Assignment 03 C Programming

Devid Duma

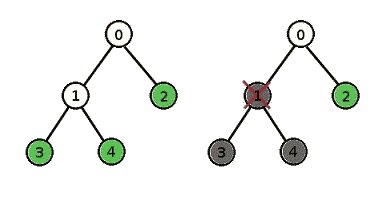
**Exercise 4**

50701 – Cell Removal

@ acm.epoka.edu.al:8888/en/problem-pid-c60d?ps=15&smt=7&smpwid=0

The problem:

In biology, organisms have the following property: Starting from the first cell (the zygote), each cell during an organism's development process either divides into 2 other cells or does not divide at all. An organism is mature when all of its cells will not divide any further.  
  
Let's assign a unique number to each cell in an organism's development process. For example, consider a species in which each organism starts with cell 0, which divides into cells 1 and 2. Cell 1 divides into cells 3 and 4. Cells 2, 3 and 4 do not divide. Every mature organism of this species will consist of exactly 3 cells - 2, 3 and 4.



During the development process, if we kill a cell, it will be absent in the mature form of the organism. If that cell happens to be a cell that divides, then the mature organism will be missing all of the cell's descendants as well because the cell is killed before it has a chance to divide. For example, in the organism described above, if we kill cell 1 during the development process, the mature organism will contain only cell 2.  
  
You are given *N* cells numerated from 0 to *N*-1. For each cell you know the index of it's parent. The zygote's parent is -1. Return the number of cells the mature form of this organism would have if you killed cell *deletedCell* during the development process.

**Input**  
The first line contains two numbers *N* and *deletedCell* (1 ≤ *N* ≤ 50, 0 ≤ *deletedCell* ≤ *N*-1). Next *N* lines will contain parents of corresponding cells. Each parent will be between -1 and *N*-1 inclusive. It is guaranteed, that there will be at least one line. Also it is guaranteed that this array of parents will form a binary tree without any cycles.

**Output**  
Output single number - the number of cells the mature form of this organism would have if you kill cell *deletedCell* during the development process.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input 1** | **Input 2** | **Input 3** | **Input 4** |
| 1 0 -1 | 3 0 -1 0 0 | 3 2 1 -1 1 | 7 3 5 3 6 6 3 -1 5 |
| **Output 1** | **Output 2** | **Output 3** | **Output 4** |
| 0 | 0 | 1 | 2 |

The Solution:

// dduma19

// http://acm.epoka.edu.al:8888/en/problem-pid-c60d?ps=15&smt=7&smpwid=0

// ProblemID-35: 3ed-Accepted - Cell Removal

#include <iostream>

#include <vector>

using namespace std;

struct Children {

int children[2]; //we either have 0 or 2 children in this problem

int currentSize = 0;

};

Children \*a;

int SearchTree(int pos, int &delcell, int counter) {

// if the cell was not killed

if(pos!=delcell) {

// if the cell does not have any children, it is a terminal cell

if(a[pos].currentSize==0)

counter++; // count terminal cell

// if the cell does have children, it is not terminal

else

// ierate through all it's children

for(int i=0;i<a[pos].currentSize;i++)

// continue counting for subtree and update current counter

counter= SearchTree(a[pos].children[i],delcell,counter);

}

// return our counter for subtree with head pos

return counter;

}

int main()

{

int n,delcell;

cin>>n>>delcell;

a= new Children[n];

int x,head;

// input stuff

for(int i=0;i<n;i++) {

cin>>x;

if(x==-1) head=i;

else {

a[x].children[a[x].currentSize++] = i;

}

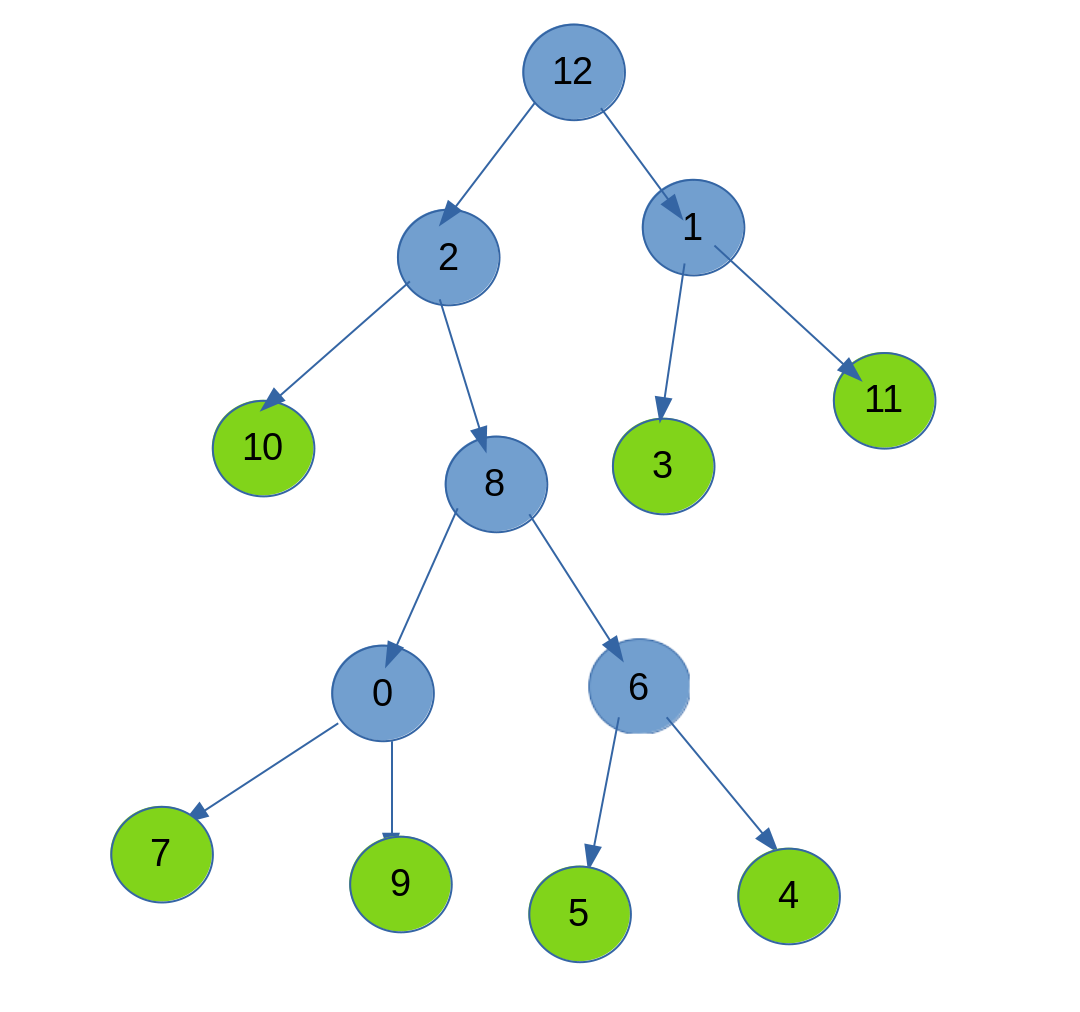
}

// start from head cell, specify the killed cell and start counting from 0 how many cells we have in total

cout<<SearchTree(head,delcell,0);

return 0;

}

Explanation:

The solution makes use of a recursive function called

SearchTree(int pos, int &delcell, int counter).

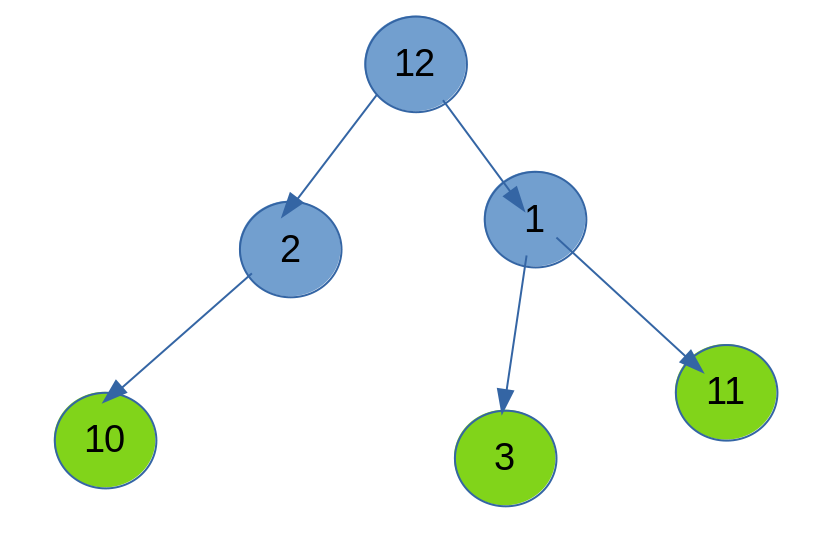
Let us suppose that the organism’s development looks something like the picture in the right:

What we would do is that we would start at cell 12 and then work our way down, counting the leaves only (nodes in green).

The way that we save the information of our tree is by using an array of structure “Children”. Each element at index x in the array of Children denotes the node with index x in the tree. Notice that we do not have a specific order for the nodes, we can see from above that node 12 could be the parent to nodes 2 and 1. For each node, so element in the array, we push back each of it’s children for a maximum of 2 children. We actually improvise a push pack by using the variable currentSize in the structure Children.

We start our SearchTree function from the head, specify the deleted node’s index and initalize the counter to 0. We continue our way down the tree using the recursive function. During our way down though we would not visit the cell that we specified as dead / killed during the organism’s development and all it’s children. So the tree we would visit it would be only a part of the whole big tree.

If the cell that was killed was, say cell 8, than the recursive function’s process would look like this:

So the recursive function TreeSearch() would start at 12, go to 2, go to 10, counter++, go back to 2, go back to 12, go to 1, go to 3, counter++, go back to 1, go to 11, counter++, go back to 1, go back to 12, return counter.

As simple as that.

The solution was tried in the acm.epoka.edu.al website and was accepted.