Review of the Lectures 21-26, 30-32

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM
2. Examples of Questions
   - recursion and memoization
   - enumeration
   - trees, binary search trees, Huffman codes
   - hashing and sorting

MCS 360 Lecture 42
Introduction to Data Structures
Jan Verschelde, 6 December 2017

Review of the Lectures 21-26, 30-32

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - recursion and memoization
   - enumeration
   - trees, binary search trees, Huffman codes
   - hashing and sorting

The Final Exam

- Monday 11 December, BSB 337, from 8AM to 10AM.
- The exam is closed book. No calculators, no computers.
- This review has the focus on the first 18 lectures,
  the material before the first midterm exam:
  1. Lectures 21 to 26, 30 to 32, chapters 7 to 10 in the textbook.
  2. Recursion, memoization, and enumeration.
  3. Trees and sorting algorithms.
  4. Focus on data structure concepts,
     not so much on C++ programming.
- Questions on this review are representative,
  but the list is by no means exhaustive.
- Review the quizzes, homework problems, and midterm exams.

Review of the Lectures 21-26, 30-32

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - recursion and memoization
   - enumeration
   - trees, binary search trees, Huffman codes
   - hashing and sorting

define a recursive function

1. Chebyshev polynomials C_n, with $\deg(C_n) = n$, are defined recursively as:

$$C_0(x) = 1, C_1(x) = x, \text{ for } n > 1 : C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x).$$

   1. Define a recursive function with prototype

```
double C ( int n, double x );
```

      that returns the value of the `n`-th Chebyshev polynomial at `x`.

execute a recursive definition

1. Chebyshev polynomials C_n, with $\deg(C_n) = n$, are defined recursively as:

$$C_0(x) = 1, C_1(x) = x, \text{ for } n > 1 : \; C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x).$$

   2. Simulate the call `C(5,1.0)`.
   Leave the formulas, do not evaluate them.

memoization

1. Chebyshev polynomials C_n, with $\deg(C_n) = n$, are defined recursively as:

$$C_0(x) = 1, C_1(x) = x, \text{ for } n > 1 : \ C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x).$$

   3. Make a more efficient version of your function via memoization, by storing the results of previous function calls in a vector, maintained by the function. The vector is initialized to hold up to `n = 50` when calling the function with $\texttt{n} = -1$.

use a stack to make an interative version

1. Chebyshev polynomials C_n, with $\deg(C_n) = n$, are defined recursively as:

$$C_0(x) = 1, C_1(x) = x, \text{ for } n > 1 : \ C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x).$$

4. Use this stack to give an iterative version of this function.

make an interative version without a stack

1. Chebyshev polynomials C_n, with $\deg(C_n) = n$, are defined recursively as:

$$C_0(x) = 1, C_1(x) = x, \text{ for } n > 1 : \ C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x).$$

5. Without using a stack, give an iterative version of this function.

Review of the Lectures 21-26, 30-32

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - recursion and memoization
   - enumeration
   - trees, binary search trees, Huffman codes
   - hashing and sorting

enumeration

2. To enumerate all numbers divisible by three, we propose a function with prototype

```
void numbers3
 ( int n, int k, vector<int> accu, int radix,
   int sum );
/*
   Writes all possible numbers of n digits long,
   every digit is in the range 0..radix-1,
   where the sum of the digits is divisible 3.
   The parameter k controls the recursion depth. */
```

Give a recursive definition of this function.

the base case

```
void numbers3
  ( int n, int k, vector<int> accu, int radix,
    int sum )
{
    if(k == n)
    {
        if(sum % 3 == 0)
        {
            for(int i=0; i<n; i++) cout << accu[i];
            cout << endl;
        }
    }
    else // the general case
```

the general case

```
    else
    {
        for(int i=0; i<radix; i++)
        {
            accu[k] = i;
            numbers3(n, k+1, accu, radix, sum+i);
        }
    }
}
```

more enumeration

3. To enumerate all possible license plates,
   we propose a function with prototype

```
void plates ( int n, int k, string accu );
/*
   Generates and prints all license plates
   of n characters, accumulating in accu.
   The k controls the recursion depth. */
```

   Give a recursive definition of this function.

the base case

```
void plates ( int n, int k, string accu )
{
   if(k == n)
      cout << accu << endl;
   else
   {
      // the general case
   }
}
```

the general case

```
for(int i=0; i<26; i++)     // letters
{
   char letter = 'A'+i;
   plates(n, k+1, accu + letter);
}
for(int i=0; i<10; i++)     // numbers
{
   char number = '0'+i;
   plates(n, k+1, accu + number);
}
```

Review of the Lectures 21-26, 30-32

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

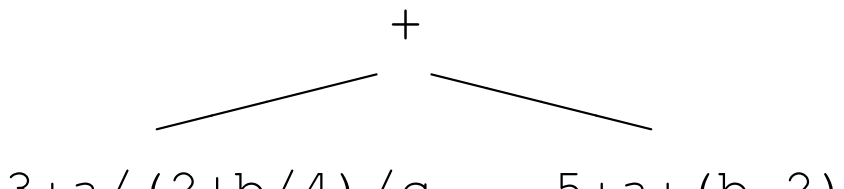
2. Examples of Questions
   - recursion and memoization
   - enumeration
   - trees, binary search trees, Huffman codes
   - hashing and sorting

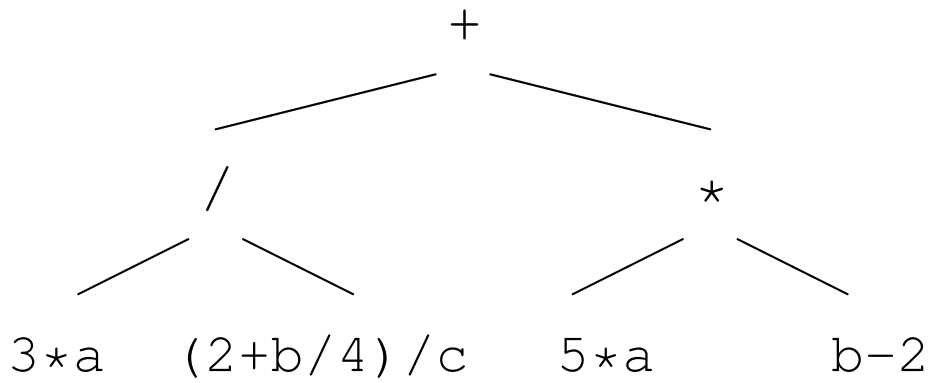
expression trees

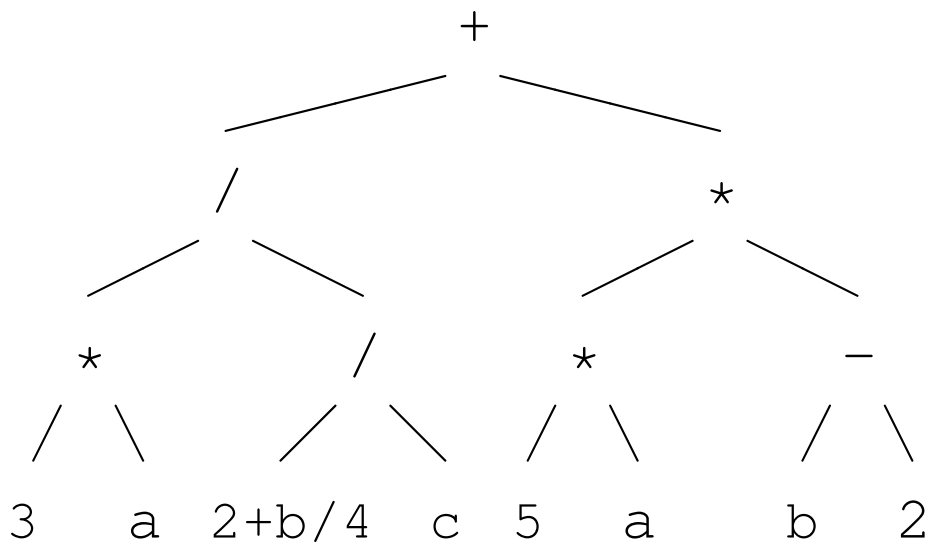
4. Use a tree to convert the expression
`3*a/(2 + b/4)/c + (5*a)*(b-2)` into postfix form.

expression trees

`3*a/(2 + b/4)/c + (5*a)*(b-2)`

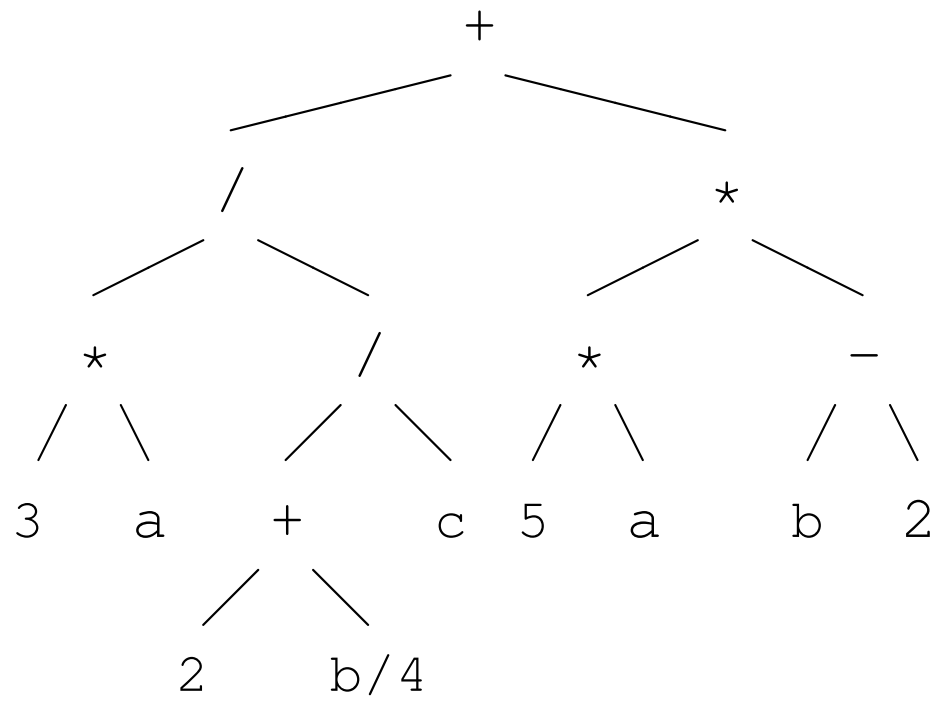


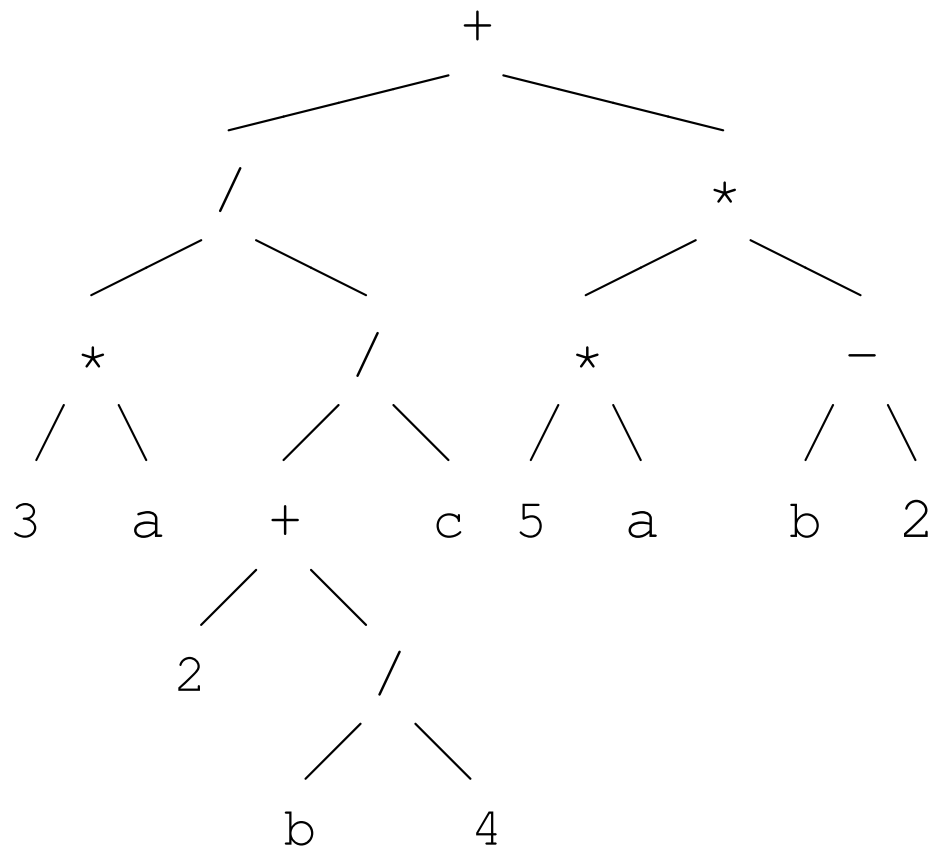




expression trees

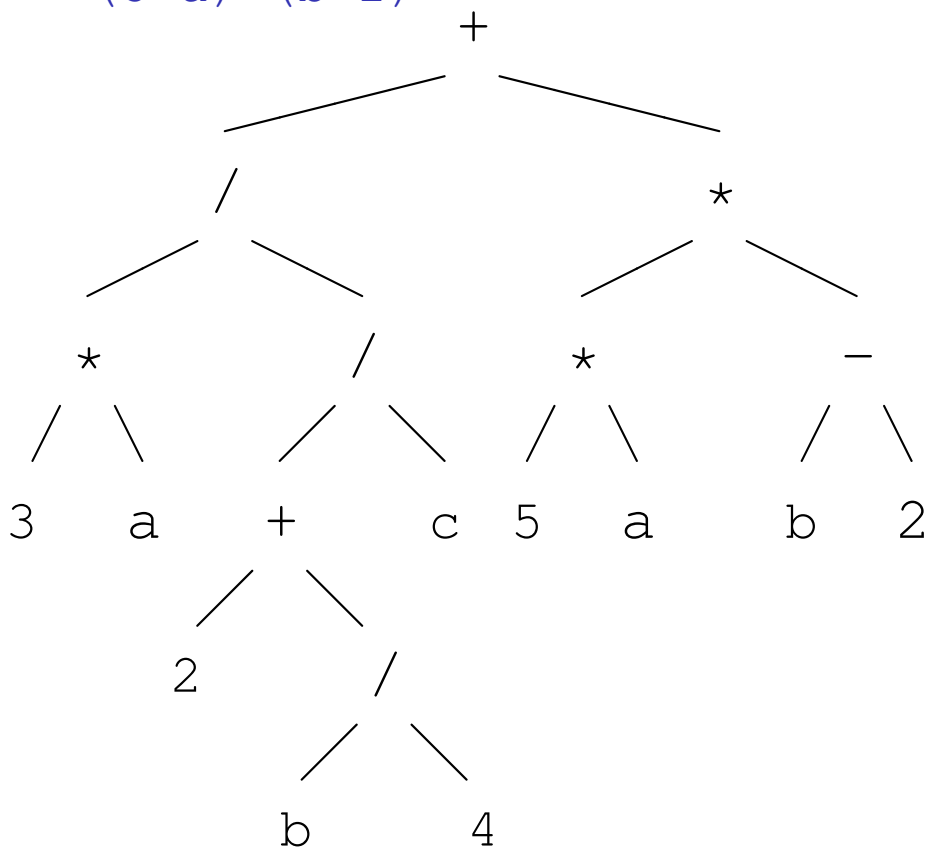
`3*a/(2 + b/4)/c + (5*a)*(b-2)`





postorder traversal

`3*a/(2 + b/4)/c + (5*a)*(b-2)`



A postorder traversal yields the postfix expression:

```
3 a * 2 b 4 / + c / 5 a * b 2 - * +
```

Huffman codes

5. Create a Huffman code for the message `"structure"`.
   1. Give code to use the appropriate data structure of the STL to make a frequency table for the characters in the message.
   2. Given the frequency table, make the Huffman code.
      Draw all intermediate stages in the creation of the Huffman code.
   3. Show how to encode and decode the message.

code to make a frequency table with STL map

```
map<char, int> frequency_table ( string s )
{
   map<char, int> M;

   for(int i=0; i<s.size(); i++)
   {
      char c = s[i];
      if(M.find(c) == M.end())
         M[c] = 1;
      else
         M[c]++;
   }
   return M;
}
```

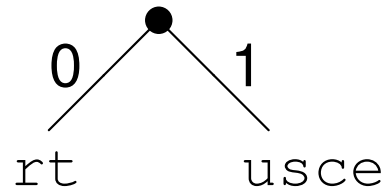
contracting frequency tables

5. Create a Huffman code for the message `"structure"`.

·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$
c	1	s	1	r	2	u	2	rt	4
e	1	ce	2	t	2	sce	3	usce	5
s	1	r	2	u	2	rt	4		
r	2	t	2	sce	3				
t	2	u	2						
u	2								

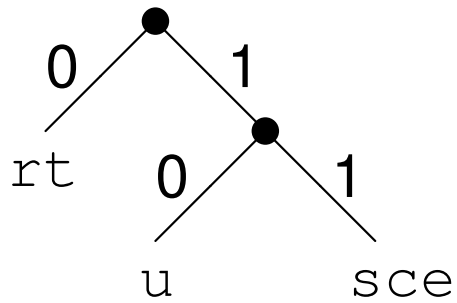
making the tree

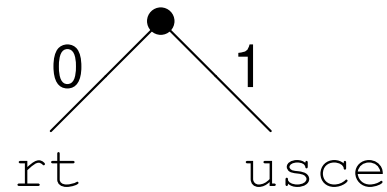
·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$
c	1	s	1	r	2	u	2	rt	4
e	1	ce	2	t	2	sce	3	usce	5
s	1	r	2	u	2	rt	4		
r	2	t	2	sce	3				
t	2	u	2						
u	2								



splitting the leaf `usce`

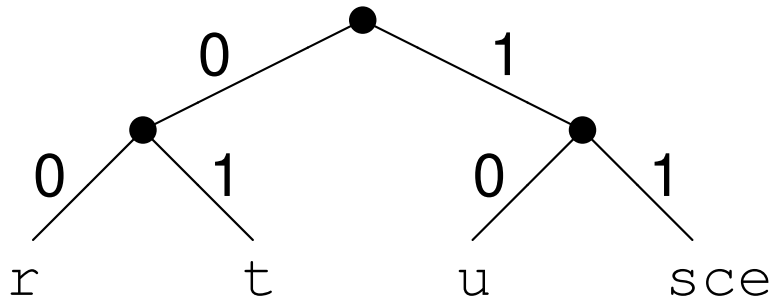
.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$
c	1	s	1	r	2	u	2	rt	4
e	1	ce	2	t	2	sce	3	usce	5
s	1	r	2	u	2	rt	4		
r	2	t	2	sce	3				
t	2	u	2						
u	2								

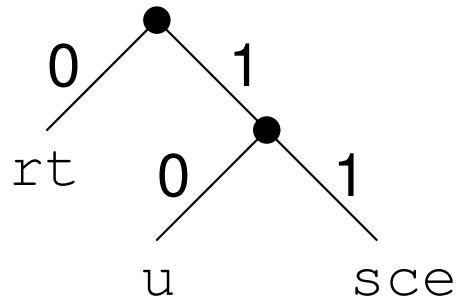




splitting the leaf `rt`

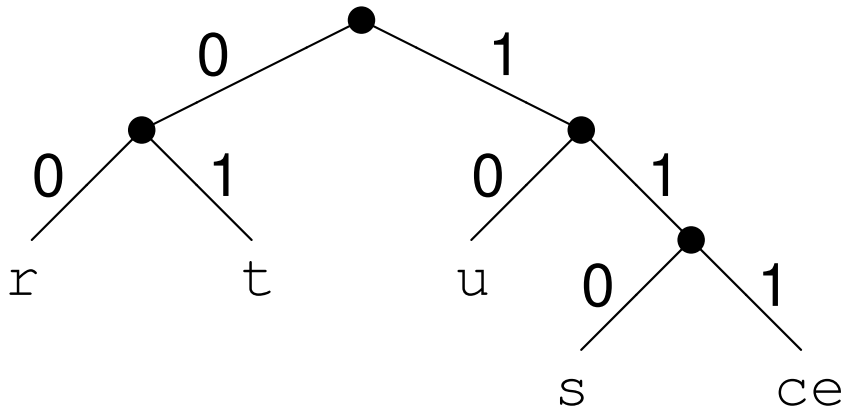
.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$
c	1	s	1	r	2	u	2	rt	4
e	1	ce	2	t	2	sce	3	usce	5
s	1	r	2	u	2	rt	4		
r	2	t	2	sce	3				
t	2	u	2						
u	2								

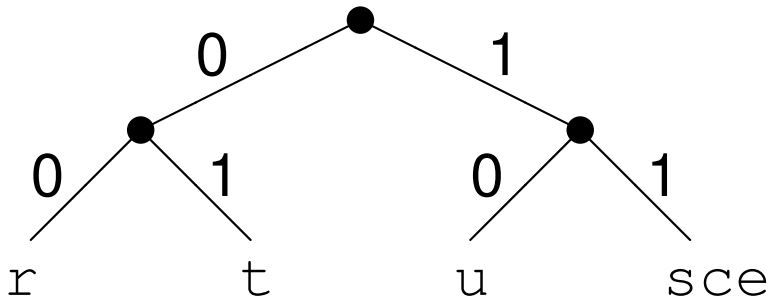




splitting the leaf sce

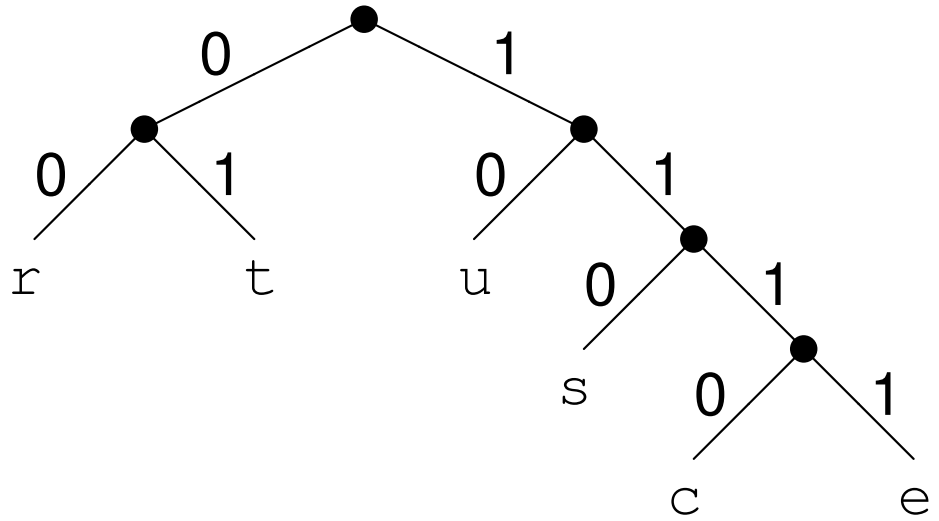
.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$	.	$f(\cdot)$
c	1	s	1	r	2	u	2	rt	4
e	1	ce	2	t	2	sce	3	usce	5
s	1	r	2	u	2	rt	4		
r	2	t	2	sce	3				
t	2	u	2						
u	2								





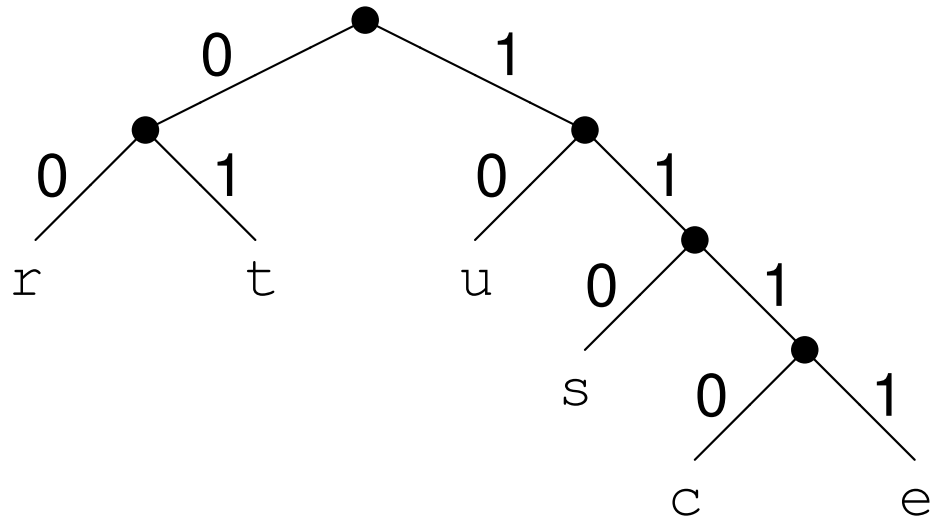
splitting the leaf ce

·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$	·	$f(\cdot)$
c	1	s	1	r	2	u	2	rt	4
e	1	ce	2	t	2	sce	3	usce	5
s	1	r	2	u	2	rt	4		
r	2	t	2	sce	3				
t	2	u	2						
u	2								



a Huffman code for `structure`

char	code
r	00
t	01
u	10
s	110
c	1110
e	1111



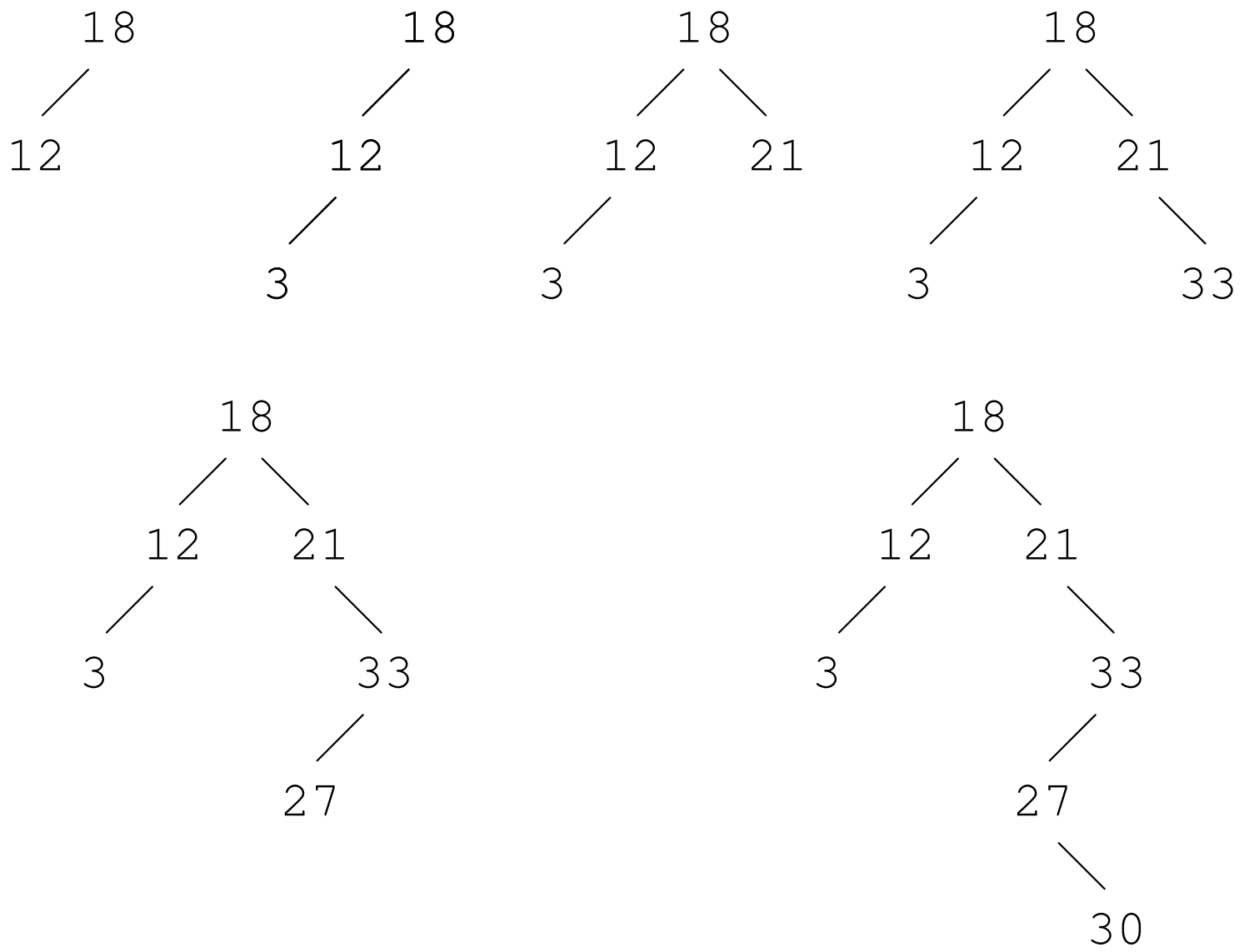
The encoding of `structure` is `11001001011100110001111`.

binary search trees

6. Consider the sequence `18 12 3 21 33 27 30`.
   1. Insert the elements in the sequence into a binary search tree.
   2. Delete 21 from the binary search tree.
   3. Delete 18 from the binary search tree.

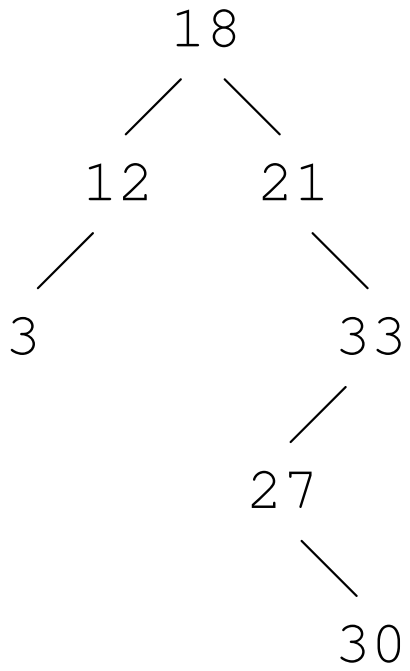
inserting numbers into a binary search tree

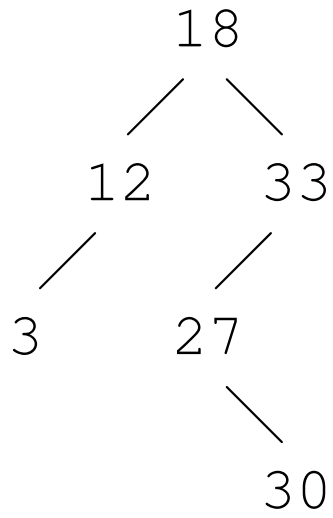
consider the sequence `18 12 3 21 33 27 30`.



deleting a number from a binary search tree

Delete `21` from:

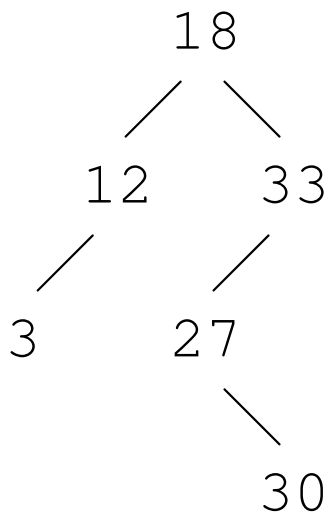


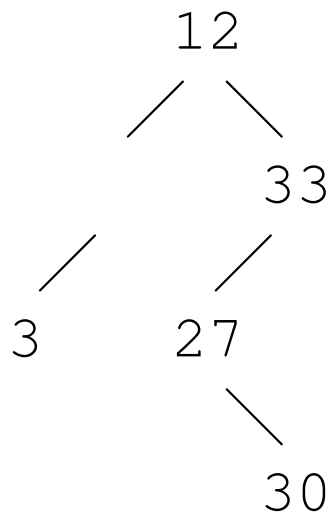


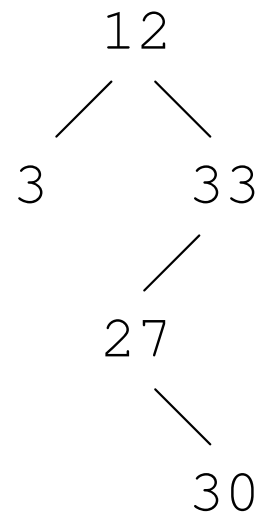
Observe that `21` had no left child:
replacing `21` with its right child yields a binary search tree.

deleting a number from a binary search tree

Delete `18` from:







Observe that `18` had two children:

- replace `18` with the largest in its left child: `12`
- `12` has no right child, its removal is by replacement by its left child.

Review of the Lectures 21-26, 30-32

1. The Final Exam
   - Monday 11 December, BSB 337, from 8AM to 10AM

2. Examples of Questions
   - recursion and memoization
   - enumeration
   - trees, binary search trees, Huffman codes
   - hashing and sorting

hashing

7. What is open addressing? What is chaining?
   Explain the differences between open addressing and chaining.
   Give an example of both techniques to illustrate the differences.

sorting algorithms

7. Consider the sequence `43 19 12 31 62 81 77 10`.
   1. Sort the numbers in ascending order with bubble sort. Show all intermediate passes.
   2. Sort the sequence with quicksort. Show all intermediate stages in the sorting algorithm.
   3. Show how to sort the numbers, while selecting each time the smallest element in the sequence and swapping if necessary.
   4. Sort the sequence by inserting in a binary search tree.
   5. Show how to build the heap in heapsort. Once the heap is built show how removal produces a sequence in ascending order.
   6. Apply Shell sort to the sequence, using values 5,3, and 1 for the gap.
   7. Show how to apply merge sort to this sequence.
   8. Sort the sequence with insertion sort.

balance of a tree

8. Give the code to compute the balance of a tree, using the definitions

```
typedef struct node tree;
struct node
{
   int info;
   tree *left;
   tree *right;
};
```

and the prototype

```
void write_balance ( int k, tree *t );
/* writes the info and balance of every node
   in inorder traversal, k controls the depth
   of the recursion (call with k == 0) */
```

the function `write_balance`

```
void write_balance ( int k, tree *t )
/* writes the info and balance of every node
   in inorder traversal, k controls the depth
   of the recursion (call with k == 0) */
{
   if(t != NULL)
   {
      for(int i=0; i<k; i++) cout << "   ";
      cout << t->info
           << " balance : " << balance(t) << endl;
      write_balance(k+1,t->left);
      write_balance(k+1,t->right);
   }
}
```

the function `balance`

```
int balance ( tree *t );
// returns the balance of the tree

int balance ( tree *t )
{
   if(t == NULL)
      return 0;
   else
      return depth(t->left) - depth(t->right);
}
```

computing the depth of a tree

```
int depth ( tree *t )
{
   if(t == NULL)
      return -1;
   else if(t->left == NULL)
      return 1+depth(t->right);
   else if(t->right == NULL)
      return 1+depth(t->left);
   else
   {
      int L = depth(t->left);
      int R = depth(t->right);
      if(L > R)
         return L + 1;
      else
         return R + 1;
   }
}
```