Paradigmas de Programação

Prof. Maicon R. Zatelli

Prolog - Programação Lógica
Predicados Dinâmicos

Universidade Federal de Santa Catarina
Florianópolis - Brasil

Prolog

Em Prolog, é possível adicionar e remover fatos e regras em tempo de execução.

Inicialmente, precisamos definir quais predicados são dinâmicos. Para isso, informamos o predicado e a sua aridade.

```
:- dynamic genitor/2, homem/1, pai/2.
```

Prolog

Adicionando novos fatos e regras

- **asserta(Termo)** permite adicionar um novo termo (que pode ser uma regra ou um fato) antes de todos os demais do mesmo tipo.
- **assertz(Termo)** permite adicionar um novo termo (que pode ser uma regra ou um fato) depois de todos os demais do mesmo tipo.

```
?- assertz(homem(tom)).
true.
?- asserta(homem(bob)).
true.
?- asserta(genitor(tom,bob)).
true.
?- assertz((pai(X,Y) :- genitor(X,Y), homem(X))).
true.
```

Prolog

Removendo fatos e regras

- **retract(Termo)** remove cada fato ou regra que unifica com o termo passado como parâmetro.
- **retractall(Termo)** remove todos os fatos ou regras que unificam com o termo passado como parâmetro.

```
?- retract(homem(X)).
X = bob ;
X = tom.

?- retractall(homem(X)).
true.
```

Programação Dinâmica

Prolog - Programação Dinâmica

Características dos Problemas

- São problemas de otimização (maximizar, minimizar)
- Tomar uma série de decisões
- Possibilidade de quebrar o problema em subproblemas semelhantes, **não independentes**, e menores que o problema original
- A solução desses subproblemas menores será utilizada para construir a solução para o problema original

Propriedades dos Problemas

- Sub-estrutura ótima: a solução do problema original é composta por soluções ótimas dos subproblemas
- Sobreposição de subproblemas: há uma pequena quantidade de subproblemas que devem ser resolvidos muitas vezes (repetição)

Prolog - Programação Dinâmica

Principais Passos para Construir uma Solução

1. Identificar os subproblemas. Quais problemas menores a partir do original existem?
2. Identificar as escolhas. Que opções tenho para resolver cada subproblema?
3. Relacionar os subproblemas (recorrência), ou seja, selecionar a melhor escolha dentre todas as existentes.
4. Identificar uma ordem topológica dos subproblemas (de acordo com a dependência de um com o outro), que significa definir a ordem em que os subproblemas devem ser resolvidos.
5. Responder ao problema original.

Prolog - Programação Dinâmica

Estratégias de Implementação

- **Estratégia bottom-up e tabela**. Normalmente é iterativa e começa diretamente pelos menores subproblemas em direção aos maiores, considerando a ordem topológica.

- **Estratégia recursiva e memoização**. Normalmente é recursiva e começa pelo problema original, que tenta recursivamente resolver os subproblemas menores que ele. Ao final da recursão estarão os menores subproblemas.

Prolog - Programação Dinâmica

Exemplos de Problemas

- Problema da mochila (*knapsack*)
- Maior subsequência crescente (*longest increasing subsequence (LIS)*)
- Problema do troco (*coin change*)
- Empilhando caixas (*box stacking*)
- Cortando torras (*rod cutting*)
- Multiplicação de cadeia de matrizes (*matrix chain multiplication*)
- Maior palíndrome (*longest palindromic subsequence*)
- Maior subsequência comum (*longest common subsequence (LCS)*)
- Árvore binária de busca ótima (*optimal binary search tree (OBST)*)

Prolog - Programação Dinâmica

Computar o n-ésimo número da sequência de Fibonacci.

```
fib(0,0) :- !.
fib(1,1) :- !.
fib(N,K) :-
    N1 is N - 1,
    N2 is N - 2,
    fib(N1,K1),
    fib(N2,K2),
    K is K1 + K2.
```

- Note que ao executar algo como **fib(35,X)** irão ser efetuados muitos cálculos repetidos, uma vez que não há nenhum fato que diga quais são os $n-1$ primeiros números da sequência de Fibonacci, exceto para F_0 e F_1.
- Tente executar **fib(35,X)** em seu computador e veja quando tempo irá demorar para calcular.

Prolog - Programação Dinâmica

Agora vamos armazenar os números da sequência de Fibonacci já encontrados.

```
:- dynamic fib2/2.
:- retractall( fib2(_,_) ).

fib2(0,0) :- !.
fib2(1,1) :- !.
fib2(N,K) :-
    N1 is N - 1, N2 is N - 2,
    fib2(N1,K1), fib2(N2,K2),
    K is K1 + K2,
    asserta(fib2(N,K) :- !).
```

- Inicialmente digo que **fib2/2** é um predicado de aridade 2 e é dinâmico.
- Depois removo todos os **fib2/2** já computados anteriormente (em outra execução).
- A cada novo número da sequência de Fibonacci encontrado, adiciono uma regra para ele: **asserta(fib2(N,K) :- !)**

O problema do troco

Dados valores de moedas V_i e um valor de troco T, o problema consiste em encontrar o menor número de moedas para dar o troco.

Por exemplo, assuma os valores de moedas 20, 30, e 60 e deve-se dar um troco de 110. Qual o menor número de moedas que pode ser usado para dar esse troco?

O problema do troco

Dados valores de moedas V_i e um valor de troco T, o problema consiste em encontrar o menor número de moedas para dar o troco.

Por exemplo, assuma os valores de moedas 20, 30, e 60 e deve-se dar um troco de 110. Qual o menor número de moedas que pode ser usado para dar esse troco?

- 3 moedas, uma de cada tipo.

Prolog - Programação Dinâmica

Subproblemas

Subproblemas de encontrar a quantidade mínima de moedas para dar trocos de $0..i$. Vamos chamar cada subproblema de $DP[i]$, ou seja, queremos descobrir o mínimo de moedas para dar um troco i.

Temos n subproblemas diferentes, com $n = T$.

Prolog - Programação Dinâmica

Escolhas

Dado que já tenho resolvido os subproblemas menores, de $0..i-1$, e estou querendo resolver o subproblema i, ou seja $DP[i]$ (identificar a menor quantidade de moedas para dar o troco **i**), basta identificar qual valor de moeda Vj será a última moeda utilizada para dar esse troco i.

Temos $|V|$ escolhas diferentes, no pior caso.

Prolog - Programação Dinâmica

Recorrência

Dentre todas as escolhas de Vj possíveis, escolho aquele Vj que minimize a quantidade de moedas necessárias para dar o troco i.

Assim, $DP[i]$ pode ser 1, caso tenha uma moeda com valor exatamente i ou o mínimo de moedas necessárias para dar o troco de $DP[i - Vj]$ mais a moeda Vj, para todo Vj.

Um caso especial que podemos inicializar é $DP[0] = 0$, visto que não há nenhuma moeda com valor 0.

$DP[0] = 0$
$DP[i] = min(DP[i - Vj] + 1)$, para todo valor de de moeda Vj, desde que $i - Vj \geq 0$

Ordem Topológica

A ordem em que os problemas devem ser resolvidos depende de qual problema depende de qual. Assim, os problemas devem ser resolvidos de maneira que todas as suas dependências já tenham sido resolvidas.

Se estamos resolvendo o problema $DP[i]$, então temos que ter resolvido todos os $DP[j]$, com $j < i$. Assim, a ordem em que resolvemos os subproblemas é: $DP[0], DP[1], .., DP[i], .., DP[T]$.

Prolog - Programação Dinâmica

Problema Original

O problema original é descobrir a menor quantidade de moedas para dar o troco T, portanto basta acessar a resolução de $DP[T]$ para obter essa informação.

Prolog - Programação Dinâmica

Estratégia de solução - Resumo

Estratégia de solução - Resumo

- Descubro qual é a última moeda que deve ser utilizada para dar o troco T.

Estratégia de solução - Resumo

- Descubro qual é a última moeda que deve ser utilizada para dar o troco T.
- Somo 1 ao total de moedas.

Estratégia de solução - Resumo

- Descubro qual é a última moeda que deve ser utilizada para dar o troco T.
- Somo 1 ao total de moedas.
- Ao utilizar a última moeda, irá sobrar um valor restante de troco T', que vem do uso da última moeda, então repito esse processo até que o valor de troco restante T' seja 0 ou impossível de se usar qualquer moeda para completar o troco.

Estratégia de solução - Resumo

- Descubro qual é a última moeda que deve ser utilizada para dar o troco T.
- Somo 1 ao total de moedas.
- Ao utilizar a última moeda, irá sobrar um valor restante de troco T', que vem do uso da última moeda, então repito esse processo até que o valor de troco restante T' seja 0 ou impossível de se usar qualquer moeda para completar o troco.
- Por exemplo, ao ter o troco de 110 para dar, posso querer utilizar como última moeda, a moeda de valor 20, então o troco restante T' será 90. Se usar como última moeda a moeda de valor 30, o troco restante T' será 80. Se usar como última moeda a moeda de valor 60, o troco restante T' será 50.

Estratégia de solução - Resumo

- Descubro qual é a última moeda que deve ser utilizada para dar o troco T.
- Somo 1 ao total de moedas.
- Ao utilizar a última moeda, irá sobrar um valor restante de troco T', que vem do uso da última moeda, então repito esse processo até que o valor de troco restante T' seja 0 ou impossível de se usar qualquer moeda para completar o troco.
- Por exemplo, ao ter o troco de 110 para dar, posso querer utilizar como última moeda, a moeda de valor 20, então o troco restante T' será 90. Se usar como última moeda a moeda de valor 30, o troco restante T' será 80. Se usar como última moeda a moeda de valor 60, o troco restante T' será 50.
- Continua ...

Estratégia de solução - Resumo

- Depois, tento descobrir o menor número de moedas necessárias para dar o troco restante, de 90, 80 e 50, e assim sucessivamente.

Estratégia de solução - Resumo

- Depois, tento descobrir o menor número de moedas necessárias para dar o troco restante, de 90, 80 e 50, e assim sucessivamente.
- Cuidados: o troco restante de 80 pode ser gerado com 4 moedas de 20, ou 2 moedas (uma de 60 e outra de 20), ou 3 moedas (duas de 30 e outra de 20). Assim, note que por diversas vezes iremos ter de recalcular os mesmos valores.

Estratégia de solução - Resumo

- Depois, tento descobrir o menor número de moedas necessárias para dar o troco restante, de 90, 80 e 50, e assim sucessivamente.
- Cuidados: o troco restante de 80 pode ser gerado com 4 moedas de 20, ou 2 moedas (uma de 60 e outra de 20), ou 3 moedas (duas de 30 e outra de 20). Assim, note que por diversas vezes iremos ter de recalcular os mesmos valores.
- Solução: salve sempre o menor número de moedas para computar algum valor!

Prolog - Programação Dinâmica

```
:- dynamic minimoMoedas/2.
:- retractall( minimoMoedas(_,_) ).

moeda(20). %Alguns valores de moedas
moeda(30).
moeda(60).

minimoMoedas(N,1) :- moeda(N), !.
minimoMoedas(N,K) :-
    N > 0,
    findall(KResto,
    (moeda(X), X < N, Resto is N - X, minimoMoedas(Resto, KResto)),
    ListaKResto),
    min_list(ListaKResto, KMinResto),
    K is KMinResto + 1,
    asserta(minimoMoedas(N,K) :- !).
```

- Tento adivinhar a última moeda a ser utilizada.
- Escolho a que utilizar o menor número de moedas e somo 1.
- Salvo o resultado.

Prolog - Programação Dinâmica

```
[debug]  ?- minimoMoedas(110,X).
```

Prolog - Programação Dinâmica

```
[debug]  ?- minimoMoedas(110,X).
X = 3.
```

Prolog - Alguns Links Úteis

- `http://www.swi-prolog.org/pldoc/man?section=dynpreds`
- `http://www.swi-prolog.org/pldoc/man?section=recdb`

Prolog

Ver atividade no Moodle