，来实现双向链表。

## 双链表的类定义、初始化操作

我们还是先说双链表的类定义以及初始化操作。

复制代码

```
1 //双链表中每个节点的定义
2 template <typename T> //T代表数据元素的类型
3 struct DbtNode
4 {
5     T          data; //数据域，存放数据元素
6     DbtNode<T>* prior; //前趋指针，指向前一个同类型（和本节点类型相同）节点
7     DbtNode<T>* next;  //后继指针，指向下一个同类型（和本节点类型相同）节点
8 };
```

shikey.com转载分享

接着定义双链表、书写双链表构造函数的代码。

复制代码


```
1 //双链表的定义
2 template <typename T>
3 class DbtLinkList
4 {
```

```

5 public:
6     DbLinkedList();           //构造函数
7     ~DbLinkedList();         //析构函数
8
9 public:
10    bool ListInsert(int i, const T& e); //在第i个位置插入指定元素e
11    bool ListDelete(int i);           //删除第i个位置的元素
12
13    bool GetElem(int i, T& e);        //获得第i个位置的元素值
14    int LocateElem(const T& e);       //按元素值查找其在双链表中第一次出现的位置
15
16    void DisplList();                //输出双链表中的所有元素
17    int ListLength();                //获取双链表的长度
18    bool Empty();                    //判断双链表是否为空
19
20 private:
21     DbListNode<T>* m_head; //头指针（指向链表第一个节点的指针，如果链表有头结点则指向头结点）
22     int m_length;         //双链表当前长度（当前有几个元素），为编写程序更加方便和提高程序运行效率而
23 };
24
25 //通过构造函数对双链表进行初始化
26 template <typename T>
27 DbLinkedList<T>::DbLinkedList()
28 {
29     m_head = new DbListNode<T>; //先创建一个头结点
30     m_head->next = nullptr; //该值暂时为nullptr，因为还没有后继节点
31     m_head->prior = nullptr; //该值一直为nullptr
32     m_length = 0; //头结点不计入双链表的长度
33 }

```

之后，在 main 主函数中加入代码创建一个双链表对象就可以了，如果你此时编译代码，可能会遇到错误提示，只需要 [参考课件](#) 把提示缺少的代码补足即可。

 复制代码

```
1 DbLinkedList<int> sdbllinkobj;
```

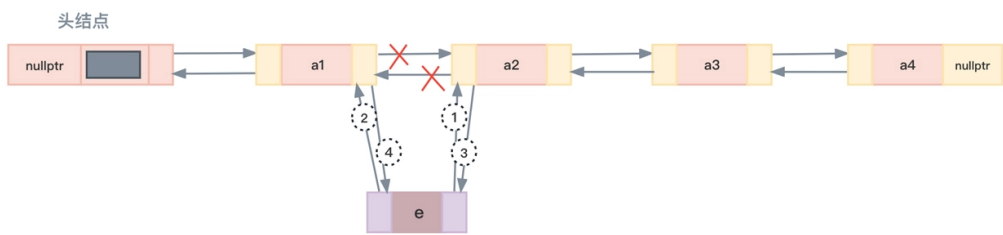
shikey.com 转载分享

## 双链表元素插入操作

双链表的元素插入操作实现代码与单链表大部分相同。值得注意的是，因为引入了前趋指针，所以必须正确设置原有节点与新插入节点的前趋指针指向。

设想一下，如果想要在 a1 和 a2 节点之间插入一个新节点 e，那么这个节点 e 的指向示意图应该是什么样的呢？

如图 10 所示：



极客时间

图10 在双链表的a1和a2节点之间插入新节点e示意图

再说实现代码。和单链表的同名实现代码 ListInsert 相比，我们需要新增几行代码来对新插入的节点 e 和 a2 节点的前趋指针赋值。

下面的代码中，几行标有数字的代码行所实现的功能与图 10 中所标记的数字一致，完整的实现代码，可以[参考课件](#)。为了防止代码执行出现错误，代码的执行顺序也要严格遵照下面的示例。


复制代码

```
1 DbtNode<T>* node = new DbtNode<T>;
2 node->data = e;
3 ①node->next = p_curr->next; //让新节点链上后续链表，因为pcurr->next指向后续的链表节点
4 ②node->prior = p_curr;
5 if(p_curr->next != nullptr)
6 ③ p_curr->next->prior = node;
7 ④p_curr->next = node; //让当前位置链上新节点，因为node指向新节点
```

shikey.com转载分享


在我们上节课讲解的单链表中，如果我们要向某个已知节点之前插入新节点，那么需要利用头指针 m\_head 从前向后找到该已知节点的前趋节点。不过，有了双链表之后，就可以直接利用该已知节点的前趋指针实现这个功能了。

你可以自行实现这个操作相关的算法代码，下面是算法命名和相关参数。

 复制代码

```
1 template<class T>
2 bool DbllinkList <T>::InsertPriorNode(DblNode<T>* pcurr, DblNode<T>* pnewnode)
3 {
4     //在节点pcurr之前插入新节点pnewnode, 请自行添加相关代码.....
5 }
```

在 main 主函数中，我们继续增加代码测试元素插入操作（如果编译代码遇到错误提示，只需要 [🔗 参考课件](#)把提示缺少的代码补足即可）。

 复制代码

```
1 sdbllinkobj.ListInsert(1, 12);
2 sdbllinkobj.ListInsert(1, 24);
3 sdbllinkobj.ListInsert(3, 48);
4 sdbllinkobj.ListInsert(2, 100);
5 sdbllinkobj.DispList();
```

新增代码的执行结果如下：

```
成功在位置为1处插入元素12!
成功在位置为1处插入元素24!
成功在位置为3处插入元素48!
成功在位置为2处插入元素100!
24 100 12 48
```

## 双链表元素删除操作

那如果是要删除双链表的第  $i$  个位置的元素呢？操作同样和单链表的相应操作类似，只是要注意前趋指针的设置。

设想一下，如果想把刚刚插入在 a1 和 a2 节点之间节点 e 删除，示意图应该怎么画呢？

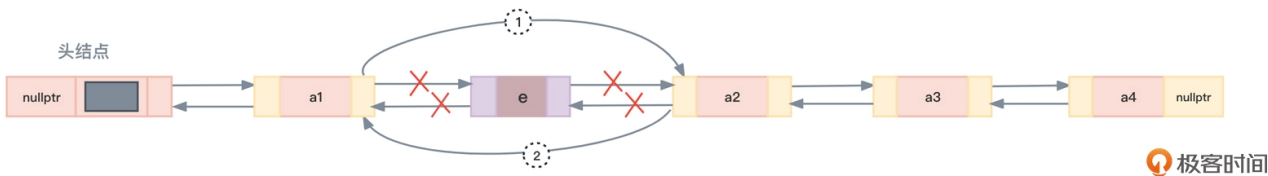


图11 删除a1和a2节点之间的节点e示意图

你会发现，我们需要做的，就是让 a1 节点的 next 指针指向 a2 节点，让 a2 节点的 prior 指针指向 a1 节点。

理解了删除操作的思路，我们再说代码。与单链表的同名实现代码 ListDelete 相比，我们需要新增一行代码来对被删除节点后继节点（a2）的前趋指针赋值。

完整的实现代码可以 [参考课件](#)，先看下核心代码。

[复制代码](#)

```
1 DbtNode<T>* p_willdel = p_curr->next; //p_willdel指向待删除的节点
2 DbtNode<T>* p_willdelNext = p_willdel->next; //p_willdelNext指向待删除节点的下一个节点
3 ①p_curr->next = p_willdel->next; //第i-1个节点的next指针指向第i+1个节点
4 if (p_willdelNext != nullptr)
5     ② p_willdelNext->prior = p_curr; //第i+1个节点的prior指针指向第i-1个节点
```

在 main 主函数中，我们继续增加代码测试元素删除操作。

[复制代码](#)

```
1 sdbllinkobj.ListDelete(4);
2 sdbllinkobj.ListDelete(2);
3 sdbllinkobj.DispList();
```

shikey.com转载分享


新增代码的执行结果如下：

成功删除位置为4的元素，该元素的值为48!

成功删除位置为2的元素，该元素的值为100!

24 12

在单链表中，如果删除某个已知节点，那么需要利用头指针 `m_head` 从前向后找到该将被删除节点的前趋节点。有了双链表之后，我们直接就可以利用该已知节点的前趋指针实现这个功能，你可以自行实现这个操作相关的算法代码。我把算法命名和相关参数也放在了这里。

 复制代码

```
1  template<class T>
2  bool DbLLinkList<T>::DeleteNode(DblNode<T>* pdel)
3  {
4      //删除pdel所指向的节点，请自行添加相关代码.....
5  }
```

双链表的翻转操作就不单独给出代码了，相信你参考单链表的翻转操作，再对比一下单链表和双链表的不同，就完全能够独立完成。这里提醒你一下，每个节点的前趋指针，是不是正确设置了呢？

## 双链表的特点

这节课的操作讲解并没有上节课多，但你一定发现了，它们都是在单链表的基础上进行了变化和改动。所以这里，我们来对照着单链表，总结一下双链表的特点，帮助你更好地理解今天的内容。

在单链表中，因为指向后继节点指针的存在，所以如果给定一个已知的节点，寻找其后继节点的时间复杂度为  $O(1)$ ，但如果寻找前趋节点，则因为每次都要从链表头出发开始寻找，所以算法的最坏情况时间复杂度为  $O(n)$ 。

而双向链表因为前趋指针的存在，寻找一个已知节点的前趋节点的时间复杂度变为了  $O(1)$ ，大大提高了寻找效率。

双链表中的某节点 `p` 前趋节点的后继指针以及后继节点的前趋指针，代表的都是 `p` 节点本身。也就是：

$$p->prior->next = p->next->prior = p$$

需要注意的是，存放前趋指针要额外消耗存储空间。

另外，我们想一想，如果在单链表或者双链表中引入一个 last 指针，让它始终指向链表的尾节点，因为有了这个 last 指针的存在，在链表的**尾部**插入数据，是不是就会非常简单呢？不过我们要注意，在插入和删除节点时要维护这个 last 指针的有效指向，此外，对于带头节点的链表，因为刚开始链表中并没有实际的数据节点，所以 last 指针也和 head 指针一样，都指向头节点。

不但如此，对于双链表，当按位置查找某个节点时，若位置不超过链表长度（m\_length）的一半，还可以利用 head 指针从左到右查找，而如果位置超过了链表的一半，就可以利用 last 指针辅助 prior 指针从右到左查找。

## 小结

这节课，我们讲解了双链表的相关内容，包括带头节点双链表的类定义及初始化操作、元素插入操作、元素删除操作等。同时还给出了双链表的特点总结。

这一系列的讲解，都是为了让我们更了解双链表的工作原理。除非有特殊需要，一般不需要自己实现单链表和双链表，因为标准模板库中已经提供了。

另外，标准模板库中的 list 容器的内部实现就是一个**双链表**。通过本节的讲解，你一定也会对 list 容器的工作原理以及优缺点有了非常清晰的认识了。想一想，如果让你自己来实现本节中所讲述的各种代码，你做得到吗？

## 归纳思考

1. 想一想单链表与双链表的区别在哪里？
2. 你能参考单链表的翻转操作代码，写出双链表的翻转操作代码吗？
3. C++ 标准模板库中的 vector 容器和 list 容器的区别有哪些（这可是面试中常被问到的问题）？list 容器和 forward\_list 容器的区别又是什么？你可以尝试总结一下。



欢迎你在留言区和我互动。如果觉得有所收获，也欢迎你把课程分享给更多的朋友一起学习进步。我们下节课见！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

## 精选留言 (1)



**Geek\_7ba740**

2023-04-08 来自四川

可以理解为读多写少用vector，写多读少用list，不确定的话用vector吗

作者回复：可以这么理解。🤔



shikey.com转载分享