RECURSION AND TREE STRUCTURE

Lecture notes of the course "*Programming Techniques*"

Lê Hồng Phương[1]

[1]Department of Mathematics, Mechanics and Informatics
Hanoi University of Science, VNUH
<*phuonglh@gmail.com*>

10/2012

Content

Content

Content

Content

1. Recursion
   - Factorial
   - Fibonacci numbers
   - Towers of Hanoi
2. Binary trees
   - Definition
   - Creation
   - Traversal
3. Exercises

Recursion

- Recursion in computer science is a method where the solution to a problem depends on solutions of smaller instances of the same problem.
- The approach can be applied to many types of problems and is one of the central ideas of computer science.
- Recursion technique is a basis for more advanced and efficient techniques in designing algorithms.
- Most computer programming languages support recursion by allowing a function to call itself within the program.

Content

Factorial

- When using a recursive function, we need to pay attention to the termination condition.

```
int factorial(int n) {
  if (n <= 0)
    return 1;
  else
    return n * factorial(n - 1);
}
```

- Note the base case when $n \leq 0$. If you miss this, the function never returns.

Factorial

Recurrence relation:

$$b_n = n * b_{n-1}$$
$$b_0 = 1.$$

Computing the recurrence relation for $n = 4$:

$$\begin{aligned} b_4 &= 4 * b_3 \\ &= 4 * 3 * b_2 \\ &= 4 * 3 * 2 * b_1 \\ &= 4 * 3 * 2 * 1 * b_0 \\ &= 4 * 3 * 2 * 1 * 1 \\ &= 4 * 3 * 2 * 1 \\ &= 4 * 3 * 2 \\ &= 4 * 6 \\ &= 24. \end{aligned}$$

Content

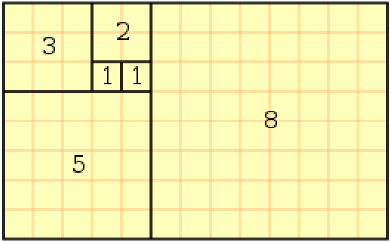
Fibonacci numbers

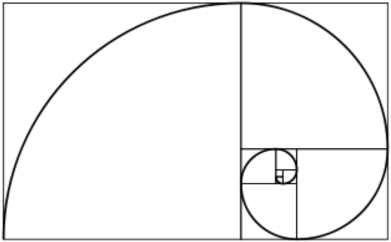
- The first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

$$F(k) = \begin{cases} k, & \text{if } k < 2, \\ F(k-1) + F(k-2), & \text{if } k \geq 2. \end{cases}$$

- The Fibonacci sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$





Fibonacci numbers

- The ratio of consecutive Fibonacci numbers converges:

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \varphi,$$

where φ is the **golden ratio**:

$$\varphi = \frac{1+\sqrt{5}}{2} = 1.6180339887\ldots$$

- See more about the golden ratio and the Fibonacci sequence in nature at:
  - `http://www.youtube.com/watch?v=wTlw7fNcO-0`
  - `http://www.youtube.com/watch?v=kkGeOWYOFoA`
  - `http://en.wikipedia.org/wiki/Golden_ratio`

Fibonacci numbers

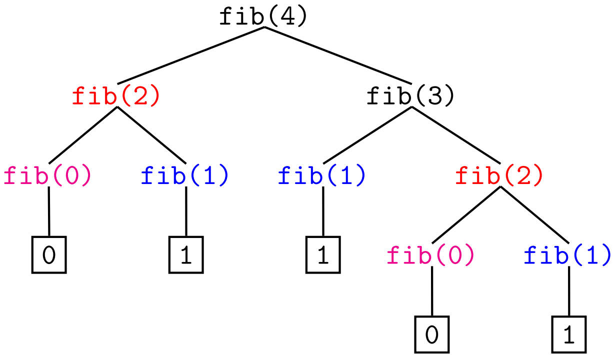
Recursive function to compute the k-th Fibonacci number:

```
int fib(int k) {
  // base cases:
  if (k < 2) {
    return k;
  }
  // recursive case:
  else {
    return fib(k - 1) + fib(k - 2);
  }
}
```

This function is very memory expensive!

Fibonacci numbers

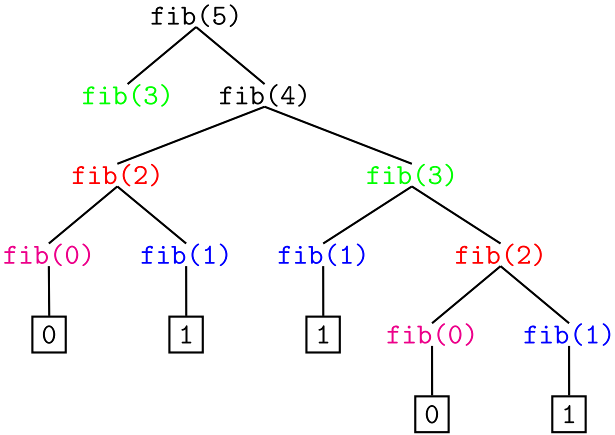
The tree structure of function calls when $k = 4$:



The subtrees whose root nodes have the same color are repeatedly evaluated.

Fibonacci numbers

The tree structure of function calls when $k = 5$:



If k is large, there is an explosion of memory needed to evaluated $F(k)$.

Content

1. Recursion
   - Factorial
   - Fibonacci numbers
   - Towers of Hanoi
2. Binary trees
   - Definition
   - Creation
   - Traversal
3. Exercises

Towers of Hanoi

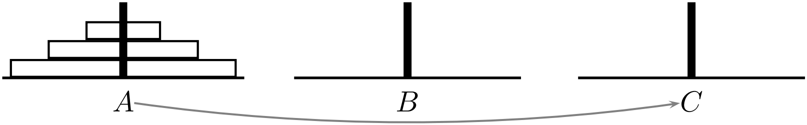
- The tower of Hanoi is a mathematical puzzle.
- We have 3 rods and a number of disks of different sizes which can slide onto any rod.
- Starting with the disks in a neat stack in ascending order of size on one rod, the smallest at the top.
- The objective is to move the entire stack to another rod obeying the following rules:
  1. Only one disk may be moved at a time;
  2. Each move consists of taking the upper disk from one rod and sliding it onto another rod;
  3. No disk can be placed on top of a smaller disk.

Towers of Hanoi

- The tower of Hanoi is a mathematical puzzle.
- We have 3 rods and a number of disks of different sizes which can slide onto any rod.
- Starting with the disks in a neat stack in ascending order of size on one rod, the smallest at the top.
- The objective is to move the entire stack to another rod obeying the following rules:
  1. Only one disk may be moved at a time;
  2. Each move consists of taking the upper disk from one rod and sliding it onto another rod;
  3. No disk can be placed on top of a smaller disk.

Towers of Hanoi

Solution for $n = 3$?

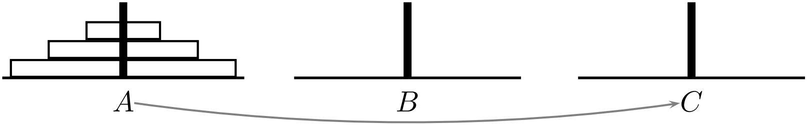


1. $A \to C$
2. $A \to B$
3. $C \to B$
4. $A \to C$
5. $B \to A$
6. $B \to C$
7. $A \to C$

Solution for $n = 10$?

Towers of Hanoi

Solution for $n = 3$?

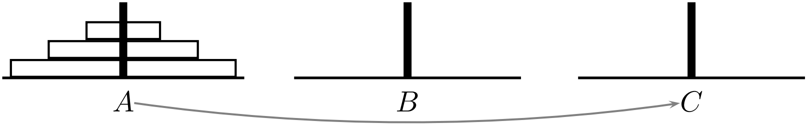


1. $A \to C$
2. $A \to B$
3. $C \to B$
4. $A \to C$
5. $B \to A$
6. $B \to C$
7. $A \to C$

Solution for $n = 10$?

Towers of Hanoi

Solution for $n = 3$?



1. $A \to C$
2. $A \to B$
3. $C \to B$
4. $A \to C$
5. $B \to A$
6. $B \to C$
7. $A \to C$

Solution for $n = 10$?

Towers of Hanoi

- *Divide and conquer* techniques
- Break down the problem into a collection of smaller problems. To move n disks from A to C:
  1. Move $n-1$ disks from A to B. This leaves disk n alone on rod A;
  2. Move disk n from A to C;
  3. Move $n-1$ disks from B to C.
- Prove that this algorithm must make at least $2^n - 1$ moves.

Towers of Hanoi

- *Divide and conquer* techniques
- Break down the problem into a collection of smaller problems. To move n disks from A to C:
  1. Move $n-1$ disks from A to B. This leaves disk n alone on rod A;
  2. Move disk n from A to C;
  3. Move $n-1$ disks from B to C.
- Prove that this algorithm must make at least $2^n - 1$ moves.

Towers of Hanoi

```
void move(int n, char a, char b, char c) {
  if (n==1) {
    printf("%c -> %c\n", a, c);
  } else {
    move(n-1, a, c, b);
    move(1, a, ' ', c);
    move(n-1, b, a, c);
  }
}

int main(int argc, char **argv) {
  move(3, 'A', 'B', 'C');
  return 0;
}
```

Towers of Hanoi

- According to a legend, there is a set of 64 gold disks, the universe is supposed to end when the task is complete.
- If done correctly, it will take $2^{64} - 1 = 1.84 * 10^{19}$ moves.
- If a move takes 1 second, that's $5.85 * 10^{11}$ years.
- Since the largest disks must be very heavy, we need about 1 minute per move, it would take $1.5 * 10^{14}$ years.
- The current age of universe is estimated at 10^{10} years.
  - Is the legend correct?

Content

Content

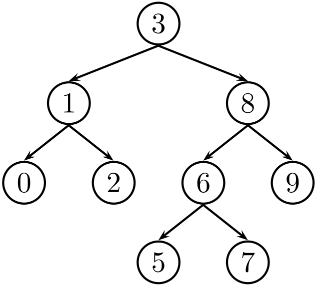
Binary trees

- A **binary tree** is a dynamic data structure where each node has at most two children.
- Any node in the tree can be reached by starting at root node and repeatedly following references to either the left or right child.
- **A binary search tree** is a binary tree with ordering among its children:
  - All elements in the left subtree are assumed to be "less" than the root element.
  - All elements in the right subtree are assumed to be "greater" than the root element.

Binary trees

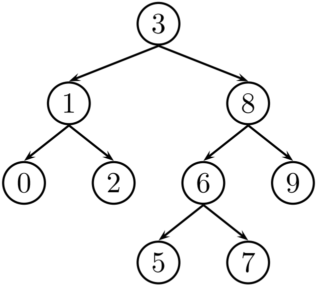
```
struct Node {
  int datum;
  struct Node *left;
  struct Node *right;
};

typedef struct Node *Tree;
Tree root = NULL;
```



Some definitions

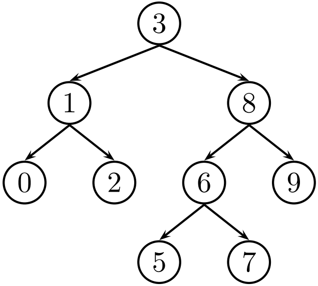
- The **root node** is the node with no parent. There is at most one root node. A **leaf node** is a node which has no children.
- The **height** of a tree is the length of the path from the root to the deepest node in the tree. If the tree has only one (root) node, its height is 0.
- The **depth** of node n is the length of the path from the root to n. The depth of the root node is 0.



- The height of the tree?
- The depth of node 2?
- The depth of node 7?

Some definitions

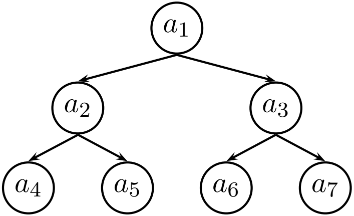
- **Siblings** are node that share the same parent node.
- A node p is an **ancestor** of a node q if it exists on the path from node p to node q. The node q is then called the **descendant** of p.
- The **size** of a node is the number of descendants that it has, including itself.



- Is node 1 ancestor of node 7?
- The size of node 8?
- The size of node 3?

Some definitions

- A **full binary tree** is a tree in which every node other than the leaves has two children.
- A **perfect binary tree** is a full binary tree in which all leaves are at the same depth.



- *Claim*: The number of nodes n of a perfect binary tree is $n = 2^{h+1} - 1$, where h is the height of the tree. (Prove that claim!)

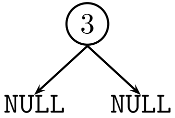
Content

Creating a node

```c
typedef struct Node *Tree;
Tree root = NULL;

Tree createNode(int datum) {
  Tree node = (Tree)malloc(sizeof(struct Node));
  node->datum = datum;
  node->left = NULL;
  node->right = NULL;
  return node;
}
```

createNode(3) $\Rightarrow$



Adding node to a binary search tree

Idea: Creates a node of a given datum and recursively searches for its correct parent to add.

```
void addNode(Tree *root, int datum) {
  if ((*root) == NULL) {
    Tree node = createNode(datum);
    *root = node;
    return;
  }
  if (datum < (*root)->datum) {
    addNode(&(*root)->left, datum);
  } else {
    addNode(&(*root)->right, datum);
  }
}
```

Creating a binary search tree

Idea: Repeatedly add nodes to the tree. Initially, the root is `NULL`.

```
int a[] = {3, 1, 8, 0, 2, 6, 9, 5};
Tree root = NULL;

int i;
for (i = 0; i < 8; i++) {
  addNode(&root, a[i]);
}
```

- Explain the code
- Give more examples on different data

Content

Tree traversal

- Linear data structures like linked list or one-dimensional array have a canonical method of traversal.
- Tree structures can be traversed in different ways.
- There are three main steps that can be performed and the order in which they are performed defines the traversal type:
  1. Performing an action on the current node ("visiting" the node).
     ```
     void visit(Tree tree) {
       printf("%3d", tree->datum);
     }
     ```
  2. Traversing to the left subtree
  3. Traversing to the right subtree

Tree traversal

Three types of tree traversal:

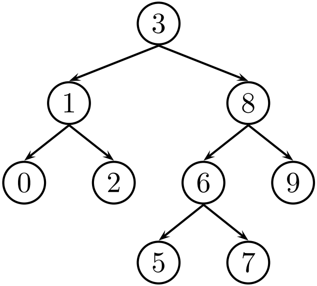
- **pre-order**: root, left subtree, right subtree
- **in-order**: left subtree, root, right subtree
- **post-order**: left subtree, right subtree, root

Tree traversal – Pre-order

Pre-order traversal:

1. Visit the root node
2. Visit its left subtree
3. Visit its right subtree

```
void preOrder(Tree tree) {
  if (tree != NULL) {
    visit(tree);
    preOrder(tree->left);
    preOrder(tree->right);
  }
}
```



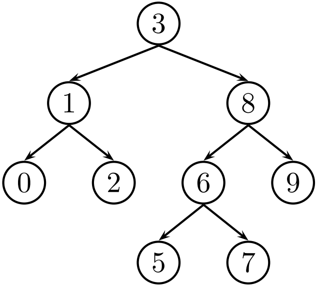
Traversal order: `3, 1, 0, 2, 8, 6, 5, 7, 9`

Tree traversal – In-order

In-order traversal:

1. Visit its left subtree
2. Visit the root node
3. Visit its right subtree

```
void inOrder(Tree tree) {
  if (tree != NULL) {
    inOrder(tree->left);
    visit(tree);
    inOrder(tree->right);
  }
}
```



Traversal order: `0, 1, 2, 3, 5, 6, 7, 8, 9`

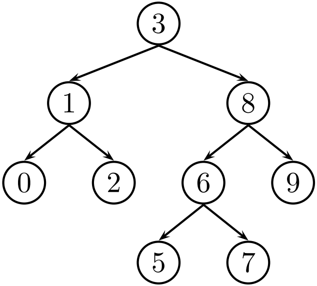
(The numbers are sorted!)

Tree traversal – Post-order

Post-order traversal:

1. Visit its left subtree
2. Visit its right subtree
3. Visit the root node

```
void postOrder(Tree tree) {
  if (tree != NULL) {
    postOrder(tree->left);
    postOrder(tree->right);
    visit(tree);
  }
}
```



Traversal order: `0, 2, 1, 5, 7, 6, 9, 8, 3`

Content

Exercise 1. Write a program which computes the Fibonacci numbers less than a positive integer n.

- Add these numbers to a linked list (a stack or a queue of your choice);
- Print out the list to verify the result.

Exercise 2. Write a repetitive function to calculate the Fibonacci numbers up to a positive integer n. Compare the running time of this function with that of the recursive version.

Exercise 3. Empirically verify the following result:

$$\lim_{n\to\infty} \frac{F(n+1)}{F(n)} = \varphi,$$

where

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.6180339887.$$

Exercises

Exercise 4. Given two nodes p and q of a binary tree, write a function which tests whether p is the ancestor of q or not.

Exercise 5. Write a function which computes the size of a node in a binary tree.

Exercise 6. Write a function which checks whether a binary tree

- is a full binary tree or not;
- is a perfect binary tree or not.