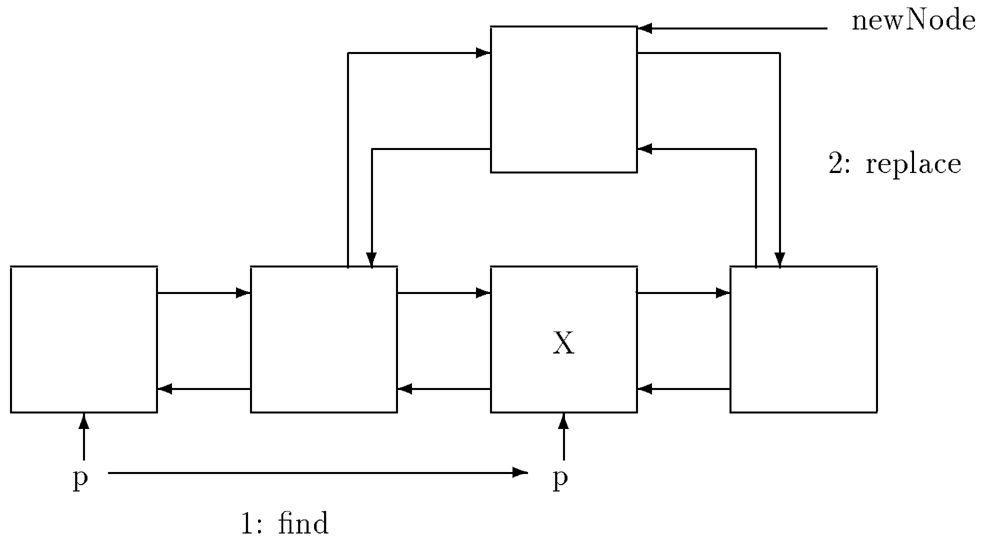
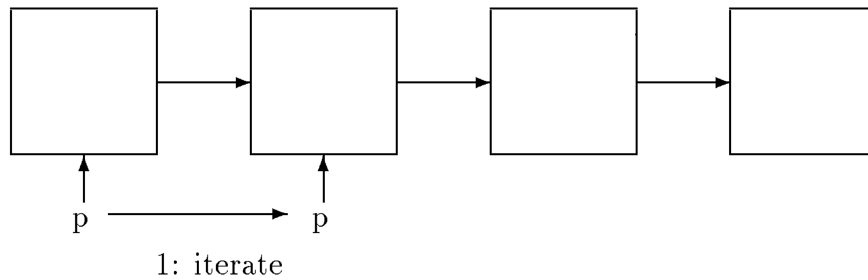
UNSORTED LINKED LISTS

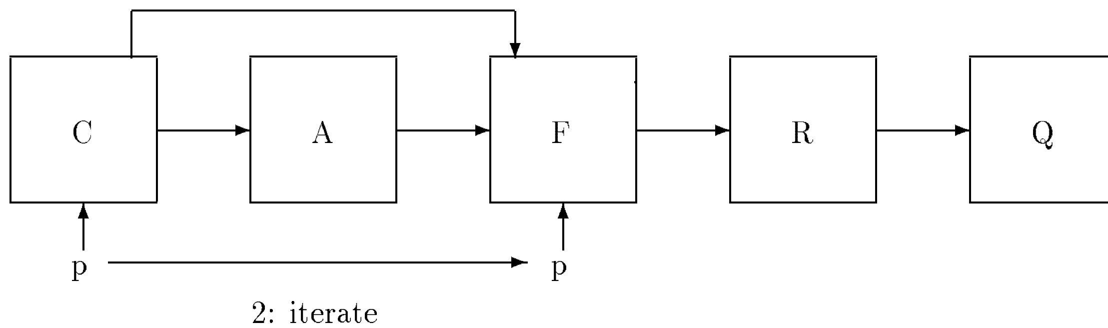
1. FALL 2007 #2: FIND_REPLACE



2. FALL 2006 #2: IS_LIST_EVEN

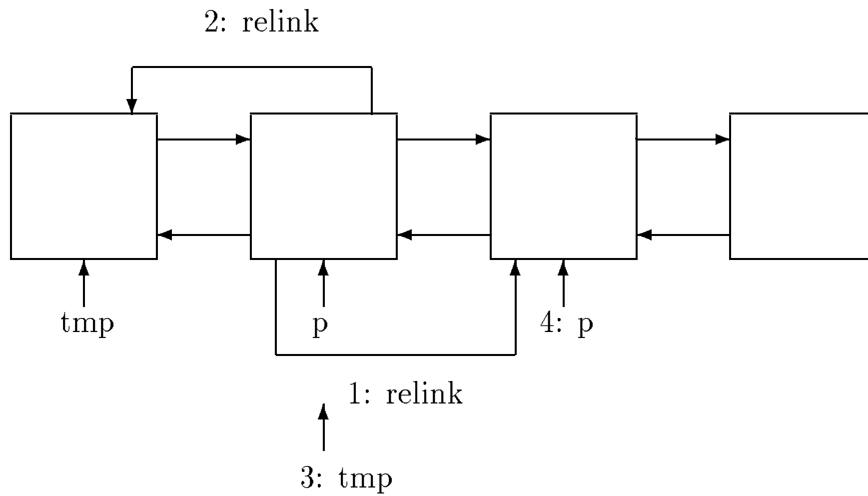


3. SPRING 2006 #3: REMOVE_EVERY_OTHER_NODE

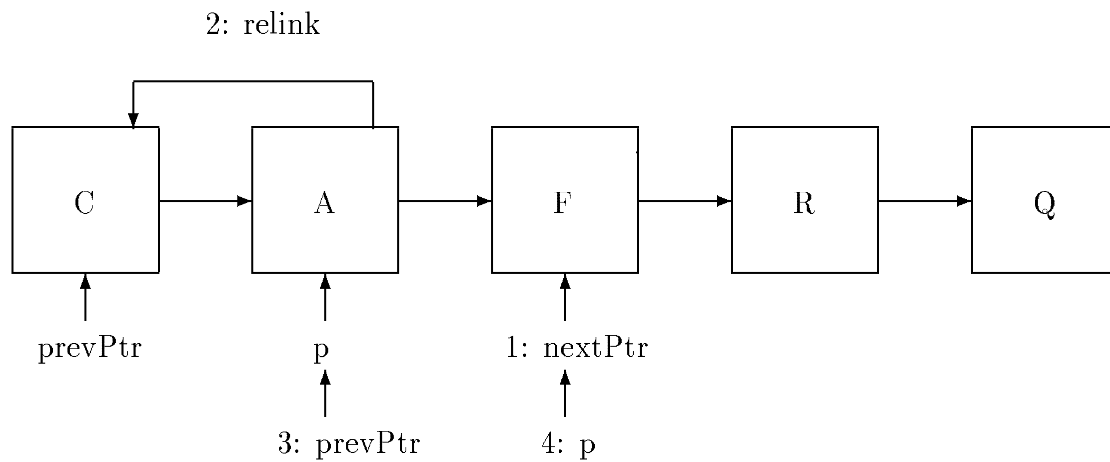



```
void rm_every_other(node_ptr p) {
  while (p && p->next) {
    p->next = p->next->next;
    p = p->next;
  }
}
```

5. FALL 2005 #2: REVERSE_DOUBLY_LINKED_LIST (REVISED 4/4/08)

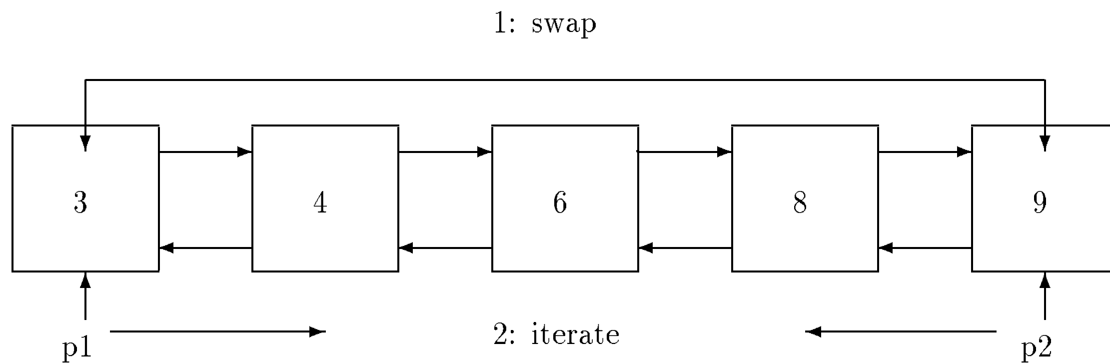


6. FALL 2004 #1: REVERSE_SINGLY_LINKED_LIST



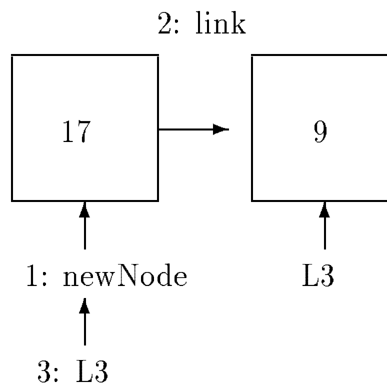
SORTED LINKED LISTS

1. SPRING 2007 #2: REVERSE_ELEMENTS_DOUBLY_LINKED_LIST



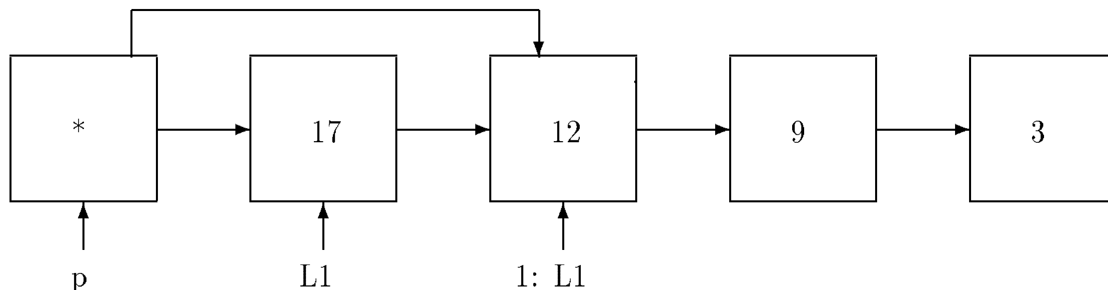
2. SPRING 2003 #2: REVERSE_INTERSECTION

```
While neither list is exhausted:

  if one value is smaller : increment its pointer

  if both values are equal: push value onto L3, increment both L1, L2

--------------------------

[2]->[9]->[17]

L1

[3]->[9]->[12]->[17]->[20]

L2

--------------------------

[2]->[9]->[17]

     L1

[3]->[9]->[12]->[17]->[20]

L2

--------------------------

[2]->[9]->[17]

     L1

[3]->[9]->[12]->[17]->[20]

     L2

--------------------------

Assume 9 has already been pushed, now push 17:
```



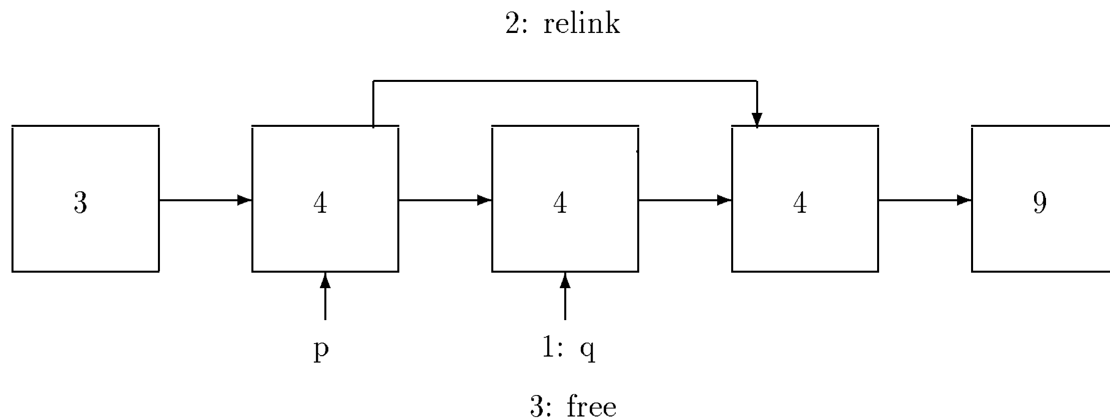
3. SPRING 2002 #2: ELIMINATE_DUPLICATES_ACROSS_LISTS

```
init previous p to L1's dummy node [*]

increment L1, L2

While neither list is exhausted:

  if L1 value is larger   : increment L1 pointer

  if L2 value is larger   : increment L2 pointer

                            increment p

  if both values are equal: increment both L1, L2

                            relink using p

                            do NOT increment p
-------------------------------

[*]->[25]->[17]->[10]->[9]->[2]

      L1

[*]->[17]->[12]->[9]->[3]

      L2

p
-------------------------------

[*]->[25]->[17]->[10]->[9]->[2]

            L1

[*]->[17]->[12]->[9]->[3]

      L2

p
-------------------------------

Since L1, L2 values are both 17, relink around L2's 17:
```



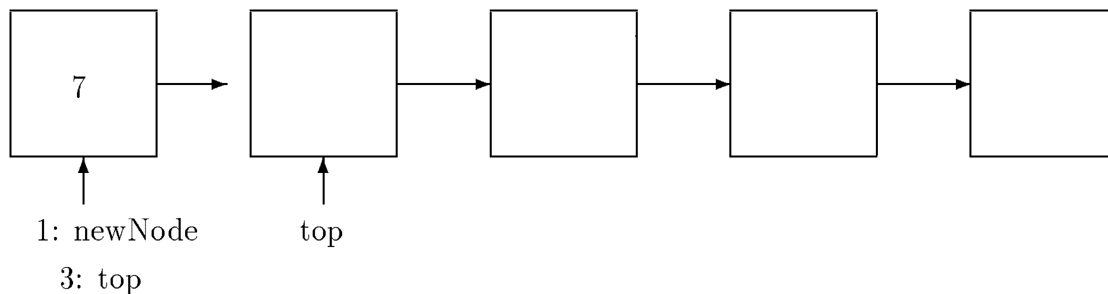
4. FALL 1999 #2: ELIMINATE_DUPLICATES_WITHIN_LIST

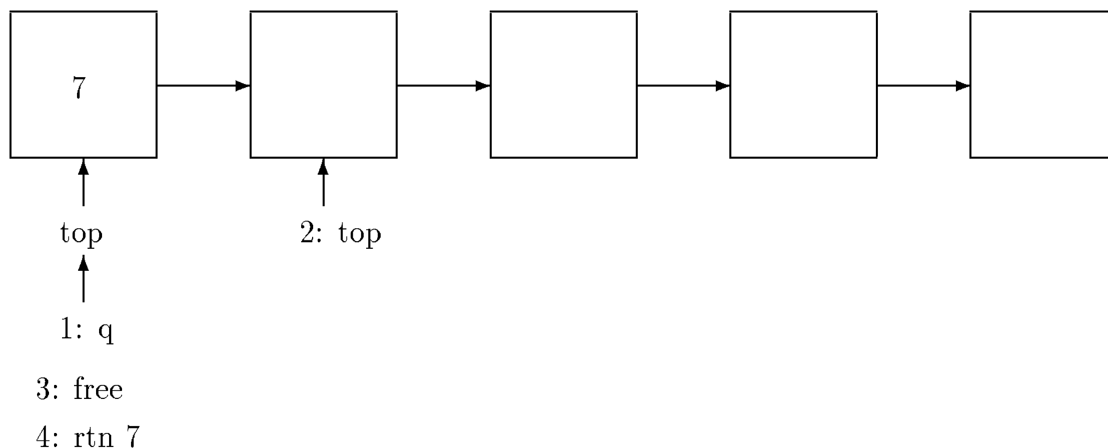
```
if adjacent values are NOT equal: increment p
if adjacent values ARE equal    : relink
                                  free q
                                  do NOT increment p
```



STACK LINKED LISTS

1. FALL 2001 #2: STACK_ADT_AS_LIST

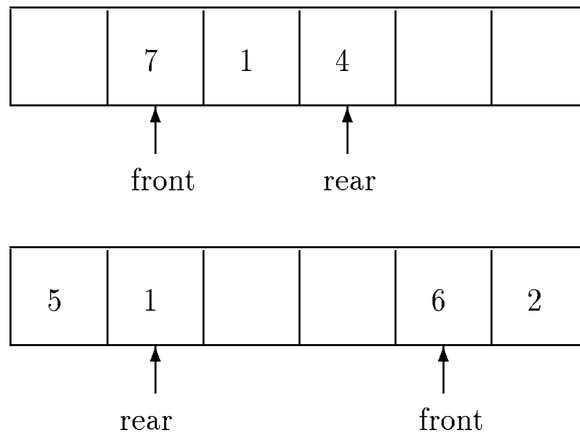




QUEUE ARRAY LISTS

1. SPRING 2005 #1: CIRCULAR_QUEUE_AS_ARRAY

```
iterate from front to rear looking for x

need a successor function to handle both cases: i++ with wrap-around
```



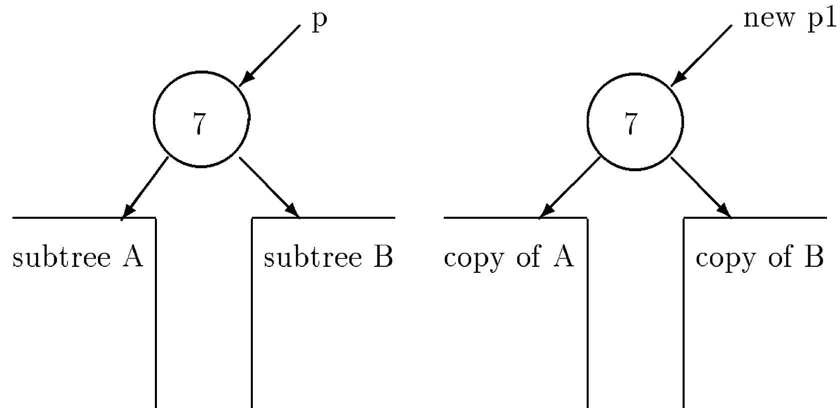
GENERAL TREES

1. FALL 2001 #1: IS_TREE_BINARY

```
iterate across child list:

  count and if > 2 then abort

  recursion on subtree

  if not binary then abort

success


A

|

v

B---->C---->D

|     |

v     v

E->F  G
```

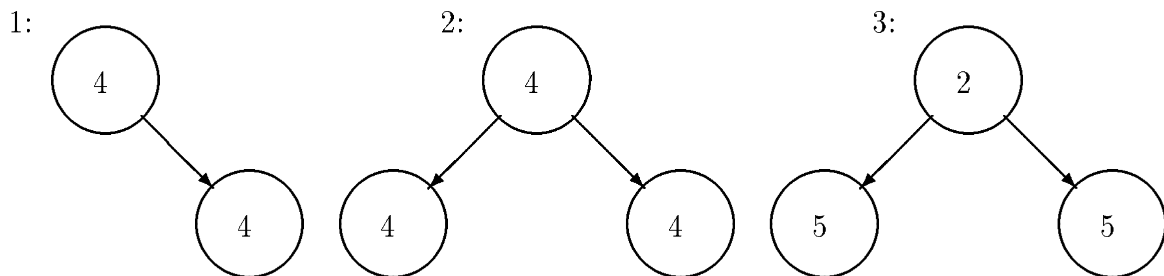
BINARY TREES

1. FALL 2007 #3: COPY_TREE

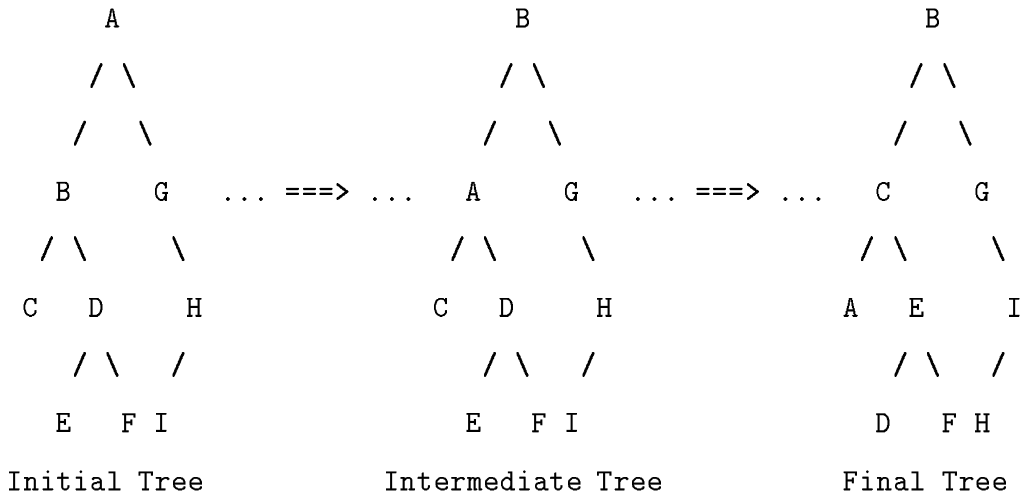


3. FALL 2006 #3: IS_TREE_ALL_SAME

```
1: left does not exist
   right exists and agrees
   continue with recursion
2: left and right exist and agree
   continue with recursion
3: left and right exist but do NOT agree
   abort
```



5. SPRING 2005 #2: SWAP_WITH_LEFT_CHILD



swap A and B

recursion left, recursion right

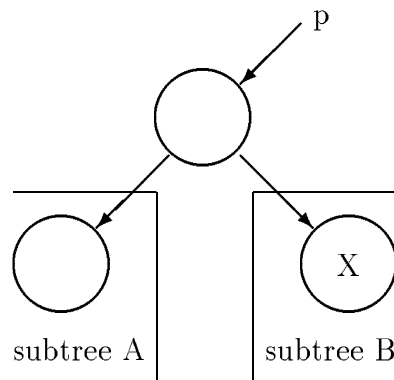
6. SPRING 2004 #2: FIND_PARENT_OF_X

left exists but is NOT x

right exists and IS x

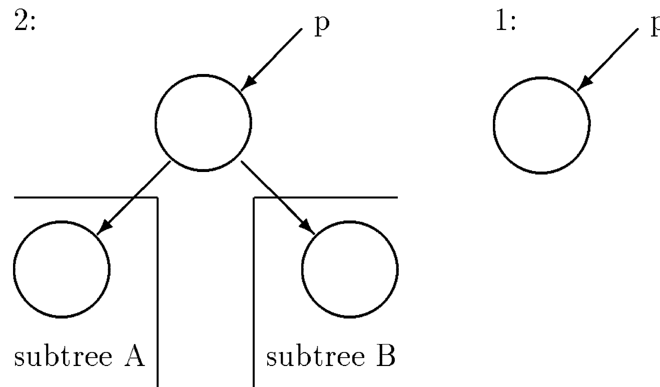
otherwise recursion A

if not found recursion B



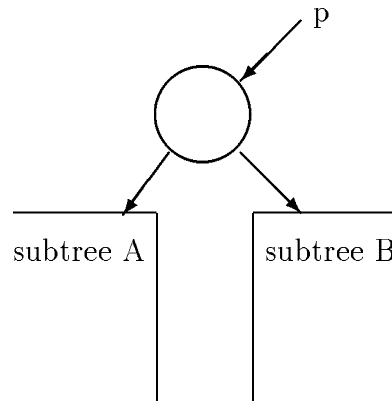
7. SPRING 2003 #1: COUNT_INTERIOR_NODES

```
1: neither left or right exist: NOT interior

2: otherwise node IS interior: 1 + A + B
```



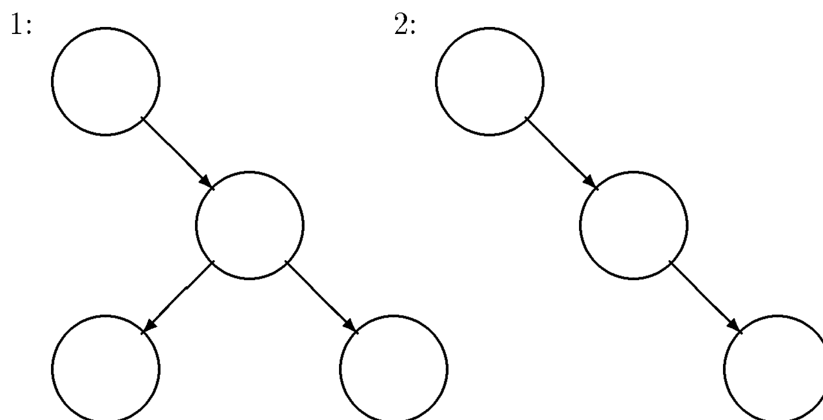
8. FALL 2002 #1: IS_TREE_EVEN

```
A, B EVEN: subtree p is ODD

A, B ODD : subtree p is ODD

A <> B   : subtree p is EVEN
```



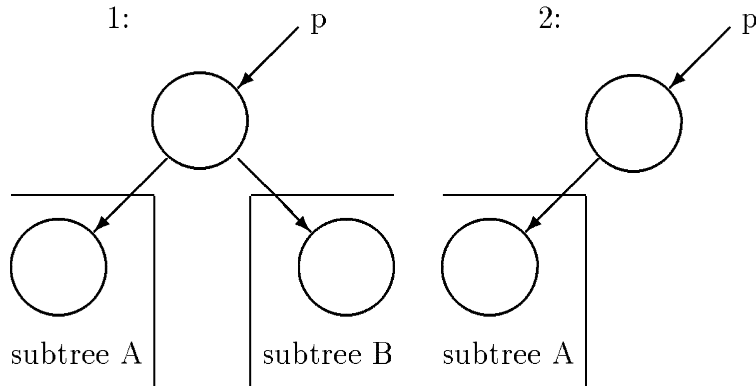
9. SPRING 2000 #1: NO_LEFT_CHILDREN

```
1: found a LEFT      : FALSE

2: never found a LEFT: TRUE
```



10. SPRING 1999 #2: COUNT_FULL_NODES

```
1: A AND B exist      : 1 + A + B

2: A OR B do NOT exist: 0 + A + B
```



Alternatively: COUNT_LEAF_NODES:

```
int count_leaf_nodes(BINARY_TREE T) {
  if (T == NULL)
    return 0;
  else
    if (T->left == NULL && T->right == NULL)
      return 1;
    else
      return count_leaf_nodes(T->left) + count_leaf_nodes(T->right);
}
```