



Starfleet - Day 03

Trees and Graphs

Staff 42 pedago@42.fr

Summary: This document is the day03's subject for the Starfleet Piscine.

Contents

I	General rules	2
II	Day-specific rules	3
III	Exercise 00: Toy factory	4
IV	Exercise 01: It's a secret	6
V	Exercise 02: Tree of life	8
VI	Exercise 03: Planet of the Apes	10
VII	Exercise 04: Planet of the Apes 2	12
VIII	Exercise 05: The jo?e of the y?a? !	13

Chapter I

General rules

- Every instructions goes here regarding your piscine
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- The exercises must be done in order. The evaluation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The subject can be modified up to 4 hours before the final turn-in time.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.
- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter II

Day-specific rules


- If asked, you must turn-in a file named `bigo` describing the time and space complexity of your algorithm as below. You can add to it any additional explanations that you will find useful. When your algorithm involves going through a tree or a graph, specify if it is breadth or depth first search, iterative or recursive.

```
$> cat bigo
O(n) time
O(1) space
iterative BFS
$>
```

- Your work must be written in C. You are allowed to use all functions from standard libraries.
- For each exercise, you must provide a file named `main.c` with all the tests required to attest that your functions are working as expected.

Chapter III

Exercise 00: Toy factory

	Exercise 00
Exercise 00: Toy factory	
Turn-in directory : <code>ex00/</code>	
Files to turn in : <code>info.c header.h main.c</code>	
Allowed functions : <code>all</code>	
Notes : <code>n/a</code>	

Thanks to your **Poker** addiction, you find yourself in the middle of nowhere working in a strange toy factory for a few days.

One day, you notice a toy which is a **plastic tree** for children. Apparently, it's a normal toy, but when you get closer, you notice some **numbers on the branches**.

You know what it is and you shout it at loud: **It's a binary tree!**

At the same time, the boss of the factory notices you during this **euphoric** moment and **immediatly** comes to you, not for **reprimand**, but to **thank** you. You were the **first** one to notice the **computer science** elements in his **toys**, it's in fact, his strange little **fantasy**.

Now he ask you to go further: Is it a **binary search tree**? What is its **height**? Is it **balanced**? You answer correctly. Then, with a **smile**, he brings 100 **different tree** toys...

You know that the creation of a **little software** is necessary to answer all his **ques-tions!**

The boss is asking you the following informations about a tree :

- The **minimum** value in the tree.
- The **maximum** value in the tree.
- The **number of nodes** in the tree.

- The height of the tree, the number of edges on the longest path from the root node to a leaf.
- Is it a Binary Search Tree (BST)?
- Is it balanced ?

Given the following structure:

```
struct s_info {  
    int min;           // the minimum value in the tree  
    int max;           // the max value in the tree  
    int elements;      // the number of nodes in the tree  
    int height;        // the height of the tree  
    int isBST;         // 0 = FALSE, 1 = TRUE  
    int isBalanced;    // 0 = FALSE, 1 = TRUE  
};
```

Create a function that returns the informations related to a tree, with his root node passed as parameter:

```
struct s_info getInfo(struct s_node *root);
```




If you're beginning with the tree, we advise you to do one function per information.



A binary search tree (BST) is a binary tree in which every node fits a specific ordering property : all left descendants $\leq n <$ all right descendants. This must be true for each node n .

Chapter IV

Exercise 01: It's a secret

	Exercise 01
Exercise 01: It's a secret	
Turn-in directory : <i>ex01/</i>	
Files to turn in : <code>createBST.c</code> <code>main.c</code> <code>header.h</code> <code>bigo</code>	
Allowed functions : <code>all</code>	
Notes : <code>n/a</code>	

One day left before you can go back home !

But before you leave, the **boss** needs one last thing from you.

He has on a **paper**, an array of **ordered** integer, and he would like that from this array, you create a **balanced Binary Search Tree Toy**.

You immediatly wonder why he would want that, he tells you:

That's a **secret**!

Hum ok... In one day you leave the factory, **common**, you can do it!

Given the following structure which represents a node of a tree:

```
struct s_node {  
    int      value;  
    struct s_node *right;  
    struct s_node *left;  
};
```

and given as parameter an array of integers ordered in ascending order and its size **n**,
implement a function which returns a **balanced Binary search tree**:

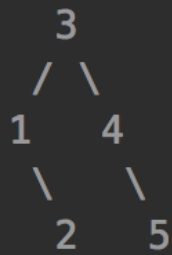
```
struct s_node *createBST(int *arr, int n);
```

Note: the array will always be sorted.

Example:

```
input: arr = [1, 2, 3, 4, 5], n = 5
```


```
output:
```



We strongly advise you to do it using a divide and conquer approach.

Chapter V

Exercise 02: Tree of life

	Exercise 02
Exercise 02: Tree of life	
Turn-in directory : <i>ex02/</i>	
Files to turn in : <i>findParent.c header.h main.c bigo</i>	
Allowed functions : <i>all</i>	
Notes : <i>n/a</i>	

Back to the civilization, you decide to take a trip to Sequoia Park, on the road to the Giant Sequoia... All of a sudden, you find an old deserted house. You decide to go inside and then, you discover an old dusty computer...

Intrigued, you turn it on and discover something amazing! It contains the life work of a searcher. One of his work is a family tree of all the living species since the creation of the earth ! An idea comes to your mind: what is the common ancestor of the man and the dinasaur?

The tree of life is an n-ary tree using the following structure :

```
struct s_node {
    char *name;
    struct s_node **children;
};
```

Create a function wich returns the first common ancestor between two species, by taking as parameters the root node of a tree and 2 strings which are the names of two species:

```
struct s_node *findParent(struct s_node *root, char *firstSpecies, char *secondSpecies);
```

Where children is a null-terminated array containing pointers to child nodes.


If one the the two species cannot be found in the tree, the function returns NULL.

Here are some valid examples with the 'tree of life' file :

```
findParent(root, "Dinosauria", "Homo sapiens"); // returns the "Amniota" node  
findParent(root, "Lynx", "Marsupialia"); // returns the "Mammalia" node  
findParent(root, "Dinosauria", "I do not exist !"); // returns NULL
```

Chapter VI

Exercise 03: Planet of the Apes

	Exercise 03
Exercise 03: Planet of the Apes	
Turn-in directory : <i>ex03/</i>	
Files to turn in : <code>saveTheSequoia.c</code> <code>main.c</code> <code>header.h</code> <code>bigo</code>	
Allowed functions : <code>all</code>	
Notes : <code>n/a</code>	

Once you reach the giant Sequoia, the stupor!

Lots of monkeys escaped from a zoo nearby! These monkeys are now hiding in the Giant Sequoia. It could be ok, but currently the **bigger** monkeys are on the lower branches (closer to the ground) and the most **lightweight**ed monkeys are at the top of the tree. The firemen are at the ball, the Sequoia will **break off** soon (because the branches are not solid)... What a calamity!

You have to save the day, you pull out your emergency **drone**, your drone can make 2 monkeys **swap** their places (by frightening them). It's now up to you to reorganize the tree to prevent it from **breaking off**.

Each node contains the **weight** of the monkey who is on it, you must create a function able to switch a **min-heap** to a **max-heap**.

Given the following structure:

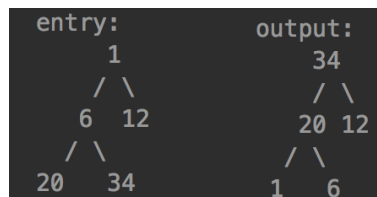
```
//binary tree node
struct s_node {
    int value;
    struct s_node *right;
    struct s_node *left;
};
```

Given a pointer to the root node of a min-heap, implement a function that transform the min-heap into a max-heap :

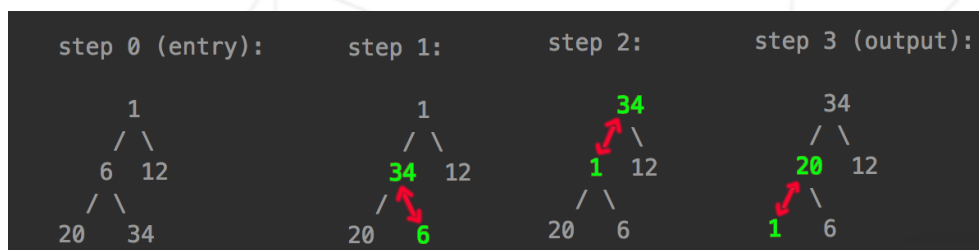
```
void saveTheSequoia(struct s_node **root);
```

Be careful, you can't recreate a new tree, you must only **swap** the monkeys.

Example:




Same example showing the steps:



By swapping the monkeys we mean swapping the nodes, not just the weights!

Chapter VII

Exercise 04: Planet of the Apes 2

	Exercise 04
Exercise 04: Planet of the Apes 2	
Turn-in directory : <i>ex04/</i>	
Files to turn in : <i>insertMonkey.c main.c header.h bigo</i>	
Allowed functions : <i>all</i>	
Notes : <i>n/a</i>	

Oh no, a new wave of monkeys is approaching, use your drone to make sure that the tree stays in max-heap with the newcomers!

Given the following structure:

```
//binary tree node
struct s_node {
    int value;
    struct s_node *right;
    struct s_node *left;
};
```

Create a function that inserts a new ape in the tree, given as parameters the root node of a tree, and the node of the new incoming monkey :

```
void insertMonkey(struct s_node *root, struct s_node *monkey);
```


The tree must stay a max-heap !



Be careful, having a max-heap means also having a complete tree !

Chapter VIII

Exercise 05: The jo?e of the y?a? !

	Exercise 05
Exercise 05: The jo?e of the y?a? !	
Turn-in directory : <i>ex05/</i>	
Files to turn in : <i>understand.c main.c header.h bigo</i>	
Allowed functions : <i>all</i>	
Notes : <i>n/a</i>	

After your return from vacation, you receive a letter from one of your friends, he says he needs you **immediately**.

Here you go again for an adventure in ... **Paris!**

In the **plane**, you are so much bored. The power cut that **turned off all the tablets** to watch movies does not help! You have no choice but to **talk to the person next to you**. However **this one** has difficulty articulating, some words are not understandable.

Argh! He just told you a **joke** that seemed very funny ...

You decide to create a **program to understand the joke**, which is the perfect time to create a **Prefix Tree** (also called **Trie**).

A **Prefix Tree** is a **n-ary tree** where each node stores :

- A character '**c**'.
- If the sequence of characters from the root to the node is a valid word, in a variable '**isWord**'.
- A null-terminated array of pointers to **child nodes**.

Here are the structures for the Prefix Tree:

```
struct s_node {
    char    c;
    unsigned int  isWord;1; // 0 = FALSE, 1 = TRUE
    struct s_node **child;
};

struct s_trie {
    struct s_node *node;    // root of the trie
};
```

Implement 2 functions:

- `createTrie(dictionary)` creates a Prefix Tree with the dictionary of english words passed as parameter (null-terminated array).
- `understand(word, trie)` returns the correct word from the Prefix Tree by substituting the characters '?' by other characters. If no match is found the function returns the incomplete word. If multiple matches are possible, the function returns the first one found.

These functions must be declared as follows :

```
struct s_trie *createTrie(char **dictionary);

char *understand(char *word, struct s_trie *trie);
```

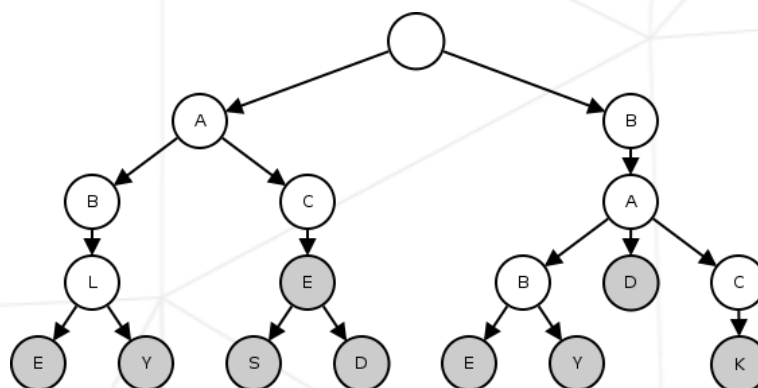
The word passed as parameter will always be lowercased (characters from 'a' to 'z' and of course '?').

Examples:

For the following dictionary :

```
char *dictionary[] = {"able", "ably", "ace", "aces", "aced", "babe", "baby", "bad", "back", NULL};
```

The Prefix Tree created by the function `createTrie` would be :



```
$> compile understand.c
$> ./understand 'what is the differ???? betw??? a figh??? pilo? and god'
what is the difference between a fighter pilot and god
$> ./understand 'i am goi?? to paris to eat some che?s?, yum?? !'
i am going to paris to eat some cheese, yummy !
```

The Adventure to Paris will begin next week! :)