**Dosar de laborator**

**Ielciu Mara**

**Grupa 924/1**

**Documentatie Laborator1 PLF (R1)**

**Ielciu Mara**

**Data: 14.10.2024**

**Cerinta R1 - 7:**

1. Test the equality of two lists
2. Determine the intersection of two sets represented as lists

**Modelul matematic pentru recursivitate:**

a)

b)

Source code:

class Nod:  
 def \_\_init\_\_(self, e):  
 self.e = e  
 self.urm = None  
  
  
class Lista:  
 def \_\_init\_\_(self):  
 self.prim = None  
  
 def cauta(self, e):  
 return self.cauta\_rec(e, self.prim)  
  
 def cauta\_rec(self, e, nod):  
 if nod is None:  
 return False  
 if nod.e == e:  
 return True  
 return self.cauta\_rec(e, nod.urm)  
  
 def submultime(self, lista2):  
 return self.submultime\_rec(self.prim, lista2.prim)  
  
 def submultime\_rec(self, nod1, nod2):  
 if nod1 is None:  
 return True  
 if nod2 is None:  
 return False  
 if self.cauta\_rec(nod1.e, nod2):  
 return self.submultime\_rec(nod1.urm, nod2)  
 return False  
  
 def lungime(self):  
 return self.lungime\_rec(self.prim)  
  
 def lungime\_rec(self, nod):  
 if nod is None:  
 return 0  
 return 1 + self.lungime\_rec(nod.urm)  
  
 def egale(self, lista2):  
 if self.lungime() != lista2.lungime():  
 return False  
 return self.submultime(lista2) and lista2.submultime(self)  
  
 def intersectie(self, lista2):  
 return self.intersectie\_rec(self.prim, lista2)  
  
 def intersectie\_rec(self, nod1, lista2):  
 if nod1 is None:  
 return None  
 if lista2.cauta(nod1.e):  
 new\_nod = Nod(nod1.e)  
 new\_nod.urm = self.intersectie\_rec(nod1.urm, lista2)  
 return new\_nod  
 else:  
 return self.intersectie\_rec(nod1.urm, lista2)  
  
 def creare\_din\_nod(self, nod):  
 lista = Lista()  
 lista.prim = nod  
 return lista  
  
  
'''  
crearea unei liste din valori citite pana la 0  
'''  
  
  
def creareLista():  
 lista = Lista()  
 lista.prim = creareLista\_rec()  
 return lista  
  
  
def creareLista\_rec():  
 x = int(input("x="))  
 if x == 0:  
 return None  
 else:  
 nod = Nod(x)  
 nod.urm = creareLista\_rec()  
 return nod  
  
  
  
  
'''  
tiparirea elementelor unei liste  
'''  
  
  
def tipar(lista):  
 tipar\_rec(lista.prim)  
  
  
def tipar\_rec(nod):  
 if nod is not None:  
 print(nod.e)  
 tipar\_rec(nod.urm)  
  
  
def main():  
 print("Creare lista 1:")  
 lista1 = creareLista()  
 tipar(lista1)  
  
 print("Creare lista 2:")  
 lista2 = creareLista()  
 tipar(lista2)  
  
 print("Lista 1 egala cu lista 2?", lista1.egale(lista2))  
 print("Intersectie dintre lista 1 si lista 2:")  
 intersectie\_nod = lista1.intersectie(lista2)  
 intersectie\_lista = lista1.creare\_din\_nod(intersectie\_nod)  
 tipar(intersectie\_lista)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Documentatie Laborator2 PLF (P1)**

**Ielciu Mara**

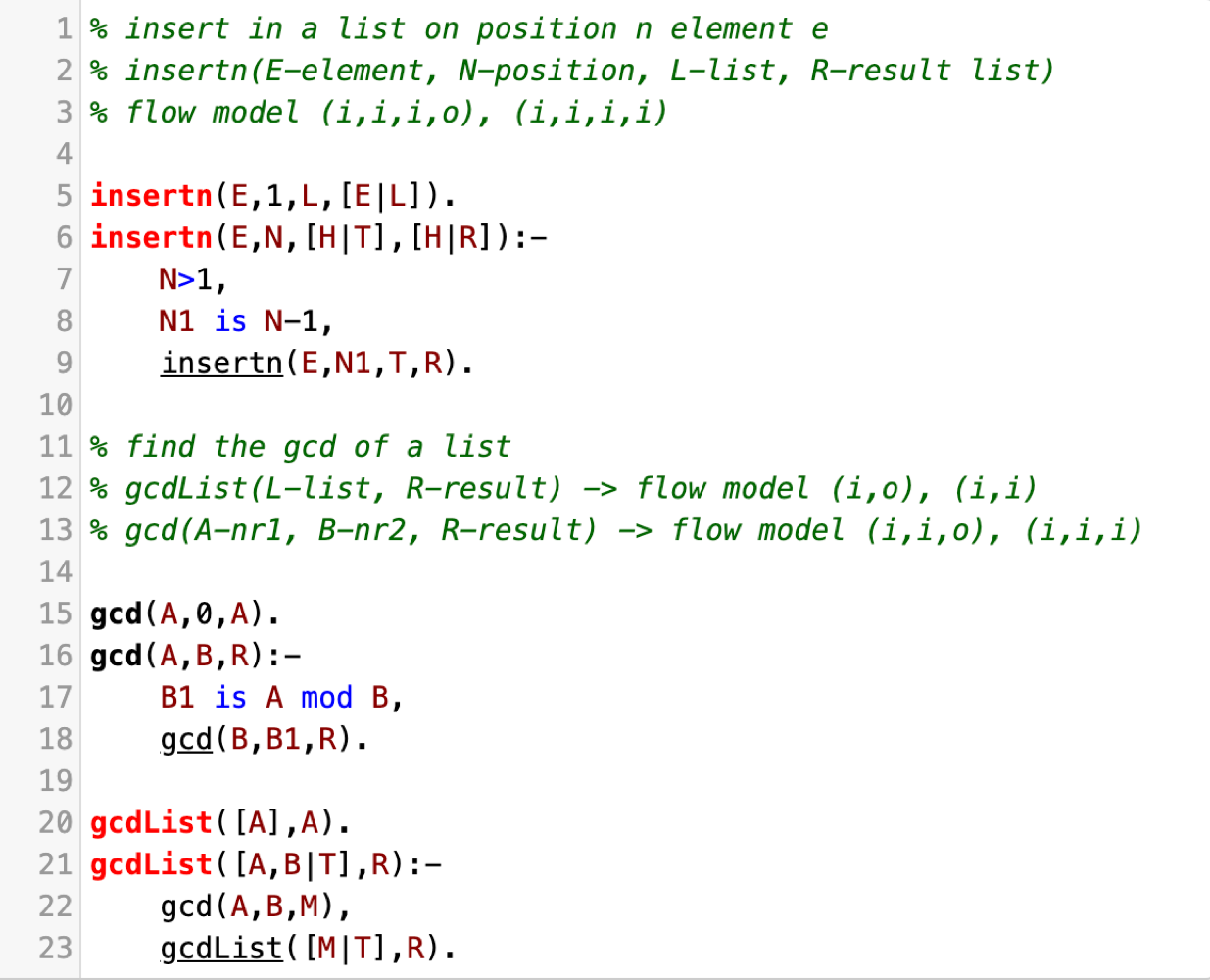
**Data: 14.10.2024**

**Cerinta P1 - 9:**

1. Insert an element on the position n in a list
2. Define a predicate to determine the greatest common divisor of a list

**Modelul matematic pentru recursivitate:**

Sorce code:



Test examples: insertn(4,2,[1,2,1,1],R). -> R=[1,4,2,1,1]

gcdList([9,18,12,21],R). -> R=3

**Documentatie Laborator3 PLF (P2)**

**Ielciu Mara**

**Data: 28.10.2024**

**Cerinta P2 - 11:**

1. Replace all occurences of an element from a list with another element e.
2. For a heterogenous list, formed from integer numbers and list of numbers, define a predicate to determine the maximum number of the list, and then to replace this value in sublists with the maximum value of the sublist.

**Modelul matematic pentru recursivitate:**

Source code:

% replaceAll(E1-elem to replace, E2-replacer, L-list, R-result list)

% flow model(i,i,i,i), (i,i,i,o)

replaceAll(\_,\_,[],[]).

replaceAll(E1,E2,[E1|T],[E2|R]):-

replaceAll(E1,E2,T,R),!.

replaceAll(E1,E2,[H|T],[H|R]):-

replaceAll(E1,E2,T,R).

% maxList(L-list, M-max, R-result)

% flow model (i,i,i), (i,i,o)

maxList([],M,M).

maxList([H|T],M,R):-

H>M,

NewM=H,

maxList(T,NewM,R),!.

maxList([H|T],M,R):-

M>=H,

maxList(T,M,R).

mainMaxList(L,R):-

maxList(L,0,R).

% maxHeteroList(L-list, M-max, R-result)

% flow model (i,i,i), (i,i,o)

maxHeteroList([],M,M).

maxHeteroList([H|T],M,R):-

is\_list(H),

maxHeteroList(T,M,R).

maxHeteroList([H|T],M,R):-

\+ is\_list(H),

H>M,

NewM=H,

maxHeteroList(T,NewM,R),!.

maxHeteroList([H|T],M,R):-

\+ is\_list(H),

M>=H,

maxHeteroList(T,M,R).

mainMaxHeteroList(L,R):-

maxHeteroList(L,0,R).

% replaceMax(L-list, M-max, R-result list)

% flow model (i,i,i), (i,i,o)

replaceMax([],\_, []).

replaceMax([H|T], M,[NewH|R]) :-

is\_list(H),

mainMaxList(H, Max),

replaceAll(M, Max, H, NewH),

replaceMax(T,M,R),!.

replaceMax([H|T],M, [H|R]) :-

\+ is\_list(H),

replaceMax(T,M,R).

mainReplaceMax(L,R):-

mainMaxHeteroList(L,M),

replaceMax(L,M,R).

Test examples:

mainReplaceMax([1,[2,5,7],4,5,[1,4],3,[1,3,5,8,5,4],5,[5,9,1],2],R). -> R = [1, [2, 7, 7], 4, 5, [1, 4], 3, [1, 3, 8, 8, 8, 4], 5, [9, 9, 1], 2]

replaceAll(1,3,[1,7,8,1,2,5,1],R). -> R=[3,7,8,3,2,5,3]

**Documentatie Laborator4 PLF (P3)**

**Ielciu Mara**

**Data: 11.11.2024**

**Cerinta P3 - 7:**

A player wants to choose the predictions for 4 games. The predictions can be 1, X, 2. Write a predicate to generate all possible variants considering that: last prediction can’t be 2 and no more than two possible predictions of X.

**Modelul matematic pentru recursivitate:**

% prediction(E-element counter, R-result list, Len-length)

% flow model (i,o,i)

% Base case - when the list has exactly 4 predictions, it is valid.

prediction(\_, [], 4).

% Case for adding '1'

prediction(X, [1|Rest], Length) :-

Length < 4,

NewLength is Length + 1,

prediction(X, Rest, NewLength).

% Case for adding 'X'

prediction(XCount, [x|Rest], Length) :-

Length < 4,

XCount < 2, % Only allow up to 2 'X' predictions

XNewCount is XCount + 1,

NewLength is Length + 1,

prediction(XNewCount, Rest, NewLength).

% Case for adding '2'

prediction(X, [2|Rest], Length) :-

Length < 3, % Ensure '2' is not the last element

NewLength is Length + 1,

prediction(X, Rest, NewLength).

predictions(Result) :-

findall(R, (prediction(0, R, 0), length(R, 4)), Result).

**Documentatie Laborator5 PLF (L1)**

**Ielciu Mara**

**Data: 25.11.2024**

**Cerinta L1 - 9:**

1. merge 2 sorted lists keeping the duplicates
2. replace an element E with all elements of list L1 at all levels of list L
3. sum of two numbers in list representation
4. greatest common divisor (GCD) of all numbers in a list

**Modelul matematic pentru recursivitate:**

a)

b)

c)

d)

**Cod sursa:**

; a) merge 2 sorted lists keeping the duplicates

(defun merge\_lists(l1 l2)

(cond

((and (null l1) (null l2)) nil)

((null l1) l2)

((null l2) l1)

((< (car l1) (car l2)) (cons (car l1) (merge\_lists (cdr l1) l2)))

(T (cons (car l2) (merge\_lists l1 (cdr l2))))

)

)

;(print (merge\_lists '(1 2 3 3) '(3 4 4 5 5 7)))

; b) replace an element E with all elements of list L1 at all levels of list L

(defun replace\_element(e l1 l)

(cond

((null l) nil)

((atom (car l)) (cond

((equal e (car l)) (append l1 (replace\_element e l1 (cdr l))))

(T (cons (car l) (replace\_element e l1 (cdr l))))

))

(T (cons (replace\_element e l1 (car l)) (replace\_element e l1 (cdr l))))

)

)

;(print (replace\_element 'b '(1 2 3) '(a (b c) a (a b c) b)))

; c) sum of two numbers in list representation

(defun digit (l k c)

(cond

((null l) (mod (+ k c) 10))

((null k) (mod (+ l c) 10))

(T (mod (+ l k c) 10))

)

)

(defun carry (l k c)

(cond

((null l) (floor (+ k c) 10))

((null k) (floor (+ l c) 10))

(T (floor (+ l k c) 10))

)

)

(defun sum\_lists (l k c)

(cond

((and (null l) (null k)) (if (= 1 c) (list 1) nil))

(T (append (sum\_lists (cdr l) (cdr k) (carry (car l) (car k) c)) (list (digit (car l) (car k) c))))

)

)

(defun compute\_sum(l1 l2)

(cond

(t (sum\_lists (reverse l1) (reverse l2) 0))

)

)

;(print (compute\_sum '(1 1 1) '(2 3 5)))

; d) greatest common divisor (GCD) of all numbers in a list

(defun \_gcd(a b)

(

cond

((and (not (numberp a)) (not (numberp b))) nil)

((not (numberp a)) b)

((not (numberp b)) a)

((= b 0) a)

(T (\_gcd b (mod a b)))

)

)

(defun gcd\_list(l)

(cond

((null l) nil)

((null (cdr l)) (car l))

(T (\_gcd (car l) (gcd\_list (cdr l))))

)

)

(print (gcd\_list '(24 16 12 A B 72)))

**Documentatie Laborator6 PLF (L2)**

**Ielciu Mara**

**Data: 9.12.2024**

**Cerinta L2 - 11:**

Return the level (and the list of nodes) with the maximum nodes for a tree of representation (2)

**Modelul matematic pentru recursivitate:**

(defun myMax (a b)

"Return the maximum of two numbers, or the valid one if one is not a number."

(cond

((not (numberp a)) b)

((not (numberp b)) a)

((>= a b) a)

(t b)))

(defun countNodesOnLevel (vector level &optional (currentLevel 0))

"Count the number of nodes on a specific level in the binary tree."

(cond

((null vector) 0)

((= level currentLevel) 1)

(t (+ (countNodesOnLevel (cadr vector) level (1+ currentLevel))

(countNodesOnLevel (caddr vector) level (1+ currentLevel))))))

(defun getTotalLevels (vector &optional (currentLevel 0))

"Calculate the total number of levels in the binary tree."

(cond

((null vector) currentLevel)

(t (myMax (getTotalLevels (cadr vector) (1+ currentLevel))

(getTotalLevels (caddr vector) (1+ currentLevel))))))

(defun getMaxChildren (vector &optional (maxChildren 0) (totalLevels (getTotalLevels vector)) (currentLevel 0))

"Find the maximum number of nodes at any level."

(cond

((>= currentLevel totalLevels) maxChildren)

(t (let ((currentCount (countNodesOnLevel vector currentLevel)))

(if (> currentCount maxChildren)

(getMaxChildren vector currentCount totalLevels (1+ currentLevel))

(getMaxChildren vector maxChildren totalLevels (1+ currentLevel)))))))

(defun getMaxLevel (vector &optional (maxChildren 0) (totalLevels (getTotalLevels vector)) (currentLevel 0))

"Find the level with the maximum number of nodes."

(cond

((>= currentLevel totalLevels) currentLevel)

(t (let ((currentCount (countNodesOnLevel vector currentLevel)))

(if (= currentCount maxChildren)

currentLevel

(getMaxLevel vector maxChildren totalLevels (1+ currentLevel)))))))

(defun getNodesOnLevel (vector level &optional (currentLevel 0))

"Get all nodes on a specific level in the binary tree."

(cond

((null vector) nil)

((= level currentLevel) (list (car vector)))

(t (append (getNodesOnLevel (cadr vector) level (1+ currentLevel))

(getNodesOnLevel (caddr vector) level (1+ currentLevel))))))

(defun main (vector)

"Find the level with the most nodes and return the nodes on that level."

(let\* ((totalLevels (getTotalLevels vector))

(maxChildren (getMaxChildren vector 0 totalLevels))

(maxLevel (getMaxLevel vector maxChildren totalLevels)))

(format t "Nivelul cu cele mai multe noduri: ~s~%" maxLevel)

(getNodesOnLevel vector maxLevel)))

(print (main '(A (B (D (G) (H)) (E (I))) (C (F) (X)))))

**Documentatie Laborator7 PLF (L3)**

**Ielciu Mara**

**Data: 6.1.2025**

**Cerinta L3- 7:**

Replace every occurrence of the element E with the list l1 on every level of the list l.

**Modelul matematic pentru recursivitate:**

**Cod sursa:**

(defun replace-element-with-list (e l1 l)

(mapcar (lambda (x)

(cond

((and (atom x) (equal x e)) l1)

((listp x) (replace-element-with-list e l1 x))

(t x)))

l)

)

(print (replace-element-with-list 'a '(1 2 3) '(b a (c a (d a)))))

(defun repl(e l1 l)

(cond

((and (atom l) (equal l e)) l1)

((listp l) (mapcar (lambda(x) (repl e l1 x)) l))

(t l)

)

)

(print (repl 'a '(1 2 3) '(b a (c a (d a)))))