CT332: TRÍ TUỆ NHÂN TẠO
BÀI THỰC HÀNH SỐ 5
Lập trình đệ quy và Tìm kiếm sâu rộng

Trong buổi này, trước hết sinh viên sẽ thực hành phương pháp tư duy lập trình theo kiểu đệ quy để giải quyết bài toán. Mọi thứ đều quy về đệ quy vì Prolog không hỗ trợ vòng lặp như các ngôn ngữ lập trình cấu trúc.

Phần tiếp theo sẽ thực hành về các thao tác trên danh sách và dùng danh sách để biểu diễn dữ liệu.

Phần 1: Lập trình đệ quy và danh sách

1. Lập trình đệ quy

1. Định nghĩa vị từ *luythua* cho phép tính a lũy thừa n. Ví dụ: **luythua(2, 3, X)** sẽ cho kết quả X = 8.
2. Định nghĩa vị từ tính số fibonacy thứ N.
3. Định nghĩa vị từ tính tổ hợp chập k của n phần tử.

2. Thao tác trên danh sách

1. Định nghĩa vị từ thempt(L1, X, L2), thực hiện thêm phần tử X vào cuối danh sách L1 tạo thành danh sách L2.
2. Định nghĩa vị từ noids(L1, L2, L3) nối hai danh sách L1 và L2 tạo thành danh sách L3.
3. Định nghĩa vị từ lấy các phần tử ở vị trí lẻ của 1 danh sách. Ví dụ ptvtle([2,4,3,1], L), L = [2, 3].
4. Định nghĩa vị từ tính tổng các phần tử lẻ trong danh sách.
5. Định nghĩa vị từ lấy các phần tử nhỏ hơn một giá trị N nào đó trong một danh sách.

3. Sử dụng danh sách để biểu diễn cấu trúc dữ liệu

Với danh sách ta có thể biểu diễn bất cứ kiểu dữ liệu nào. Ví dụ để biểu diễn 1 sinh viên có tên là Nam có tuổi là 25, ta có thể dùng danh sách [‘Nam’, 25]. Điều biểu diễn danh sách các sinh viên (gồm tên và tuổi) biểu diễn như sau: [[‘Nam’, 25], [‘Bắc’, 19], [‘Trung’, 21]].

Giả sử L là một danh sách các sinh viên, L = [[‘Nam’, 25], [‘Bắc’, 19], [‘Trung’, 21]]. Ta có thể viết vị từ **tuoi(L, Ten, Tuoi)** trả về tuổi của sinh viên có tên là **Ten** trong danh sách **L**.

- Trường hợp thứ nhất: phần tử đầu của danh sách có dạng [A, B], và A = Ten, ta gán Tuoi = B và dừng lại không tìm tiếp nữa bằng vị từ cắt !

tuoi([[A, B] | _], Ten, Tuoi) :- Ten = A, Tuoi = B, !.

- Trường hợp thứ hai, phần đầu của danh sách có dạng [A, B] và A \= B, ta bỏ qua phần tử này và tìm trong phần còn lại của danh sách.
  tuoi([[A, B] | T], Ten, Tuoi) :- Ten \= A, tuoi(T, Ten, Tuoi).

Tóm lại vị từ **tuoi(L, Ten, Tuoi)** được định nghĩa trong GNU Prolog như sau:

```
tuoi([[A, B] | _ ], Ten, Tuoi) :- Ten = A, Tuoi = B, !.
tuoi([[A, B] | T ], Ten, Tuoi) :- Ten \= A, tuoi(T, Ten,
Tuoi).
```

Nạp chương trình và truy vấn bằng các câu hỏi sau, ghi nhận kết quả cho từng câu hỏi.

L = [['Nam', 25], ['Bac', 19], ['Trung', 21]], tuoi(L, 'Nam', Tuoi).

L = [['Nam', 25], ['Bac', 19], ['Trung', 21]], tuoi(L, 'Bac', Tuoi).

L = [['Nam', 25], ['Bac', 19], ['Trung', 21]], tuoi(L, 'Tuan', Tuoi).

Hãy cải tiến chương trình trên để trong trường hợp tìm không thấy tên trong danh sách, ta có Tuổi = -1. *Gợi ý:* xét thêm trường hợp danh sách rỗng.

Phần 2: Tìm kiếm sâu rộng

1. Tìm kiếm theo chiều rộng

Khung chương trình để giải quyết vấn đề bằng phương pháp tìm kiếm theo chiều rộng như sau:

```
/*định nghĩa các phép toán thông qua vị từ successor*/
successor(X,S) :- …
…
/*Vị từ succ gọi vị từ successor sinh ra trạng thái mới S từ trạng thái
X và thêm S vào đường đi hiện hành*/
Succ1([X|Path], [S,X|Path]) :- successor(X,S), \+member(S, Path).

/*Giải thuật tìm kiếm theo chiều rộng*/
bfs(B, [[B|Path]|_], [B|Path]).
bfs(B, [[X|Path]|Y], Solution) :- B \=
      X, findall(S, succ1([X|Path], S),
      L), append(Y, L, Open), bfs(B, Open
       , Solution).

/*Giải quyết vấn đề bằng cách tìm kiếm theo chiều rộng*/
find(A, B, P) :- bfs(B, [[A]], Q), reverse(Q, P).
```

Vị từ **bfs** có 3 tham số:

B: Trạng thái đích

Open: danh sách các trạng thái chưa duyệt

Solution: lời giải

B là trạng thái đích được biểu diễn bằng một cấu trúc nào đó. Tùy theo bài toán mà B có thể là 1 phần tử hoặc một danh sách hoặc bất kỳ một cấu trúc phức tạp nào.

Để có thể truy vết tìm đường đi từ trạng thái bắt đầu đến trạng thái kết thúc. Với mỗi trạng thái, ta sẽ lưu *toàn bộ đường đi từ trạng thái bắt đầu đến nó* bằng một danh sách (thứ tự ngược, trạng thái đang xét sẽ nằm đầu danh sách).

Như vậy lịch biểu Open là một hàng đợi được biểu diễn bằng danh sách các trạng

thái chưa duyệt. Mỗi trạng thái được biểu bằng một danh sách lưu đường đi từ trạng thái bắt đầu đến nó như được trình bày ở trên. Vậy Open là *danh sách của các danh sách.*

Solution là một lời giải = đường đi từ trạng thái bắt đầu đến trạng thái kết thúc được biểu diễn giống như các phần tử trong danh sách Open.

succ([X|Path], S) là một vị từ mô tả các phép toán để sinh ra trạng thái kế tiếp từ trạng thái X. Sẽ có dạng [Y, X | Path] với Y là trạng thái sinh ra từ

X. Xét vị từ:

succ([X|Path], [S,X|Path]) :- successor(X,S), \+member(S, Path).

Phép kiểm tra **\+member(S, Path)** được thêm vào để đảm bảo rằng trạng thái mới S chưa có trên đường đi Path. Điều này giúp ta tránh đi lòng vòng theo một chu trình mà không đến đích.

Vị từ **findall** là một vị từ định nghĩa sẵn trong Prolog cho phép trả về một danh sách các trạng thái S có thể có từ trạng thái [X|Path].

Giờ ta xét đến định nghĩa của vị từ bfs.

Trong trường hợp đầu tiên, phần tử đầu của danh sách Open là phần tử [B|Path] trong đó B là trạng thái đích thì solution kết quả sẽ chính là [B|Path].

Trường hợp thứ hai, phần tử đầu là [X|Path] (B \= X, X chưa phải là trạng thái kết thúc), danh sách còn lại là Y. Ta sẽ tìm các trạng thái mới sinh ra từ X, và thêm nó vào cuối danh sách Y (Open là hàng đợi) và gọi đệ quy tìm lời giải với Open mới.

Vị từ find được thêm vào để dễ sử dụng. tìm đường đi từ A đến B và lưu đường đi tìm được vào P. Ta sẽ đưa A vào Open và gọi bfs với trạng thái đích là B. bfs trả về một đường đi từ A đến B nhưng theo thứ tự ngược. Để có được thứ tự đúng ta phải đảo ngược danh sách bằng vị từ *reverse*.

Với khung giải thuật này ta có thể giải bất cứ bài toán tìm kiếm nào. Để sử dụng ta chỉ cần cung cấp một cách biểu diễn trạng thái và định nghĩa phép toán *successor*.

Ví dụ: Áp dụng khung lời giải trên vào bài toán 6 cái ly.

```
/*O1*/
successor([X1, X2, X3, X4, X5, X6], [Y1, Y2, Y3, X4, X5, X6]) :-
          Y1 is 1- X1, Y2 is 1 - X2, Y3 is 1 - X3.


/*O2*/
successor([X1, X2, X3, X4, X5, X6], [X1, Y2, Y3, Y4, X5, X6]) :-
          Y2 is 1- X2, Y3 is 1 - X3, Y4 is 1 - X4.


/*O3*/
successor([X1, X2, X3, X4, X5, X6], [X1, X2, Y3, Y4, Y5, X6]) :-
          Y3 is 1- X3, Y4 is 1 - X4, Y5 is 1 - X5.


/*O4*/
successor([X1, X2, X3, X4, X5, X6], [X1, X2, X3, Y4, Y5, Y6]) :-
          Y4 is 1- X4, Y5 is 1 - X5, Y6 is 1 - X6.

succ1([X|Path], [S,X|Path]) :- successor(X,S), \+member(S, Path).


bfs(B, [[B|Path]|_], [B|Path]).
bfs(B, [[X|Path]|Y], Solution) :- B \=
     X, findall(S, succ1([X|Path], S),
     L), append(Y, L, Open),
     bfs(B, Open , Solution).

find(A, B, P) :- bfs(B, [[A]], Q), reverse(Q,
P).
```

2. Tìm kiếm theo chiều sâu

Tương tự tìm kiếm theo chiều rộng, khung chương trình để giải quyết vấn đề bằng phương pháp tìm kiếm theo chiều sâu như sau:

```
/*định nghĩa các phép toán thông qua vị từ successor*/
successor(X,S) :- …
…
/*Vị từ succ gọi vị từ successor sinh ra trạng thái mới S từ trạng
thái X và thêm S vào đường đi hiện hành*/ succ1([X|Path],
[S,X|Path]) :- successor(X,S), \+member(S, Path).

/*Giải thuật tìm kiếm theo chiều sâu*/
dfs(B, [[B|Path]|_], [B|Path]).
dfs(B, [[X|Path]|Y], Solution) :- B \=
      X, findall(S, succ1([X|Path], S),
      L), append(L, Y, Open), bfs(B, Open
      , Solution).

/*Giải quyết vấn đề bằng cách tìm kiếm theo chiều sâu*/
find(A, B, P) :- dfs(B, [[A]], Q), reverse(Q, P).
```

Ta chỉ cần thay đổi hàm append một chút. Đó là thêm các trạng thái mới vào đầu danh sách thay vì thêm vào cuối danh sách (vì Open là stack).