**江西师范大学计算机信息工程学院学生实验报告（3）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **专业：** | **数据科学与大数据技术2班** | **姓名：** | **赖丽婷** | **学号：** |  | **日期：** | **2021.9.30** |

|  |  |  |  |
| --- | --- | --- | --- |
| 课程名称 | 数据结构 | 实验室名称 | 计算机综合实验室 |
| 实验名称 | 带头节点的单链表 | | |
| 指导教师 |  | 成绩 |  |

**1.实验目的**（结出本次实验所涉及并要求掌握的知识点）

掌握带头节点的插入，删除，倒置，排序等操作，注意一些小细节，尽量减少时间复杂度与空间复杂度

1. **实验内容**（结出实验内容具体描述）
2. 删除第一个值为x的节点
3. 倒置带头节点的单链表
4. 将x插入升序单链表中保持升序状态
5. 删除所有值为x的节点
6. 将链表升序排列
7. 将两个升序链表放入另一个链表中，保持升序状态
8. 取交集
9. 奇数在前偶数在后
10. 带头节点的插入删除倒置取交集，找倒数第k个节点，升序排列，等操作

**3.算法描述及实验步骤**（用适当的形式表达算法设计思想与算法实现步骤）

1. 前面四题跟不带头节点的删除操作差不多，只是不用考虑在头部删除的特殊情况
2. 第五题使用冒泡排序
3. 第六题利用单链表尾节点数值来判定插入结束标志
4. 第七题取交集用其中一个链表的值与另一个单链表的值做比对，找到了就存进另一个节点
5. 第八题奇偶互换，从前往后找，如果找到了偶数就往后面找奇数，找到了就互换值，这样的复杂度比较大，可以从前往后找偶数，找到了让它排到链表的最后，注意结束标志
6. 找倒数第k个节点，可以设置两个节点，先让其中一个先走k个，然后另一个与前一个保持k个间隔同步走

**4.调试过程及运行结果**（详细记录在调试过程中出现的问题及解决方法。记录实验执行的结果）

**5. 总结**（对实验结果进行分析，问题回答，实验心得体会及改进意见）

注意最后应该要释放内存

要注意特殊情况的考虑

奇偶位置调整要注意时间复杂度与空间复杂度，可以找到偶数调整到指针的末尾，但是遇到原链表的最后一个节点，就可以停止操作了

1. **附录**（程序源代码等）

1

void delx(linklist head, datatype x){

linklist p, pre;

p = head->next;

pre = head;

while(p && p->info != x){

pre = p;

p = p->next;

}

if(p){

//头节点地址将一直保存

pre->next = p->next;

free(p);

}

}

2

void reverse(linklist head){

linklist pre, p, q;

p = head->next;

pre = NULL;

while(p){

q = p;

p = p->next;

q->next = pre;

pre = q;

}

head->next = pre;

}

void reverse2(linklist \*head){

linklist pre, p, q;

pre = NULL;

p = (\*head)->next;

while(p){

q = p;

p = p->next;

q->next = pre;

pre = q;

}

(\*head)->next = pre;

}

3

void insert(linklist head, datatype x){

linklist pre, p, s;

pre = head;

p = head->next;

s = (linklist) malloc(sizeof(node));

s->info = x;

while(p && p->info < x){

pre = p;

p = p->next;

}

if(p){

s->next = pre->next;

pre->next = s;

}else if(p == NULL){

//最后这里还是要建立连接的

pre->next = s;

s->next = NULL;

}

}

4

void delallx(linklist head, datatype x){

linklist pre, p, q;

pre = head;

p = head->next;

do{

while(p&&p->info != x){

pre = p;

p = p->next;

}

if(p){

pre->next = p->next;

q = p;

p = p->next;

free(q);

}

}while(p);

}

5

void sort(linklist head){

linklist pre, p, q, s;

int temp;

pre = head;

p = head->next;

while(p){

q = pre->next;

s = q->next;

while(s){

if(s->info <= q->info){

temp = q -> info;

q -> info = s -> info;

s -> info = temp;

}

s = s->next;

}

pre = pre->next;

p = p->next;

}

}

6

linklist mergeAscend(linklist L1, linklist L2){

linklist L3;

//没有分配内存空间怎么放东西呀

L3= (linklist) malloc(sizeof(node));

L3->next = NULL;

linklist p, q, r;

q = L1->next;

r = L2->next;

p = L3;

while(r&&q){

if(q->info>r->info){

p->next = r;

p = p->next;

r = r->next;

}else{

p->next = q;

p = p->next;

q = q->next;

}

}

while(q){

p->next = q;

p = p->next;

q = q->next;

}

while(r){

p->next = r;

p = p->next;

r = r->next;

}

reverse(L3);

return L3;

}

7

linklist interSection(linklist L1, linklist L2){

linklist L3, p, q, r;

p = L1->next;

q = L2->next;

L3 = (linklist) malloc(sizeof(node));

L3->next = NULL;

r = L3;

while(p){

q = L2->next;

while(q){

if(p->info == q->info){

r->next = p;

r = r->next;

p = p->next;

break;

}else{

q = q->next;

}

}

if(!q){

p = p->next;

}

}

r->next = NULL;

return L3;

}

8

//先找到在前面的偶数

//再往后找奇数，找到则换

//保存换完的偶数位，再往后找

void partion(linklist head){

linklist p, r, q;

int k=0;

p = head->next;

while(p)

{

while(p && p->info % 2 == 1){

p = p->next;

}

q = p;

if(q){

r = q->next;

while(r && r->info % 2 == 0){

r = r->next;

}

if(r)

{

k = r->info;

r->info = q->info;

q->info = k;//这里程序写错了

}

}

if(q)

{

p = q->next;//注意这里也要进行非空判断，不然会访问到不确定的地址，导致程序终止

}

}

}

//节省时间复杂度的奇偶互换

void partion1(linklist head){

linklist pre, p, end, beend;

pre = head;

p = head->next;

while(p){

end = p;

p = p->next;

}

beend = end;

//判断是否到了愿链表的末尾

p = head -> next;

while(p && p != beend)

{

while(p && p->info % 2 != 0)

{

pre = p;

p = p->next;

}

if(p)

{

pre->next = p->next;

end->next = p;

p->next = NULL;

end = p;

p = pre->next;

}

}

}

9

linklist search(linklist head, linklist k){

linklist p, pre;

pre = p = head->next;

int count;

count = 0;

while(p && count < k){

p = p->next;

count++;

}

//要判断一下k是否大于链表长度

if(count < k){

return NULL;

}

while(p){

pre = pre->next;

p = p->next;

}

return pre;

}