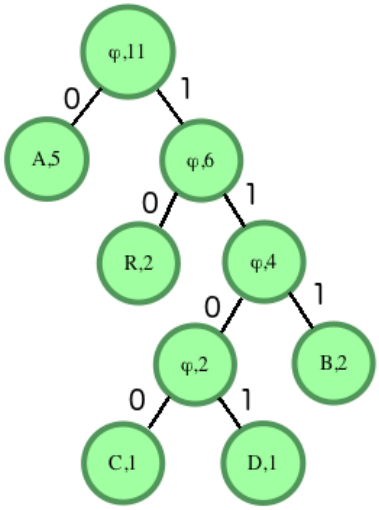
Tree: Huffman Decoding

Huffman coding assigns variable length codewords to fixed length input characters based on their frequencies. More frequent characters are assigned shorter codewords and less frequent characters are assigned longer codewords. All edges along the path to a character contain a code digit. If they are on the left side of the tree, they will be a *0* (zero). If on the right, they'll be a *1* (one). Only the leaves will contain a letter and its frequency count. All other nodes will contain a null instead of a character, and the count of the frequency of all of it and its descendant characters.

For instance, consider the string *ABRACADABRA*. There are a total of 11 characters in the string. This number should match the count in the ultimately determined root of the tree. Our frequencies are $A = 5, B = 2, R = 2, C = 1$ and $D = 1$. The two smallest frequencies are for C and D, both equal to 1, so we'll create a tree with them. The root node will contain the sum of the counts of its descendants, in this case $1 + 1 = 2$. The left node will be the first character encountered, C, and the right will contain D. Next we have 3 items with a character count of 2: the tree we just created, the character B and the character R. The tree came first, so it will go on the left of our new root node. B will go on the right. Repeat until the tree is complete, then fill in the 1's and 0's for the edges. The finished graph looks like:



Input characters are only present in the leaves. Internal nodes have a character value of ϕ (NULL). We can determine that our values for characters are:

```
A - 0
B - 111
C - 1100
D - 1101
R - 10
```

Our Huffman encoded string is:

```
A B    R  A C     A D     A B    R  A
0 111 10 0 1100 0 1101 0 111 10 0
```

```
or
01111001100011010111100
```

To avoid ambiguity, Huffman encoding is a prefix free encoding technique. No codeword appears as a prefix of any other codeword.

To decode the encoded string, follow the zeros and ones to a leaf and return the character there.

You are given pointer to the root of the Huffman tree and a binary coded string to decode. You need to print the decoded string.

Function Description

Complete the function decode_huff in the editor below. It must return the decoded string.

decode_huff has the following parameters:

- *root*: a reference to the root node of the Huffman tree
- *s*: a Huffman encoded string

Input Format

There is one line of input containing the plain string, s. Background code creates the Huffman tree then passes the head node and the encoded string to the function.

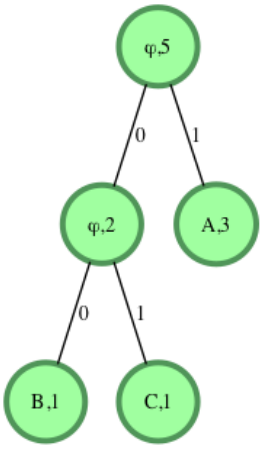
Constraints

$1 \leq |s| \leq 25$

Output Format

Output the decoded string on a single line.

Sample Input




```
s="1001011"
```

Sample Output

```
ABACA
```

Explanation

```
S="1001011"
Processing the string from left to right.
S[0]='1' : we move to the right child of the root. We encounter a leaf node with value 'A'. We add 'A' to the
decoded string.
We move back to the root.
```

```
S[1]='0' : we move to the left child.
S[2]='0' : we move to the left child. We encounter a leaf node with value 'B'. We add 'B' to the decoded string.
We move back to the root.

S[3] = '1' : we move to the right child of the root. We encounter a leaf node with value 'A'. We add 'A' to the
decoded string.
We move back to the root.

S[4]='0' : we move to the left child.
S[5]='1' : we move to the right child. We encounter a leaf node with value C'. We add 'C' to the decoded string.
We move back to the root.

 S[6] = '1' : we move to the right child of the root. We encounter a leaf node with value 'A'. We add 'A' to the
decoded string.
We move back to the root.

Decoded String = "ABACA"
```

Contest ends in a day

Submissions: 129
Max Score: 60
Difficulty: Medium

Rate This Challenge:

More

C++

```
//
//  main.cpp
//  Huffman
//
//  Created by Vatsal Chanana

#include<bits/stdc++.h>
using namespace std;

typedef struct node {
    int freq;
    char data;
    node * left;
    node * right;
} node;

struct deref:public binary_function<node*, node*, bool> {
    bool operator()(const node * a, const node * b)const {
        return a->freq > b->freq;
    }
};

typedef priority_queue<node *, vector<node*>, deref> spq;

node * huffman_hidden(string s) {

    spq pq;
    vector<int>count(256,0);

    for(int i = 0; i < s.length(); i++ ) {
        count[s[i]]++;
    }

```

```
    for(int i=0; i < 256; i++) {

        node * n_node = new node;
        n_node->left = NULL;
        n_node->right = NULL;
        n_node->data = (char)i;
        n_node->freq = count[i];

        if( count[i] != 0 )
            pq.push(n_node);

    }

    while( pq.size() != 1 ) {

        node * left = pq.top();
        pq.pop();
        node * right = pq.top();
        pq.pop();
        node * comb = new node;
        comb->freq = left->freq + right->freq;
        comb->data = '\0';
        comb->left = left;
        comb->right = right;
        pq.push(comb);

    }

    return pq.top();

}

void print_codes_hidden(node * root, string code, map<char, string>&mp) {

    if(root == NULL)
        return;

    if(root->data != '\0') {
        mp[root->data] = code;
    }

    print_codes_hidden(root->left, code+'0', mp);
    print_codes_hidden(root->right, code+'1', mp);

}
```

```
/*
The structure of the node is

typedef struct node {

    int freq;
    char data;
    node * left;
    node * right;

} node;

*/


void decode_huff(node * root, string s) {
    // creating a tempporary node
    node *temp = root;

    // iterating the characters of the input string
    for (auto ch : s){
```

```
100         // checking for the edge number
101         // traversing through the tree
102         if (ch == '1'){
103             temp = temp->right;
104         } else {
105             temp = temp->left;
106         }
107
108         // output the character if the traversal reaches the leaf
109         if (temp->right == NULL && temp->left == NULL){
110             cout << temp->data;
111             temp = root; // resetting the tree
112         }
113     }
114 }
115 int main() {}
```

Line: 111 Col: 47

Upload Code as File Test against custom input Run Code Submit Code

Testcase 0 ✔ | Testcase 1 ✔ | Testcase 2 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Compile Time

Compile Message

```
Solution.cpp: In function ‘node* huffman_hidden(std::__cxx11::string)’:
Solution.cpp:30:22: warning: comparison of integer expressions of different signedness: ‘int’ and ‘std::__
cxx11::basic_string<char>::size_type’ {aka ‘long unsigned int’} [-Wsign-compare]
     for(int i = 0; i < s.length(); i++ ) {
                    ~~^~~~~~~~~~~~
Solution.cpp: In function ‘int main()’:
Solution.cpp:123:23: warning: comparison of integer expressions of different signedness: ‘int’ and ‘std::_
_cxx11::basic_string<char>::size_type’ {aka ‘long unsigned int’} [-Wsign-compare]
     for( int i = 0; i < s.length(); i++ ) {
                     ~~^~~~~~~~~~~~
```

Run Time

Input (stdin)

```
ABACA
```

Your Output (stdout)

```
ABACA
```

Expected Output

```
ABACA
```