Introduction to Algorithms

CELEN086

Seminar 4
(w/c 28/10/2024)

Outline

In this seminar, we will study and review on following topics:

- Basic list operations
- Design recursive algorithms for list

You will also learn useful Math/CS concepts and vocabularies.

Home Work Question :

An abundant number is a natural number whose distinct proper factors have a sum exceeding that number.
Thus, 12 is abundant because $1+2+3+4+6 > 12$.

Write an algorithm **isabundant** which tests whether or not a number is abundant.

Note : You may /may not use helper algorithm to complete the task.

Algorithm: isabundant(n)
Requires: a positive integer n
Returns: True if n is abundant otherwise False.

1. Let sum = isabundant_helper(n,1,0)
2. if sum>n then
3. return True
4. else
5. return False
6. endif

Algorithm: isabundant_helper(x,c,s)

Requires: three integers x,c and s.

Returns: sum of all proper factors of x.

1. If c ==x then
2. return s
3. else if(x mod c ==0)then
4. return helper(x,c+1,s+c)
5. else
6. return helper(x,c+1, s)
7. endif

Trace : if n =12
x=12, c =1 , s =0

Algorithm: isabundant(n)
Requires: a positive integer n
Returns: True if n is abundant otherwise False.

1. Let sum = isabundant_helper(n,1,0)
2. if sum>n then
3. return True
4. else
5. return False
6. endif

Algorithm: isabundant_helper(x,c,s)

Requires: three integers x,c and s.

Returns: sum of all proper factors of x.

1. If i==x then
2. return s
3. else if(x mod c ==0)then
4. return helper(x,c+1,s+i)
5. else
6. return helper(x,c+1, s)
7. endif

List command

nil or [ ]	returns empty list
value(list)	returns the first element (value) of a list
tail(list)	returns the rest of a list (except the value element)
isEmpty(list)	returns True if the list is empty, False otherwise.
cons(x, list)	returns a list with element x added to the front

Note: value( ) and tail( ) only work on nonempty list.

Practice

1. Create the list L = [1, 2, 3, 4] using the basic list commands.

   cons(1, cons(2, cons(3, cons(4, nil))))

2. What does the pseudocode return?

cons(value([1,2,3,4]), cons(value(tail([1,2,3,4])),cons(5, tail(tail([1,2,3,4])))))

= cons(1, cons(2, cons(5, [3, 4])))

= cons(1, cons(2, [5, 3, 4]))

= cons(1, [2, 5, 3, 4])

= [1, 2, 5, 3, 4]

Example

Design the recursive algorithm to count how many times the number 5 appears in a list using the functions isEmpty, value, and tail.

Algorithm: countFives(list)
Requires: A list
Returns: The count of occurrences of the number 5 in the list
1. if isEmpty(list)
2.   return 0 // Base case: if the list is empty, return 0
3. else if value(list) == 5
4.   return 1 + countFives(tail(list)) // Count this occurrence and recurse on the rest of the list
5. else
6.   return countFives(tail(list)) // No occurrence here, recurse on the rest of the list
7. endif

Explanation:
1. The base case checks if the list is empty using isEmpty(list). If it is, the function returns 0 since there are no elements left to count.
2. If the first element (obtained using value(list)) is 5, the function adds 1 to the result of the recursive call on the rest of the list (tail(list)).
3. If the first element is not 5, the function makes a recursive call on the rest of the list (tail(list)) without incrementing the count.

Trace countFives([5,3,51,5])

For the list [5, 3, 5, 1, 5]:

1. Initial call: countFives([5, 3, 5, 1, 5])
First element is 5, so:
1 + countFives([3, 5, 1, 5])
2. Recursive call: countFives([3, 5, 1, 5])
First element is not 5, so:
countFives([5, 1, 5])
3. Recursive call: countFives([5, 1, 5])
First element is 5, so:
1 + countFives([1, 5])
4. Recursive call: countFives([1, 5])
First element is not 5, so:
countFives([5])
5. Recursive call: countFives([5])
First element is 5, so:
1 + countFives([ ])
6. Base case: countFives([ ])
List is empty, return 0.

Backtracking:

Now, we backtrack and sum up the counts:

- countFives([ ]) returns 0.
- countFives([5]) becomes 1 + 0 = 1.
- countFives([1, 5]) becomes 0 + 1 = 1.
- countFives([5, 1, 5]) becomes 1 + 1 = 2.
- countFives([3, 5, 1, 5]) becomes 0 + 2 = 2.
- countFives([5, 3, 5, 1, 5]) becomes 1 + 2 = 3.

Final result:
The function returns 3, as there are three occurrences of the number 5 in the list.

Example

Write a recursive algorithm index(x,L) that returns the position number of element x in the list L.

Example: index(8, [2,9,8,5]) = 3

Algorithm: index(x,L)
Requires: a number x and a list L containing x
Returns: position number (index) of x in the list

```
1. if  x==value(L)
2.      return  1
3. else
4.      return  1+index(x,tail(L))
5. endif
```

Trace index(8,[2,9,8,5])

x=8, L=[2,9,8,5]

8==2? False.   return 1+index(8, [9,8,5]) =1+2=3

x=8, L=[9,8,5]

8==9? False.   return 1+index(8, [8,5]) =1+1=2

x=8, L=[8,5]

8==8? True.   return 1   (backtracking)

Practice

Write a recursive algorithm getNth(n,L) that returns the n-th element of list L.

Algorithm: getNth(n,L)
Requires: a positive integer n and a list L
Returns: the n-th element of the list

Example: getNth(3, [2,9,8,5]) = 8

---

1. if n==1
2.   return value(L)
3. else
4.   return getNth(n-1, tail(L))
5. endif

Practice

Write a recursive algorithm delNth(n,L) that deletes the n-th element of list L.

Algorithm: delNth(n,L)
Requires: a positive integer n and a list L
Returns: a list after deletion of the n-th element in L

Example: delNth(3, [2,9,8,5]) = [2,9,5]

```
1.  if  n==1
2.      return  tail(L)
3.  else
4.      return  cons(value(L), delNth(n-1, tail(L)))
5.  endif
```

Trace delNth(3,[2,9,8,5])

n=3, L=[2,9,8,5]

return cons(2, delNth(2,[9,8,5]))

=cons(2, [9,5])=[2,9,5]

n=2, L=[9,8,5]

return cons(9, delNth(1,[8,5]))

=cons(9, [5])=[9,5]

n=1, L=[8,5]

return [5]

Homework Exercise

Write a recursive algorithm cut(L, i, j) that takes a nonempty list L and cuts the elements from the i-th to j-th positions ($1 \leq i \leq j \leq length(L)$) and returns the rest of the list.

Example: cut([2,9,8,5,7,4,3], 3,5) = [2,9,4,3]

(You can call the function delNth(n,L) and use it as a sub-algorithm directly.)

Solution

Algorithm: cut(L, i, j)
Requires: a nonempty list L and two positive integer i, j with $1 \leq i \leq j \leq length(L)$
Returns: a list after deletion of the i-th to j-th elements in L.

```
if  i==j then
    return  delNth(i,L) // base case
else
    return  cut(delNth(i,L),i,j-1) // recursive step
endif
```

You should trace this algorithm with the given example.