Θεόδωρος – Νικόλαος Παπαγεωργίου

ics20025

ics20025@uom.edu.gr

1) List Processing

Έστω το **Prolog** κατηγόρημα **indicate_change(List, Change)** στο οποίο η λίστα **List** περιέχει ακεραίους και το οποίο επιτυγχάνει όταν η λίστα **Change** περιέχει την αλλαγή (**no_c**, **up**, **down**) για τις συνεχόμενες τιμές της λίστας **List** (**no_c** = no_change).

Κώδικας:

```
%%% indicate_change/2 where it succeeds when the Change list contains the changes
%%% between the numbers of the first list (contains up, down, no_c (no change))
indicate_change([H], []).

indicate_change([H1, H2|T], [Change|Changes]):-
      changes(H1, H2, Change),
      indicate_change([H2|T], Changes).

%%% The changes between the numbers, if both numbers are equal returns no_c (no change)
changes(Number, Number, no_c).

%%% The changes between the numbers, if the second number is greater, returns up
changes(Number1, Number2, up):-
      Number1 < Number2.

%%% The changes between the numbers, if the second number is less, returns down
changes(Number1, Number2, down):-
      Number1 > Number2.
```

Σχόλια:

Το κατηγόρημα **indicate_change/2**, δέχεται μία λίστα και ανάλογα με τα δύο πρώτα στοιχεία της γεμίζει την δεύτερη λίστα με τις απαντήσεις **up**, **down** και **no_c** εάν τα στοιχεία έχουν διαφορά προς τα πάνω, κάτω και καμία αλλαγή αντίστοιχα. Στην βασική περίπτωση η **indicate_change** επιστρέφει την κενή λίστα.

Παραδείγματα Εκτέλεσης:

```
[eclipse 1]: ?- indicate_change([1, 1], C).

C = [no_c]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 2]: ?- indicate_change([1, 2], C).

C = [up]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 3]: ?- indicate_change([1, 0], C).

C = [down]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 4]: ?- indicate_change([1, 2, 1, 4], C).

C = [up, down, up]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 5]: ?- indicate_change([1, 1, 3, 3, 4], C).

C = [no_c, up, no_c, up]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 6]: ?- indicate_change([1], C).

C = []
Yes (0.00s cpu, solution 1, maybe more) ?
```

Ο παραπάνω κώδικας δεν παρουσιάζει -φανερά τουλάχιστον- προβλήματα και εκτελείται όπως ζητείται.

2) List Processing

Να δώσετε τον ΜΗ αναδρομικό ορισμό του κατηγορήματος της άσκησης 1, σε νέο κατηγόρημα **indicate_change_alt(List,Change)**, το οποίο εμφανίζει ακριβώς την ίδια συμπεριφορά με το κατηγόρημα **indicate_change/2**, δηλαδή πετυχαίνει για όλες τις περιπτώσεις που περιγράφονται παραπάνω.

Κώδικας:

```
%%% Alternative indicate_change %%%

%%% Finds all the Changes when for every X with combinations of H1 and H2 that create the List
indicate_change_alt(List, Changes) :-
      findall(X, (append(_, [H1,H2|_], List), changes(H1, H2, X)), Changes).
```

Σχόλια:
Το κατηγόρημα **indicate_change_alt/2**, δέχεται μία λίστα και ανάλογα με τα στοιχεία της, ελέγχει τις αλλαγές (χρησιμοποιώντας τα changes που αναφέρονται στην (1η) άσκηση) και μέσω της **findall/3** επιστρέφει τις **Changes** ελέγχοντας για τις αλλαγές μεταξύ των δύο στοιχείων της **List**.

Παραδείγματα Εκτέλεσης:

```
[eclipse 1]: ?- indicate_change_alt([1, 1, 3, 3, 4], C).

C = [no_c, up, no_c, up]
Yes (0.00s cpu)
[eclipse 2]:
        ?- indicate_change_alt([1, 1, 3, 3, 4, 1, 1], C).

C = [no_c, up, no_c, up, down, no_c]
Yes (0.00s cpu)
[eclipse 3]:
?- indicate_change_alt([1, 1], C).

C = [no_c]
Yes (0.00s cpu)
```

```
[eclipse 4]: indicate_change_alt([1,2], C).

C = [up]
Yes (0.00s cpu)
[eclipse 5]: indicate_change_alt([1,0], C).

C = [down]
Yes (0.00s cpu)
[eclipse 6]: indicate_change_alt([1,2,1,4], C).

C = [up, down, up]
Yes (0.00s cpu)
[eclipse 7]: indicate_change_alt([1], C).

C = []
Yes (0.00s cpu)
```

Ο παραπάνω κώδικας δεν παρουσιάζει -φανερά τουλάχιστον- προβλήματα και εκτελείται όπως ζητείται.

3) Rolling Average

Ο κινητός μέσος όρος μιας χρονοσειράς κατασκευάζεται λαμβάνοντας μέσους όρους **N** διαφόρων διαδοχικών τιμών (παράθυρο). Για παράδειγμα στην ακόλουθη χρονοσειρά, ο κινητός μέσος όρος με παράθυρο 2 είναι **[10,20,30]** → **[15.0,25.0]** Αν η χρονοσειρά μοντελοποιείται με την μορφή λίστας, να υλοποιοήσετε ένα κατηγόρημα **rolling_avg(N, TimeSeq, Aver)**, όπου αν **N** είναι το μέγεθος του παραθύρου, **TimeSeq** η χρονοσειρά τότε η λίστα **Aver** αναπαριστά τον αντίστοιχο κινητό μέσο όρο.

Κώδικας:

```
%%% Checks if the N is bigger than the list and fails if it is so. If it isn't it
%%% searches the average in the whole list
rolling_avg(N, TimeSeq, Average):-
      length(TimeSeq, Len),
      N < Len,
      roll_avg(N, TimeSeq, Average).


%%% Creates a sublist of Numbers with the size of N and finds the average of them.
%%% Then it deletes the first Num and continues to find the averages
roll_avg(N, TimeSeq, [Ans|Aver]):-
      sublist([Num|Tail], TimeSeq),
      length([Num|Tail], N),
      average_num([Num|Tail], Ans),
      delete(Num, TimeSeq, NewSeq),
      roll_avg(N, NewSeq, Aver).

%%% The base case for roll_avg/3, where it returns the average of the last N numbers
roll_avg(N, TimeSeq, [Ans]):-
      length(TimeSeq, N),
      average_num(TimeSeq, Ans).
```

Σχόλια:

Το κατηγόρημα **rolling_avg/3** ελέγχει αρχικά εάν το **N** είναι μικρότερο της λίστας με τους χρόνους και έπειτα, μέσω της **roll_avg/3** ελέγχει τους κινητούς μέσους όρους. Η **roll_avg/3** βρίσκει μία υπολίστα της **TimeSeq** η οποία έχει μέγεθος ίσο με το **N**, έπειτα βρίσκει τον μέσο όρο αυτής, αφαιρεί τον πρώτο της αριθμό από την λίστα **TimeSeq** και συνεχίζει να βρίσκει τους μέσους όρους στην νέα υπολίστα **NewSeq**. Στην βασική της περίπτωση βρίσκει τον μέσο όρο ανάμεσα στα τελευταία **N** στοιχεία.

Παραδείγματα Εκτέλεσης:

```
[eclipse 1]: rolling_avg(2, [10, 20, 30], Aver).

Aver = [15.0, 25.0]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 2]: ?- rolling_avg(2, [10, 20, 30, 40, 50], Aver).

Aver = [15.0, 25.0, 35.0, 45.0]
Yes (0.00s cpu, solution 1, maybe more) ? -
Type ; for more solutions, otherwise <return> ?
[eclipse 3]:
        Use "halt." to exit ECLiPSe.
[eclipse 3]: ?- rolling_avg(3, [10, 20, 30, 40, 50], Aver).

Aver = [20.0, 30.0, 40.0]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 4]: ?- rolling_avg(3, [20, 10, 30, 50, 40, 30], Aver).

Aver = [20.0, 30.0, 40.0, 40.0]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 5]: ?- rolling_avg(1, [10, 20, 30, 40, 50], Aver).

Aver = [10.0, 20.0, 30.0, 40.0, 50.0]
Yes (0.00s cpu, solution 1, maybe more) ?
```

Για την λύση της παρούσας άσκησης, χρησιμοποιήθηκε η sublist/2 και η average_num/2, οι οποίες βοήθησαν στην επίλυση του προβλήματος (το implementation των μπορεί να βρεθεί στο τέλος του εγγράφου).***

4) Paths

Σε ένα κατευθυνόμενο γράφο, οι κόμβοι έχουν διαφορετικά βάρη, όπως φαίνεται στο ακόλουθο σχήμα:

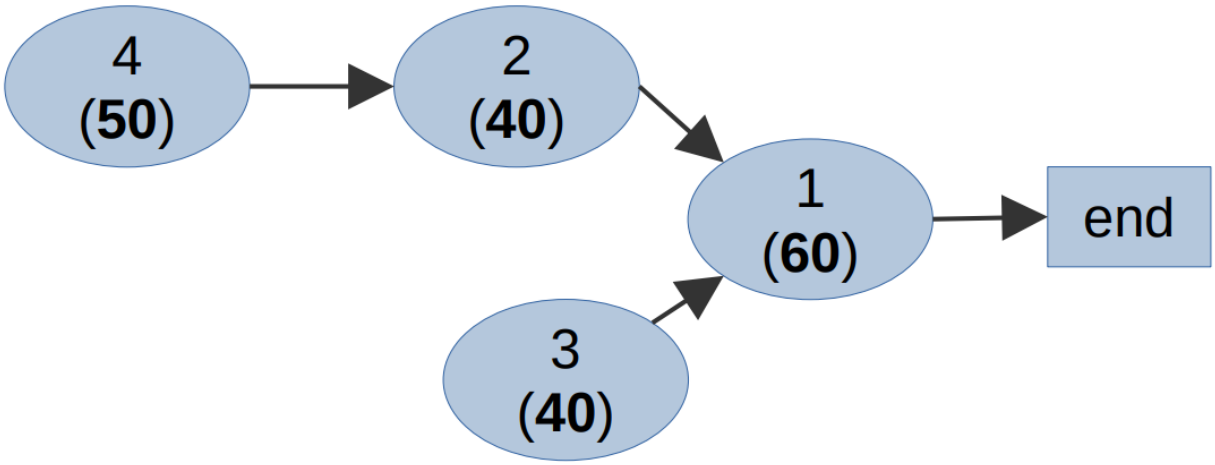

Figure 1: Παράδειγμα γράφου

Ο κόμβος **end** είναι ο ειδικός κόμβος τερματισμού κάθε μονοπατιού (**sink**). Στόχος σας είναι να βρείτε μια ακολουθία από μονοπάτια, η οποία να έχει την μέγιστη συνολική αξία (δες παρακάτω).

Ένα μονοπάτι ξεκινά από κάποιο αρχικό κόμβο, δηλαδή κόμβο στον οποίο δεν δείχνει κάποιος άλλος κόμβος (πχ στο παραπάνω παράδειγμα οι κόμβοι **3** και **4**) και καταλήγει είτε στον κόμβο **end**, είτε σε κάποιον κόμβο τον οποίο ήδη έχει επισκεφτεί κάποιο μονοπάτι, ανάλογα με την σειρά που εξετάζονται τα μονοπάτια. Για παράδειγμα, στο παραπάνω παράδειγμα, υπάρχουν 2 περιπτώσεις:

- Να ξεκινήσει πρώτα το μονοπάτι από τον κόμβο **4**, οπότε σε αυτό θα ανήκουν οι κόμβοι **4→ 2→ 1** (το **end** είναι απλά ο κόμβος τερματισμού, και έπειτα το μονοπάτι από τον κόμβο **3**, **1** (**60**) **end 2** (**40**) **4** (**50**) **3** (**40**) οπότε απλά περιέχει μόνο ένα κόμβο

- Να ξεκινήσει πρώτα το μονοπάτι από τον κόμβο **3**, οπότε σε αυτό θα ανήκουν οι κόμβοι **3→ 1** και έπειτα το μονοπάτι από τον κόμβο **4**, στο οποίο θα ανήκουν οι κόμβοι **4 → 2** (το μονοπάτι σταματά καθώς ο κόμβο **1** ήδη ανήκει στο μονοπάτι του κόμβου **3**.

Οι δύο περιπτώσεις φαίνονται στο ακόλουθο σχήμα:

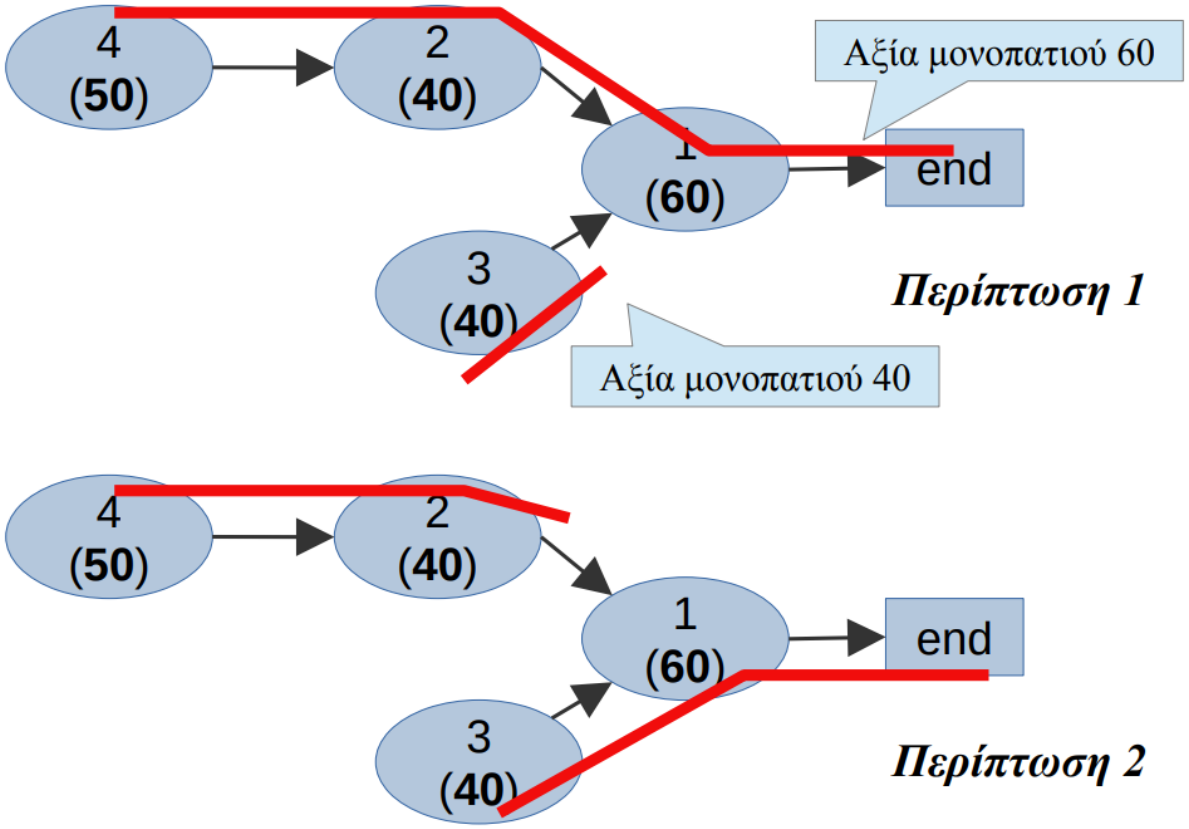

Figure 2: Οι δύο περιπτώσεις μονοπατιών

Κάθε κόμβος έχει ένα βάρος, όπως φαίνεται στο σχήμα **2**. Η αξία ενός μονοπατιού είναι ίση με την μέγιστη αξία ενός κόμβου που ανήκει σε αυτό. Για παράδειγμα το μονοπάτι **4→ 2 → 1** στην περίπτωση **1**, έχει αξία **60**, ενώ στην ίδια περίπτωση το μονοπάτι που ξεκινά από το **3** έχει αξία **40**. Η συνολική αξία της διάταξης **[4,3]** είναι το άθροισμα των αξιών των μονοπατιών, δηλαδή **100**. Στην δεύτερη περίπτωση, η αξία του μονοπατιού **4→2** είναι **50**, ενώ του **3→ 1** είναι **60**, οπότε η διάταξη **[3,4]** έχει αξία 110. Η πληροφορία για γράφους σαν τον παραπάνω κωδικοποιείται σε μια λίστα, όπου κάθε μέλος περιγράφει ένα κόμβο **Node** της είναι της μορφής **node(Node,Value,Next)**, όπου **Node** ο αριθμός του κόμβου που περιγράφεται, **Value** η αξία του, και Next ο αριθμός του επόμενου κόμβου. Για παράδειγμα ο κόμβος **4** αντιστοιχεί στο **node(4,50,2)**. Η αναπαράσταση του παραπάνω γράφου είναι:

[node(1,60,end), node(2,40,1), node(3,40,1), node(4,50,2)]

Δοθέντων τριών γράφων/γεγονότων:

(α) Να υλοποιήσετε ένα κατηγόρημα **graph_start_nodes(Graph,StartNodes)**, το οποίο δεδομένου ενός γράφου **Graph** με την μορφή λίστας όπως φαίνεται παραπάνω, βρίσκει τους αρχικούς κόμβους του γράφου, δηλαδή εκείνους στους οποίους δεν "δείχνει" κανένας άλλος κόμβος.

(β) Να υλοποιήσετε το κατηγόρημα **evaluate_path(Node,Visited,Graph,Values,NewVisited)**, το οποίο ξεκινώντας από ένα κόμβο **Node**, σε ένα γράφο **Graph**, δημιουργεί ένα μονοπάτι μέχρι να φτάσει είτε (α) στον κόμβο end ή (β) σε ένα κόμβο που ανήκει στην λίστα **Visited**. Η λίστα **NewVisited** περιέχει τους κόμβους του νέου μονοπατιού.

(γ) Να υλοποιήσετε το κατηγόρημα **max_seq(Graph,Seq,Value)**, το οποίο πετυχαίνει όταν η λίστα **max_seq(Graph,Seq,Value)** είναι η ακολουθία των αρχικών κόμβων, του γράφου **Graph**, η οποία πρέπει να ακολουθηθεί ώστε να μεγιστοποιείται η συνολική αξία **Value** (όπως περιγράφεται παραπάνω). Αν υπάρχουν πολλές λύσεις με την ίδια αξία επιστρέψτε μια από αυτές.

Κώδικας:

```
%%% Representations of graphs mentioned in the exercise. %%%

example(1,[node(1,60,end), node(2,40,1), node(3,40,1), node(4,50,2)]).
example(2,[node(1,3,end), node(2,2,1), node(3,1,1), node(4,4,1), node(5,5,end)]).
example(3,[node(1,100,end), node(2,100,1), node(3,100,2), node(4, 90, 1), node(5,80,2),
node(6,100,3), node(7, 90, 1), node(8,100,3)]).


%%% (a) %%%

graph_start_nodes(Graph, StartNodes):-
      findall(X, (member(node(X,_ ,_ ), Graph), not(member(node(_,_,X), Graph))), StartNodes).


%%% (b) %%%

%%% Base case for evaluate_path/5, when the next node is the end node return the Value
%%% and the current StartingNode
evaluate_path(StartingNode, Visited, Graph, [Value], [StartingNode]):-
      member(node(StartingNode, Value, end), Graph).

%%% Base case for evaluate_path/5, when the next node is a part of the Visited nodes,
%%% return the Value and the current StartingNode
evaluate_path(StartingNode, Visited, Graph, [Value], [StartingNode]):-
      member(EndNode, Visited),
      member(node(StartingNode, Value, EndNode), Graph).
```

```
%%% Traces the Graph for the Nodes and keeps the Values past, but cannot trace back to
%%% the Visited nodes
evaluate_path(StartingNode, Visited, Graph, [Value|Values], [StartingNode|NewSeq]):-
      member(node(StartingNode, Value, EndNode), Graph),
      not(member(EndNode, Visited)),
      evaluate_path(EndNode, [EndNode|Visited], Graph, Values, NewSeq).


%%% (c) %%%
max_seq(Graph, Seq, Value):-
      graph_start_nodes(Graph, Seq),
      evaluations(Seq, [], Graph, [], Value).

evaluations([StartNode|T], Visited, Graph, MaxVals, Value):-
      evaluate_path(StartingNode, Visited, Graph, Values, Sequence),
      max(Values, Max),
      evaluations(T, Sequence, Graph, [Max|MaxVals], Value).

evaluations([StartNode], Visited, Graph, MaxVals, Value):-
      evaluate_path(StartNode, Visited, Graph, Values, Seq),
      max(Values, Max),
      sum_list([Max|MaxVals], Value).
```

Σχόλια:
Στο (α) ερώτημα, το graph_start_nodes/2 ψάχνει μέσω της findall/3 όλα τα Nodes για τα οποία υπάρχουν μέσα στον γράφο, αλλά δεν δείχνονται από κάποιο άλλο Node. Εν συνεχεία, στο (β) ερώτημα, στις δύο βασικές του περιπτώσεις, εάν ο επόμενος κόμβος είναι είτε το end, είτε ένας κόμβος που υπάρχει στα Visited lists, επιστρέφει σαν μοναδική τιμή στα Values το Value του StartingNode και θέτει σαν βάση στα NewVisited το StartingNode. Στις υπόλοιπες περιπτώσεις, βρίσκουμε το EndNode που δεν έχουμε επισκεφτεί και δείχνει το StartingNode και συνεχίζουμε ψάχνοντας από το EndNode, προσθέτοντας το στους περασμένους κόμβους.

Παραδείγματα Εκτέλεσης:

(α)

```
[eclipse 1]: example(1, Graph), graph_start_nodes(Graph, StartNodes).

Graph = [node(1, 60, end), node(2, 40, 1), node(3, 40, 1), node(4, 50, 2)]
StartNodes = [3, 4]
Yes (0.00s cpu)
[eclipse 2]: example(2, Graph), graph_start_nodes(Graph, StartNodes).

Graph = [node(1, 3, end), node(2, 2, 1), node(3, 1, 1), node(4, 4, 1), node(5, 5, end)]
StartNodes = [2, 3, 4, 5]
Yes (0.00s cpu)
```

(β)

```
[eclipse 1]: ?-example(1,Graph), evaluate_path(3, [], Graph, Values, NewVisited).

Graph = [node(1, 60, end), node(2, 40, 1), node(3, 40, 1), node(4, 50, 2)]
Values = [40, 60]
NewVisited = [3, 1]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 2]: ?-example(1,Graph),evaluate_path(4, [], Graph, Values, NewVisited).

Graph = [node(1, 60, end), node(2, 40, 1), node(3, 40, 1), node(4, 50, 2)]
Values = [50, 40, 60]
NewVisited = [4, 2, 1]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 3]: ?-example(1,Graph),evaluate_path(4, [1], Graph, Values, NewVisited).

Graph = [node(1, 60, end), node(2, 40, 1), node(3, 40, 1), node(4, 50, 2)]
Values = [50, 40]
NewVisited = [4, 2]
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 4]: ?-example(1,Graph),evaluate_path(3,[4,2,1],Graph,Values,NewVisited).

Graph = [node(1, 60, end), node(2, 40, 1), node(3, 40, 1), node(4, 50, 2)]
Values = [40]
NewVisited = [3]
Yes (0.00s cpu, solution 1, maybe more) ?
```

(γ)

```
[eclipse 1]: ?- example(1, Graph), max_seq(Graph, Seq, Value).

Graph = [node(1, 60, end), node(2, 40, 1), node(3, 40, 1), node(4, 50, 2)]
Seq = [3, 4]
Value = 110
Yes (0.00s cpu, solution 1, maybe more) ?
[eclipse 2]: ?- example(2, Graph), max_seq(Graph, Seq, Value).

Graph = [node(1, 3, end), node(2, 2, 1), node(3, 1, 1), node(4, 4, 1), node(5, 5, end)]
Seq = [2, 3, 4, 5]
Value = 14
Yes (0.00s cpu, solution 1, maybe more) ?
```

Για την λύση των evaluation χρησιμοποιήθηκε η sum_list (η οποία έχει παραπλήσιο implementation με την sum/2 της Prolog, απλώς για διασφάληση λειτουργίας του προγράμματος, ο κώδικας παρέμεινε ως έχει.***

***

Βοηθητικά Κατηγορήματα Που Χρησιμοποιήθηκαν

***

```
%%% Succeeds when L1 is a sublist of L2 (L2 contains L1)
sublist(L1,L2):-
      append(Ltemp, _S2, L2),
      append(_S1, L1,Ltemp).


%%% Finds the average of a List
average_num(List, Average):-
      sum_list(List, Sum),
      length(List, N),
      Average is Sum / N.


%%% The base case for sum_list/2, if there's an empty List, the Sum is 0
sum_list([], 0).

%%% Finds the Sum of all the elements of the List
sum_list([H|T], Sum) :-
      sum_list(T, Rest),
      Sum is H + Rest.
```