**Advanced Algorithms**

**Exercise for Lecture 8,9,10**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** | **周思源** | **Student ID** | 24S151092 |
| **Lecture 5** |  | | |
| **Lecture 6** |  | | |
| **Lecture 7** |  | | |
| **Total Score** |  | | |
| **Notes** | Deadline: **2024-10-17 24:00**  Submission Format: ‘**Lecture8910\_Name\_ID.docx**’, and please send to:**aa\_24fall\_hw@163.com**  This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. | | |

**Lecture 8**

**Problem 8.1[15 points]** Observe that whenever we reference the size attribute of a node in either OS-SELECT or OS-RANK, we use it only to compute a rank. Accordingly, suppose we store in each node its rank in the subtree of which it is the root. Show how to maintain this information during insertion and deletion. (Remember that these two operations can cause rotations.)

**Solution**:

In addition to the attributes x.key,x.color,x.p,x.left,x.right, each node x has a new attribute x.rank, which indicates the rank of x in the subtree with x as the root node. It is named rank to distinguish it from rank. Considering the sizesizesize attribute of the node, we have x.rank′=x.size+1. When inserting node z, we search for the correct position of node z in the tree. For each node x on the path, if z is inserted in the left subtree of x, add 1 to x.rank, and if z is inserted in the right subtree of x, keep x.rank′unchanged.

RB-INSERT(T, z)

x = T.root

y = T.nil

while x ≠ T.nil // descend until reaching the sentinel

y = x

if z.key < x.key

x = x.left

x.rank' = x. rank' + 1

else x = x.right

z.p = y

if y == T.nil

T.root = z

else if z.key < y.key

y.left = z

else y.right = z

z.left = T.nil

z.right = T.nil

z.color = RED

z.rank' = 1

RB-INSERT-FIXUP(T, z)

RB-DELETE(T, z)

y = z

y-original-color = y.color

if z.left == T.nil

x = z.right

RB-TRANSPLANT(T, z, z.right) // replace z by its right child

else if z.right == T.nil

x = z.left

RB-TRANSPLANT(T, z, z.left)

else y = RB-MINIMUM(T, z.right)

y-original-color = y.color

x = y.right

if y ≠ z.right

RB-TRANSPLANT(T, y, y.right)

y.right = z.right

y.right.p = y

else x.p = y

RB-TRANSPLANT(T, z, y) // replace z by its successor y

y.left = z.left

y.left.p = y

y.color = z.color

t = x // modify rank' ...

while t.p ≠ T.nil

if t == t.p.left

t.p.rank' = t.p.rank' - 1

t = t.p // ... modify rank'

if y-original-color == BLACK

RB-DELETE-FIXUP(T, x)

LEFT**-**ROTATE(T, x)

y **=** x.right

x.right **=** y.left

**if** y.left ≠ T.nil

y.left.p **=** x

y.p **=** x.p

**if** x.p **==** nil

T.root **=** y

**else** **if** x **==** x.p.left

x.p.left **=** y

**else** x.p.right **=** y

y.left **=** x

x.p **=** y

y.rank' **=** y.rank' **+** x.rank'

RIGHT**-**ROTATE(T, x)

y **=** x.left

x.left **=** y.right

**if** y.right ≠ T.nil

y.right.p **=** x

y.p **=** x.p

**if** x.p **==** nil

T.root **=** y

**else** **if** x **==** x.p.right

x.p.right **=** y

**else** x.p.left **=** y

y.right **=** x

x.p **=** y

x.rank' **=** x.rank' **-** y.rank'

**Problem 8.2[10 points]** Given an interval tree T and an interval i, describe how to list all intervals in T that overlap i in O(min(n,klgn)) time, where k is the number of intervals in the output list. (Hint: One simple method makes several queries, modifying the tree between queries. A slightly more complicated method does not modify the tree.)

**Solution**:

INTERVALS**-**SEARCH(T, i)

let A be an empty array

f **=** false

**while** f ≠ true

x **=** T.root

**while** x ≠ T.nil and i does not overlap x.**int**

**if** x.left ≠ T.nil and x.left.max ≥ i.low

x **=** x.left

**else** x **=** x.right

**if** x ≠ T.nil

APPEND(A, x)

**if** x ≠ T.root

**if** x **=** x.p.left

x.p.left **=** T.nil

**else** x.p.right **=** T.nil

**else** T.root **=** nil

**else** f **=** true

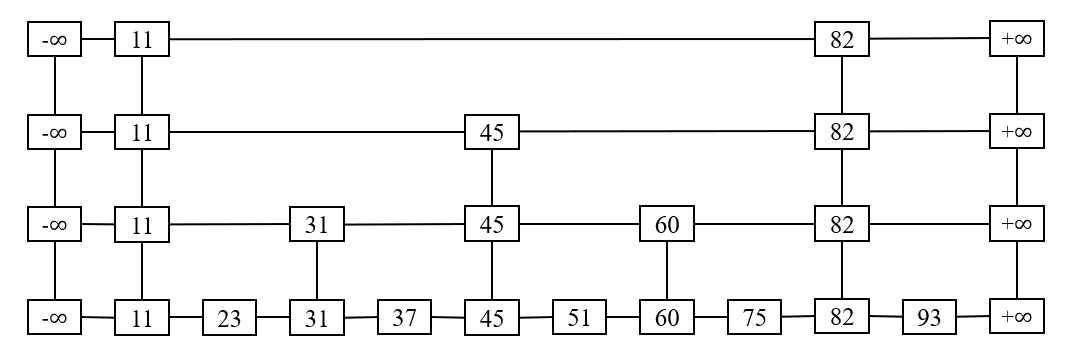
**return** A

**Lecture 9**

**Problem 9.1[20 points]**

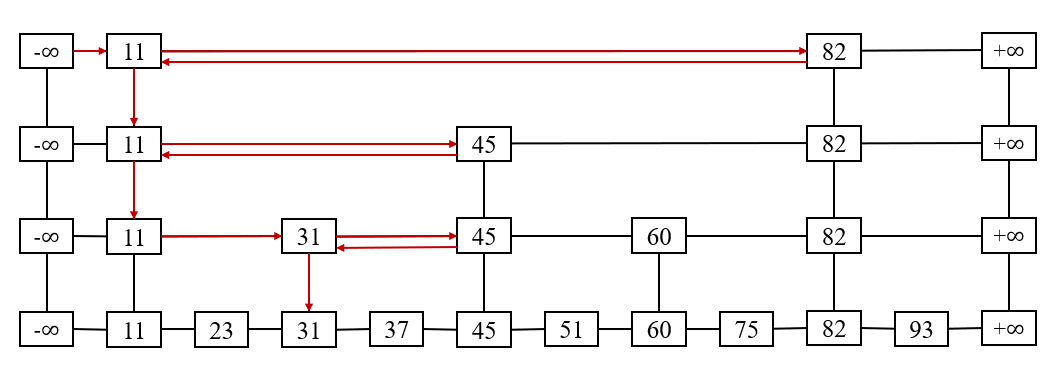
(1) Please show how to search 51 in the next skip list. Each comparison is an intermediate step, and you need to provide each step. The answer can be shown in one picture and the example is shown as follows:

(2) Please show how to search 93 in the next skip list.



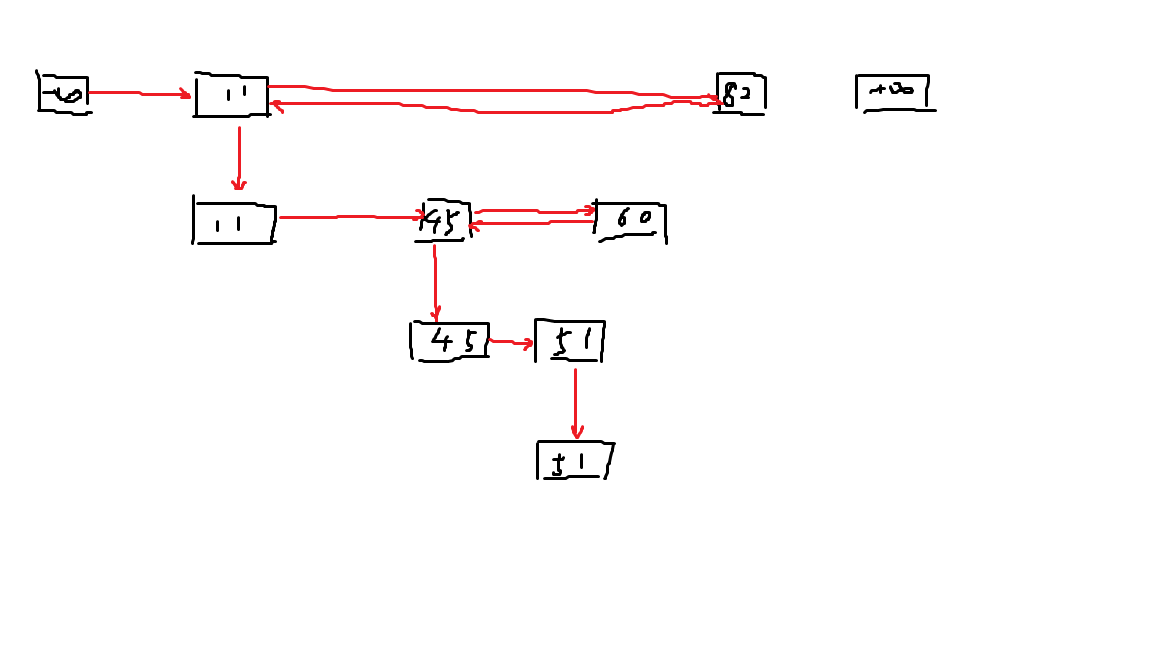
**Picture 1. The skip list**

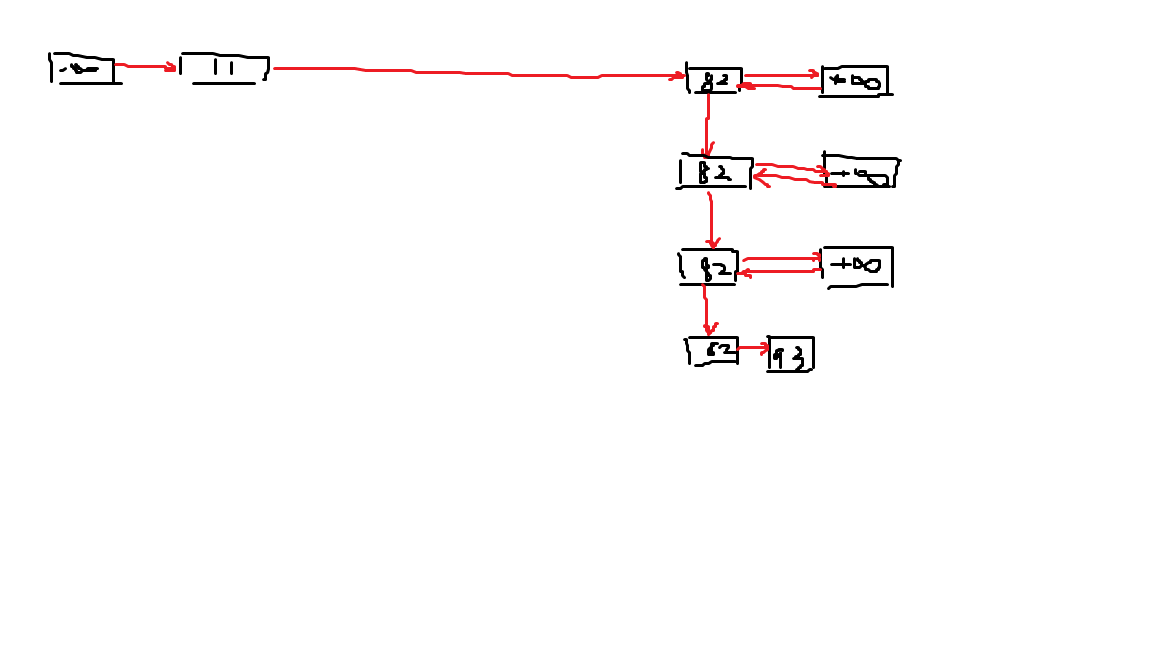
**Search 31**



**Picture 2. The example of solution**

**Solution:**

****

****

**Problem 9.2[20 points]**

(1) Please design the data structure of the nodes in the skip list and the data structure of skip list separately.

(2) Please give the code for deleting operation in the skip list. Please give the effective C code directly instead of pseudo code whose name should be Skip\_List\_Delete().

**Solution**:  
//定义key和value的类型

typedef int KeyType;

typedef int ValueType;

//定义结点

typedef struct nodeStructure\* Node;

struct nodeStructure{

KeyType key;

ValueType value;

Node forward[1];

};

//定义跳跃表

typedef struct listStructure\* List;

struct listStructure{

int level;

Node header;

};

bool Delete(const KeyType& key,

ValueType& value){

Node update[MAX\_LEVEL];

int i;

Node x = list\_->header;

//寻找要删除的结点

for(i = list\_->level; i >= 0; --i){

while(x->forward[i]->key < key){

x = x->forward[i];

}

update[i] = x;

}

x = x->forward[0];

//结点不存在

if(x->key != key){

return false;

}else{

value = x->value;

//调整指针

for(i = 0; i <= list\_->level; ++i){

if(update[i]->forward[i] != x)

break;

update[i]->forward[i] = x->forward[i];

}

//删除结点

free(x);

while(list\_->level > 0

&& list\_->header->forward[list\_->level] == NIL\_){

--list\_->level;

}

--size\_;

return true;

}

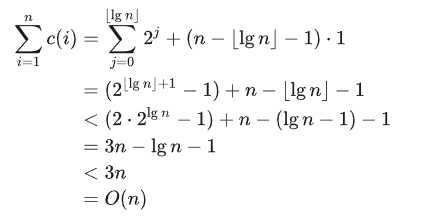
}

**Lecture 10**

**Problem 10.1[20 points]** Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2,and 1 otherwise. Choose any two of three methods to determine the amortized cost per operation, respectively.

**Solution**:

The total actual cost is:



Because the total amortized cost is asymptotically equal to the total actual cost, the amortized cost of each operation is:O(n)/n = O(1)

**Problem 10.2[15 points]** Show how to implement a queue with two ordinary stacks so that the amortized cost of each ENQUEUE and each DEQUEUEDEQUEUE operation is O(1).

**Solution**:

Suppose there are two stacks s1 and s2, objects are queued from s1 and dequeued from s2, the pseudo code is as follows:

BUILD-QUEUE()

Let Q.S1 and Q.S2 be new stacks

ENQUEUE(Q, x)

PUSH(Q.S1, x)

DEQUEUEDEQUEUE(Q)

If STACK-EMPTY(Q.S2) == TRUE

While STACK-EMPTY == FALSE

PUSH(Q.S2, POP(Q.S1))

Return POP(Q.S2)

When performing a certain operation, the number of objects in the stack S1 is s1, and the number of objects in the stack S2 is s2. The actual cost and amortized cost of the queue operation are:

|  |  |  |
| --- | --- | --- |
| operaion | Actual cost | Amortized cost |
| ENQUEUE | 1 | 3 |
| DEQUEUEDEQUEUE | 2\*s1 | 0 |

Suppose there are n operations in total, the total amortized cost is at most O(n), and the amortized cost of an operation in the operation sequence is

O(n)/n = O(1).