Nível 2

Tema: Recursão e Manipulação de Listas — Família Real Britânica

Descrição do Problema: O problema resolvido por este código é permitir que um sistema lógico determine automaticamente relações familiares diretas e indiretas dentro da árvore genealógica da Família Real Britânica, utilizando regras recursivas e manipulação de listas para realizar inferências complexas de parentesco.

Fatos: Foram definidos todos os pais e mães dos membros da família real britânica usada no modelo.

Regras:

1. parent(X, Y) — unifica pai e mãe.
2. irmaos(X, Y) — identifica irmãos biológicos (mesmo pai e mesma mãe).
3. avos(X, Y) — identifica avós.
4. ancestral(X, Y) — determina se **X está acima de Y** na árvore genealógica
    - pai/mãe (1 geração)
    - avô/avó (2 gerações)
    - bisavô/bisavó (3 gerações)
    - e assim por diante...
    a. Exemplo: `?- ancestral('Elizabeth', 'George').` →
        - O prolog entende a cadeia: $Elizabeth \rightarrow CharlesIII \rightarrow William \rightarrow George$
            - Elizabeth → Charles III (parent)
            - Charles III → William (parent)
            - William → George (parent)
5. listaAncestrais(Y, Lista) — constrói uma **lista completa** com todos os ancestrais de Y em ordem de descoberta.

- `findall(Elemento, Condição, Lista)`:
    - coleta todas as soluções possíveis da condição;
    - armazena tudo em `Lista`;

a. Exemplo: `?- listaAncestrais('George', L).` →

- Retorna uma lista: `L = ['Philip', 'Elizabeth', 'Charles III', 'Diana', 'William'].`

6. descendente(Y, X) — a **versão espelhada** da regra ancestral:
    - `ancestral(X, Y)` → X está acima
    - `descendente(Y, X)` → Y está abaixo

    a. Exemplo: `?- descendente('Louis', 'Philip').`

    - O prolog entende a cadeia: $Philip \rightarrow CharlesIII \rightarrow William \rightarrow Louis$
7. listaDescendentes(X, Lista) — **retorna todos os descendentes de X**, usando a regra recursiva `descendente`.
    - Coleta todos os elementos que estão abaixo de X

    a. Exemplo: `?- listaDescendentes('Elizabeth', L).`

    - Retorna uma lista: `L = ['Charles III','Anne','Andrew','Edward', ... ].`
8. ancestralEmComum(X, Y, Z) — Z é ancestral comum de X e Y **se for ancestral de ambos**.
    - Para cada candidato Z:
        1. Verifica se é ancestral de X
        2. Verifica se é ancestral de Y
        3. Garante que X e Y não são a mesma pessoa

    a. Exemplo: `?- ancestralEmComum('George','Archie', Z).`

    - Retorna elementos Z, que possuem `'George'` e `'Archie'` como ancestrais em comum:
        - `Z = 'Elizabeth' ;`
        - `Z = 'Philip'.`
9. distMembros(X, Y, Caminho) — gera uma **lista representando o caminho genealógico** entre dois membros.

- **Caso Base:** `Caminho = [X, Y]` → Se X é pai ou mãe de Y:
- **Caso Recursivo:** `Caminho = [X | CaminhoEntre(Z e Y)]` → Se X é pai/mãe de Z, e Z leva até Y:

a. Exemplo: `?- distMembros('Philip', 'George', C).`

- Retorna uma lista: `C = ['Philip', 'Charles III', 'William', 'George'].`

Consultas:

1. `?- ancestral('Elizabeth', X).` → "Descobrir **todas as pessoas para as quais Elizabeth é ancestral**, ou seja, todos os seus descendentes diretos e indiretos." | Como o Prolog faz o raciocínio dessa regra?
   - Procura pessoas Y que sejam filhos diretos de Elizabeth → `Charles III`, `Anne`, `Andrew`, `Edward`.
   - Para cada um desses, aplica a recursão e procura seus filhos, netos, bisnetos etc.
   - Exemplo do caminho lógico:
     - Elizabeth → Charles III → William → George
     - Elizabeth → Charles III → Harry → Archie

   a. Resultado final:

Interpretação da Regra: A consulta lista **toda a linhagem abaixo de Elizabeth**, funcionando como uma busca em profundidade recursiva.

2. `?- descendente(X, 'Philip').` → “Encontrar **todos os descendentes** de Philip (inclusive filhos, netos, bisnetos etc.).” | Como o Prolog faz o raciocínio dessa regra?
    - Primeiro encontra os filhos diretos:
        - Charles III
        - Anne
        - Andrew
        - Edward
    - Depois aplica a recursão:
        - Descendentes de Charles III → William, Harry, George, Charlotte, Louis

- Descendentes de Anne → Peter, Zara, Savannah, Isla, Mia, Lucas, Lena
- Descendentes de Andrew → Beatrice, Eugenie, Sienna, Athena, August, Ernest
- Descendentes de Edward → Louise, Earl

a. Resultado final:

```
?- descendente(X, 'Philip').
X = 'Charles III' ;
X = 'Anne' ;
X = 'Andrew' ;
X = 'Edward' ;
X = 'William' ;
X = 'Harry' ;
X = 'George' ;
X = 'Charlotte' ;
X = 'Louis' ;
X = 'Archie' ;
X = 'Lilibet' ;
X = 'Peter Phillips' ;
X = 'Zara Phillips' ;
X = 'Savannah' ;
X = 'Isla' ;
X = 'Mia' ;
X = 'Lucas' ;
X = 'Lena' ;
X = 'Beatrice' ;
X = 'Eugenie' ;
X = 'Sienna' ;
X = 'Athena' ;
X = 'August' ;
X = 'Ernest' ;
X = 'Louise' ;
X = 'Earl' ;
false.
```

Interpretação da Regra: Essa consulta mostra o caminho inverso da ancestralidade, comprovando que Philip é o patriarca acima de várias gerações.

3. `?- listaDescendentes('Elizabeth', L).` → "Gerar uma **lista completa** contendo todos os descendentes de Elizabeth." | Como o Prolog faz o raciocínio dessa regra?
   - `findall(Y, Condição, Lista)` coleta **todas** as soluções que satisfazem a condição.

- A condição é `descendente(Y, 'Elizabeth')`, que utiliza recursão.

a. Resultado final:

```
?- listaDescendentes('Elizabeth', L).
L = ['Charles III', 'Anne', 'Andrew', 'Edward', 'William', 'Harry', 'George', 'Charlotte', 'Louis'|...].
```

Interpretação da Regra: Diferente da consulta anterior (que imprime um por vez), aqui o Prolog constrói **uma única lista contendo todos os descendentes**, cumprindo a exigência de manipulação de listas do Nível 2.

4. `?- listaAncestrais('George', L).` → "Listar **todos os ancestrais** de George em ordem de descoberta." | Como o Prolog faz o raciocínio dessa regra?
   - O Prolog verifica:
     - Quem é pai de George → `'William'`
     - Quem é pai de William → `'Charles III'`
     - Quem é pai de Charles III → `'Philip'`
     - Quem é mãe de Charles III → `'Elizabeth'`
     - Quem é mãe de William → `'Diana'`
   - Todos os ancestrais encontrados são colocados na lista.

   a. Resultado final:

```
?- listaAncestrais('George', L).
L = ['William', 'Catherine', 'Philip', 'Charles III', 'Elizabeth', 'Diana'].
```

Interpretação da Regra: A consulta constrói a "árvore acima" de George, provando que o algoritmo de ancestralidade está correto.

5. `?- ancestralEmComum('George', 'Archie', Z).` → "Encontrar **ancestrais comuns** entre George e Archie." | Como o Prolog faz o raciocínio dessa regra?
   - Ancestrais de George são → `William, Charles III, Philip, Elizabeth, Diana`
   - Ancestrais de Archie são → `Harry, Charles III, Philip, Elizabeth, Diana, Meghan`
   - É feita uma interseção nas duas listas → `Philip, Elizabeth, Charles III, Diana`

   a. Resultado final:

```
?- ancestralEmComum('George', 'Archie', Z).
Z = 'Philip' ;
Z = 'Charles III' ;
Z = 'Elizabeth' ;
Z = 'Diana' ;
false.
```

Interpretação da Regra: Essa consulta demonstra inferência lógica envolvendo **duas linhas familiares simultaneamente**, algo mais avançado.

6. `?- distMembros('Philip', 'George', Caminho).` → "Construir a **sequência de gerações** ligando Philip até George." | Como o Prolog faz o raciocínio dessa regra?
   - O Prolog faz:
     1. Philip → Charles III
     2. Charles III → William
     3. William → George
   - Como cada etapa corresponde ao caso recursivo, o caminho final é: `['Philip', 'Charles III', 'William', 'George']`

   a. Resultado final:

```
?- distMembros('Philip', 'George', Caminho).
Caminho = ['Philip', 'Charles III', 'William', 'George'] ;
false.
```

Interpretação da regra: Essa consulta demonstra: recursão; construção de listas; encadeamento lógico profundo. Simplificadamente, ela demonstra a distância entre dois membros da família e o caminho para chegar até eles.