**哈尔滨工业大学（深圳）2023年春《数据结构》**

**第一次作业 线性结构**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 |  | 姓名 |  | 成绩 |  |

**1. 简答题**

* 1. **Big-O**

For each of the functions *f(N)* given below, indicate the tightest bound possible (in other words,giving O(2N) as the answer to every question is not likely to result in many points). Unless otherwise specified, all logs are base 2. **You MUST choose your answer from the following** (not given in any particular order), each of which could be re-used (could be the answer for more than one of a) – h)):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| O(N2) | O(N½) | O(N3 logN) | O(N logN) | O(N) | O(N2 logN) | O(N5) |
| O(2N) | O(N3) | O(log N) | O(1) | O(N4) | O(N12) | O(NN) |
| O(N6) | O(N8) | O(N9) | O(N10) |  |  |  |

You do not need to explain your answer.

【参考答案】

|  |  |
| --- | --- |
| Functions | Big-O |
| a) *f(N)* = (1/2) (N logN) + (logN) 2 | O(NlogN) |
| b) *f(N)* = N2·(N + N logN + 1000) | O(N3logN) |
| c) *f(N)* = N2 logN + 2N | O(2N) |
| d) *f(N)* = ( (1/2) (3N + 5 + N) ) 4 | O(N4) |
| e) *f(N)* = (2N + 5 + N4 ) / N | O(N3) |
| f) *f(N)* = log10(2N) | **O(N)** |
| g) *f(N)* = N! + 2N | **O(NN)** |
| h) *f(N)* = (N·N·N·N + 2N)2 | **O(N8)** |

**1.2 Big-O and Run Time Analysis**

Describe the worst case running time of the following pseudocode functions in Big-O notation in terms of the variable n. ***Showing your work is not required*** (although showing work ***may*** allow some partial credit in the case your answer is wrong – don’t spend a lot of time showing your work.). You MUST choose your answer from the following (not given in any particular order), each of which could be re-used (could be the answer for more than one of I. – IV.):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| O(n2) | O(n3logn) | O(nlogn) | O(n) | O(n2log n) | O(n5) |
| O(2n) | O(n3) | O(logn) | O(1) | O(n4) | O(n·n) |

|  |  |  |
| --- | --- | --- |
|  | functions | RunTime |
| Ⅰ | void silly(int n) {  for (int i = 0; i < n; ++i) {  j = n;  while (j > 0) {  System.out.println(”j = ” + j);  j = j - 2;  }  }  } | O(n2) |
| Ⅱ | void silly(int n, int x, int y) {  for (int k = n; k > 0; k--)  if (x < y + n) {  for (int i = 0; i < n; ++i)  for (int j = 0; j < i; ++j)  System.out.println(”y = ” + y);  } else { System.out.println(”x = ” + x);  }  } | O(n3) |
| Ⅲ | void silly(int n) {  for (int i = 0; i < n; ++i) {  for (int j = 0; j < n; ++j)  System.out.println(”j = ” + j);  for (int k = 0; k < i; ++k) {  System.out.println(”k = ” + k)  for (int m = 0; m < 100; ++m)  System.out.println(”m = ” + m);  }  }  } | O(n2) |
| Ⅳ | int silly(int n, int m) {  if (m < 2) return m;  if (n < 1) return n;  else if (n < 10)  return silly(n/m, m);  else  return silly(n - 1, m);  } | O(n) |

1.3 对链表设置头结点的作用是什么?简述线性链表头指针，头结点，首元结点(第一个结点) 三个概念的区别；

【参考答案】

对带头结点的链表,在表的任何结点之前插入结点或删除任何位置的结点,所要做的都是修改前一个结点的指针域,因为在带头结点的链表中任何元素结点都有前驱结点;如果没有头结点,在首元结点前插入结点或删除首元结点都要修改头指针,其算法要比带头结点的算法复杂些;

其次,带头结点的链表结构,初始化后的头指针就固定了,除撤销算法外,所有算法都不会修改头指针,可以减少出错的可能性;

头指针是指向链表中第一个结点的指针。首元结点是指链表中存储第一个数据元素的结点。头结点是在首元结点之前附设的一个结点,该结点不存储数据元素,其指针域指向首元结点,其作用主要是为了方便对链表的操作。它可以对空表、非空表以及首元结点的操作进行统一处理。

1.4 在什么情况下用顺序表比链表好，什么时候用链表比顺序表好？请给出你的分析和理解;

【参考答案】

当线性表的数据元素在物理位置上是连续存储的时候,用顺序表比用链表好,其特点是可以进行随机存取。

1.5 若频繁地对一个线性表进行插入和删除操作，则该线性表宜采用何种存储结构，为什么?

【参考答案】

若频繁地对一个线性表进行插入和删除操作,则该线性表宜采用链式存储结构;因为链式存储结构在插入和删除数据元素时不需要移动数据元素,只需要修改结点的指针域就可以改变数据元素之间的逻辑关系;

1.6 队列可以用单循环链表来实现，故可以只设一个头指针或只设一个尾指针，请分析用哪种方案最合适?

【参考答案】

尾指针是指向终端结点的指针，用它来表示单循环链表可以使得头结点的单循环链表，其尾指针为rear，则开始结点和终端结点的位置分别是rear->next->next和rear，查找时间复杂度都是O(1)。

若用头指针来表示该链表，则查找终端结点的时间复杂度为O(n)。

1.7 设数组A[50][80]，其基地址为2000，每个元素占2个存储单元，以行序为主序顺序存储，回答下列问题：

(1) 该数据组有多少个元素？

(2) 该数组占用多少存储单元？

(3) 数组元素a[30][30]的存储地址是多少?

【参考答案】

(1)该数组有：50\*80=4000个元素;

(2)该数组占用4000\*2=8000个存储单元

(3)loc(30,30)=2000+(30\*80+30)\*2=6860

**2．算法设计**

针对本部分的每一道题，要求：

(1) 给出算法的基本设计思想；

(2) 采用类C或类C++语言描述算法，关键之处给出注释；

(3) 分析算法的时间复杂度和空间复杂度。

【注】可用类语言描述，给出伪码，无需上级调试。

2.1 输入一个已经按升序排序过的数组和一个数字，在数组中查找两个数，使得它们的和正好是输入的那个数字。

【参考答案】

（1）设计思想：

1. 让指针指向数组的头部和尾部，相加，如果小于M，则增大头指针， 如果大于则减小尾指针
2. 退出的条件，相等或者头部=尾部

（2）算法实现

void function(int a[],int n,int M）

{ int i=0, j=n-1;

　 while(i!=j)

{ if(a[i]+a[j]==M)

{　printf("%d,%d",a[i],a[j]);

break;

　　　 }

　　　 a[i]+a[j]>M? j--:i++;

　 }

}

（3）算法分析

利用了折半查找的思想，T(*n*)=O(*n*)，S(*n*)=O(1)。

2.2 How to implement a queue using stack?

【参考答案】

A queue can be implemented using two stacks. Let q be the queue andstack1 and stack2 be the 2 stacks for implementing q. We know that stack supports push, pop, and peek operations and using these operations, we need to emulate the operations of the queue - enqueue and dequeue. Hence, queue q can be implemented in two methods (Both the methods use auxillary space complexity of O(n)):

（1）By making enqueue operation costly:

• Here, the oldest element is always at the top of stack1 which ensures dequeue operation occurs in O(1) time complexity.

• To place the element at top of stack1, stack2 is used.

• Pseudocode:

o Enqueue: Here time complexity will be O(n)

enqueue(q, data):

While stack1 is not empty:

Push everything from stack1 to stack2.

Push data to stack1

Push everything back to stack1.

• Dequeue: Here time complexity will be O(1)

deQueue(q):

If stack1 is empty then error else

Pop an item from stack1 and return it

（2）By making the dequeue operation costly:

• Here, for enqueue operation, the new element is pushed at the top of stack1. Here, the enqueue operation time complexity is O(1).

• In dequeue, if stack2 is empty, all elements from stack1 are moved to stack2 and top of stack2 is the result. Basically, reversing the list by pushing to a stack and returning the first enqueued element. This operation of pushing all elements to a new stack takes O(n) complexity.

• Pseudocode:

o Enqueue: Time complexity: O(1)

enqueue(q, data):

Push data to stack1

• Dequeue: Time complexity: O(n)

dequeue(q):

If both stacks are empty then raise error.

If stack2 is empty:

While stack1 is not empty:

push everything from stack1 to stack2.

Pop the element from stack2 and return it.

2.3 设计一个算法,利用栈来实现带头结点的单链表h的逆序。

【参考答案】

（1）设计思想

将单链表结点依次放入**链栈**中,链栈本身就是一个单链表,即实现了原单链表的逆序;假设链栈不带头结点,再加上原来的头结点,就完成了原单链表的逆序;

（2）算法实现

Void revertSNode(h)

{ SNode st=NULL;

p=h->next,q;

While(p)

{ q=p->next;

p->next=st;

st=p;

p=q;

}

h->next=st;

}

3、算法分析

T(n)=O(n); S(n)=O(1)

2.4 分别用顺序表、单项链表作为线性表的存储结构，元素类型为整型（int）。设计算法，将线性表中所有的负数放在正数之前，亦可理解为线性表的左端元素值均小于0，而右端元素值均大于或等于0。

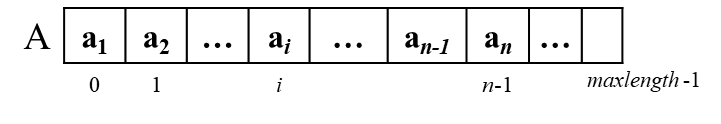
要求算法的空间复杂度S(n)=O(1);

已知数据类型定义如下：

# define maxlength 100

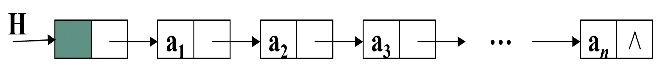
Typedef struct {

int data[maxlength];

 int last;

} SLIST;

SLIST A;

Typedef struct NodeType {

int data ;

NodeType \*next ;

}LLIST ;

LLIST H；

【参考答案】

* 顺序表：

（1）设计思想：

令游标 L 从左（L=1）向右扫描，越过key小于 v的记录，直到A[L]data ≥v 为止；同时令游标 R 从右(R=A.last）开始向左扫描，越过大于等于 v 的记录,直到A[R].data＜v 的记录A[R]为止；

若L>R（L=R+1），成功划分，L是右边子序列的起始下表；

若L<R, 则swap( A[L].data, A[R].data )；

重复上述操作，直至过程进行到L>R（L=R+1）为止。

（2）算法实现：

Void Partition (SLIST &A )

{ int L, R ;

L = 1; R = A.last ;

do {

swap ( &A[L].data, &A[R].data ) ;

while ( A.data[L]<0 )

L = L +1 ;

while (A.data[R] >= 0 )

R = R -1 ;

} while ( L <= R );

}

（3）算法分析

T(n)=O(n); S(n)=O(1).

* 单向链表

1. 设计思想

从表头到表尾，越过值>=0的结点，把值<0的结点插入到表头位置（头插）

（2）算法实现

Void Partition (LLIST &H )

{ LLIST \*p,\*q;

P=H->next;

While(p->next)

{

If(!p->next) return;

If(p->next->data>=0)

P=p=>next;

else

{

q=p->next;

p->next=q->next;

q->next=H->next;

H->next=q

}

}

（3）算法分析

T(n)=O(n); S(n)=O(1).

2.5 已知一个带有表头结点的单链表，结点结构为（data,next），假设该链表只给出了头指针list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第k个位置上的结点（k为正整数）。若查找成功，算法输出该结点的data域的值，并返回1；否则，只返回0。

【参考答案】

（1）设计思想

定义两个指针变量p和q，初始时均指向头结点的下一个结点。p指针沿链表移动；当p指针移动到第k个结点时，q指针开始与p指针同步移动；当p指针移动到链表最后一个结点时，q指针所指元素为倒数第k个结点。

以上过程对链表仅进行一遍扫描。

（2）算法实现

typedef struct LNode {

int data;

struct LNode \* next;

} \* LinkList;

int SearchN (LinkList list, int k)

{

LzinkList p, q;

int count = 0; /\* 计数器赋初值 \*/

p = q = list-> next; /\* p和q指向链表表头结点的下一个结点 \*/

while (p!= NULL)

{

if (count < k)

count++; /\* 计数器+1 \*/

else

q = q-> next; /\* q移到下一个结点 \*/

p = p-> next; /\* p移到下一个结点 \*/

}

if (count < k) /\* 如果链表的长度小于k \*/

return (0); /\* 查找失败 \*/

else

{

printf(“%d”, q->data); /\* 查找成功 \*/

return (1);

}

}

（3）算法分析

T(n)=O(n)，S(n)=O(1).

**2.6 设二维数组A[1..m][1..n]含有m\*n个整数，**设计一个算法，判断A中所有元素是否互不相同，输出相关信息（yes/no）。

【参考答案】

（1）设计思想

在当前位置，每个元素要同本行后面的元素比较一次，然后同第i+1行及以后各行元素比较一次。如何达到每个元素同其它元素比较一次且只一次？

（2）算法实现

int JudgEqual(int A[m][n], int m,n)

//判断二维数组中所有元素是否互不相同，如果是返回1，否则返回0

{ for(i=0;i<m;i++)

for(j=0;j<n-1;j++)

{ for(p=j+1;p<n;p++)

if(A[i][j]==A[i][p]) { count<<”no”;return(0);} //发现一个相同

for(k=i+1;k<m;k++) //和第i+1行及以后元素比较

for(p=0;p<n;p++)

if(A[i][j]==A[k][p]) { count<<”no”;return(0);}

}

count<<”yes”;

return(1); //元素互不相同

} //JudgEqual

1. 算法分析

第1个元素需和其它m\*n-1个元素比较

第2个元素需和其它m\*n-2个元素比较

……

第m\*n-1个元素需和最后1个元素（m\*n）比较

总的最多比较次数为：

(m\*n-1)+(m\*n-2)+……+2+1=(m\*n)(m\*n-1)/2。

假设任何一个位置上的具有相同元素的概率相同，则平均比较次数为：

(m\*n)(m\*n-1)/4

所以总的时间复杂度T(n)=O((m\*n)2)=O(n4)。