INE5416
Paradigmas de Programação

Ricardo Azambuja Silveira
INE-CTC-UFSC
E-Mail: silveira@inf.ufsc.br
URL: www.inf.ufsc.br/~silveira

Estrutura de dados

- Listas
- Pilhas
- Conjuntos
- Árvores
- Grafos

Classificação em listas

- Algoritmo bubblesort
  - Encontre dois elementos adjacentes, X e Y, nesta ordem em Lista, tais que X @> Y. Troque as posições de X e Y, obtendo Lista1 e, depois, classifique Lista1;
  - Se não houver nenhum par de elementos adjacentes, X e Y, nesta ordem em Lista, então esta já está classificada.
  - Neste algoritmo tanto o melhor caso, como o pior caso tem ordem "n^2" porque os ciclos são sempre realizados até ao fim, mesmo quando os elementos já estão ordenados.

Classificação em listas

- Bubblesort

```
bubblesort(Lista, Saída) :-
    troca(Lista, Lista1), !, bubblesort(Lista1, Saída).
bubblesort(Saída, Saída).
troca([X, Y | Resto], [Y, X | Resto]) :-
    X@>Y.
troca([Z | Resto], [Z | Resto1]) :-
    troca(Resto, Resto1).
```

Classificação em listas

- Algoritmo insertsort
  - Para classificar uma lista não vazia, L = [X | R]:
    - Classifique o corpo R da lista L;
    - Insira a cabeça, X, de L no corpo classificado em uma posição tal que a lista resultante esteja classificada.

Classificação em listas

- Algoritmo insertsort

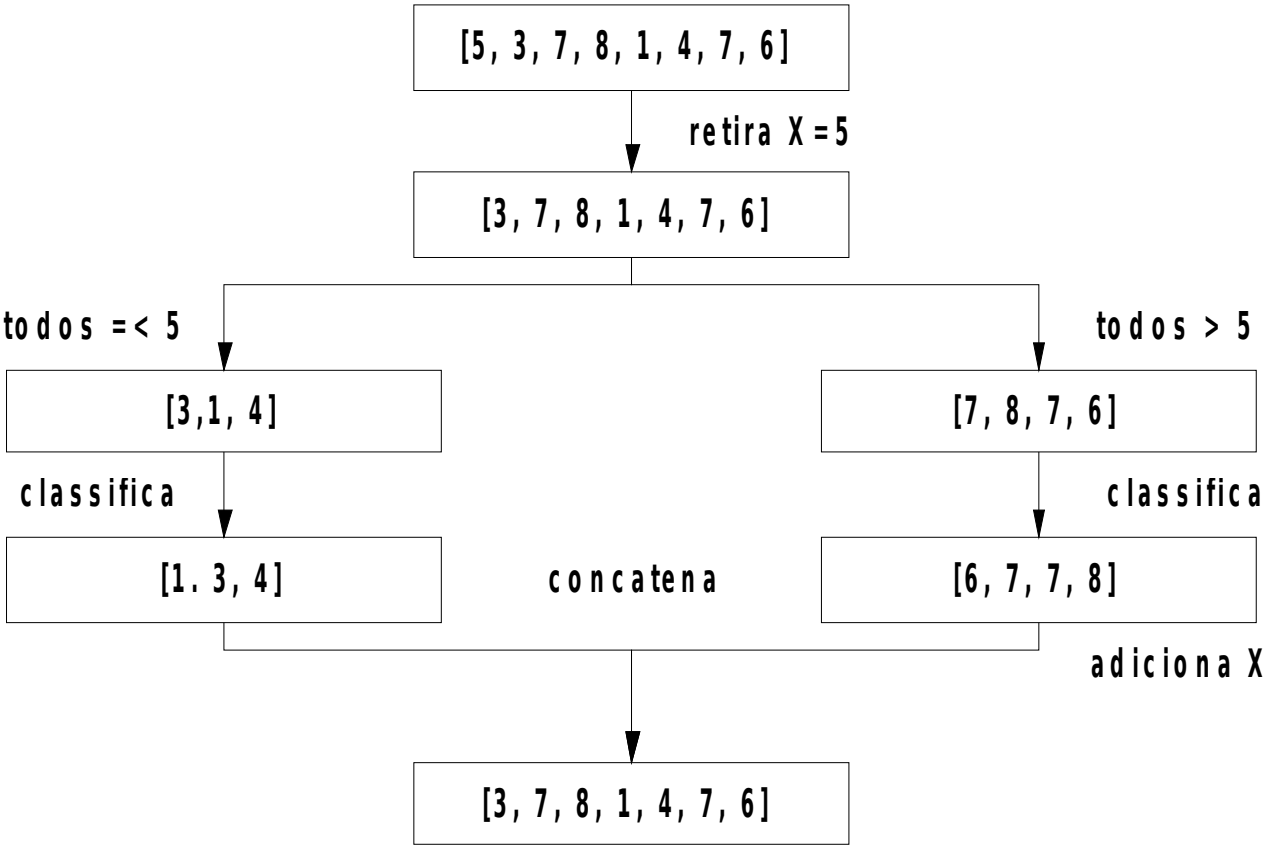
```
insertsort([], []).
insertsort([X | Resto], Saída) :-
    insertsort(Resto, Resto1), insere(X, Resto1, Saída).
insere(X, [Y | Saída], [Y | Saída1]) :-
        X@>Y, !, insere(X, Saída, Saída1).
insere(X, Saída, [X | Saída]).
```

Classificação em listas

- Algoritmo Quicksort
  - Para classificar uma lista não vazia L:
    - Separe algum elemento X de L e divida o restante em duas listas, denominadas Menor e Maior, da seguinte maneira: Todos os elementos de L que são maiores do que X pertencem a Maior e os restantes pertencem a Menor;
    - Classifique Menor, obtendo Menor1;
    - Classifique Maior, obtendo Maior1;
    - A lista completa é a concatenação de Menor1 com [X | Maior1].

Classificação em listas

- Algoritmo Quicksort



Classificação em listas

- Algoritmo Quicksort

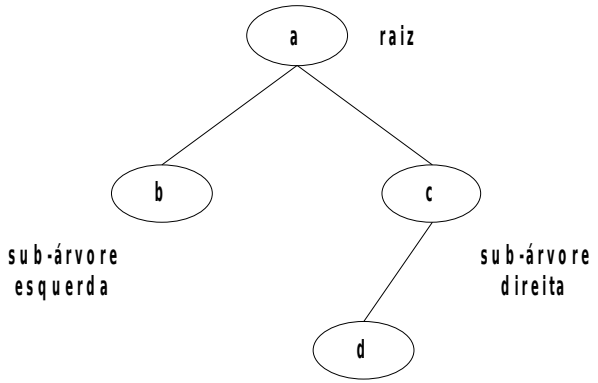
```
quicksort([X | R], Saída) :-
    divide(X, R, Maior, Menor),
    quicksort(Menor, Menor1),
    quicksort(Maior, Maior1),
    conc(Menor1, [X | Maior1], Saída).

divide(X, [], [], []).
divide(X, [Y | R], [Y | Menor], Maior) :-
 X@> Y, !, divide(X, R, Menor, Maior).
divide(X, [Y | R], Menor, [Y | Maior]) :-
 divide(X, R, Menor, Maior).

conc([], L, L).
conc([X | L1], L2, [X | L3]) :-
 conc(L1, L2, L3).
```

Conjuntos

- Representação através de árvores binárias
  - Uma árvore binária, ou é vazia, ou é constituída por três argumentos:
    - uma raiz,
    - uma sub-árvore esquerda, e
    - uma sub-árvore direita.
  - árvore mostrad ınto {a, b, c, d}



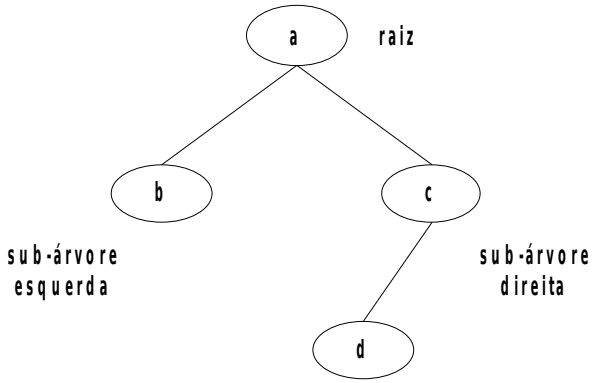
Arvores binárias

- Representação em prolog
  - Uma forma é tornar a raiz da árvore o functor principal do termo e as sub-árvores os seus argumentos: a(b, c(d)).
  - Outra alternativa é usar um símbolo para representar a árvore vazia e um functor para as árvores comtrês componentes (a raiz e as duas sub-árvores).

Arvores binárias

- Representação em prolog
  - A árvore vazia será representada pelo átomo "nil",
  - usa-se functor t, de forma que a árvore que possui uma raiz R, uma sub-árvore esquerda E e uma sub-árvore direita D: t(E, R, D)
  - Nessa representação a árvore do exemplo fica:

    t(t(nil, b, nil), a, t(t(nil, d, nil), c, nil))



Arvores binárias

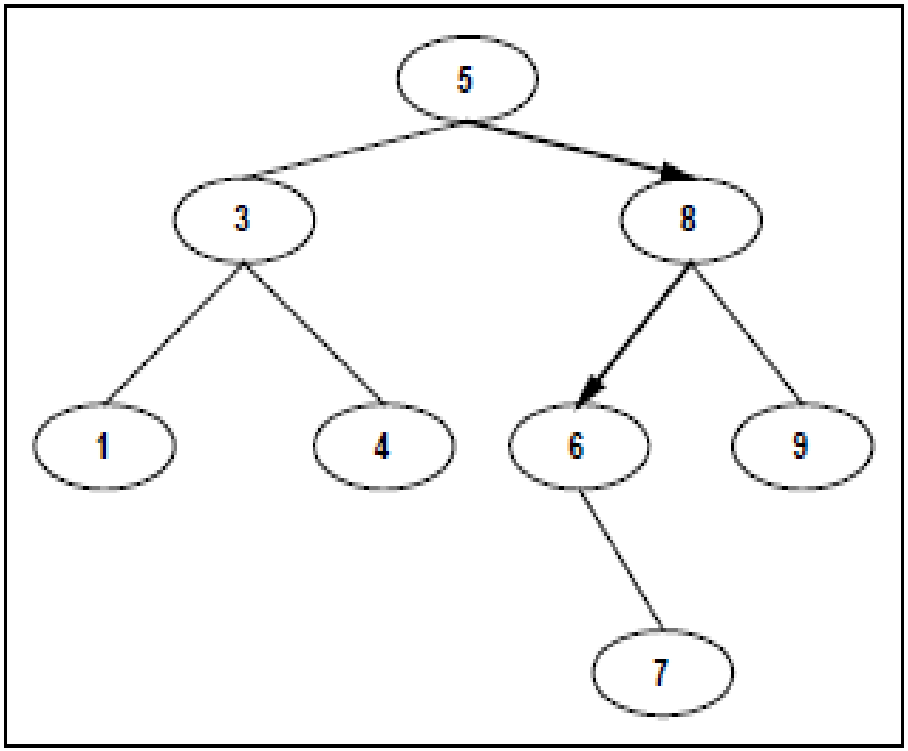
- Relação de pertinência
  - X pertence a uma árvore T se:
    - A raiz de T é X
    - X está na sub-árvore esquerda
    - X está na sub-árvore direita

```
pertence(X, t(_, X, _)) :- !.
pertence(X, t(E, _, _)) :-
   pertence(X, E).
pertence(X, t(_, _, D)) :-
   pertence(X, D).
```

Árvores ordenadas

- uma árvore não-vazia t(E, R, D) está ordenada da esquerda para a direita se:
  - Todos os nodos na sub-árvore E são menores do que X
  - Todos os nodos na sub-árvore D são maiores do que X
  - Ambas as sub-árvores estão também ordenadas.
- Tal árvore binária é denominada um "dicionário binário".

Árvores ordenadas



Dicionário binário

- Para encontrar um item X em um dicionário binário D:

  Se X é a raiz de D, então X já foi encontrado, senão

  Se X é menor do que a raiz de D, então X deve ser procurado na sub-árvore esquerda de D, senão

  Procurar X na sub-árvore direita de D, e

  Se D estiver vazio a pesquisa falha.

```
pertence(X, t(_,X,_)).
pertence(X, t(E, R, _)) :-
        R@>X, pertence(X, E).
pertence(X, t(_, R, D)) :-
        X@> R, pertence(X, D).
```

Dicionário binário

- O procedimento pertence/2 pode também ser empregado para construir um dicionário binário
- a consulta abaixo irá construir um dicionário binário D que contém os elementos 5, 3 e 8:

```
?-pertence(5, D), pertence(3, D), pertence(8, D).

D=t(t(D1, 3, D2), 5, t(D3, 8, D4))
```

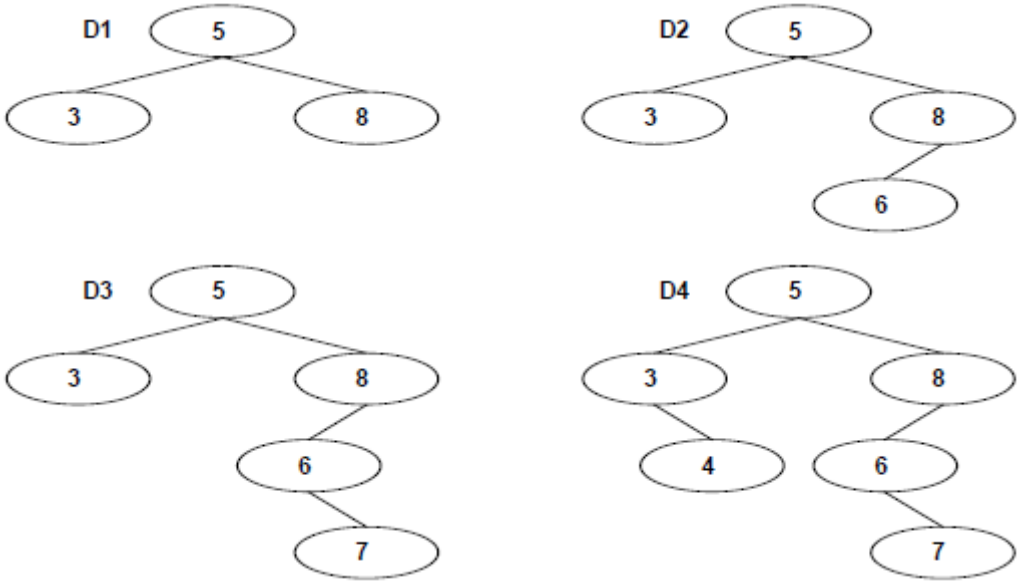
  - As variáveis D1, D2, D3 e D4 são sub-árvores não especificadas, que podem conter qualquer coisa. O dicionário que será construído irá depender da ordem dos objetivos na consulta.

Dicionário binário

- Insersão
  - O resultado da inserção de X a uma árvore vazia é a árvore t(nil,X,nil);
  - Se X é a raiz de D, D1=D (itens duplicados não são inseridos);
  - Se a raiz de D é maior do que X, então X deve ser inserido na sub-árvore esquerda de D. Caso contrário X deve ser inserido na sub-árvore direita de D.

Dicionário binário

- as árvores correspondem a seguinte sequência de inserções:
  - insFolha(D1, 6, D2),
  - insFolha(D2, 7, D3),
  - insFolha(D3, 4, D4)



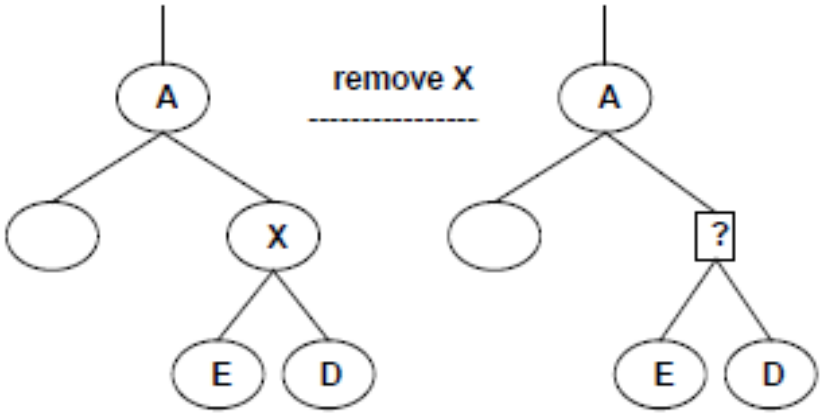
Dicionário binário

- Inserindo folhas em prolog

```
insFolha(nil, X, t(nil, X, nil)).
insFolha(t(E, X, D), X, t(E, X, D)).
insFolha(t(E, R, D), X, t(E1, R, D)) :-
  R@> X, insFolha(E, X, E1).
insFolha(t(E, R, D), X, t(E, R, D1)) :-
  X@>R, insFolha(D, X, D1).
```

Dicionário binário

- Removendo nodos



Dicionário binário

- Se uma das duas sub-árvores, E ou D, estiver vazia,a sub-árvore não-vazia é conectada a A.
- Se ambas forem não-vazias o nodo mais à esquerda de D ocupa a lacuna
- A transferência do nodo mais à esquerda da sub-árvore direita é realizada pela relação trans(T, Y, T1) onde Y é o nodo mais à esquerda de T e T1 é T após remover Y.

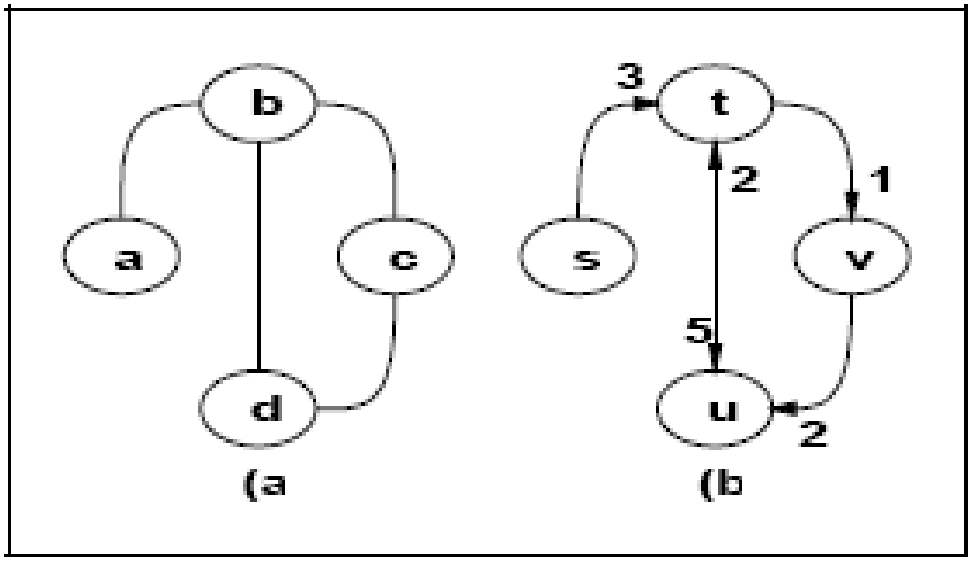
Dicionário binário

```
remove(t(nil, X, D), X, D).
remove(t(E, X, nil), X, E).
remove(t(E, X, D), X, t(E, Y, D1)) :-
  trans(D, Y, D1).
remove(t(E, R, D), X, t(E1, R, D)) :-
  R@> X, remove(E, X, E1).
remove(t(E, R, D), X, t(E, R, D1)) :-
  X@> R, remove(D, X, D1).

trans(t(nil, Y, D), Y, D).
trans(t(E, R, D), Y, t(E1, R, D)) :-
  trans(E, Y, E1).
```

Representação de Grafos

- Um grafo um conjunto de nodos arestas, onde cada aresta interliga um par de nodos.
- Quando as arestas são direcionadas, são denominadas arcos, formando grafos direcionados.
- Aos arcos podem ser associados custos, nomes etc.



Representação de Grafos

- Os grafos podem ser representados em Prolog de diversas maneiras.
  - Um método é representar cada aresta ou arco separadamente, por meio de uma cláusula.

```
conecta(a, b).
conecta(b, c).
conecta(c, d).
conecta(d, b).
```

```
arco(s, t, 3).
arco(t, u, 5).
arco(t, v, 1).
arco(v, u, 2).
arco(u, t, 2).
```

CTC • UFSC
Informática e Estatística

Representação de Grafos

Representação de Grafos

- Outro método é representar o grafo completo como um único objeto representado por um par de conjuntos: nodos e arcos

G1 = grafo([a,b,c,d],[ar(a,b),ar(b,c),ar(b,d),ar(c,d)])

G2 = grd([s,t,u,v],[a(s,t,3),a(t,v,1),a(t,u,5),a(u,t,2), a(v,u,2)])

- Outro método é associar a cada nodo a lista de todos os nodos adjacentes, usando um operador infixo

G1 = [a ® [b], b ® [a, c, d], c ® [b, d], d ® [b, c]]

G2 = [s ® [t/3], t ® [u/5, v/1], u ® [t/2], v ® [u/2]]

Caminhamento em grafos

- Dado o grafo G, os nodos A eZ, o caminho C é dado por:

```
caminho(A, Z, G, C) :-
    caminho1(A, [Z], G, C).
caminho1(A, [A | C1], _, [A | C1]).
caminho1(A, [Y | C1], G, C) :-
    adjacente(X, Y, G),
    not membro(X, C1),
    caminho1(A, [X, Y | C1], G, C).
adjacente(X, Y, grafo(Nodos, Arestas)) :-
    membro(ar(X, Y), Arestas);
    membro(ar(Y, X), Arestas).
```

Caminho Hamiltoniano

- Percorre todos os nodos do grafo

```
hamiltoniano(Grafo, Caminho) :-
    caminho( _, _, Grafo, Caminho),
    cobre(Caminho, Grafo).
cobre(Caminho, Grafo) :-
    not(nodo(N, Grafo), not membro(N, Caminho).
```

Custo do caminho

```
caminho(A, Z, G, C, Custo) :-
    caminho1(A, [Z], 0, G, C, Custo).
caminho1(A, [A | C1], Custo1, G, [A | C1], Custo1).
caminho1(A, [Y | C1], Custo1, G, C, Custo) :-
    adjacente(X, Y, CustoXY, G),
    not membro(X, C1),
    Custo2 is Custo1+CustoXY,
    caminho1(A, [X, Y | C1], Custo2, G, C, Custo).
```

Cronograma

DATA	ASSUNTO
30/10	Estruturas de dados
06/11	Estruturas de dadosEnunciado do t rabalho final (SECCOM)
13/11	TRABALHO
20/11	TRABALHO
27/11	PROVA
04/12	Prova de recuperação
11/12	