

Yiyun Zhang
7/14/2016

2.2

a. array

```
8 #include <iostream>
9 using namespace std;
10
11
12 //insert
13 void insert(int x, int p, int* L){
14     int maxlen, last;
15     if(last>=maxlen){
16         cout << "list is full" << endl;
17     }
18     else if(p>last+1 || p<1){
19         cout << "position does not exits" << endl;
20     }
21     else{
22         for(int q=last;q>=p;q--){
23             L[q+1] = L[q];
24             last += 1;
25             L[p] = x;
26         }
27     }
28 }
29
30 //delete
31 void del(int p, int* L){
32     int maxlen, last;
33     if(p>=maxlen || p<0){
34         cout << "position does not exist" << endl;
35     }
36     else{
37         last -= 1;
38         for(int q=p;q<=last;q++){
39             L[q] = L[q+1];
40         }
41     }
42 }
43
44 //locate
45 int locate(int x, int* L){
46     int last;
47     for(int q=1;q<=last;q++){
48         if(L[q] == x){
49             return q;
50         }
51     }
52     return last+1;
53 }
```

Running time:

insert:O(1)

delete:O(1)

locate:O(n)

b. pointer

```
2 //  pointer.cpp
3 //  CS260
4 //
5 //  Created by Yiyun Zhang on 7/14/16.
6 //  Copyright © 2016 Yiyun Zhang. All rights reserved.
7 //
8
9 #include <iostream>
10 using namespace std;
11
12
13 struct node {
14     int element;
15     node *next;
16 };
17
18 //insert
19 void insert(int x, node *p){
20     node *temp;
21     temp = p->next;
22     p->next = new node;
23     p->next->element = x;
24     p->next->next = temp;
25 }
26
27 //delete
28 void del(node *p){
29     p->next = p->next->next;
30 }
31
32
33 //locate
34 node* locate(int x, node *L){
35     node *temp;
36     temp = L;
37     while(temp->next != NULL){
38         if(temp->next->element == x){
39             return temp;
40         }
41         else{
42             temp = temp->next;
43         }
44     }
45     return temp;
46 }
47 ...
```

Running time:

insert:O(1)

delete:O(1)

locate:O(n)

c. cursor-based

```
8 #include <iostream>
9 using namespace std;
10
11 struct node {
12     int element;
13     int *next;
14 };
15
16 bool move(int p, int q){
17     int temp;
18     node *space;
19     if(p==0){
20         cout << "cell does not exist";
21         return false;
22     }
23     else{
24         temp = q;
25         q = p;
26         p = *space[q].next;
27         space[q].next = &temp;
28         return true;
29     }
30     return false;
31 }
32
33 //insert
34 void insert(int x, int p, int* L){
35     node *space;
36     if(p==0){
37         if(move(p,*L)){
38             space[*L].element= x;
39         }
40     }
41     else{
42         if(move(p,*space[p].next)){
43             space[*space[p].next].element = x;
44         }
45     }
46 }
47
48
49 //delete
50 void del(int p, int *L){
51     node *space;
52     if(p==0){
53         move(*L,p);
54     }
55     else{
56         move(*space[p].next,p);
57     }
58 }
```

Running time:

insert: $O(n)$

delete: $O(n)$

2.4

```
concatenate.py
1 def concatenate(L):
2     newL = []
3     for x in L:
4         for a in x:
5             newL.append(a)
6     return newL
```

2.11

```
p := FIRST(L);           1
while p <> END(L) do begin
    q := p;
    while q <> END(L) do begin
        q := NEXT(q, L);          n
        r := FIRST(L);           1
        while r <> q do
            r := NEXT(r, L)      n-r
        end;
        p := NEXT(p, L)          n-p
End;
```

FIRST: $1 \cdot n \cdot 1 = O(n^2)$ times

NEXT: $((n-r+n-q) \cdot n + n-p) \cdot n = O(n^3)$ times

END: $n \cdot n = O(n^2)$ times

3.10

```
level-order.py

1  class Node:
2      def __init__(self, a):
3          self.data = a
4          self.left = None
5          self.right = None
6
7      def levelorder(t):
8          h = height(t)
9          for i in range(1, h+1):
10              currentlevel(t, i)
11
12     def currentlevel(x , y):
13         if x is None:
14             return
15         if y == 1:
16             print "%d" %(x.data)
17         elif y > 1 :
18             currentlevel(x.left , y-1)
19             currentlevel(x.right , y-1)
20
21     def height(t):
22         if t is None:
23             return 0
24         else :
25             lefth = height(t.left)
26             righth = height(t.right)
27             if lefth < righth :
28                 return righth+1
29             else:
30                 return lefth+1
```