פתרון מבחן

השאלה היא שאלת "פאזל" קלאסית , הרעיון מאחורי הפתרון הוא יצירת תשתית מתאימה למהדר /תיאור מרחב הבעיה ,

תחילה נחליט על ייצוג מתאים למידע , במקרה שלנו רשימת שחקנים כאשר כל שחקן מיוצג באמצעות יחס בן שלוש שדות :

Data = player(Name,From,Year)

השלב הבא הוא תיאור העובדות הידועות לנו כתשתית להתאמה :

basketball\_team(L):-

L=[\_,\_,\_,\_],

member(player(ariel,D7,\_),L),

member(player(jacob,D1,\_),L),

member(player(michael,D4,\_),L),

member(player(adam,D5,\_),L),

member(player(\_,3,\_),L),

member(player(\_,4,\_),L),

member(player(\_,9,\_),L),

member(player(\_,10,\_),L),

member(player(\_,D6,yugoslavian),L),

member(player(\_,D2,american),L),

member(player(\_,\_,israeli),L),

member(player(\_,D3,greek),L),

/\*---------1----------\*/

D2>D1,

/\*---------2----------\*/

D2<D3,

/\*---------3----------\*/

D4 > D5+D6,

/\*---------4----------\*/

D7>D3,!.

ושליפת הפרטים החסרים .

# שאלה 2

get(American\_name,Michael\_year):-

basketball\_team(L),

delete(player(American\_name,\_,american),L,L1),

delete(player(michael,Michael\_year,\_),L1,\_).

/\*

| ?- get(American\_name,Michael\_year).

American\_name = adam ,

Michael\_year = 9

\*/

בשאלה מסוג זה, אתם נדרשים להפגין שליטה בשפת פרולוג באופן עבודתה ובמרכיביה .

המפתח בשאלות אלו הוא שליטה בכל מרכיבי השפה (זכרו כי לרשותכם עומד גם ספר הלימוד ), למשל בשאלה זאת המפתח להבנה הוא הבנת השורות :

assert(tmp(Term)), retract(tmp(Term1)), Term1==Term

f([X|Xs],[X|Ys]-Tail):-

g(X),!,

f(Xs, Ys-Tail).

f([X|Xs],Ys-Tail):-

f(Xs, Ys-[X|Tail]).

f([],T-T).

g(Term):-

assert(tmp(Term)),

retract(tmp(Term1)),

Term1==Term.

ותוצאות ההרצה :

/\*

| ?- f([X,3, ab, R, 4, DD], L-[]).

X = \_ ,

R = \_ ,

DD = \_ ,

L = [3,ab,4,DD,R,X]

| ?- f([X, g(1,2), ab, g(X,f(1)), 23, f(X,Y)], L-[]).

X = \_ ,

Y = \_ ,

L = [g(1,2),ab,23,f(X,Y),g(X,f(1)),X]\*/

\*/

# שאלה 3

הבסיס למימוש של quickselect הוא הפרדיקט split (List, X, L1, L2). הוא מפצל רשימה List לשתי רשימות כך שכל האיברים ב L1 קטנים מ X או שווים לו, וכל האיברים ב L2 גדולים מ X.

הפרדיקט הראשי quickselect, מפרק את הרשימה סביב האיבר הראשון לשתי רשימות

L1

L2

X

List

X

וממשיך בהתאם לאלגוריתם :

אם |L1|>k: המשך רקורסיבית לחפש את האיבר ה-kי בגודלו בתוך L1.

אם |L1|<k: קבע k1= k-|L1|. המשך רקורסיבית לחפש את האיבר ה-k1י בגודלו בתוך L2.

אם |L1|=k החזר את X.

קובץ קוד מקור: quickselect.pl

quick\_select([X|Xs], K, Res):-

split(X,Xs,Small,Big),

length(Small,L1),

length(Big,L2),

(L1+1=:=K,!,

Res=X

;

L1>=K,!,

quick\_select(Small, K, Res)

;

K1 is K-L1-1,

quick\_select(Big, K1, Res)).

# שאלה 4

split(X, [], [] ,[]).

split(X, [Y|Tail], [Y|Small], Big):-

X>Y,!,

split(X, Tail, Small, Big).

split(X, [Y|Tail], Small, [Y|Big]):-

split(X, Tail, Small, Big).

בשאלה זאת כדאי לשם לב כי אין דרישה ממשית להימנע מפתרונות כפולים , אפילו בדוגמת הריצה יש פתרונות כפולים , לכן הפתרון הבא קביל בהחלט :

בצורה הרגילה :

וברשימות הפרש

%%%%%%%%%%%%%%%%

% Implemenation with different list

%%%%%%%%%%%%%%%%

mis\_root\_dl(nil, T-T, 0).

mis\_root\_dl(t(L,X,R), [X|List1]-T, N):-

mis\_noroot\_dl(L, List1-List2, N1),

mis\_noroot\_dl(R, List2-T, N2),

N is N1+N2+1.

mis\_noroot\_dl(nil, T-T, 0).

mis\_noroot\_dl(t(L,X,R), List1-T, N):-

mis\_dl(L, List1-List2, N1),

mis\_dl(R, List2-T, N2),

N is N1+N2.

mis\_dl(Tree, List-T ,N):-

mis\_noroot\_dl(Tree, ListA-TA, NA),

mis\_root\_dl(Tree, ListB-TB, NB),

(NA>=NB, List=ListA, T=TA, N=NA ;

NB>=NA, List=ListB, T=TB, N=NB).

mis\_root(nil, [], 0).

mis\_root(t(L,X,R), [X|List], N):-

mis\_noroot(L, List1, N1),

mis\_noroot(R, List2, N2),

append(List1, List2, List),

N is N1+N2+1.

mis\_noroot(nil, [], 0).

mis\_noroot(t(L,X,R), List, N):-

mis(L, List1, N1),

mis(R, List2, N2),

append(List1, List2, List),

N is N1+N2.

mis(Tree, List ,N):-

mis\_noroot(Tree, ListA, NA),

mis\_root(Tree, ListB, NB),

(NA>=NB, List=ListA, N=NA ;

NB>=NA, List=ListB, N=NB).

ואם נוסיף cut – ים כדי להימנע מכפילויות נקבל למעשה רק חלק מהפתרונות :

mis\_root(nil, [], 0).

mis\_root(t(L,X,R), [X|List], N):-

mis\_noroot(L, List1, N1),

mis\_noroot(R, List2, N2),

append(List1, List2, List),

N is N1+N2+1.

mis\_noroot(nil, [], 0).

mis\_noroot(t(L,X,R), List, N):-

mis(L, List1, N1),

mis(R, List2, N2),

append(List1, List2, List),

N is N1+N2.

mis(Tree, List ,N):-

mis\_noroot(Tree, ListA, NA),

mis\_root(Tree, ListB, NB),!,

(NA>=NB, List=ListA, N=NA ;

NB>=NA, List=ListB, N=NB).

build\_tree(Tree):-

Tree=t(t(t(t(nil,6,t(nil,7,nil)),3,nil),2,t(nil,4,t(nil,5,nil))),1,t(t(t(nil,10,nil),9,t(t(nil,12,nil),11,nil)),8,nil)).

/\*

| ?- build\_tree(T), mis(T, List, N).

T = t(t(t(t(nil,6,t(nil,7,nil)),3,nil),2,t(nil,4,t(nil,5,nil))),1,t(t(t(nil,10,nil),9,t(t(nil,12,nil),11,nil)),8,nil)) ,

List = [3,7,5,8,10,12] ,

N = 6 ;

T = t(t(t(t(nil,6,t(nil,7,nil)),3,nil),2,t(nil,4,t(nil,5,nil))),1,t(t(t(nil,10,nil),9,t(t(nil,12,nil),11,nil)),8,nil)) ,

List = [1,3,7,5,10,12] ,

N = 6

| ?-

\*/

נוכל למנוע כפילויות בעזרת שימוש בזיכרון :

not\_in\_memory(L):-

(retract(mem(L)),!,assert(mem(L)),fail

;

true).

mis\_root(nil, [], 0).

mis\_root(t(L,X,R), [X|List], N):-

mis\_noroot(L, List1, N1),

mis\_noroot(R, List2, N2),

append(List1, List2, List),

N is N1+N2+1.

mis\_noroot(nil, [], 0).

mis\_noroot(t(L,X,R), List, N):-

mis(L, List1, N1),

mis(R, List2, N2),

append(List1, List2, List),

N is N1+N2.

mis(nil, [], 0).

mis(t(L,X,R), List ,N):-

mis\_noroot(t(L,X,R),ListA, NA),

mis\_root(t(L,X,R), ListB, NB),

(NA>=NB, List=ListA, N=NA ; NB>=NA,List=ListB, N=NB).

up(T,List,N):-

retractall(mem(\_)),!,

mis(T,List,N),not\_in\_memory(List),assert(mem(List)).

build\_tree(Tree):-

Tree=t(t(t(t(nil,6,t(nil,7,nil)),3,nil),2,t(nil,4,t(nil,5,nil))),1,t(t(t(nil,10,nil),9,t(t(nil,12,nil),11,nil)),8,nil)).