Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Advanced Queue

Example: 1to1/1to1.c

```c
int main(void)
{
     int  funct_table[7] = { 2, 0, 0, 4, 4, 3, 5};
     int  n = sizeof(funct_table)/sizeof(int);
     int  status[sizeof(funct_table)/sizeof(int)];
     int  counter[sizeof(funct_table)/sizeof(int)];
     int  i;

     printf("\nOne-To-One Function Construction Program");
     printf("\n========================================\n");
     printf("\nDomain    Range    Status");
     printf("\n------    -----    ------");

     find_one_to_one(funct_table, status, counter, n);

     for (i = 0; i < n; i++) {
          printf("\n%4d%10d", i, funct_table[i]);
          if (status[i] == SAVED)
               printf("        SAVED");
          else
               printf("       DELETED");
     }

     printf("\n\nConstructed New 1-1 Function\n");
     printf("\nDomain    Range");
     printf("\n------    -----");
     for (i = 0; i < n; i++)
          if (status[i] == SAVED)
               printf("\n%4d%10d", i, funct_table[i]);
     printf("\n");
     return 0;
```

Example: 1to1/1to1.c

```c
void find_one_to_one(int  funct[], int status[], int counter[], int n)
{
     int  queue[QUEUE_SIZE];  /* we need a queue          */
     int  head, tail;         /* queue pointers           */
     int  i, j;

     for (i = 0; i < n; i++) { /* initialization          */
          counter[i] = 0;     /* size of inverse-images   */
          status[i]  = SAVED; /* assume all are SAVED     */
     }

     for (i = 0; i < n; i++)  /* count inverse-image size */
          counter[funct[i]]++;

     for (tail = -1, i = 0; i < n; i++) /* put all i such */
          if (counter[i] == 0) /* that counter[i]=0 to Q  */
               queue[++tail] = i;

     head = 0;                /* main loop. start from H  */
     while (head <= tail) {   /* if there have elements   */
          j = queue[head++];  /* get it and put it to j   */
          status[j] = DELETED;/* delete it. no inv-image  */
          if (--counter[funct[j]] == 0)
               queue[++tail] = funct[j];
     }
}
```

Example: 1to1/1to1.c

```
21 void find_one_to_one(int  funct[], int status[], int counter[], int n)
22 {
23      int  queue[QUEUE_SIZE];  /* we need a queue          */
24      int  head, tail;         /* queue pointers           */
25      int  i, j;
26
27      for (i = 0; i < n; i++) { /* initialization          */
28           counter[i] = 0;     /* size of inverse-images   */
29           status[i]  = SAVED; /* assume all are SAVED     */
30      }
31
32      for (i = 0; i < n; i++)  /* count inverse-image size */
33           counter[funct[i]]++;
34
35      for (tail = -1, i = 0; i < n; i++) /* put all i such */
36           if (counter[i] == 0) /* that counter[i]=0 to Q  */
37                queue[++tail] = i;
38
39      head = 0;                /* main loop. start from H  */
40      while (head <= tail) {   /* if there have elements   */
41           j = queue[head++];  /* get it and put it to j   */
42           status[j] = DELETED;/* delete it. no inv-image  */
43           if (--counter[funct[j]] == 0)
44                queue[++tail] = funct[j];
45      }
46 }
```

Think over 1to1 example

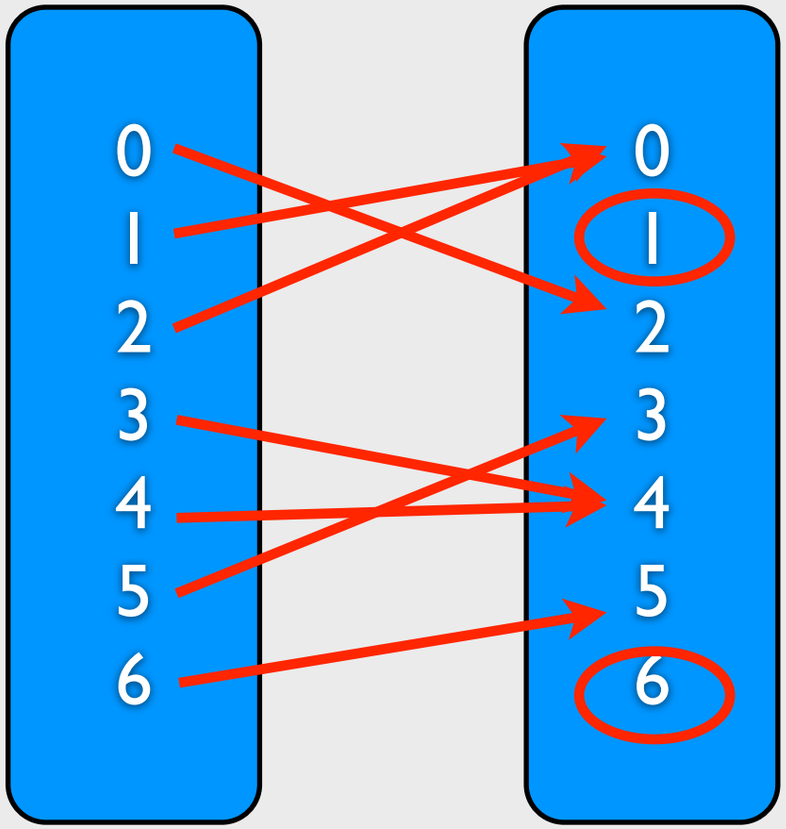
- Why we can find out the S’ when the queue is empty?
- What if we replace the queue as stack?
- What is the worst case? How many times will be executed?

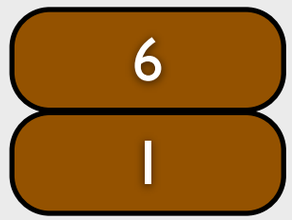
Think over 1to1 example

- Why we can find out the S’ when the queue is empty?
- That is because the queue is to store the numbers without any mappings. So, when we delete these numbers, the remaining numbers form the S’ set.

Think over 1to1 example

- What if we replace the queue as stack?

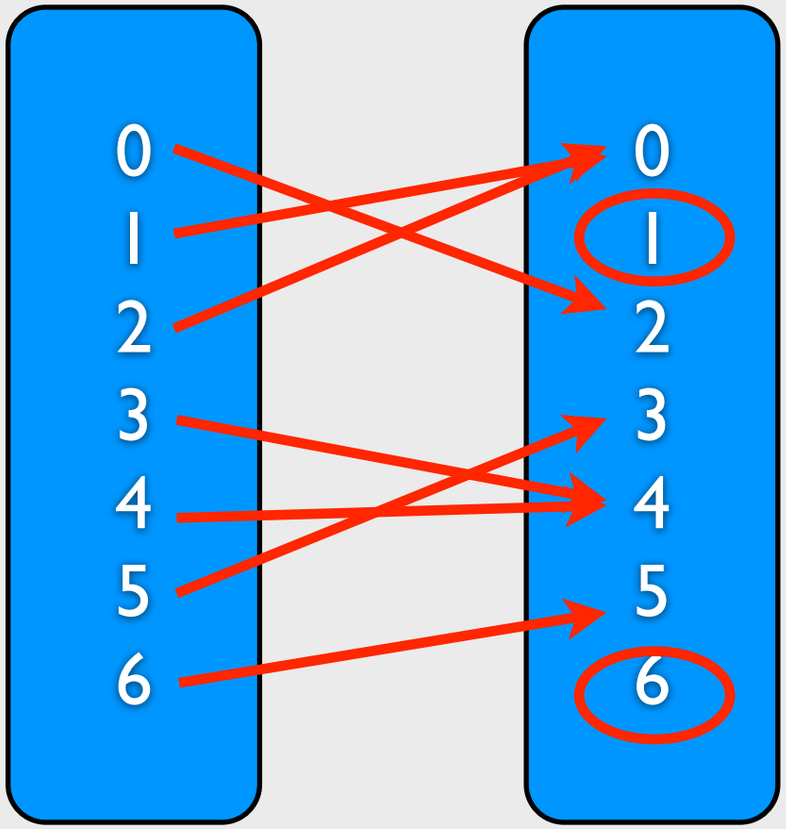




stack

Think over 1to1 example

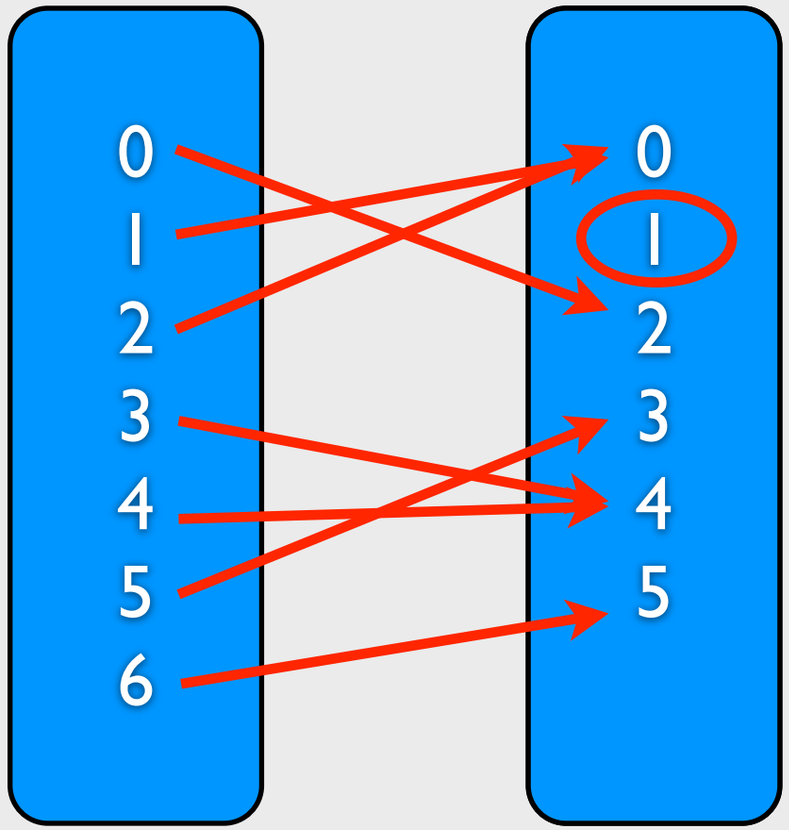
- What if we replace the queue as stack?



stack

Think over 1to1 example

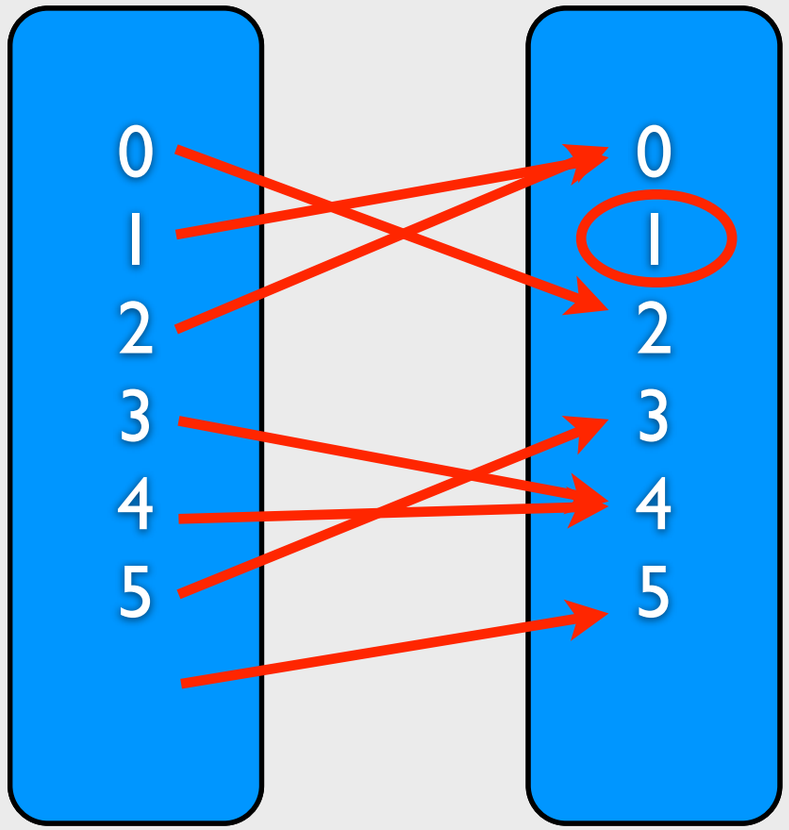
- What if we replace the queue as stack?



stack

Think over 1to1 example

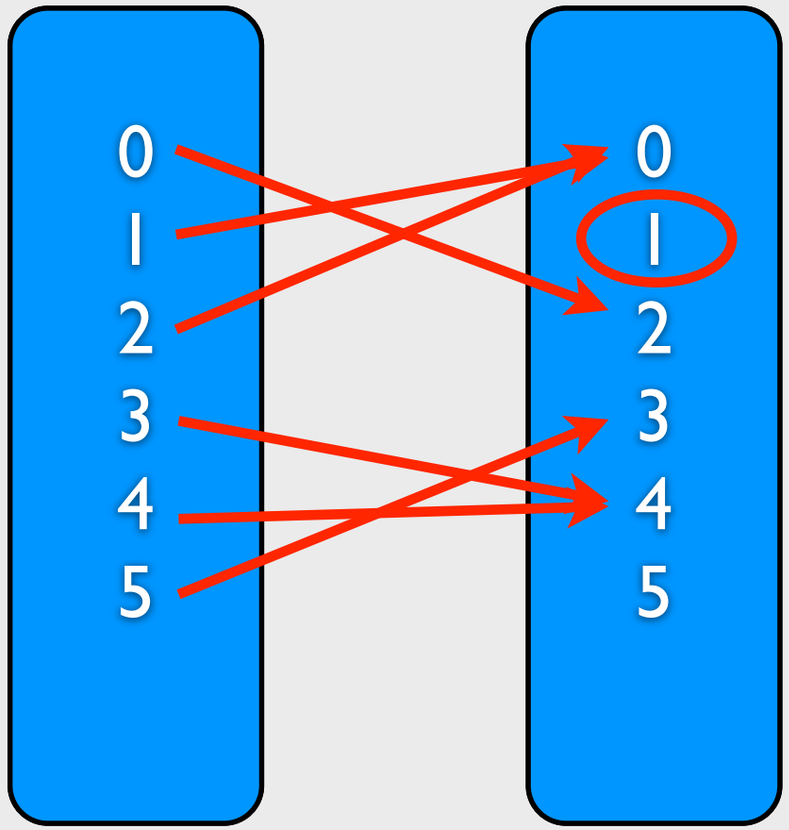
- What if we replace the queue as stack?



stack

Think over 1to1 example

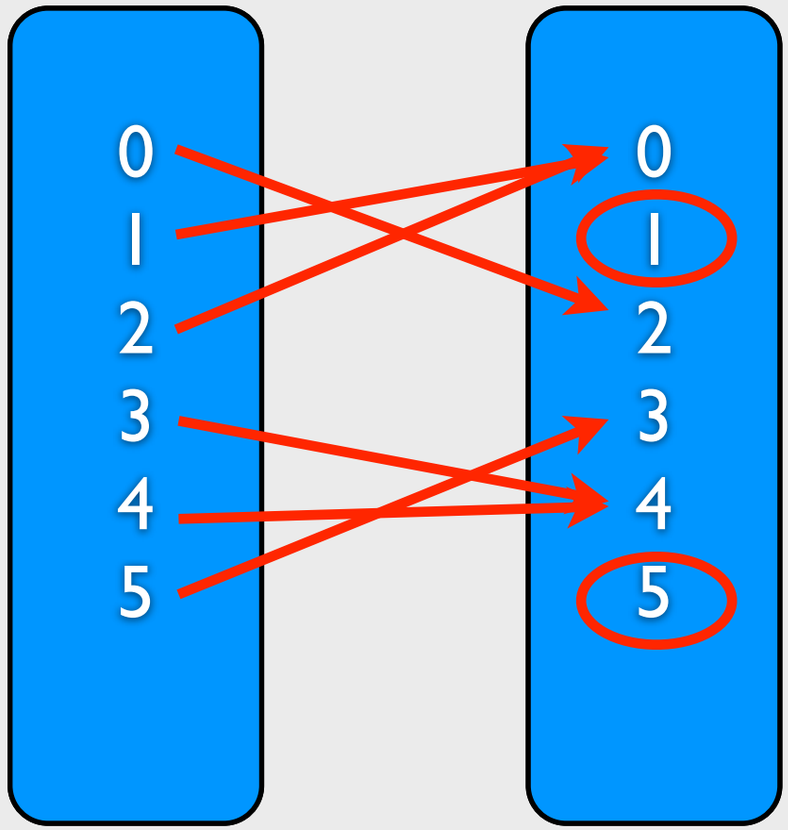
- What if we replace the queue as stack?



stack

Think over 1to1 example

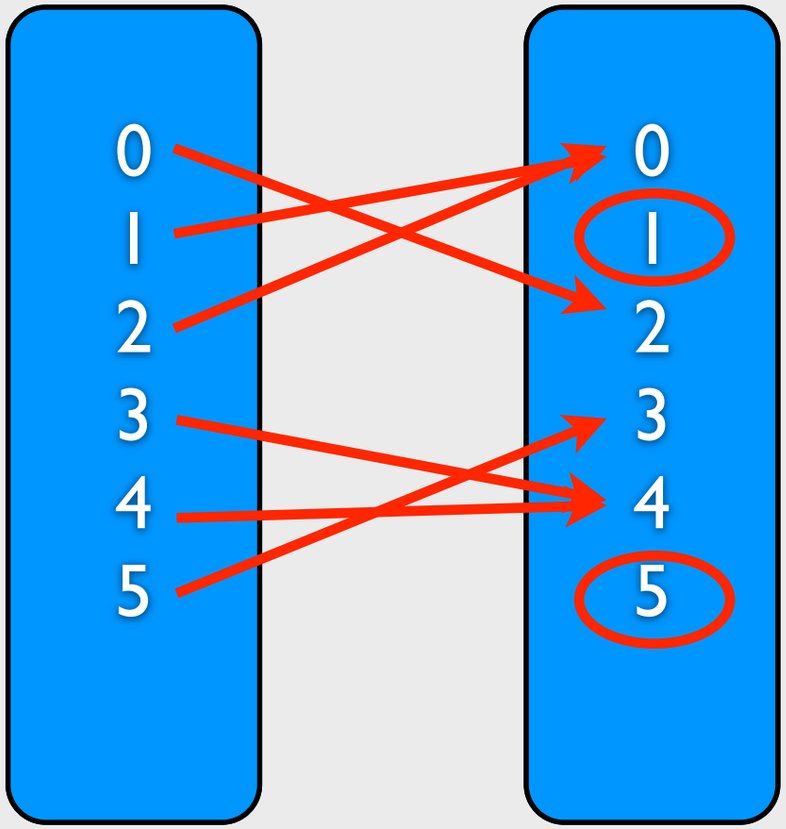
- What if we replace the queue as stack?

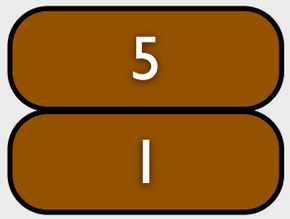


stack

Think over 1to1 example

- What if we replace the queue as stack?

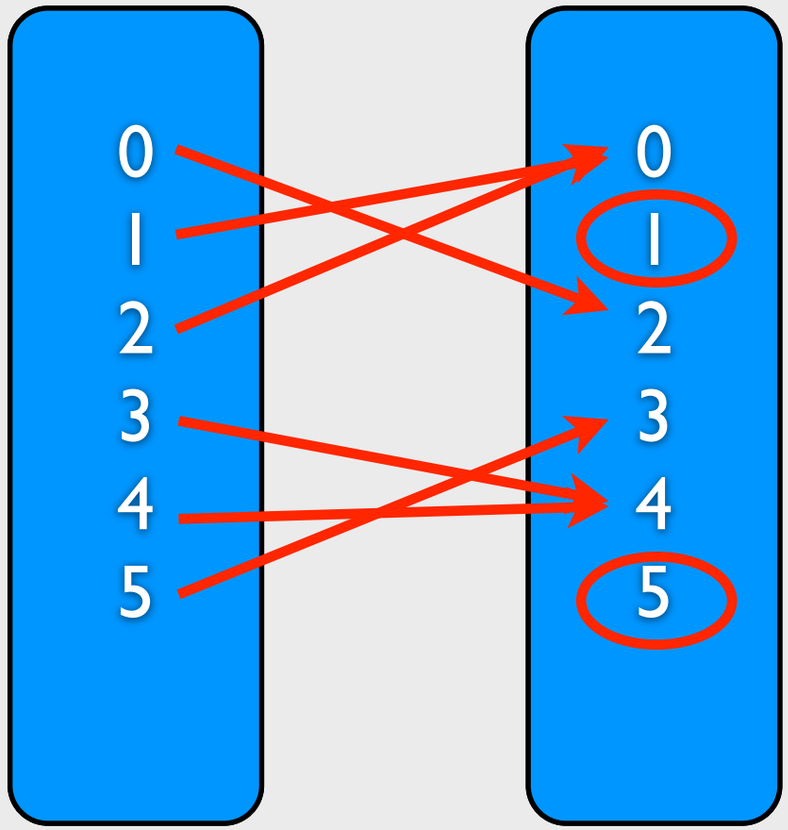




stack

Think over 1to1 example

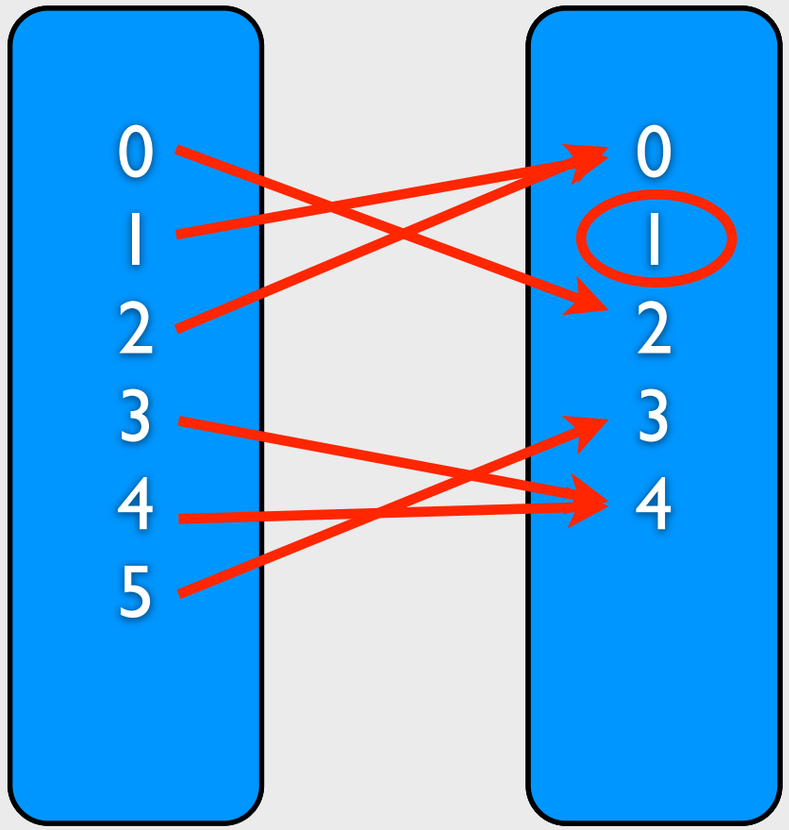
- What if we replace the queue as stack?



stack

Think over 1to1 example

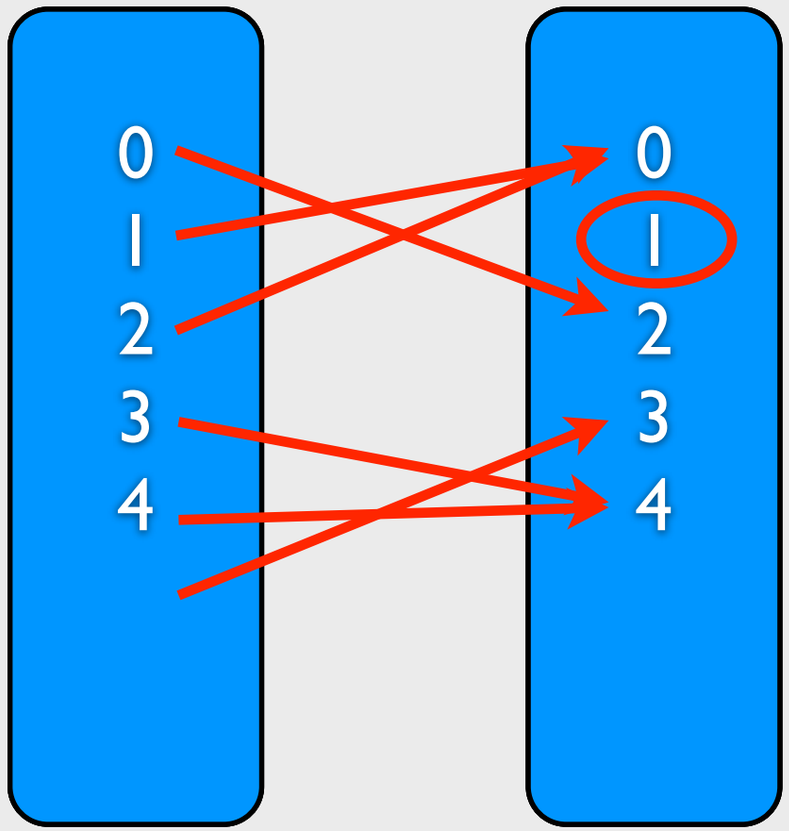
- What if we replace the queue as stack?



stack

Think over 1to1 example

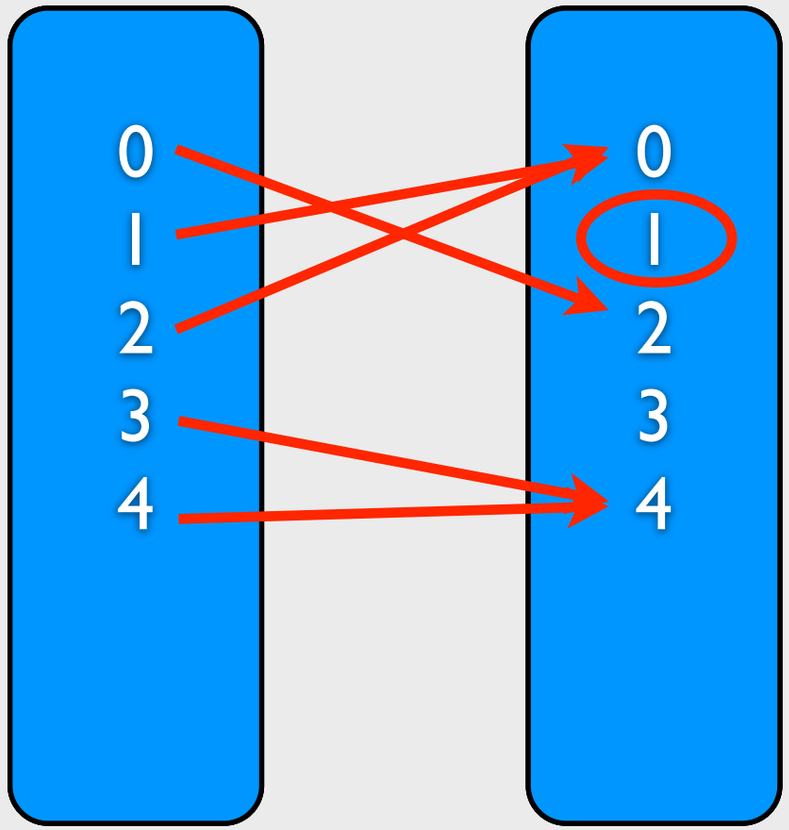
- What if we replace the queue as stack?



stack

Think over 1to1 example

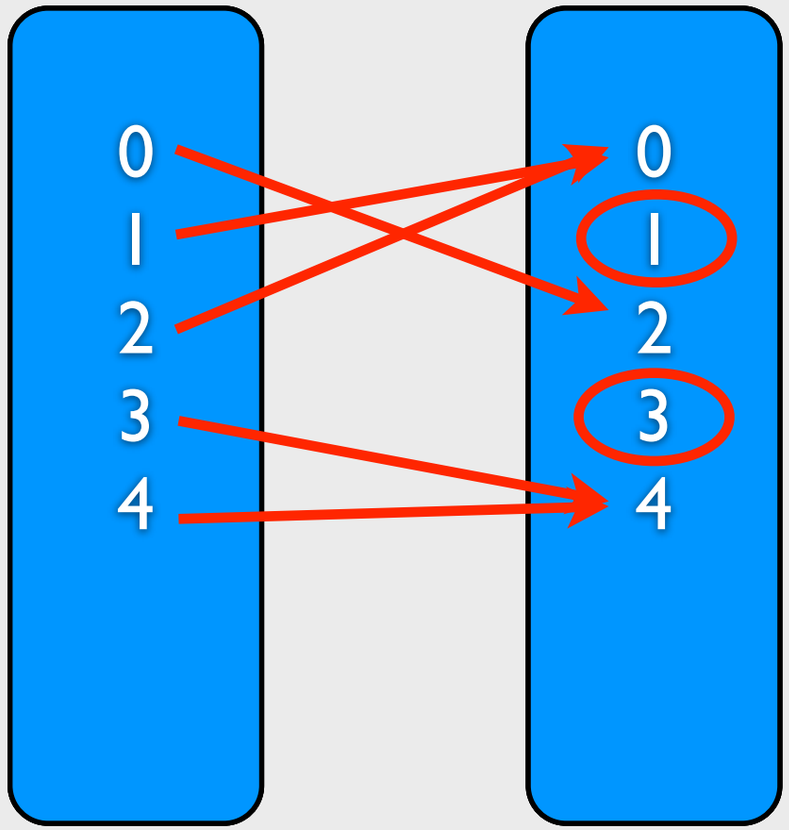
- What if we replace the queue as stack?



stack

Think over 1to1 example

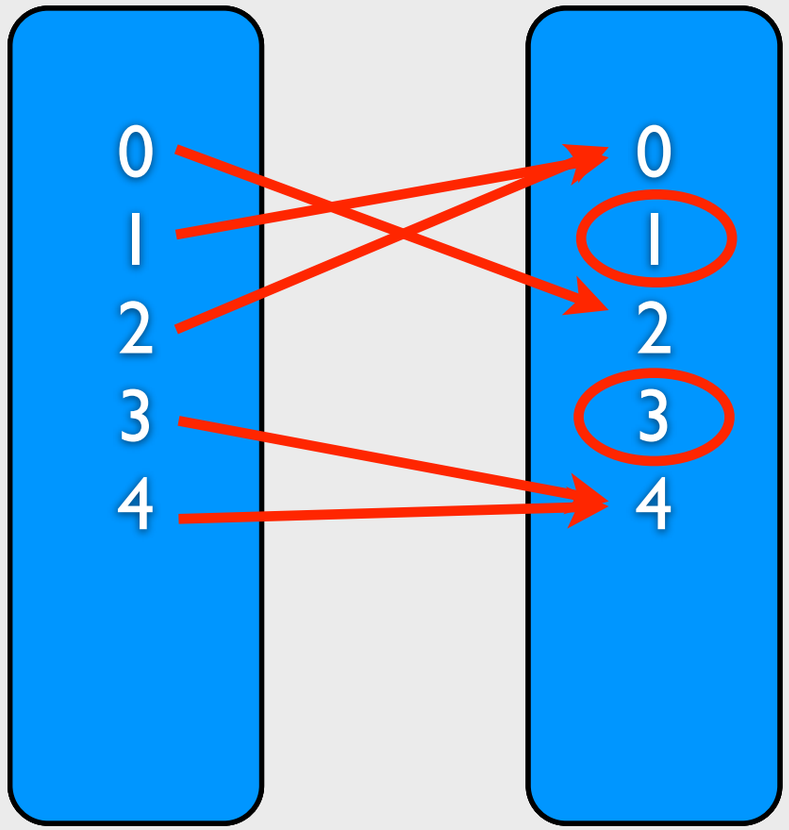
- What if we replace the queue as stack?

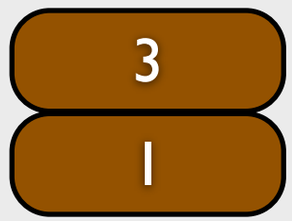


stack

Think over 1to1 example

- What if we replace the queue as stack?

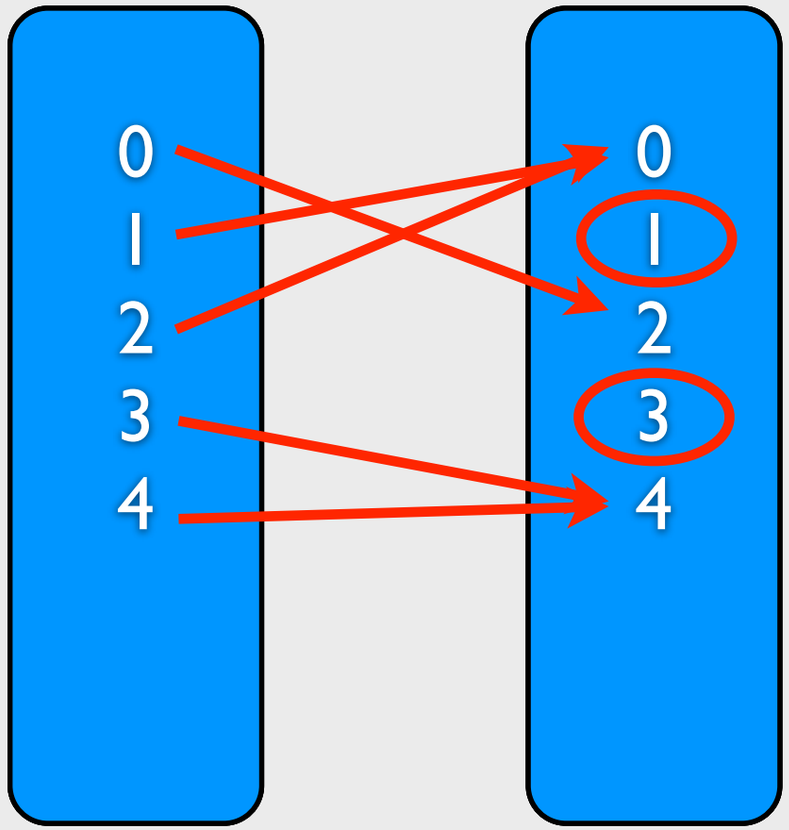




stack

Think over 1to1 example

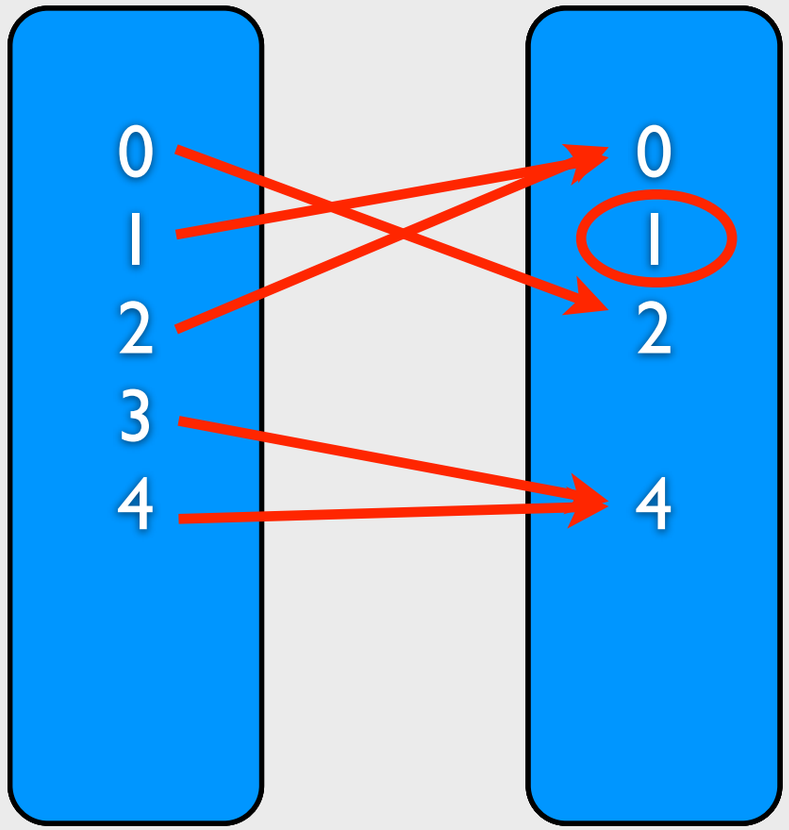
- What if we replace the queue as stack?



stack

Think over 1to1 example

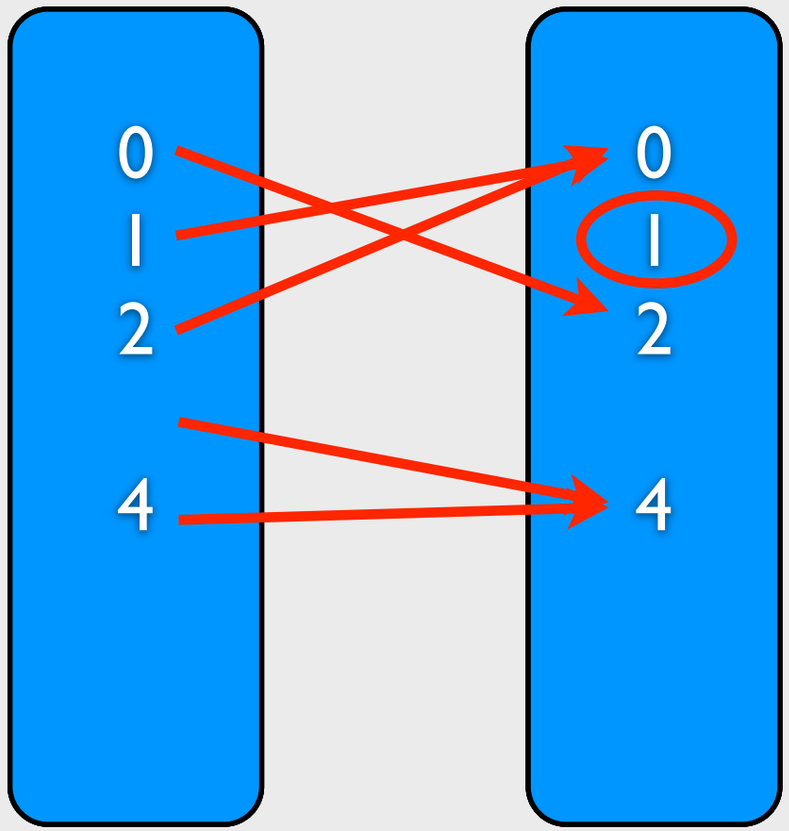
- What if we replace the queue as stack?



stack

Think over 1to1 example

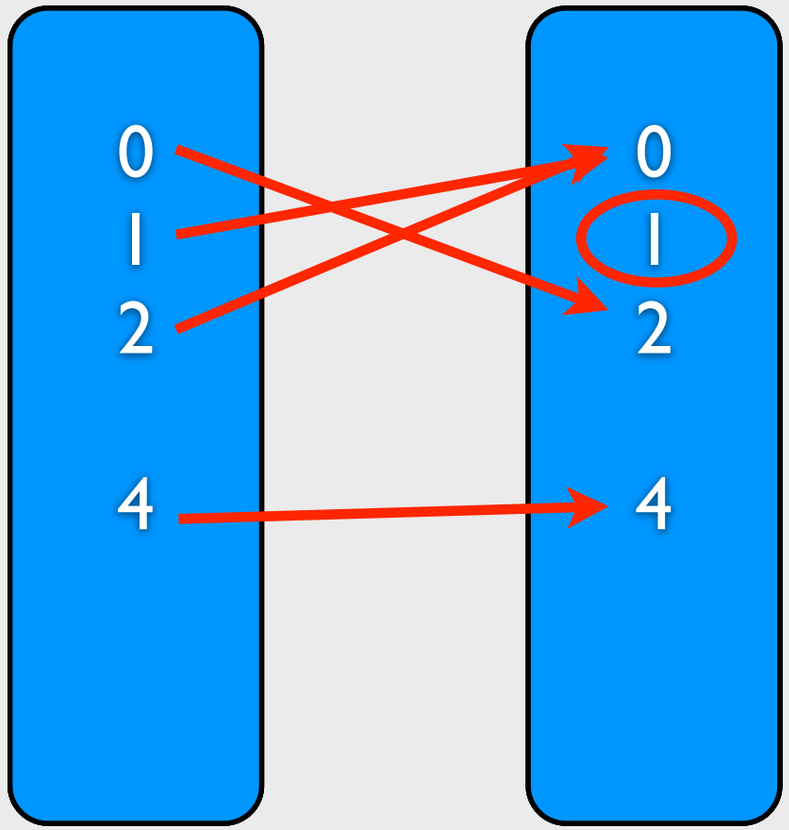
- What if we replace the queue as stack?



stack

Think over 1to1 example

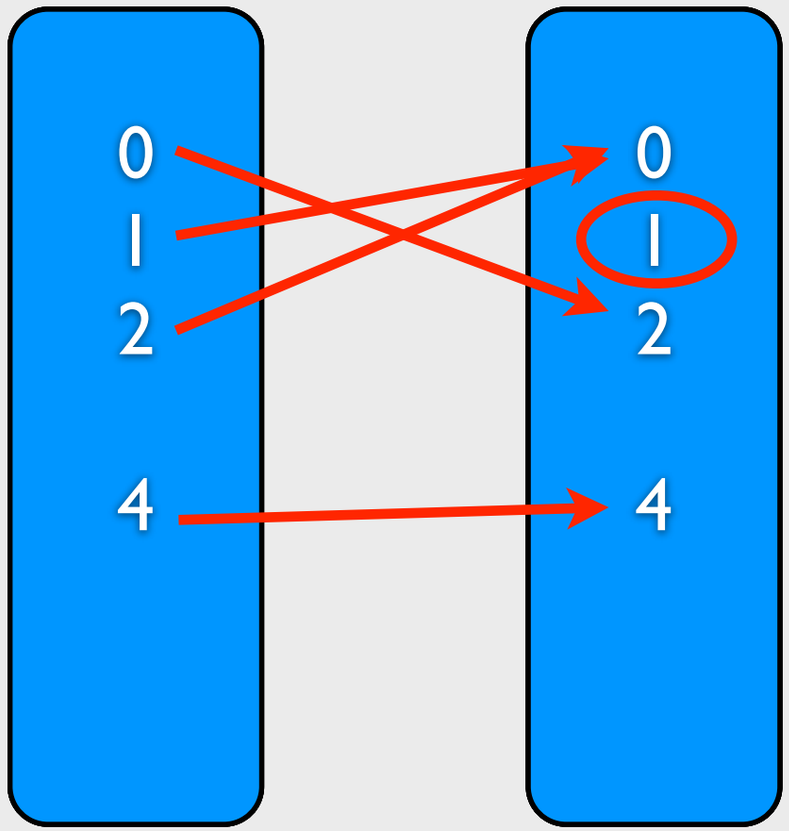
- What if we replace the queue as stack?



stack

Think over 1to1 example

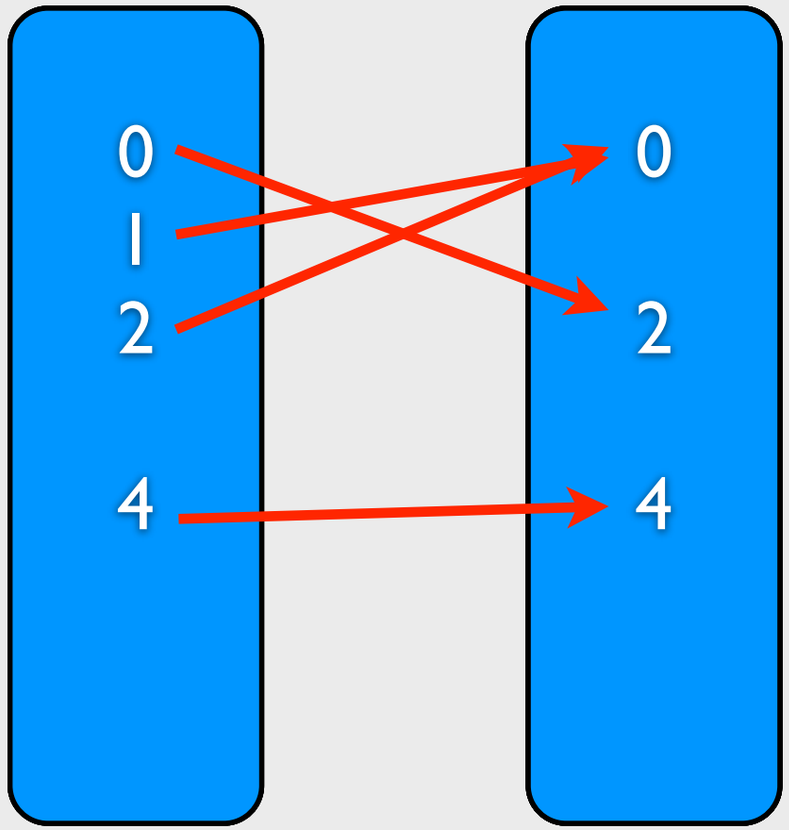
- What if we replace the queue as stack?



stack

Think over 1to1 example

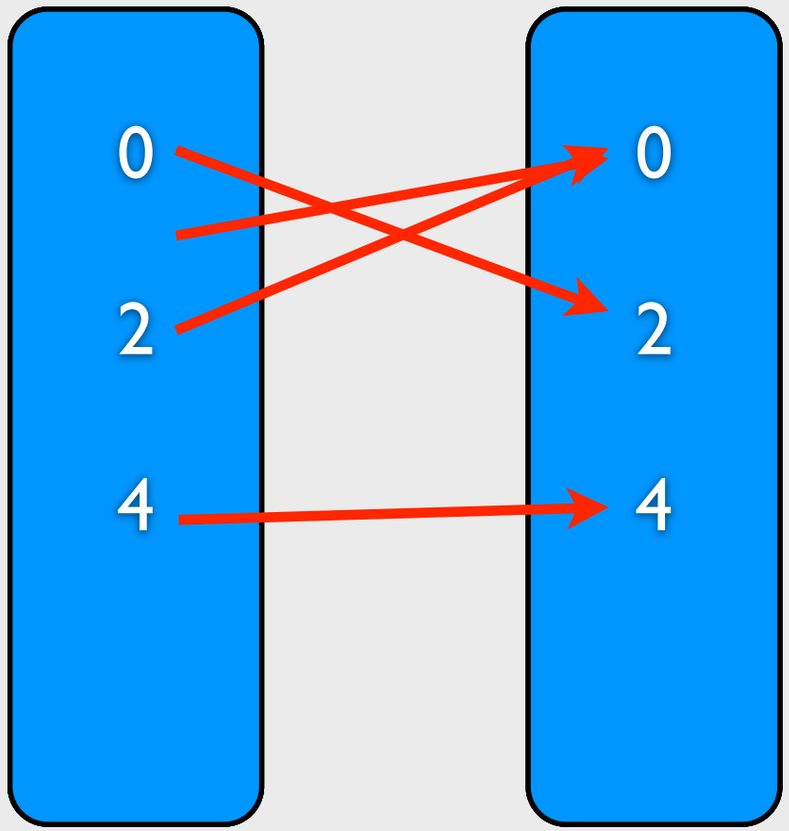
- What if we replace the queue as stack?



stack

Think over 1to1 example

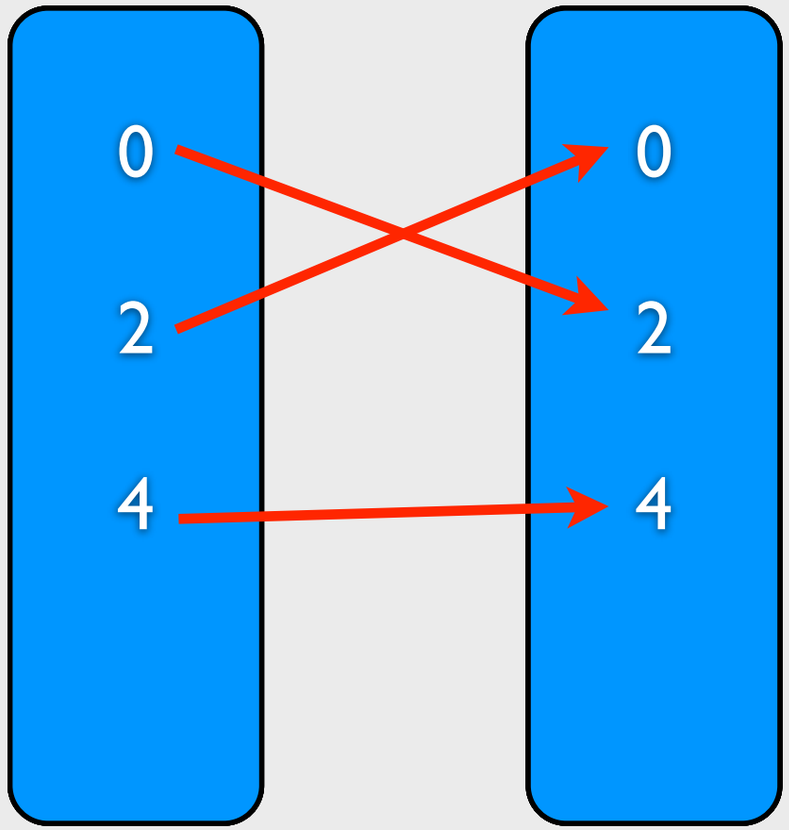
- What if we replace the queue as stack?



stack

Think over 1to1 example

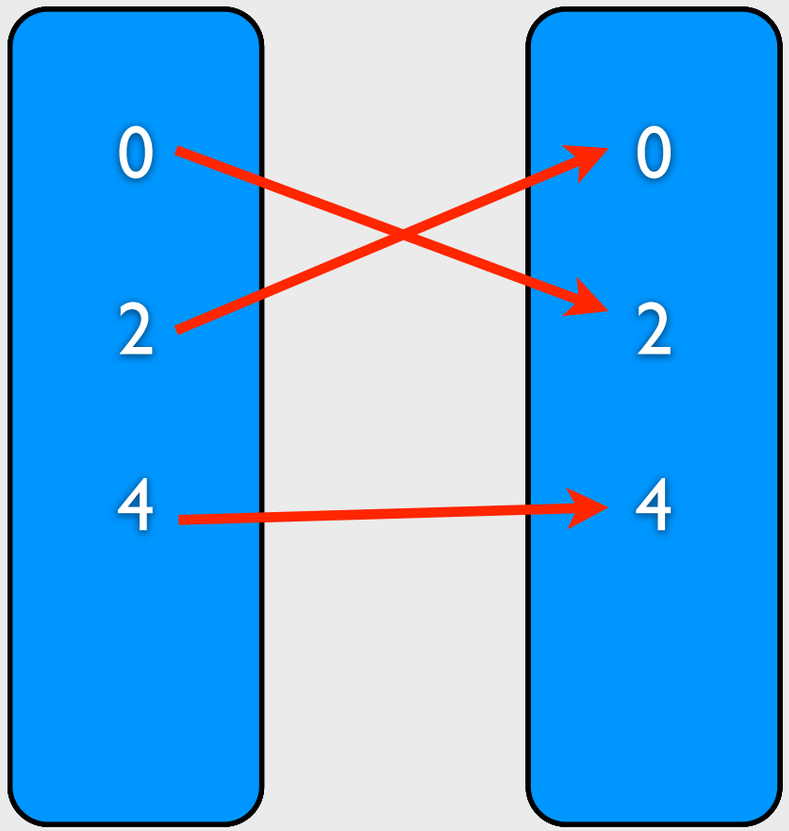
- What if we replace the queue as stack?



stack

Think over 1to1 example

- What if we replace the queue as stack?



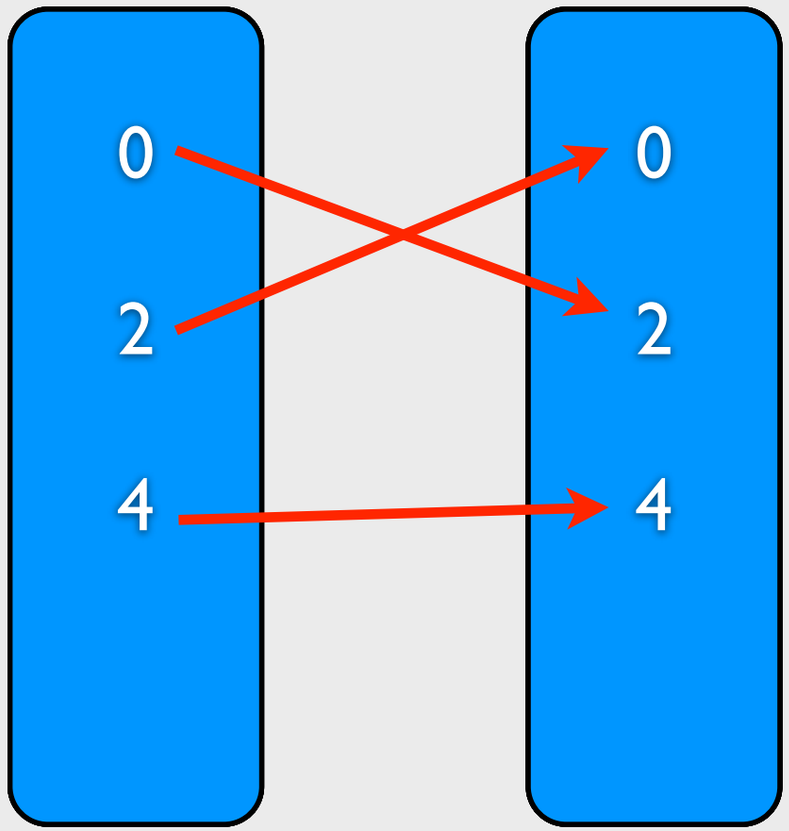
S' = {0, 2, 4}

stack

Think over 1to1 example

- What if we replace the queue as stack?

So, the result is the same!!



S' = {0, 2, 4}

stack

Think over 1to1 example

- What is the worst case? How many times will be executed?
  - the worst case: |S'| = 0
  - corresponding time complexity: O(n)

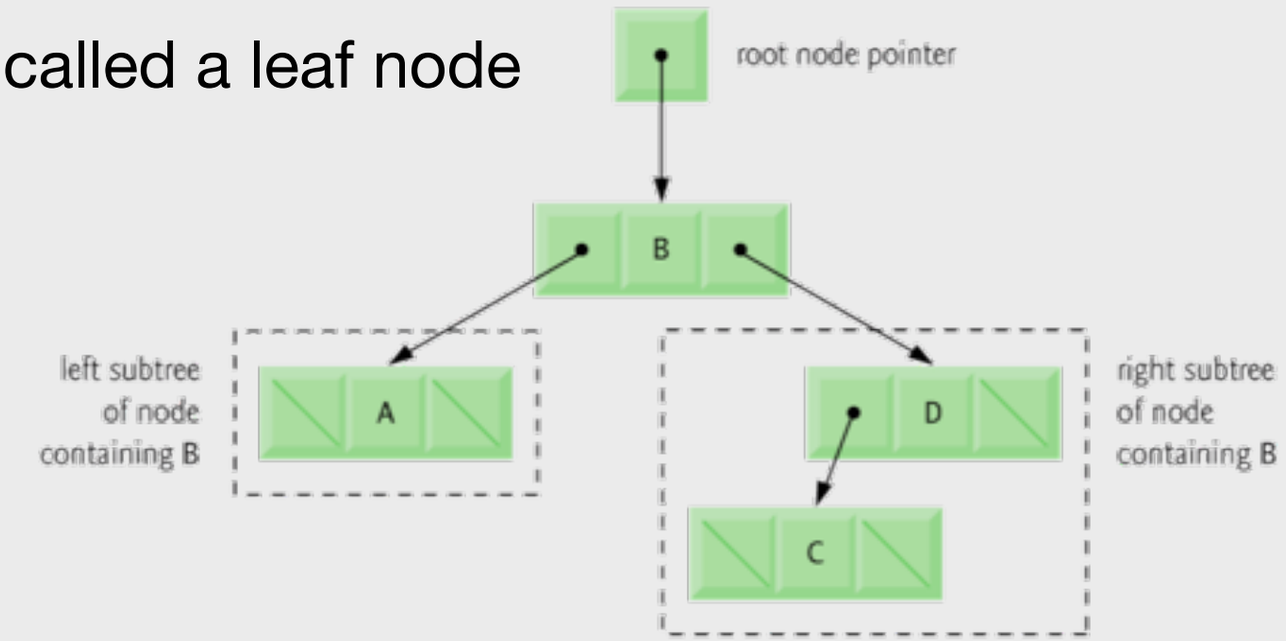
Advanced Tree

Trees

- Tree node contain two or more links
- Binary trees
  - All nodes contain two links
    - None, one, or both of which may be NULL
  - The root node is the first node in a tree
    - Each link in the root node refers to a child
    - A node with no children is called a leaf node

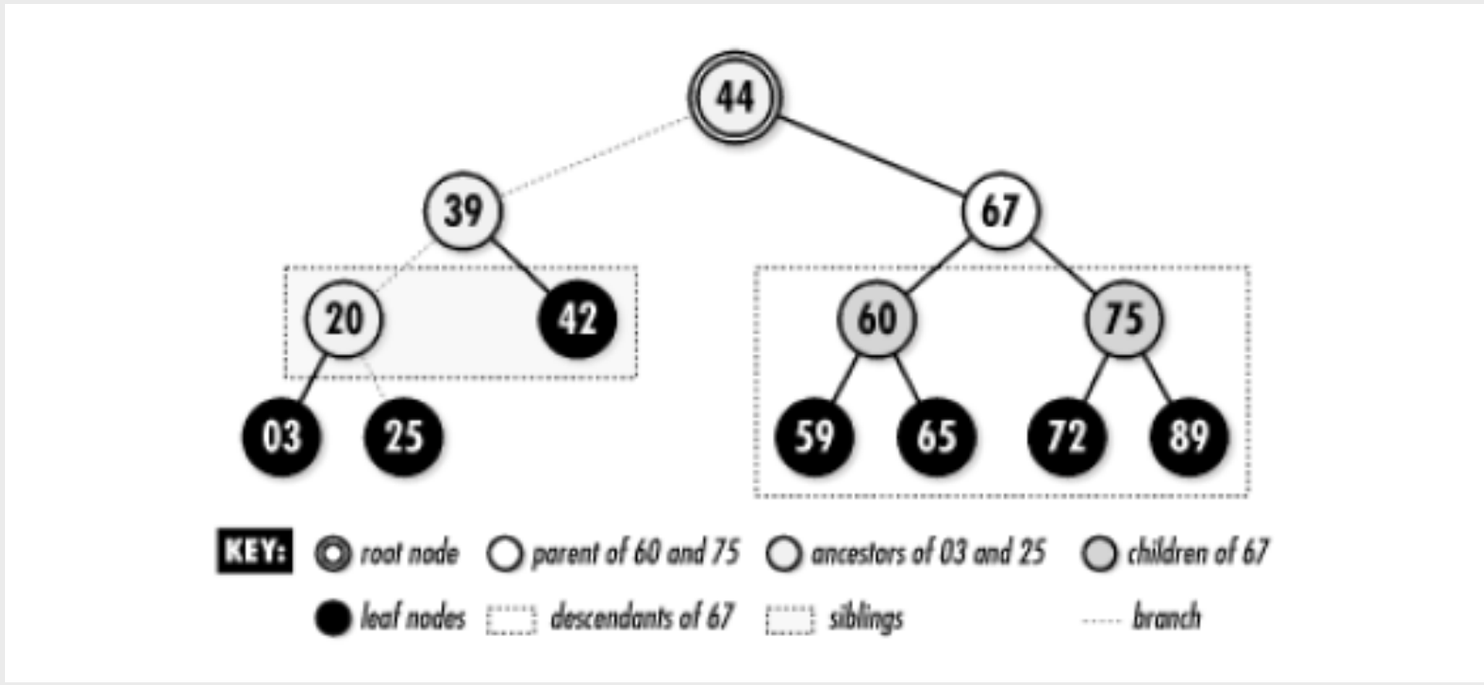
Trees

- Tree node contain two or more links
- Binary trees
  - All nodes contain two links
    - None, one, or both of which may be NULL
  - The root node is the first node in a tree
    - Each link in the root node refers to a child
    - A node with no children is called a leaf node



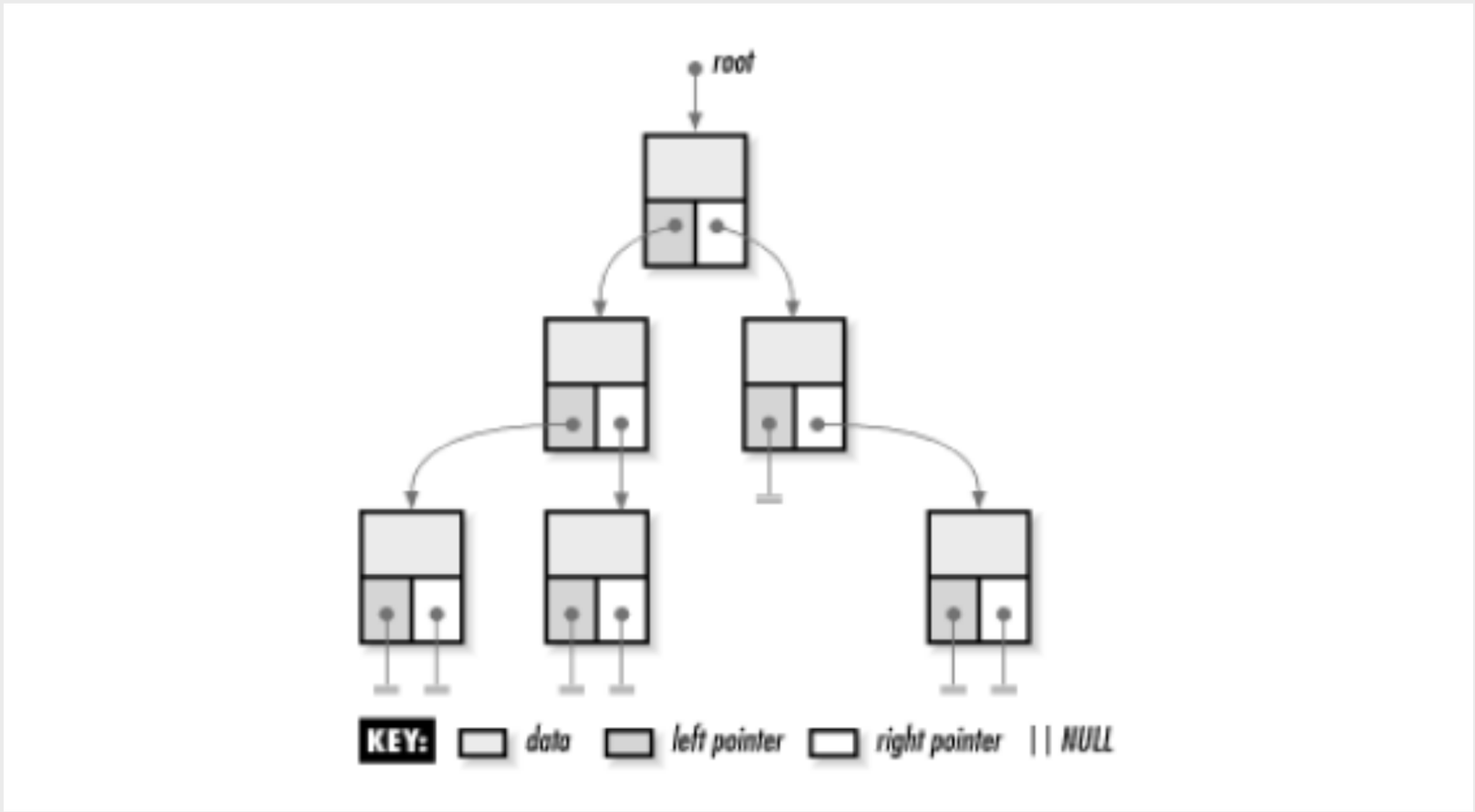
Binary Trees

- Trees containing nodes with up to two children
- Key terms
  - root, leaf, parent, children, ancestor, descendants, siblings, branch

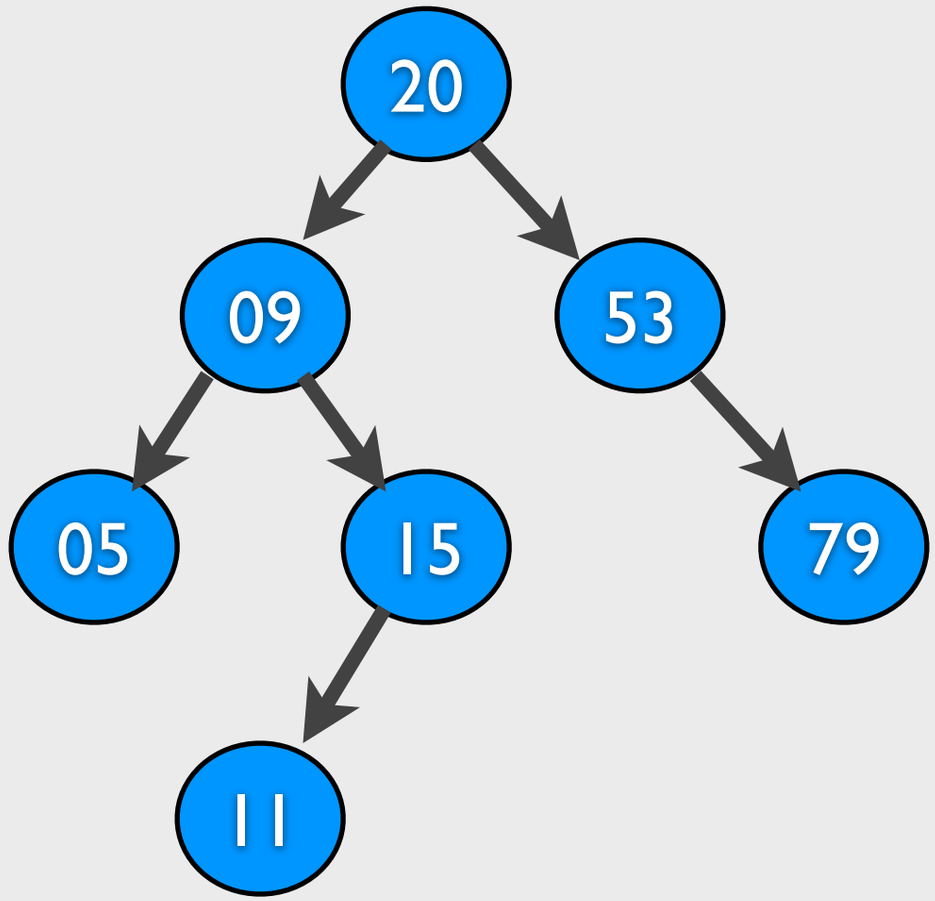


Binary Trees

- Links

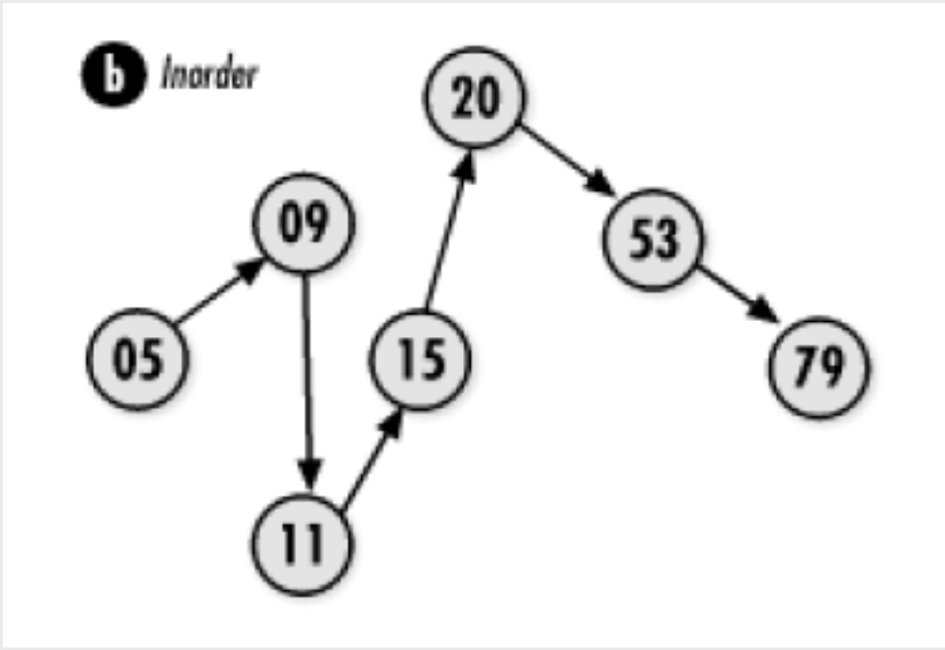


Tree Traversals



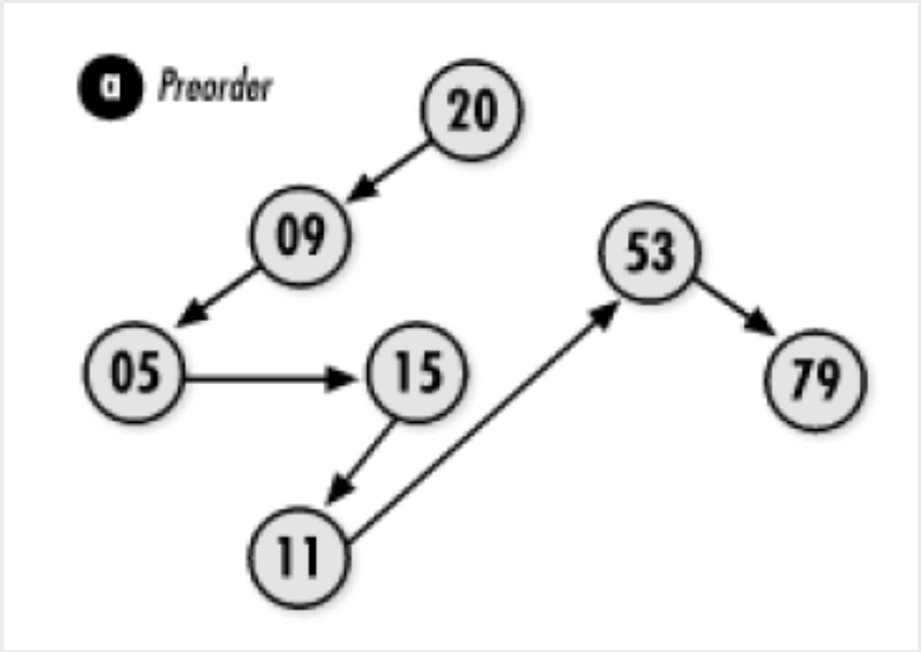
Tree Traversals

- Inorder traversal -- prints the node values in ascending order
  - Traverse the left subtree with an inorder traversal
  - Process the value in the node
  - Traverse the right subtree with an inorder traversal



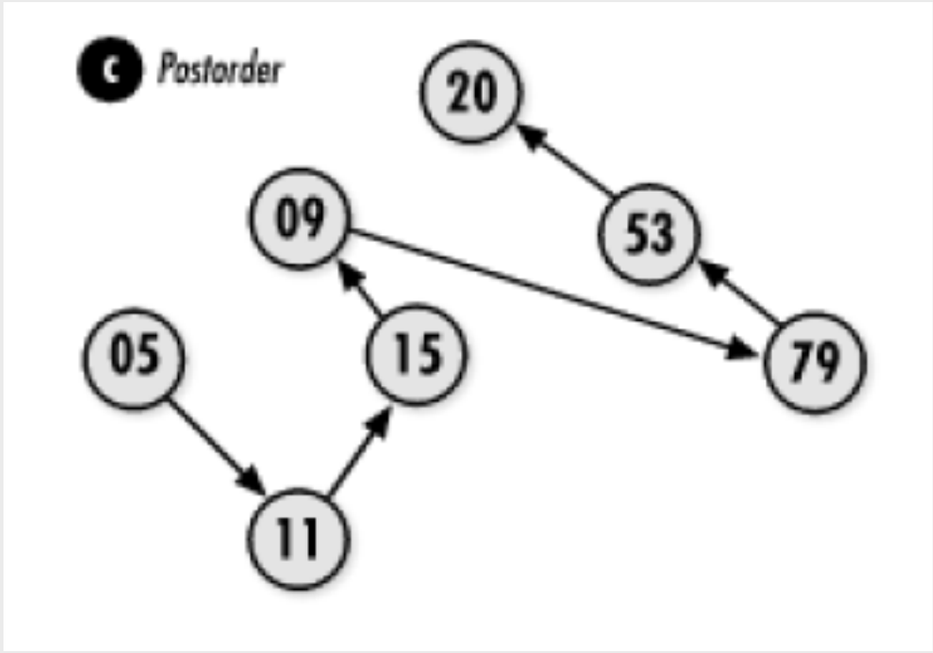
Tree Traversals

- Preorder traversal
  - Process the value in the node
  - Traverse the left subtree with a preorder traversal
  - Traverse the right subtree with a preorder traversal



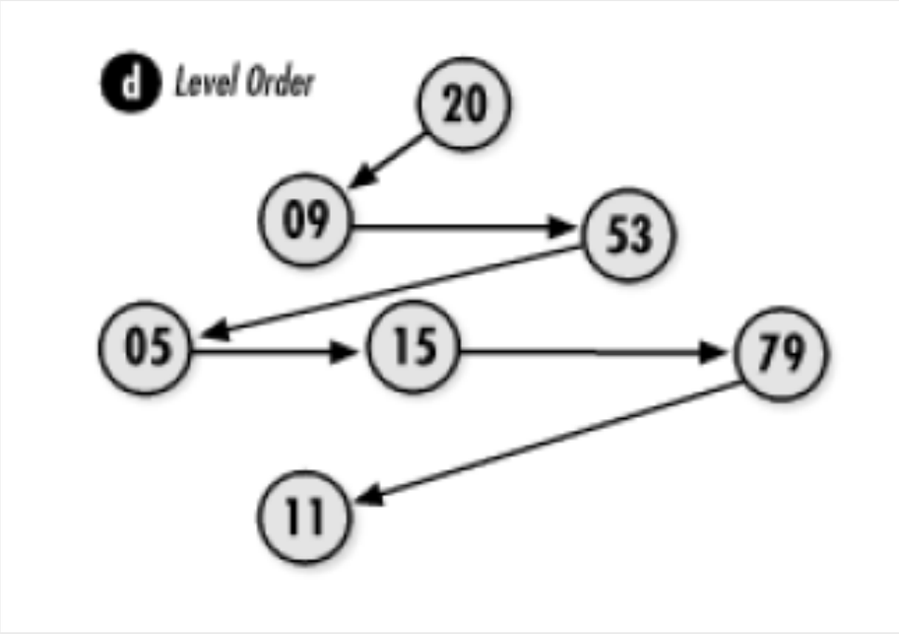
Tree Traversals

- Postorder traversal
  - Traverse the left subtree with a postorder traversal
  - Traverse the right subtree with a postorder traversal
  - Process the value in the node



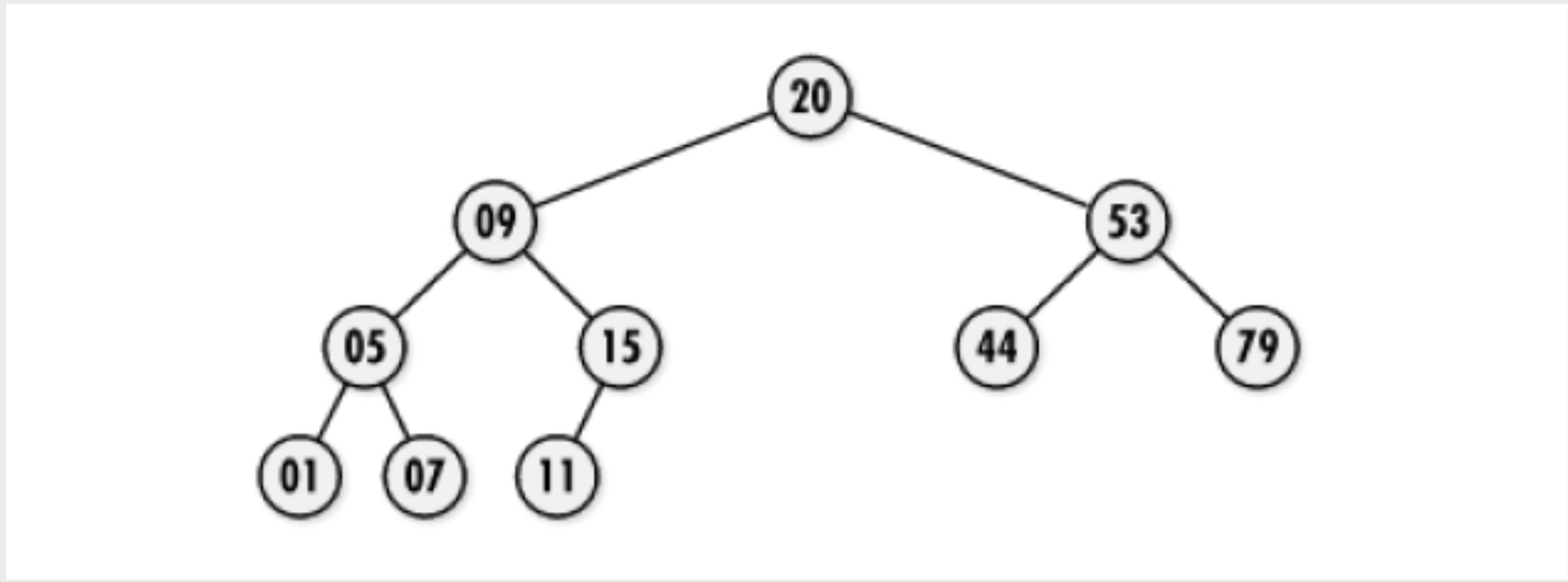
Tree Traversals

- Level order traversal
  - visit its nodes beginning at the root and proceed downward
  - visit the nodes at each level from left to right
  - breadth-first exploration



Tree Balancing

- A process of keeping a tree as short as possible for a given number of nodes
- Make sure that one level of the tree is completely full before allowing a node to exist at the next level
- Left-balanced tree if all leaves occupy only the leftmost positions in the last level



Implementation

- Example: bitree/bitree.h

```
 8 /****************************************************************
 9  *  Define a structure for binary tree nodes.
10  ****************************************************************/
11 typedef struct BiTreeNode_ {
12     void *data;
13     struct BiTreeNode_ *left;
14     struct BiTreeNode_ *right;
15 } BiTreeNode;
16
17 /****************************************************************
18  *  Define a structure for binary trees.
19  ****************************************************************/
20 typedef struct BiTree_ {
21     int size;
22     int (*compare)(const void *key1, const void *key2);
23     void (*destroy)(void *data);
24     BiTreeNode *root;
25 } BiTree;
```

Implementation

- Example: bitree/bitree.h

```
27 /************************************************************************
28  *  ------------------------- Public Interface -------------------------  *
29  ************************************************************************/
30 void bitree_init(BiTree *tree, void (*destroy)(void *data));
31 void bitree_destroy(BiTree *tree);
32 int bitree_ins_left(BiTree *tree, BiTreeNode *node, const void *data);
33 int bitree_ins_right(BiTree *tree, BiTreeNode *node, const void *data);
34 void bitree_rem_left(BiTree *tree, BiTreeNode *node);
35 void bitree_rem_right(BiTree *tree, BiTreeNode *node);
36 int bitree_merge(BiTree *merge, BiTree *left, BiTree *right, const void *data);
37 #define bitree_size(tree) ((tree)->size)
38 #define bitree_root(tree) ((tree)->root)
39 #define bitree_is_eob(node) ((node) == NULL)
40 #define bitree_is_leaf(node) ((node)->left == NULL && (node)->right == NULL)
41 #define bitree_data(node) ((node)->data)
42 #define bitree_left(node) ((node)->left)
43 #define bitree_right(node) ((node)->right)
```

Implementation

- Example: bitree/bitree.c

```c
/**********************************************************
 *  ---------------------------- bitree_init ---------------
 **********************************************************/
void bitree_init(BiTree *tree, void (*destroy)(void *data)) {
    tree->size = 0;
    tree->destroy = destroy;
    tree->root = NULL;
    return;
}
```

```c
/**************************************************************************
 *  ---------------------------- bitree_destroy ---------------------------  *
 **************************************************************************/
void bitree_destroy(BiTree *tree) {
    /**********************************************************************
     *  Remove all the nodes from the tree.
     **********************************************************************/
    bitree_rem_left(tree, NULL);

    /**********************************************************************
     *  No operations are allowed now, but clear the structure as a precaution.
     **********************************************************************/
    memset(tree, 0, sizeof(BiTree));

    return;
}
```

Implementation

```c
int bitree_ins_left(BiTree *tree, BiTreeNode *node, const void *data) {

    BiTreeNode *new_node, **position;

    /************************************************************************
     *  Determine where to insert the node.
     ************************************************************************/
    if (node == NULL) {
        /********************************************************************
         *  Allow insertion at the root only in an empty tree.
         ********************************************************************/
        if (bitree_size(tree) > 0) return -1;
        position = &tree->root;
    } else {
        /********************************************************************
         *  Normally allow insertion only at the end of a branch.
         ********************************************************************/
        if (bitree_left(node) != NULL) return -1;
        position = &node->left;
    }

    /************************************************************************
     *  Allocate storage for the node.
     ************************************************************************/
    if ((new_node = (BiTreeNode *)malloc(sizeof(BiTreeNode))) == NULL) return -1;

    /************************************************************************
     *  Insert the node into the tree.
     ************************************************************************/
    new_node->data = (void *)data;
    new_node->left = NULL;
    new_node->right = NULL;
    *position = new_node;

    /************************************************************************
     *  Adjust the size of the tree to account for the inserted node.
     ************************************************************************/
    tree->size++;

    return 0;
}
```

Implementation

```
130 void bitree_rem_left(BiTree *tree, BiTreeNode *node) {
131
132     BiTreeNode          **position;
133
134     /************************************************************************
135      *  Do not allow removal from an empty tree.
136      ************************************************************************
137     if (bitree_size(tree) == 0) return;
138
139     /************************************************************************
140      *  Determine where to remove nodes.
141      ************************************************************************
142     if (node == NULL)
143         position = &tree->root;
144     else
145         position = &node->left;
146
147     /************************************************************************
148      *  Remove the nodes.
149      ************************************************************************
150     if (*position != NULL) {
151         bitree_rem_left(tree, *position);
152         bitree_rem_right(tree, *position);
153
154         if (tree->destroy != NULL) {
155             /****************************************************************
156              *  Call a user-defined function to free dynamically allocated data.
157              ****************************************************************
158             tree->destroy((*position)->data);
159         }
160
161         free(*position);
162         *position = NULL;
163
164         /****************************************************************
165          *  Adjust the size of the tree to account for the removed node.
166          ****************************************************************
167         tree->size--;
168     }
169     return;
170 }
```

Implementation

- Example: bitree/traverse.h

```
4  #ifndef TRAVERSE_H
5  #define TRAVERSE_H
6
7  #include "bitree.h"
8  #include "list.h"
9
10 /****************************************************************
11 *  ------------------------- Public Interface -------------------
12 ****************************************************************/
13 int preorder(const BiTreeNode *node, List *list);
14 int inorder(const BiTreeNode *node, List *list);
15 int postorder(const BiTreeNode *node, List *list);
16
17 #endif
```

Implementation

- Example: bitree/traverse.c

```
10 int preorder(const BiTreeNode *node, List *list) {
11     if (!bitree_is_eob(node)) {
12         if (list_ins_next(list, list_tail(list), bitree_data(node)) != 0)
13             return -1;
14
15         if (!bitree_is_eob(bitree_left(node)))
16             if (preorder(bitree_left(node), list) != 0)
17                 return -1;
18
19         if (!bitree_is_eob(bitree_right(node)))
20             if (preorder(bitree_right(node), list) != 0)
21                 return -1;
22     }
23     return 0;
24 }
```

Implementation

- Example: bitree/traverse.c

```
29 int inorder(const BiTreeNode *node, List *list) {
30     if (!bitree_is_eob(node)) {
31         if (!bitree_is_eob(bitree_left(node)))
32             if (inorder(bitree_left(node), list) != 0)
33                 return -1;
34 
35         if (list_ins_next(list, list_tail(list), bitree_data(node)) != 0)
36             return -1;
37 
38         if (!bitree_is_eob(bitree_right(node)))
39             if (inorder(bitree_right(node), list) != 0)
40                 return -1;
41     }
42     return 0;
43 }
```

Implementation

- Example: bitree/traverse.c

```
48 int postorder(const BiTreeNode *node, List *list) {
49     if (!bitree_is_eob(node)) {
50         if (!bitree_is_eob(bitree_left(node)))
51             if (postorder(bitree_left(node), list) != 0)
52                 return -1;
53 
54         if (!bitree_is_eob(bitree_right(node)))
55             if (postorder(bitree_right(node), list) != 0)
56                 return -1;
57 
58         if (list_ins_next(list, list_tail(list), bitree_data(node)) != 0)
59             return -1;
60     }
61     return 0;
62 }
```

Implementation

- Example: bitree/test.c

Implementation

- Example: bitree/test.c

```
134     fprintf(stdout, "Inserting some nodes\n");
135     if (insert_int(&tree, 20) != 0) return 1;
136     if (insert_int(&tree, 10) != 0) return 1;
137     if (insert_int(&tree, 30) != 0) return 1;
138     if (insert_int(&tree, 15) != 0) return 1;
139     if (insert_int(&tree, 25) != 0) return 1;
140     if (insert_int(&tree, 70) != 0) return 1;
141     if (insert_int(&tree, 80) != 0) return 1;
142     if (insert_int(&tree, 23) != 0) return 1;
143     if (insert_int(&tree, 26) != 0) return 1;
144     if (insert_int(&tree, 5) != 0) return 1;
145
146     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
147     fprintf(stdout, "(Preorder traversal)\n");
148     print_preorder(bitree_root(&tree));
```

Implementation

- Example: bitree/test.c

```
134     fprintf(stdout, "Inserting some nodes\n");
135     if (insert_int(&tree, 20) != 0) return 1;
136     if (insert_int(&tree, 10) != 0) return 1;
137     if (insert_int(&tree, 30) != 0) return 1;
138     if (insert_int(&tree, 15) != 0) return 1;
139     if (insert_int(&tree, 25) != 0) return 1;
140     if (insert_int(&tree, 70) != 0) return 1;
141     if (insert_int(&tree, 80) != 0) return 1;
142     if (insert_int(&tree, 23) != 0) return 1;
143     if (insert_int(&tree, 26) != 0) return 1;
144     if (insert_int(&tree, 5) != 0) return 1;
145
146     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
147     fprintf(stdout, "(Preorder traversal)\n");
148     print_preorder(bitree_root(&tree));
```

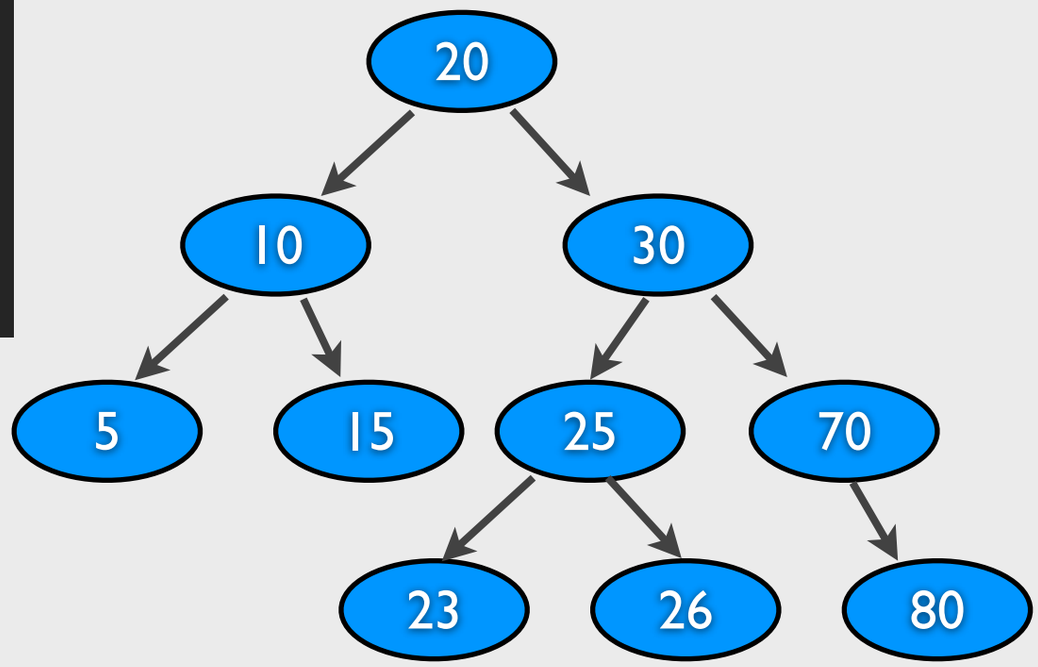
```
Inserting some nodes
Tree size is 10
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=030
Node=025
Node=023
Node=026
Node=070
Node=080
```

Implementation

- Example: bitree/test.c

```
134     fprintf(stdout, "Inserting some nodes\n");
135     if (insert_int(&tree, 20) != 0) return 1;
136     if (insert_int(&tree, 10) != 0) return 1;
137     if (insert_int(&tree, 30) != 0) return 1;
138     if (insert_int(&tree, 15) != 0) return 1;
139     if (insert_int(&tree, 25) != 0) return 1;
140     if (insert_int(&tree, 70) != 0) return 1;
141     if (insert_int(&tree, 80) != 0) return 1;
142     if (insert_int(&tree, 23) != 0) return 1;
143     if (insert_int(&tree, 26) != 0) return 1;
144     if (insert_int(&tree, 5) != 0) return 1;
145
146     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
147     fprintf(stdout, "(Preorder traversal)\n");
148     print_preorder(bitree_root(&tree));
```

```
Inserting some nodes
Tree size is 10
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=030
Node=025
Node=023
Node=026
Node=070
Node=080
```



Implementation

- Example: bitree/test.c

Implementation

- Example: bitree/test.c

```
150     i = 30;
151     if ((node = search_int(&tree, i)) == NULL) {
152         fprintf(stdout, "Could not find %03d\n", i);
153     } else {
154         fprintf(stdout, "Found %03d...Removing the left tree below it\n", i);
155         bitree_rem_left(&tree, node);
156         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
157         fprintf(stdout, "(Preorder traversal)\n");
158         print_preorder(bitree_root(&tree));
159     }
```

Implementation

- Example: bitree/test.c

```
150     i = 30;
151     if ((node = search_int(&tree, i)) == NULL) {
152         fprintf(stdout, "Could not find %03d\n", i);
153     } else {
154         fprintf(stdout, "Found %03d...Removing the left tree below it\n", i);
155         bitree_rem_left(&tree, node);
156         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
157         fprintf(stdout, "(Preorder traversal)\n");
158         print_preorder(bitree_root(&tree));
159     }
```

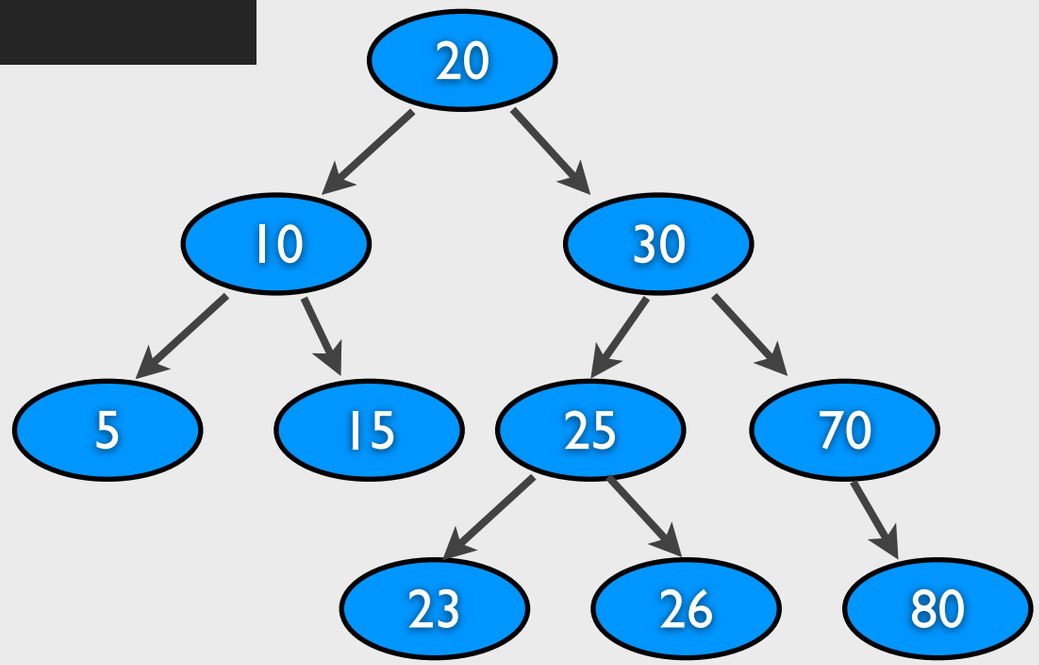
```
Found 030...Removing the left tree below it
Tree size is 7
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=030
Node=070
Node=080
```

Implementation

- Example: bitree/test.c

```
150     i = 30;
151     if ((node = search_int(&tree, i)) == NULL) {
152         fprintf(stdout, "Could not find %03d\n", i);
153     } else {
154         fprintf(stdout, "Found %03d...Removing the left tree below it\n", i);
155         bitree_rem_left(&tree, node);
156         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
157         fprintf(stdout, "(Preorder traversal)\n");
158         print_preorder(bitree_root(&tree));
159     }
```

```
Found 030...Removing the left tree below it
Tree size is 7
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=030
Node=070
Node=080
```

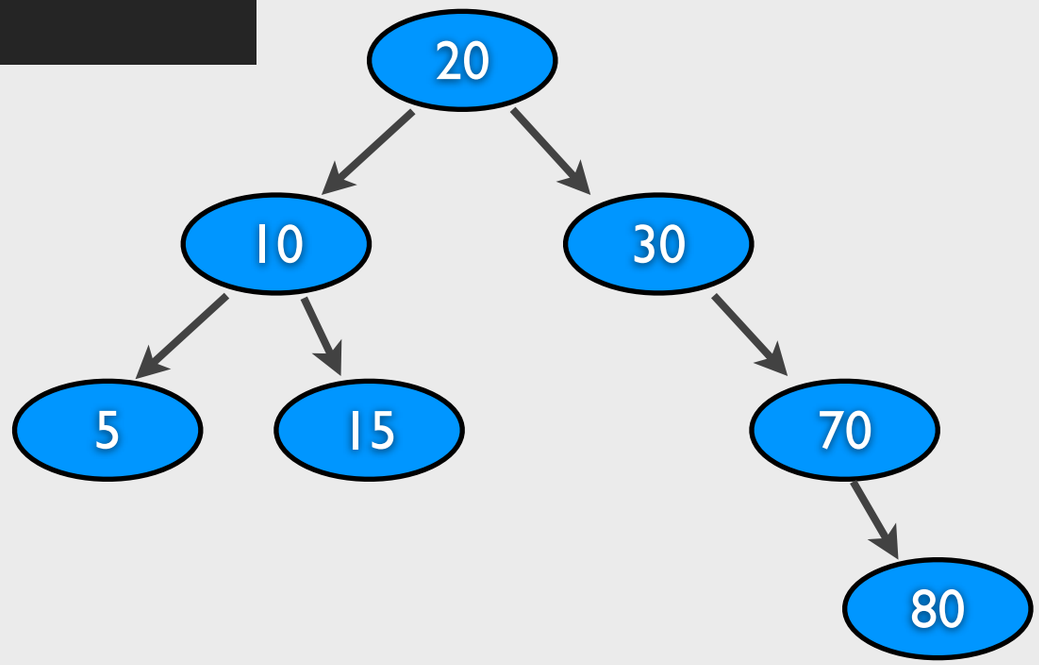


Implementation

- Example: bitree/test.c

```
150     i = 30;
151     if ((node = search_int(&tree, i)) == NULL) {
152         fprintf(stdout, "Could not find %03d\n", i);
153     } else {
154         fprintf(stdout, "Found %03d...Removing the left tree below it\n", i);
155         bitree_rem_left(&tree, node);
156         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
157         fprintf(stdout, "(Preorder traversal)\n");
158         print_preorder(bitree_root(&tree));
159     }
```

```
Found 030...Removing the left tree below it
Tree size is 7
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=030
Node=070
Node=080
```



Implementation

- Example: bitree/test.c

Implementation

- Example: bitree/test.c

```
161     i = 99;
162     if ((node = search_int(&tree, i)) == NULL) {
163         fprintf(stdout, "Could not find %03d\n", i);
164     } else {
165         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
166         bitree_rem_right(&tree, node);
167         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
168         fprintf(stdout, "(Preorder traversal)\n");
169         print_preorder(bitree_root(&tree));
170     }
```

Implementation

- Example: bitree/test.c

```
161     i = 99;
162     if ((node = search_int(&tree, i)) == NULL) {
163         fprintf(stdout, "Could not find %03d\n", i);
164     } else {
165         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
166         bitree_rem_right(&tree, node);
167         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
168         fprintf(stdout, "(Preorder traversal)\n");
169         print_preorder(bitree_root(&tree));
170     }
```

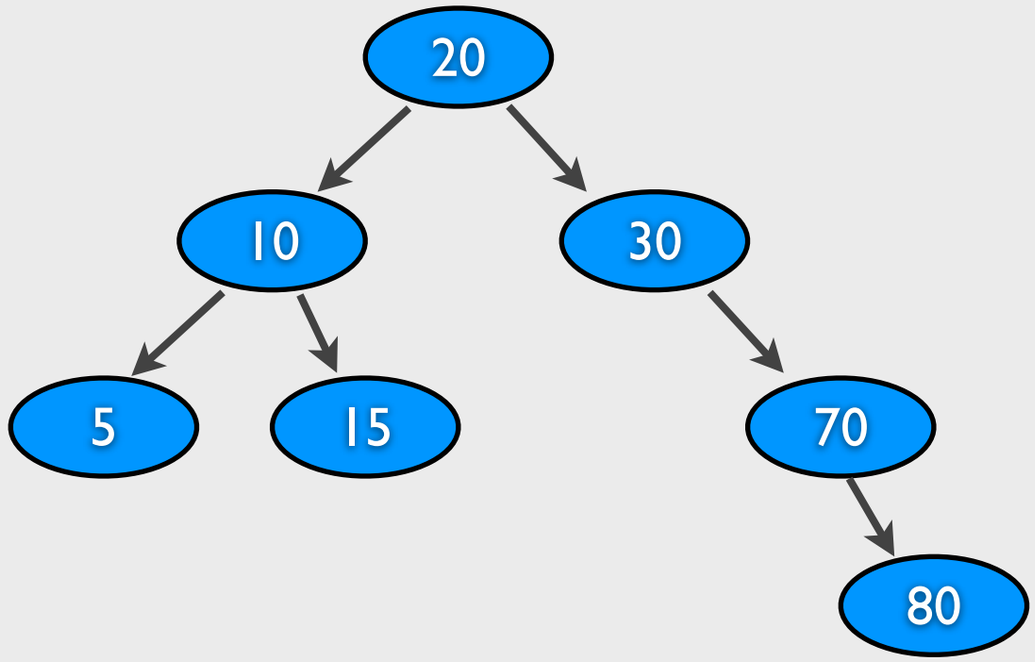
```
Could not find 099
```

Implementation

- Example: bitree/test.c

```
161     i = 99;
162     if ((node = search_int(&tree, i)) == NULL) {
163         fprintf(stdout, "Could not find %03d\n", i);
164     } else {
165         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
166         bitree_rem_right(&tree, node);
167         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
168         fprintf(stdout, "(Preorder traversal)\n");
169         print_preorder(bitree_root(&tree));
170     }
```

```
Could not find 099
```



Implementation

- Example: bitree/test.c

Implementation

- Example: bitree/test.c

```
172     i = 20;
173     if ((node = search_int(&tree, i)) == NULL) {
174         fprintf(stdout, "Could not find %03d\n", i);
175     } else {
176         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
177         bitree_rem_right(&tree, node);
178         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
179         fprintf(stdout, "(Preorder traversal)\n");
180         print_preorder(bitree_root(&tree));
181     }
```

Implementation

- Example: bitree/test.c

```
172     i = 20;
173     if ((node = search_int(&tree, i)) == NULL) {
174         fprintf(stdout, "Could not find %03d\n", i);
175     } else {
176         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
177         bitree_rem_right(&tree, node);
178         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
179         fprintf(stdout, "(Preorder traversal)\n");
180         print_preorder(bitree_root(&tree));
181     }
```

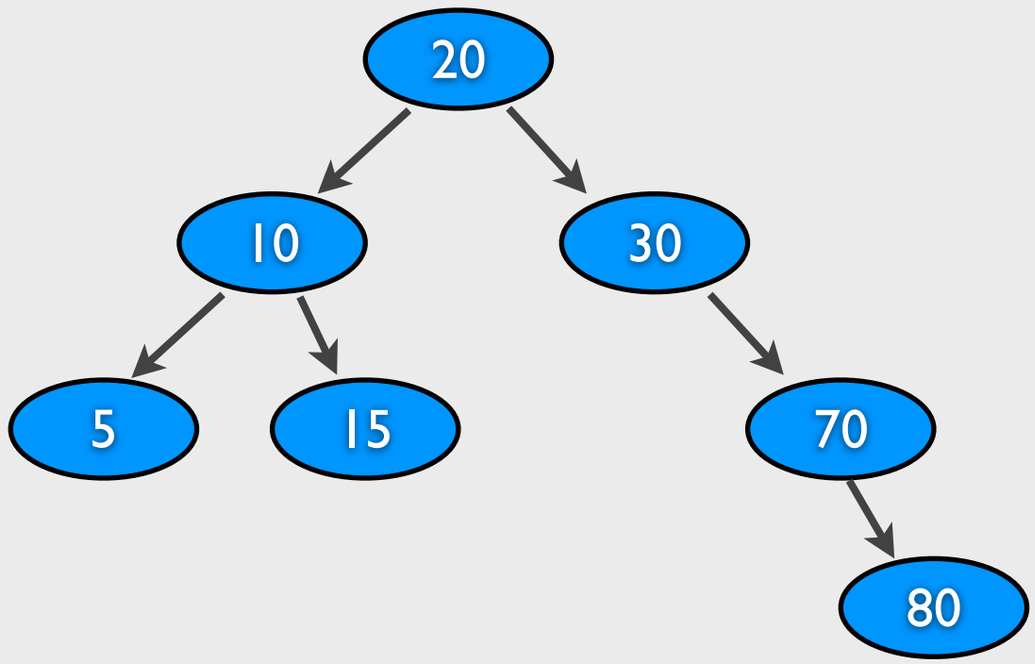
```
Found 020...Removing the right tree below it
Tree size is 4
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
```

Implementation

- Example: bitree/test.c

```
172     i = 20;
173     if ((node = search_int(&tree, i)) == NULL) {
174         fprintf(stdout, "Could not find %03d\n", i);
175     } else {
176         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
177         bitree_rem_right(&tree, node);
178         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
179         fprintf(stdout, "(Preorder traversal)\n");
180         print_preorder(bitree_root(&tree));
181     }
```

```
Found 020...Removing the right tree below it
Tree size is 4
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
```

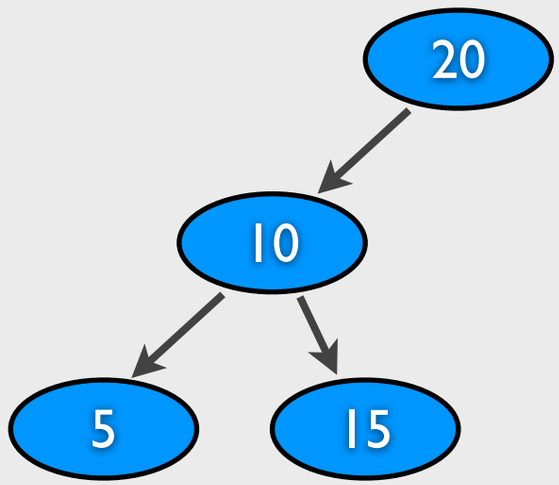


Implementation

- Example: bitree/test.c

```
172     i = 20;
173     if ((node = search_int(&tree, i)) == NULL) {
174         fprintf(stdout, "Could not find %03d\n", i);
175     } else {
176         fprintf(stdout, "Found %03d...Removing the right tree below it\n", i);
177         bitree_rem_right(&tree, node);
178         fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
179         fprintf(stdout, "(Preorder traversal)\n");
180         print_preorder(bitree_root(&tree));
181     }
```

```
Found 020...Removing the right tree below it
Tree size is 4
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
```



Implementation

- Example: bitree/test.c

```
Testing bitree_is_leaf...Value=0 (1=Yes)
Testing bitree_is_leaf...Value=0 (1=Yes)
Testing bitree_is_leaf...Value=1 (1=Yes)
Testing bitree_is_leaf...Value=1 (1=Yes)
```

Implementation

- Example: bitree/test.c

```
183     i = bitree_is_leaf(bitree_root(&tree));
184     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (0=OK)\n", i);
185     i = bitree_is_leaf(bitree_left((bitree_root(&tree))));
186     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (0=OK)\n", i);
187     i = bitree_is_leaf(bitree_left(bitree_left((bitree_root(&tree)))));
188     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (1=OK)\n", i);
189     i = bitree_is_leaf(bitree_right(bitree_left((bitree_root(&tree)))));
190     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (1=OK)\n", i);
```

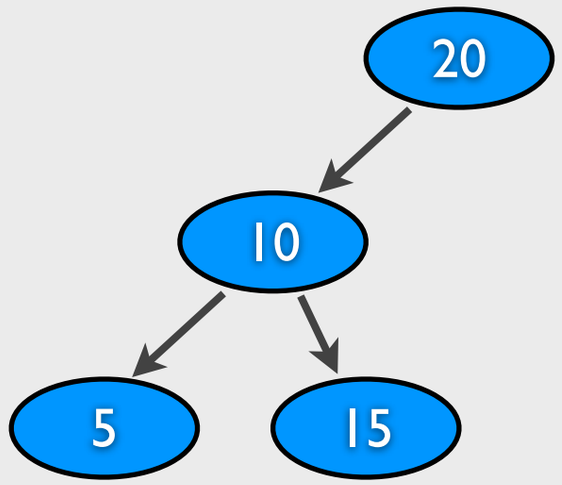
```
Testing bitree_is_leaf...Value=0 (1=Yes)
Testing bitree_is_leaf...Value=0 (1=Yes)
Testing bitree_is_leaf...Value=1 (1=Yes)
Testing bitree_is_leaf...Value=1 (1=Yes)
```

Implementation

- Example: bitree/test.c

```
183     i = bitree_is_leaf(bitree_root(&tree));
184     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (0=OK)\n", i);
185     i = bitree_is_leaf(bitree_left((bitree_root(&tree))));
186     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (0=OK)\n", i);
187     i = bitree_is_leaf(bitree_left(bitree_left((bitree_root(&tree)))));
188     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (1=OK)\n", i);
189     i = bitree_is_leaf(bitree_right(bitree_left((bitree_root(&tree)))));
190     fprintf(stdout, "Testing bitree_is_leaf...Value=%d (1=OK)\n", i);
```

```
Testing bitree_is_leaf...Value=0 (1=Yes)
Testing bitree_is_leaf...Value=0 (1=Yes)
Testing bitree_is_leaf...Value=1 (1=Yes)
Testing bitree_is_leaf...Value=1 (1=Yes)
```



Implementation

- Example: bitree/test.c

Implementation

- Example: bitree/test.c

```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```

Implementation

- Example: bitree/test.c

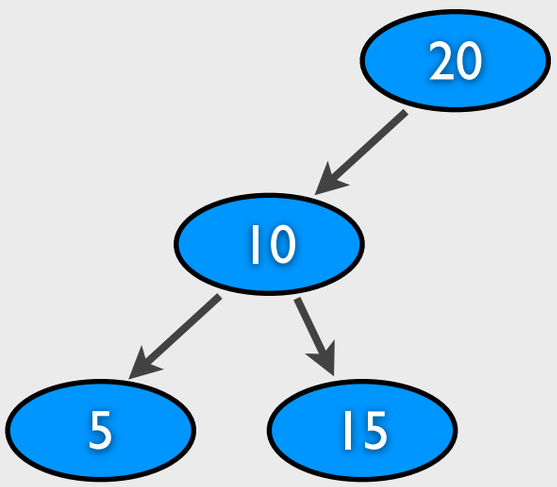
```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```

```
Inserting some nodes
Tree size is 8
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=011
Node=055
Node=044
Node=077
(Inorder traversal)
Node=005
Node=010
Node=011
Node=015
Node=020
Node=044
Node=055
Node=077
(Postorder traversal)
Node=005
Node=011
Node=015
Node=010
Node=044
Node=077
Node=055
Node=020
```

Implementation

- Example: bitree/test.c

```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```

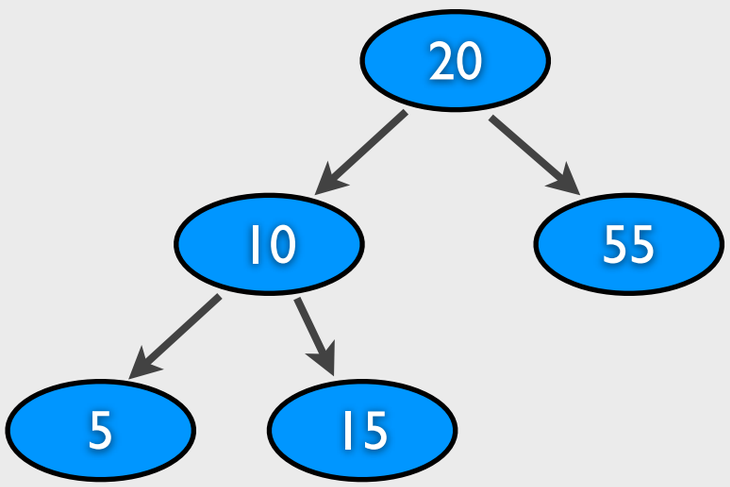



```
Inserting some nodes
Tree size is 8
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=011
Node=055
Node=044
Node=077
(Inorder traversal)
Node=005
Node=010
Node=011
Node=015
Node=020
Node=044
Node=055
Node=077
(Postorder traversal)
Node=005
Node=011
Node=015
Node=010
Node=044
Node=077
Node=055
Node=020
```

Implementation

- Example: bitree/test.c

```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```

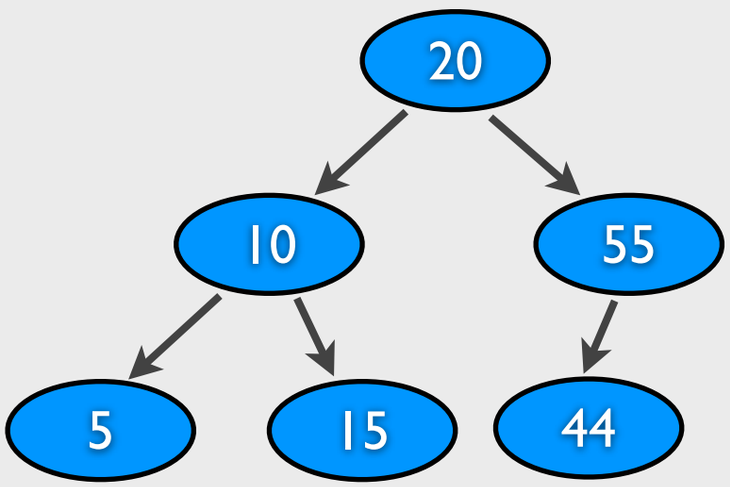



```
Inserting some nodes
Tree size is 8
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=011
Node=055
Node=044
Node=077
(Inorder traversal)
Node=005
Node=010
Node=011
Node=015
Node=020
Node=044
Node=055
Node=077
(Postorder traversal)
Node=005
Node=011
Node=015
Node=010
Node=044
Node=077
Node=055
Node=020
```

Implementation

- Example: bitree/test.c

```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```

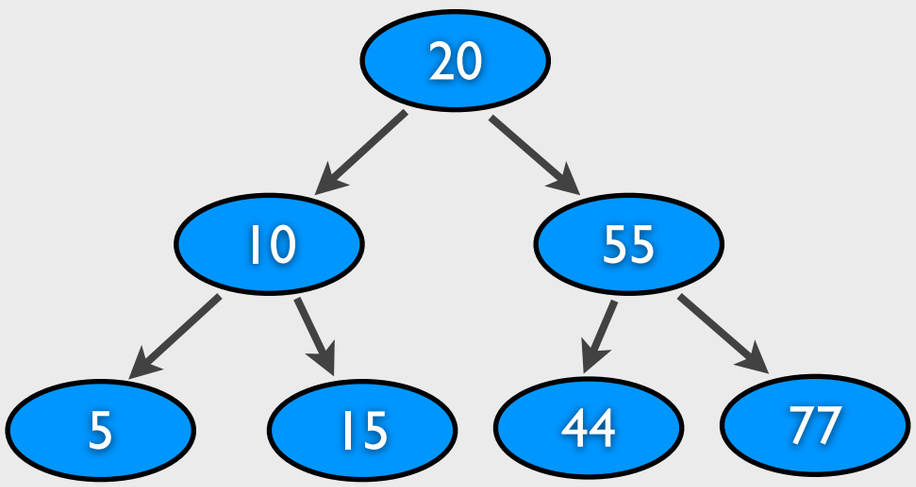



```
Inserting some nodes
Tree size is 8
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=011
Node=055
Node=044
Node=077
(Inorder traversal)
Node=005
Node=010
Node=011
Node=015
Node=020
Node=044
Node=055
Node=077
(Postorder traversal)
Node=005
Node=011
Node=015
Node=010
Node=044
Node=077
Node=055
Node=020
```

Implementation

- Example: bitree/test.c

```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```

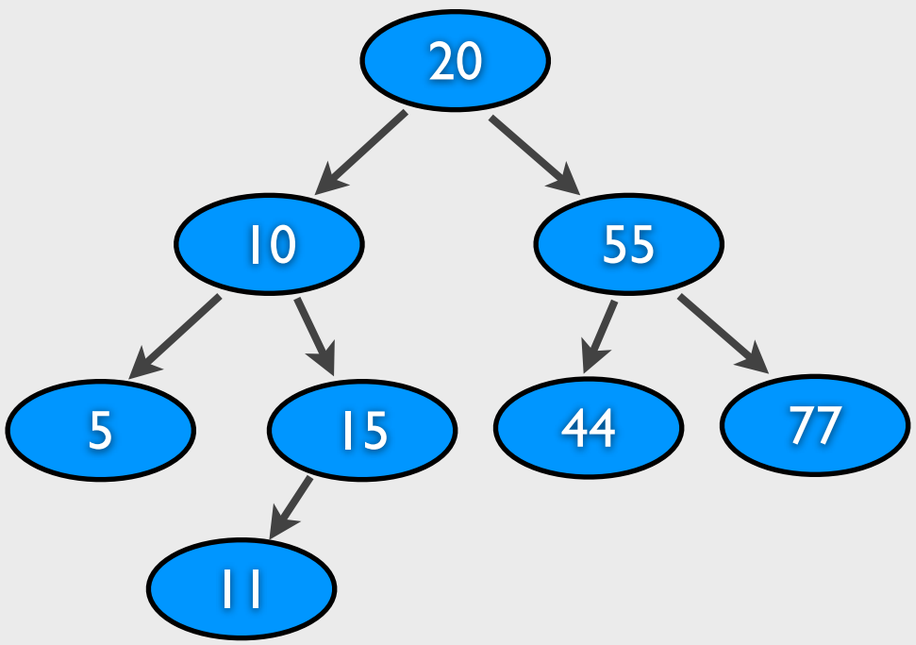



```
Inserting some nodes
Tree size is 8
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=011
Node=055
Node=044
Node=077
(Inorder traversal)
Node=005
Node=010
Node=011
Node=015
Node=020
Node=044
Node=055
Node=077
(Postorder traversal)
Node=005
Node=011
Node=015
Node=010
Node=044
Node=077
Node=055
Node=020
```

Implementation

- Example: bitree/test.c

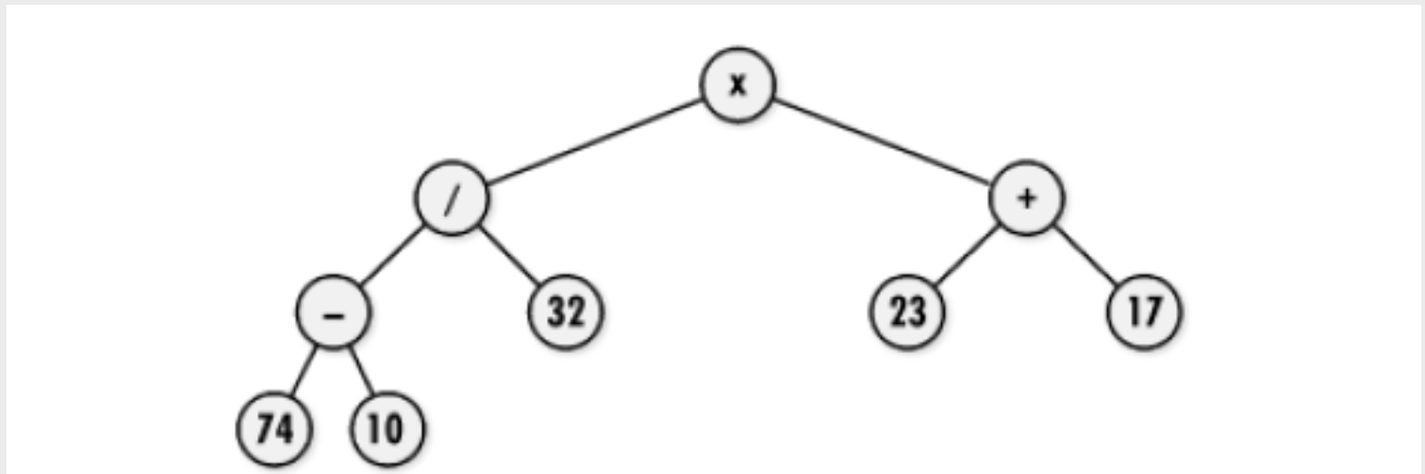
```
192     fprintf(stdout, "Inserting some nodes\n");
193     if (insert_int(&tree, 55) != 0) return 1;
194     if (insert_int(&tree, 44) != 0) return 1;
195     if (insert_int(&tree, 77) != 0) return 1;
196     if (insert_int(&tree, 11) != 0) return 1;
197     fprintf(stdout, "Tree size is %d\n", bitree_size(&tree));
198     fprintf(stdout, "(Preorder traversal)\n");
199     print_preorder(bitree_root(&tree));
200     fprintf(stdout, "(Inorder traversal)\n");
201     print_inorder(bitree_root(&tree));
202     fprintf(stdout, "(Postorder traversal)\n");
203     print_postorder(bitree_root(&tree));
```




```
Inserting some nodes
Tree size is 8
(Preorder traversal)
Node=020
Node=010
Node=005
Node=015
Node=011
Node=055
Node=044
Node=077
(Inorder traversal)
Node=005
Node=010
Node=011
Node=015
Node=020
Node=044
Node=055
Node=077
(Postorder traversal)
Node=005
Node=011
Node=015
Node=010
Node=044
Node=077
Node=055
Node=020
```

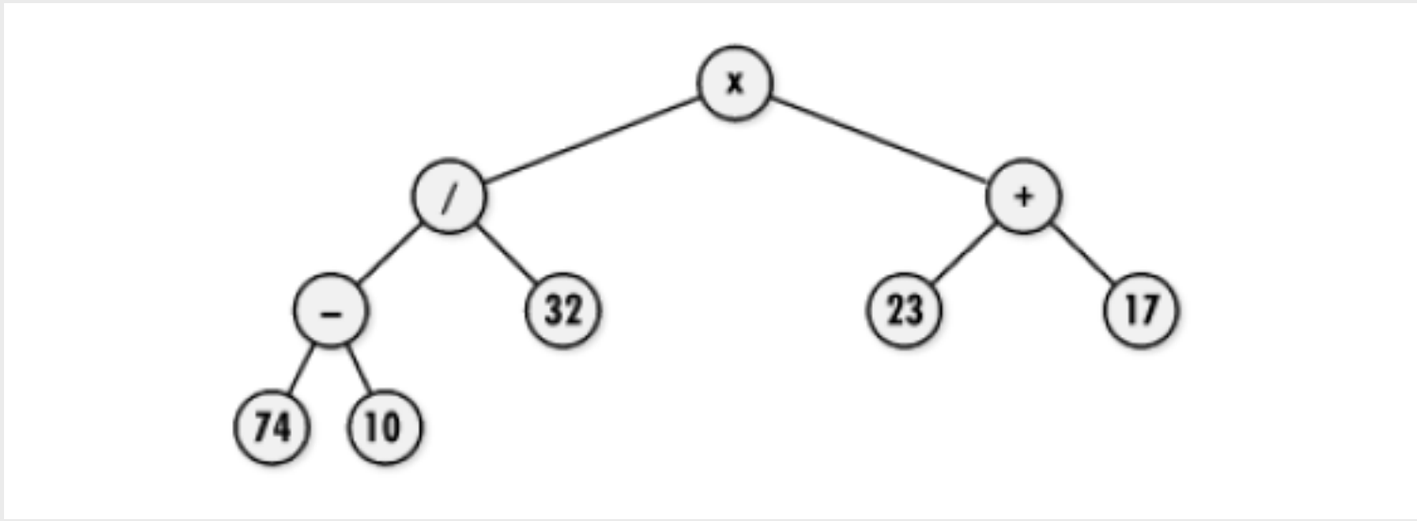
Binary Tree Example

- Expression Processing
  - ex: (((74 - 10) / 32) * (23 + 17)) = 80



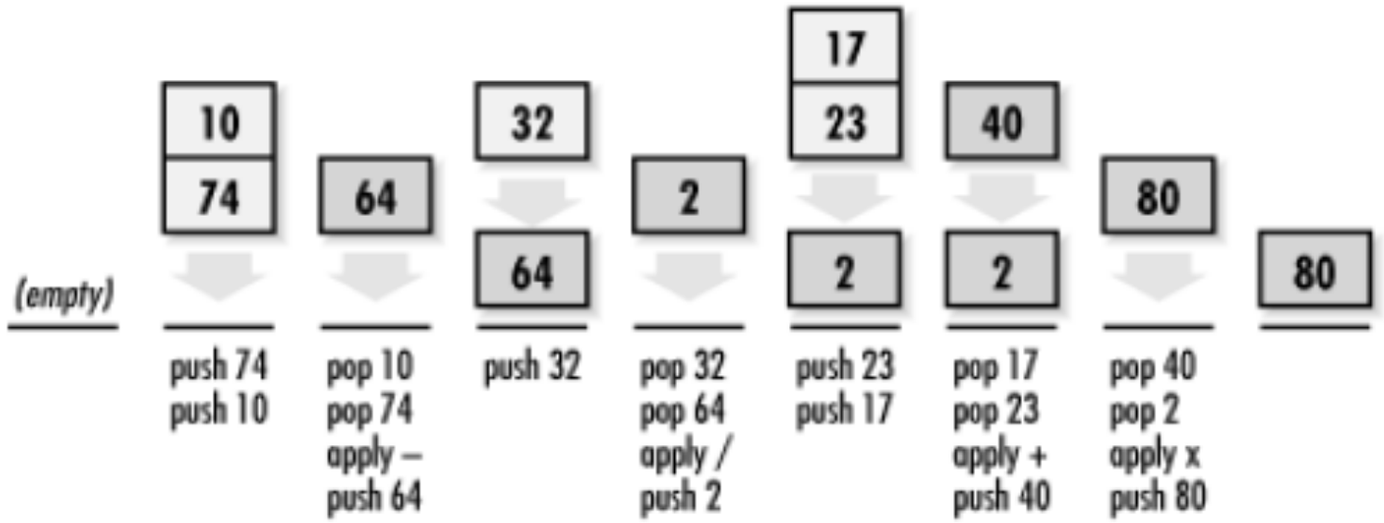
Binary Tree Example

- Expression Processing
  - preorder traversal: x / - 74 10 32 + 23 17
  - inorder traversal: 74 - 10 / 32 x 23 + 17
  - postorder traversal: 74 10 - 32 / 23 17 + x



Binary Tree Example

- postfix: 74 10 - 32 / 23 17 + x
  - use a stack machine processing the postfix expression



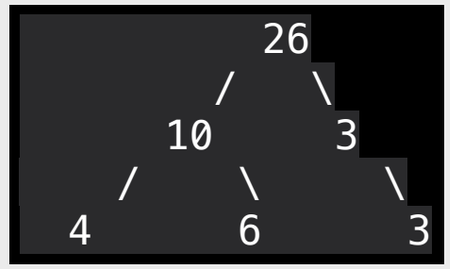
SumTree

- Example: SumTree/sumtree.c

```c
int main(void) {
    struct node *root = newNode(26);
    root->left = newNode(10);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(6);
    root->right->right = newNode(3);

    if(isSumTree(root))
        printf("The given tree is a SumTree ");
    else
        printf("The given tree is not a SumTree ");

    getchar();
    return 0;
}
```



Unix Tips

- fzf is a general-purpose command-line fuzzy finder
  - https://github.com/junegunn/fzf
  - It's an interactive Unix filter for command-line that can be used with any list; files, command history, processes, hostnames, bookmarks, git commits, etc.

Vim Tips

- A code-completion engine for Vim
  - YouCompleteMe
  - Useful links
    - https://jielite.blogspot.com/2014/09/vim-youcompleteme-c-c.html
    - https://ethans.me/zh-tw/posts/2018-09-01-installing-vim-with-youcompleteme-plugin-to-enable-autocomplete/