

Recursivitatea pe stivă vs Recursivitatea pe coadă

Vom porni de la implementarea predicatului `len/2`, care calculează recursiv lungimea unei liste primite ca input.

Create a

Program

Query

Markdown

HTML



cell here

```

1 len([],0).
2 len([_|T],N) :- len(T,M), N is M + 1.

```

≡ ?- trace, len([1,2,4],P).

Programul de mai sus are o complexitate de $O(n)$. Această implementare se numește **Recursivitate pe stivă**, deoarece folosește stiva de memorie pentru a reține rezultatele fiecărui apel.

Astfel, la fiecare **apel**, spațiu locupat pe stivă **crește**, iar la fiecare **revenire** dintr-un apel recursiv, spațiul pe stivă **scade**.

La recursivitate foarte adâncă, se va obține o eroare de tip `stack overflow`.

O modalitate mai eficientă a programului de mai sus poate fi obținută folosind un acumulator, ca parametru suplimentar al funcției.

Acumulatori în Prolog



Variabila Colectoare (Acumulator) este o metodă de a transmite eficient rezultate parțiale în cadrul calculelor Prolog.

```

1 % len_acc(L,A,N) succeeds if
2 %
3 % input A is an accumulated length so far
4 % output N is the total length of the original list.
5 len_acc([],A,A).
6 len_acc([_|T],A, N) :- A1 is A + 1, len_acc(T,A1,N).
7 %Vom inițializa variabila colectoare:
8 len2(L,N) :- len_acc(L,0,N).

```

≡ ?- trace, len2([1,2,3],P).

În această implementare, predicatul `len/3` este unul **recursiv pe coadă**.

Complexitatea programului devine $O(1)$.

Recursivitatea pe coadă se obține atunci când valoarea întoarsă de ultimul apel recursiv este valoarea de retur a programului, nefiind necesară reținerea informației pe stiva de memorie.

Recursivitatea pe coadă

1. Apel recursiv => pe stivă
2. Complexitate spațială $O(n)$
3. Scriere mai simplă

Recursivitatea pe coadă

1. Rezultatul apelului recursiv => întors direct
 2. Complexitate spațială $O(1)$
 3. Scriere mai complexă => Acumulator
- Uneori, rolul acumulatorului poate fi preluat de unul dintre parametri, caz în care nu este nevoie nici de funcția suplimentară în care se inițializează acumulatorul.

Prolog în Inteligența Artificială

Se pot diferenția două abordări ale Inteligenței Artificiale:

1. Abordarea Simbolică
2. Abordarea Statistică

Abordarea simbolică:

- este în mod obișnuit potrivită pentru **verificare formală**.
- Regulile sunt formulate în mod explicit și se poate verifica și interpreta cu ușurință dacă sunt complete sau ce exprimă.
- Folosind programarea logică inductivă (ILP), un program poate să învețe reguli bazate pe exemple pozitive și negative.
- Aplicații:
 - sisteme expert bazate pe reguli
 - demonstratori de teoreme
 - forme inteligente de căutare
 - abordări bazate pe satisfacerea unor constrângeri
 - etc.

În **Prolog** Inteligența Artificială Simbolică poate fi ilustrată prin:

- **Knowledge Representation(Reprezentarea Cunoștințelor)** - Prolog permite reprezentarea cunoștințelor folosind fapte și reguli, într-o manieră declarativă. Putem defini predicate care reprezintă diferite relații sau fapte entitățile programului:

```
1 femeie(ana).
```



```

2 femeie(maria).
3 barbat(ion).
4
5 mama(ana,maria).
6 mama(ana,ion).

```

- **Inferență:** Prolog folosește inferență logică pentru a deduce informații din baza sa de cunoștințe.

```
1 mama(X,Y).
```

- **Raționament deductiv** : pe baza regulilor și faptelor definite în baza de cunoștințe, în Prolog se pot crea cu ușurință reguli și relații noi despre entitățile implicate.

```
1 părinte(X,Y):-mama(X,Y).
```

- **Calcul simbolic:** Prolog poate utiliza concepte abstracte:

- Expresii aritmetice:

```
add(X, Y, Result) :- Result is X + Y.
```

```
subtract(X, Y, Result) :- Result is X - Y.
```

```
multiply(X, Y, Result) :- Result is X * Y.
```

```
divide(X, Y, Result) :- Y \= 0, Result is X / Y.
```

- Manipularea listelor:

```
reverse([], []).
```

```
reverse([H|T], R) :- reverse(T, RT), append(RT, [H], R).
```

- Utilizarea string-urilor
- Expresii simbolice : integrale, derivate, rezolvări de ecuații, Expresii logice.
- Expresii regulate - prin folosirea predicatului `re/2`:

- **Sisteme Expert** Prolog este adesea folosit pentru a construi sisteme expert care pot da sfaturi sau lua decizii experte în diverse domenii, cum ar fi diagnosticarea medicală sau procesarea limbajului natural.

Puzzle logice in Prolog

Ne propunem să rezolvăm problema lui Einstein în Prolog.

1. Englezul locuiește într-o casă roșie.
2. Suedezul are câini ca animale de companie.
3. Danezul bea ceai.
4. Casa verde este învecinată cu casa albă, situată la stânga acesteia.
5. Proprietarul casei verzi bea cafea.
6. Persoana care fumează Pall Mall crește păsări.
7. Proprietarul casei galbene fumează Dunhill.

8. Bărbatul care locuiește în casa din mijloc bea lapte.
9. Norvegianul locuiește în prima casă.
10. Bărbatul care fumează Blends locuiește lângă cel care are pisici.
11. Bărbatul care are cai locuiește lângă cel care fumează Dunhill.
12. Bărbatul care fumează Blue Master bea bere.
13. Germanul fumează Prince.
14. Norvegianul locuiește lângă casa albastră.
15. Bărbatul care fumează Blends are un vecin care bea apă. Întrebarea este: Cine deține peștele?

```

1 /*
2  The Brit lives in a red house.
3  The Swede keeps dogs as pets.
4  The Dane drinks tea.
5  The Green house is next to, and on the left of the White house.
6  The owner of the Green house drinks coffee.
7  The person who smokes Pall Mall rears birds.
8  The owner of the Yellow house smokes Dunhill.
9  The man living in the centre house drinks milk.
10 The Norwegian lives in the first house.
11 The man who smokes Blends lives next to the one who keeps cats.
12 The man who keeps horses lives next to the man who smokes Dunhill.
13 The man who smokes Blue Master drinks beer.
14 The German smokes Prince.
15 The Norwegian lives next to the blue house.
16 The man who smokes Blends has a neighbour who drinks water.
17 */
18
19 persons(0, []) :- !. % end when the index is 0 and the list is empty.
20 persons(N, [(_Men,_Color,_Drink,_Smoke,_Animal)|T]) :- N1 is N-1, persons(N1,
21
22 % get the Nth element if corresponding to some informations of the recursive
23 person(1, [H|_], H) :- !.
24 person(N, [_|T], R) :- N1 is N-1, person(N1, T, R).
25
26 % The Brit lives in a red house
27 hint1([(brit,red,_,_,_)|_]).
28

```

Procesarea Limbajului Natural în Prolog - Analizator sintactic

Prolog a fost creat pentru a facilita procesarea limbajului natural, prin urmare se pot implementa ușor gramatici pentru a parsă, genera, completa și verifica secvențe de text ca fiind liste.

Operatorul `-->` indica o regula dintr-o gramatică clauzal definită (DCG) și înlocuiește operatorul `:-`. Preprocesorul va adăuga doi parametri în plus: lista care se procesează și lista rezultat. De exemplu clauza:

```
sentence(S, Lrez):- np(S, Lrez1), vp(Lrez1, Lrez) .
```

se poate transforma într-o regulă DCG după cum urmează: `sentence --> np, vp .`

Fiecare element din stanga unei reguli este considerat goal si la el preprocesorul va adauga de asemenea cele doua liste ca paramentrii

- Un parser se defineste in raport cu o gramatica sau un alt mecanism de analiza a constructiilor unui limbaj natural. Rolul acestuia este de a identifica acele reguli (din gramatica) care stau la baza structurii propozitiei ce se analizeaza. Actiunea lui se concretizeaza intr-un proces de cautare a acelor structuri sintactice sintactice posibile care se potrivec cu structura propozitiei.
- Exista mai multe tipuri de cautare, cele mai uzuale fiind depth-first si breadth first. Cautarea depth-first (cautare in adancime cu revenire prin backtracking) este implementata direct in Prolog.
- Rezultatul cautarilor lansate de parser confirma sau infirma corectitudinea sintactica a propozitiei analizate in raport cu gramatica utilizata
- Traseul unui algoritm de analiza sintactica (sau al unui parser) este de cele mai multe ori reprezentat sub forma unui arbore – **parse tree**.

```

1 % Grammar rules
2 sentence --> subject, predicate.
3 subject --> subject_pronoun.
4 subject --> noun.
5 predicate --> verb.
6 predicate --> verb, noun.
7
8 % Key words for each grammatical category
9 subject_pronoun --> [he].
10 subject_pronoun --> [she].
11 noun --> [dog].
12 noun --> [cat].
13 noun --> [child].
14 verb --> [runs].
15 verb --> [eats].
16 verb --> [plays].
17
18 % Sentence analysis predicate
19 analyze_sentence(Sentence) :-
20     phrase(sentence, Sentence).
```

```
?- analyze_sentence([he, runs, dog]).
```

phrase/2 este un predicat built-in Prolog folosit pentru a aplica o regulă DCG unei liste de simboluri sau elemente, fiind o parte fundamentală a analizei gramaticale și a procesării textului în Prolog.

Predicatul phrase/2 primește două argumente:

1. Primul argument este regula DCG pe care dorim să o aplicăm listei de simboluri.
 2. Al doilea argument este lista de simboluri sau elemente căreia regula DCG ar trebui să fie aplicată. Scopul predicatului phrase/2 este de a verifica dacă lista de simboluri respectă regule de gramatică specificate de DCG.
- Dacă lista de simboluri se potrivește gramaticii, predicatul phrase/2 reușește și returnează adevărat.

- Dacă nu există nicio potrivire, predicatul `phrase/2` (/pdoc/man?predicate=phrase/2) eșuează și returnează fals.

Sisteme Expert în Prolog

```

1 % Cunoștințe despre boli și simptome
2 simptom(feбра, gripa).
3 simptom(tuse, gripa).
4 simptom(nas_infundat, gripa).
5 simptom(feбра, raceala).
6 simptom(durere_gat, raceala).
7 simptom(nas_infundat, raceala).
8 boala(gripa) :- simptom(feбра, gripa), simptom(tuse, gripa), simptom(nas_infu
9 boala(raceala) :- simptom(feбра, raceala), simptom(durere_gat, raceala), simp
10
11 % Reguli pentru diagnosticul medical
12 diagnostic(Pacient, Boala) :-
13     are_simptom(Pacient, S),!,
14     %not(diagnostic(Pacient, _)), % Pentru a evita mai multe diagnoze pentru
15     simptom(S,Boala),
16     write('Sistemul expert a diagnosticat pacientul cu: '),
17     write(Boala),
18     nl,
19     recomandare(Boala),
20     nl.
21
22 % Recomandări medicale
23 recomandare(gripa) :-
24     write('Recomandare: Odihniți-vă, luați medicamente pentru febră și tuse,
25 recomandare(raceala) :-
26     write('Recomandare: Odihniți-vă, luați medicamente pentru febră și durere
27
28

```

```

≡ ?- trace,
    diagnostic(john, Boala).

```

```

≡ ?- diagnostic(susan, Boala).

```

```

1 Your Prolog rules and facts go here ...

```

Abordarea statistică:

- poate **genera reguli implicit**, pe baza seturilor de date de antrenare, testare și validare. Acest lucru este util, nemaifiind necesar ca regulile să fie gândite apriori de om, sistemul/algorithmul fiind capabil de a deduce reguli dificil de encodat pentru om.
- Pe de altă parte, acesta poate fi și un dezavantaj:
 - devine greu de urmărit explicit raționamentul algoritmului
 - devine greu de corectat valorile obținute de program: De exemplu, poate clasifica greșit o imagine cu un penguin drept o mașină dacă se modifică valoarea unui singur pixel.
- Aplicații:
 - Rețele Neuronale
 - Instruire automată (Machine Learning)
 - data minning
 - etc.

Instruire automată în Prolog - Regresia liniară

Regresia Liniară - Prezicerea unor date liniar dependente

este o tehnică de analiză statistică folosită pentru a investiga și modela relațiile dintre două sau mai multe variabile. Scopul principal al regresiei liniare este să înțeleagă și să cuantifice modul în care o variabilă (numită variabilă dependentă) este influențată de una sau mai multe alte variabile (numite variabile independente). Aceasta se realizează prin ajustarea unei linii drepte sau unei suprafețe liniare la datele observate pentru a face predicții sau a evalua relația dintre variabile.

Date de Exemplu:

1. date inițiale `data(X,Y)` Vom defini un set de date (PUNCTE) sub forma unor perechi coordonate (X, Y).
 - `x`= variabila independentă
 - `y`- variabila dependentă
2. Calculul Mediei: Vom defini predicatul `mean/2` care calculează media unei liste de valori.
3. Calculul Sumelor Patratelor: Predicatul `sum_of_squares/3` calculează suma pătratelor abaterilor valorilor față de la media lor. Acest lucru este folosit pentru calculul coeficientului.
4. Calculul Coeficienților: Predicatul `linear_regression/2` efectuează principalele calculele regresiei liniare.
 - Acesta calculează coeficientul (a) și interceptul (b) ale modelului de regresie liniară.
 - Acesta salvează toate valorile X și Y din datele de exemplu folosind `findall/3`.
 - Calculează media pentru X și Y folosind predicatul `mean/2`.
 - Calculează covarianța dintre X și Y folosind predicatul `covariance/5`. Acesta este folosit pentru calculul coeficientului (a).
 - Calculează suma pătratelor deviațiilor X față de media lor, care este folosită în calculul coeficientului.
 - În final, calculează coeficientul (a) și interceptul (b) și le returnează.
5. Calculul Covarianței: Predicatul `covariance/5` calculează covarianța dintre două liste de valori (X și Y) cu mediile lor corespunzătoare.

```
1 % Sample data: (x, y) pairs
2 data(1, 2).
3 data(2, 3).
```

```

4 data(3, 4).
5 data(4, 5).
6 data(5, 6).
7
8 % Calculate the mean of a list of numbers
9 mean([], 0).
10 mean(List, Mean) :-
11     sum_list(List, Sum),
12     length(List, Count),
13     Mean is Sum / Count.
14
15 % Calculate the sum of squares
16 sum_of_squares([], _, 0).
17 sum_of_squares([X | Xs], Mean, Sum) :-
18     sum_of_squares(Xs, Mean, Rest),
19     Sum is (X - Mean) ** 2 + Rest.
20
21 % Calculate the covariance between X and Y
22 covariance([], [], _, _, 0).
23 covariance([X | Xs], [Y | Ys], XMean, YMean, Cov) :-
24     covariance(Xs, Ys, XMean, YMean, Rest),
25     Cov is (X - XMean) * (Y - YMean) + Rest.
26
27 % Calculate the coefficients of the linear regression
28 linear_regression(Coefficient, Intercept) :-

```

```

≡ ?- linear_regression(Coefficient, Intercept),
      format('Linear Regression: y = ~2f x + ~2f~n', [Coefficient, Intercept]).

```

```

1 % Y is sin(X) for X in 0..Max
2 lregres(Nr_pct, X, Y) :-
3     between(1, Nr_pct, X),
4     Y is X*1+1.

```

```

≡ ?- r_data_frame(df, [x=X,y=Y], lregres(5,X, Y)),
      <- library("ggplot2"),
      <- ggplot(data=df, aes(x=x, y=y)) + geom_line()+ geom_point().

```

```

1 Your Prolog rules and facts go here ...

```

Rețele neuronale adânci în Prolog

Cercetări recente (<https://arxiv.org/pdf/1805.10872.pdf> (<https://arxiv.org/pdf/1805.10872.pdf>) sau video - <https://www.youtube.com/watch?v=-b9Ze7yXuo>) (<https://www.youtube.com/watch?v=-b9Ze7yXuo>) conduc către folosire Prolog în implementarea unor structuri de învățare automată profundă. O explicație detaliată poate fi urmărită <https://www.youtube.com/watch?v=b-AC428qhdQ> (<https://www.youtube.com/watch?v=b-AC428qhdQ>)

