**Σπυρόπουλος Δημήτριος**
**it1566**
**it1566@uom.edu.gr**

1)List Processing

```
%%%indicate_change/2
indicate_change([], []).
indicate_change([_], []).
indicate_change([X,Y|Rest],[no_c|Change]):-
    X =:= Y,
    indicate_change([Y|Rest],Change).
indicate_change([X,Y|Rest],[up|Change]):-
    X < Y,
    indicate_change([Y|Rest],Change).
indicate_change([X,Y|Rest],[down|Change]):-
    X > Y,
    indicate_change([Y|Rest],Change).
```

1η και 2η περίπτωση: Κενή λίστα και λίστα με 1 στοιχείο επιστρέφεται η κενή λίστα.
2η περίπτωση:Το πρώτο και το δεύτερο στοιχείο της λίστας είναι ίδια οπότε προστίθεται το no_c στη λίστα Change και καλεί τον εαυτό του χωρίς το πρώτο στοιχείο της λίστας ώστε σταδιακά να “αδειάσει” η λίστα και να πάει στην απλή περίπτωση 1.
3η και 4η περίπτωση: Αντίστοιχα με την 2η απλά αντί για να no_c για X>Y περνιέται down και για X<Y up.

```
?- indicate_change([1, 1, 1, 2, 3, 2, 1], C).
C = [no_c, no_c, up, up, down, down]
Yes (0.00s cpu, solution 1, maybe more)
?- indicate_change([1, 2, 1, 1, 3, 1, 1, 5, 4, 3, 6, 6], C).
C = [up, down, no_c, up, down, no_c, up, down, down, up, no_c]
Yes (0.00s cpu, solution 1, maybe more)
```

2)List Processing II

```
18 change([X,Y],up):-
19   X<Y.
20 change([X,Y],down):-
21   X>Y.
22 change([X,Y],no_c):-
23   X=:=Y.
24
25 indicate_change_alt(List,Change):-
26   findall([X,Y], (append(_, [X,Y|_], List)),Result),
27   findall(X,(member(Y,Result),change(Y,X)),Change).
```

Βοηθητικό κατηγόρημα change/2 που ελέγχει αν υπάρχει αλλαγή μεταξύ των στοιχείων μια λίστας με 2 στοιχεία.
Δημιουργία μιας λίστας με όλες τις υπολίστες 2 στοιχείων, της αρχικής λίστας(1ο findall). Έλεγχος αν υπάρχει αλλαγή μεταξύ των στοιχείων κάθε λίστας και επιστροφή των αποτελεσμάτων στη λίστα Change(2ο findall).

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

3)Rolling Average

```
33 sum_list([],0).
34 sum_list([H|T],Sum):-
35    sum_list(T,RestSum),
36    Sum is H + RestSum.
37
38 sublist(L1,L2):-
39     append(S1,_,L2),
40     append(_,L1,S1).
41
42 rolling_avg(N,List,Aver):-
43    length(List,L),
44    N=<L,
45    length(List1,N),
46    findall(List1, (sublist(List1,List)),Result),
47    findall(X,(member(Y,Result),sum_list(Y,Sum),X is Sum/N),Aver).
```

Έλεγχος αν το N είναι μεγαλύτερο του μήκους της λίστας(1ο length). Δημιουργία λίστας μήκους N(2ο length). Εύρεση όλων των υπολιστών μήκους N της αρχικής λίστας(1ο findall). Υπολογισμός των μέσων όρων των υπολιστών(2ο findall).

```
?- rolling_avg(2, [10, 20, 30], Aver).
Aver = [15.0, 25.0]
Yes (0.00s cpu)
?- rolling_avg(2, [10, 20, 30, 40, 50], Aver).
Aver = [15.0, 25.0, 35.0, 45.0]
Yes (0.00s cpu)
?- rolling_avg(1, [10, 20, 30, 40, 50], Aver).
Aver = [10.0, 20.0, 30.0, 40.0, 50.0]
Yes (0.00s cpu)
?- rolling_avg(3, [10, 20], Aver).
No (0.00s cpu)
?- rolling_avg(3, [10, 20, 30, 40, 50], Aver).
Aver = [20.0, 30.0, 40.0]
Yes (0.00s cpu)
```

4)Paths

a)graph_start_nodes/2

```
graph_start_nodes(Graph,StartNodes):-
  findall(X,member(node(X,_,_),Graph),List1),
  setof(X,(A,B,Y,Z)^(member(node(X,Y,Z),Graph),
  member(node(A,B,X),Graph)),List2),
  findall(X,(member(X,List1),not(member(X,List2))),StartNodes).
```

Πρώτα βρίσκουμε όλους τους κόμβους του γράφου σε μια λίστα List1. Στη συνέχεια βρίσκουμε τους κόμβους για τους οποίους υπάρχει κόμβος που να δείχνει σε αυτούς, σε μια λίστα List2.
Τέλος, αφαιρούμε τους κόμβους της List2 από αυτούς της List1, οπότε μένουν μόνο οι κόμβοι στους οποίους δεν δείχνει κάποιος άλλος κόμβος

```
?- 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
StartNodes = [2, 3, 4, 5]
Yes (0.00s cpu)
?- example(3, Graph), graph_start_nodes(Graph, StartNodes).
Graph = [node(1, 100, end), node(2, 100, 1), node(3, 100, 2), node(4, 90, 1), no
StartNodes = [4, 5, 6, 7, 8]
Yes (0.00s cpu)
```

b)evaluate_path/5

```
evaluate_path(Node,Visited,Graph,[Cost1],[Node]):-
  member(node(Node,Cost1,NextNode),Graph),
  member(NextNode,Visited).

evaluate_path(Node,[],Graph,[Cost1,Cost2],[Node,FinalNode]):-
  member(node(Node,Cost1,FinalNode),Graph),
  member(node(FinalNode,Cost2,end),Graph).

evaluate_path(Node,Visited,Graph,[Cost1|Values],[Node|NewVisited]):-
  member(node(Node,Cost1,NextNode),Graph),
  member(node(NextNode,_,_),Graph),
  evaluate_path(NextNode,Visited,Graph,Values,NewVisited).
```

Στην πρώτη περίπτωση ελέγχεται εάν έχουμε ήδη επισκεφτεί τον αρχικό κόμβο ή τον επόμενο κόμβο μέσω άλλου μονοπατιού.
Στην δεύτερη περίπτωση ελέγχεται εάν ο επόμενος κόμβος είναι και τελικός.
Στη γενική περίπτωση βρίσκει τον επόμενο κόμβο και καλεί τον εαυτό του με αρχικό κόμβο τον επόμενο, έχοντας προσθέσει το κόστος στη λίστα Values και τον κόμβο στη λίστα NewVisited.

```
?- example(3, Graph), evaluate_path(3, [], Graph, Values, NewVisited).
Graph = [node(1, 100, end), node(2, 100, 1), node(3, 100, 2), node(4, 90, 1), no
Values = [100, 100, 100]
NewVisited = [3, 2, 1]
Yes (0.00s cpu, solution 1, maybe more)
?- example(3, Graph), evaluate_path(4, [], Graph, Values, NewVisited).
Graph = [node(1, 100, end), node(2, 100, 1), node(3, 100, 2), node(4, 90, 1), no
Values = [90, 100]
NewVisited = [4, 1]
Yes (0.00s cpu, solution 1, maybe more)
?- example(3, Graph), evaluate_path(4, [3, 2, 1], Graph, Values, NewVisited).
Graph = [node(1, 100, end), node(2, 100, 1), node(3, 100, 2), node(4, 90, 1), no
Values = [90]
NewVisited = [4]
Yes (0.00s cpu, solution 1, maybe more)
```