数据结构和算法

作者：小甲鱼

让编程改变世界

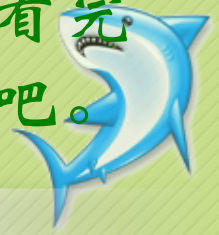
Change the world by program

双向链表

- 这货我们地球人把他称为火车，有了它，全世界每年春季最大规模的物种迁移才会发生在中国！

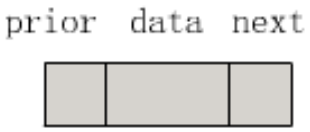
双向链表

- 大家都知道，任何事物出现的初期都显得有些不完善。例如我们的火车刚发明的时候是只有一个“头”的，所以如果它走的线路是如下：
- A->B->C->D->E->F->G->H->I->J->K->L->A
- 假设这会儿火车正停在K处呢，要他送一批货到J处，那么它将走的路线是：
- K->L->A->B->C->D->E->F->G->H->I->J
- 嗯，所以后来我们的火车就都有两个头了。看完这个例子，大家就明白双向链表的必要性了吧。

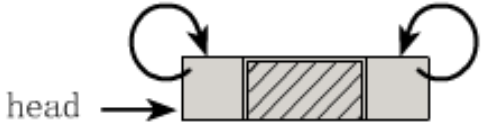


双向链表结点结构

```
typedef struct DualNode
{
    ElemType data;
    struct DualNode *prior;  //前驱结点
    struct DualNode *next;   //后继结点
} DualNode, *DuLinkList;
```



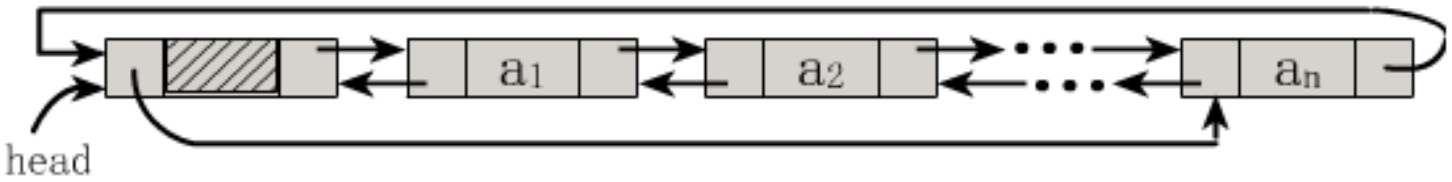
(a) 结点结构



(b) 空的双循环链表

双向链表的结点结构

- 既然单链表可以有循环链表，那么双向链表当然也可以有。

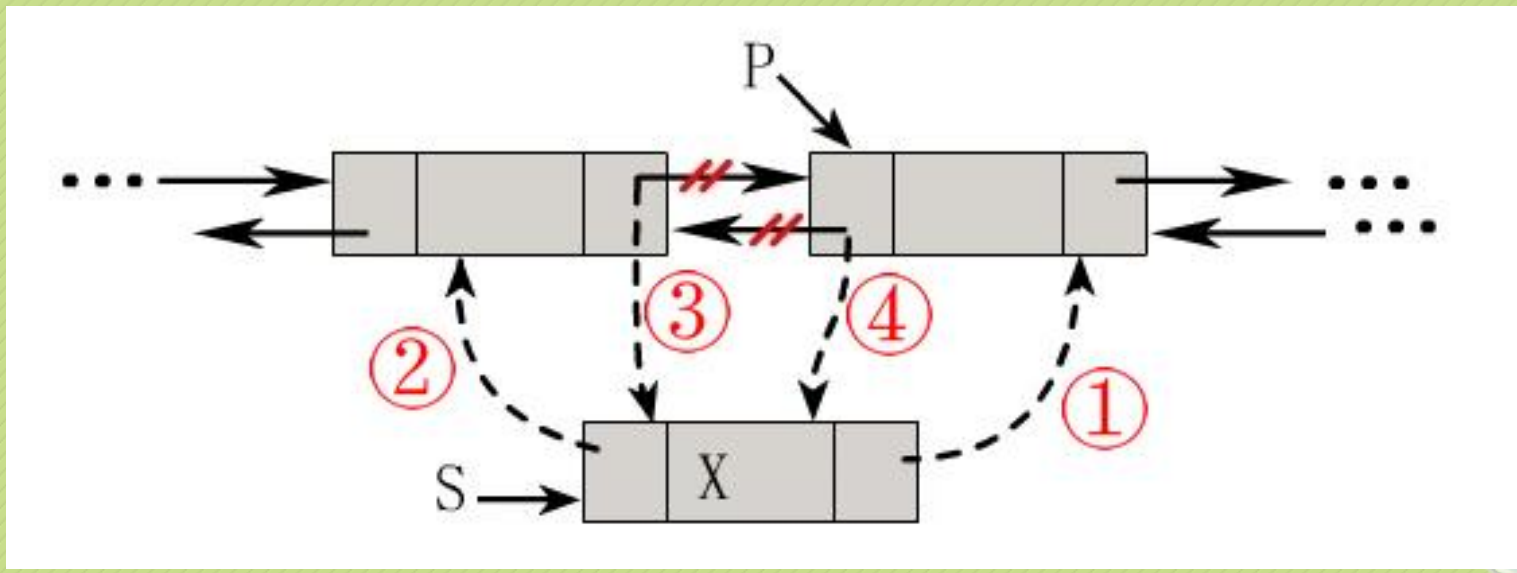


(c) 非空的双循环链表

- 在这里小甲鱼问大家一个问题：由于这是双向链表，那么对于链表中的某一个结点p，它的后继结点的前驱结点是什么？

双向链表的插入操作

- 插入操作其实并不复杂，不过顺序很重要，千万不能写反了。

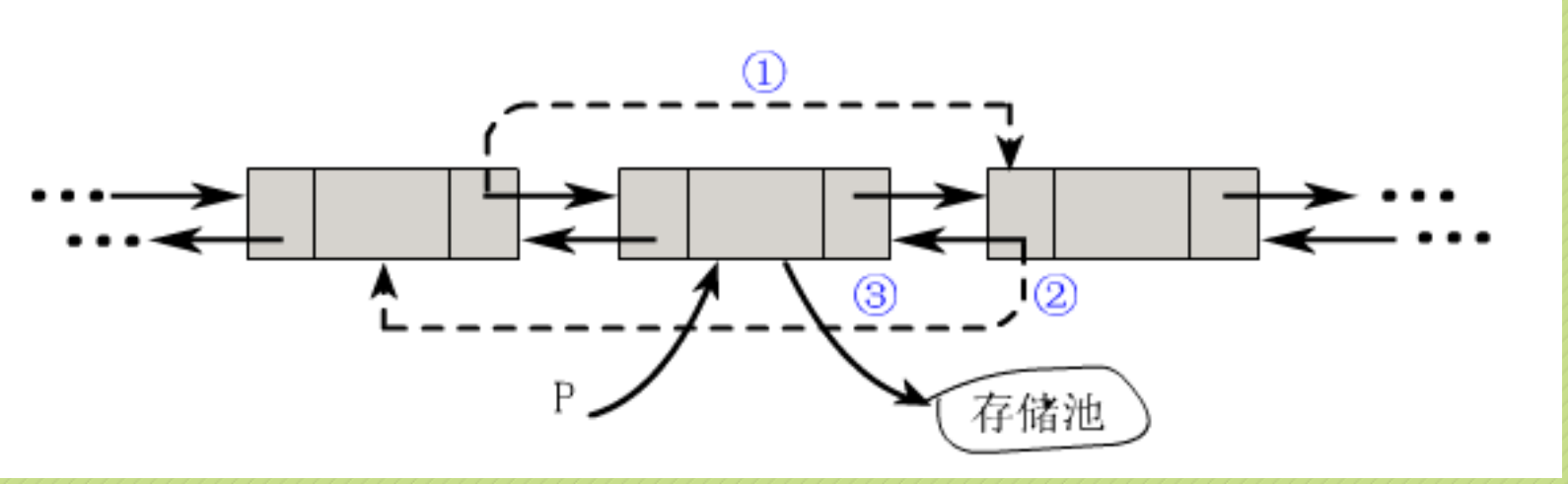


双向链表的插入操作

- 代码实现：
  - s->next = p;
  - s->prior = p->prior;
  - p->prior->next = s;
  - p->prior = s;
- 关键在于交换的过程中不要出现矛盾，例如第四步先被执行了，那么p->prior就会提前变成s，使得插入的工作出错。严重性打个比方就是打电话给老婆的时候不小心叫成小三的名字！

双向链表的删除操作

- 如果刚才的插入操作理解了，那么再来理解接下来的删除操作就容易多了。



双向链表的删除操作

- 代码实现：
  - p->prior->next = p->next;
  - p->next->prior = p->prior;
  - free(p);
- 最后总结一下，双向链表相对于单链表来说，是要更复杂一点，每个结点多了一个prior指针，对于插入和删除操作的顺序大家要格外小心。
- 不过，双向链表可以有效提高算法的时间性能，说白了就是用空间来换取时间。

本章节结束语

- 力争做一只逆流而上的小甲鱼！