PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

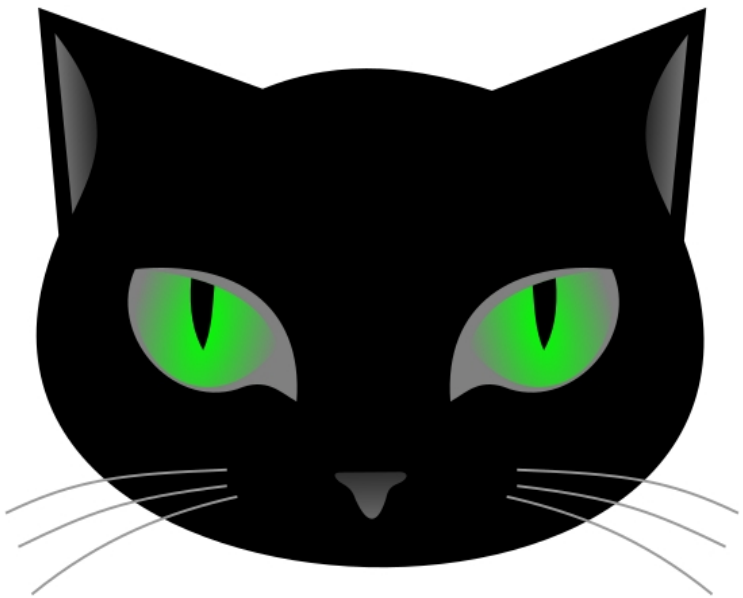
claudio.sa@udesc.br

Departamento de Ciência da Computação – DCC
Centro de Ciências e Tecnológias – CCT
Universidade do Estado de Santa Catarina – UDESC

7 de maio de 2019

Buscas

- O que é uma *busca*?
- Problemas $\Rightarrow$ buscar ...
- Buscas em estruturas quaisquer
- Listas são o suficiente!
- Núcleo das buscas
- Exemplo



- Requisito: conceitos de listas e recursividade dominados!

- Requisito: conceitos de listas e recursividade dominados!
- Além destes: noções sobre grafos, árvores, nós, etc

- Requisito: conceitos de listas e recursividade dominados!
- Além destes: noções sobre grafos, árvores, nós, etc
- Solucionar problemas implicar em percorrer estados (um caminho) que levem há um estado–solução

- Requisito: conceitos de listas e recursividade dominados!
- Além destes: noções sobre grafos, árvores, nós, etc
- Solucionar problemas implicar em percorrer estados (um caminho) que levem há um estado–solução
- Grosseiramente: **estados de problemas $\Leftrightarrow$ estruturas abstratas**

- Requisito: conceitos de listas e recursividade dominados!
- Além destes: noções sobre grafos, árvores, nós, etc
- Solucionar problemas implicar em percorrer estados (um caminho) que levem há um estado–solução
- Grosseiramente: **estados de problemas $\Leftrightarrow$ estruturas abstratas**
- Pois, problemas em geral se apresentam como uma conexão complexa tipo um *grafo*, e a varredura sob este grafo é sistemática sob uma *árvore de busca*

- Requisito: conceitos de listas e recursividade dominados!
- Além destes: noções sobre grafos, árvores, nós, etc
- Solucionar problemas implicar em percorrer estados (um caminho) que levem há um estado–solução
- Grosseiramente: **estados de problemas $\Leftrightarrow$ estruturas abstratas**
- Pois, problemas em geral se apresentam como uma conexão complexa tipo um *grafo*, e a varredura sob este grafo é sistemática sob uma *árvore de busca*
- Então, computar listas em Picat é uma estratégia de resolver problemas!

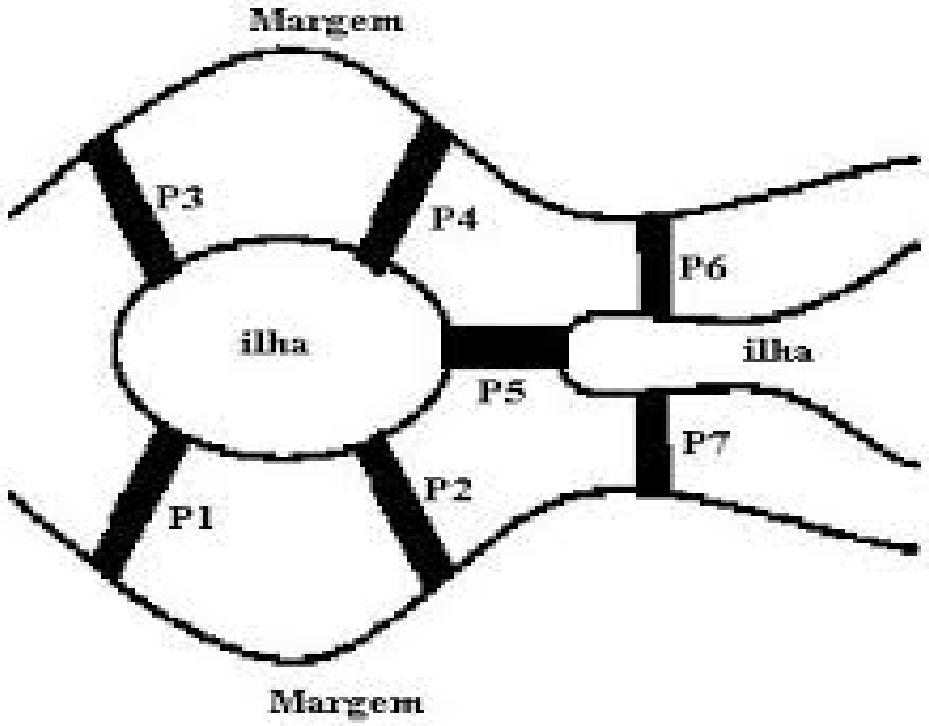


Figura 1: Ciclo Euleriano – Problema das Pontes de Königsberg

Ciclo Euleriano II

- No século 18 havia na cidade de Königsberg (antiga Prússia) um conjunto de sete pontes (identificadas pelas letras de P1 até P7 na figura ao lado) que cruzavam o rio Prególia. Elas conectavam duas ilhas entre si e as ilhas com as margens esquerda e direita.
- Os habitantes daquela cidade perguntavam-se se era possível cruzar as sete pontes numa caminhada contínua sem que se passasse duas vezes por qualquer uma das pontes.
- Embora intrigante, este problema foi atacado por Leonard Euler (1736) e demonstrou que isto não era possível para um grafo qualquer
- Curiosamente, este problema é *fácil* de resolver. Euler demonstrou uma relação entre vértices e arestas para que isto fosse possível.

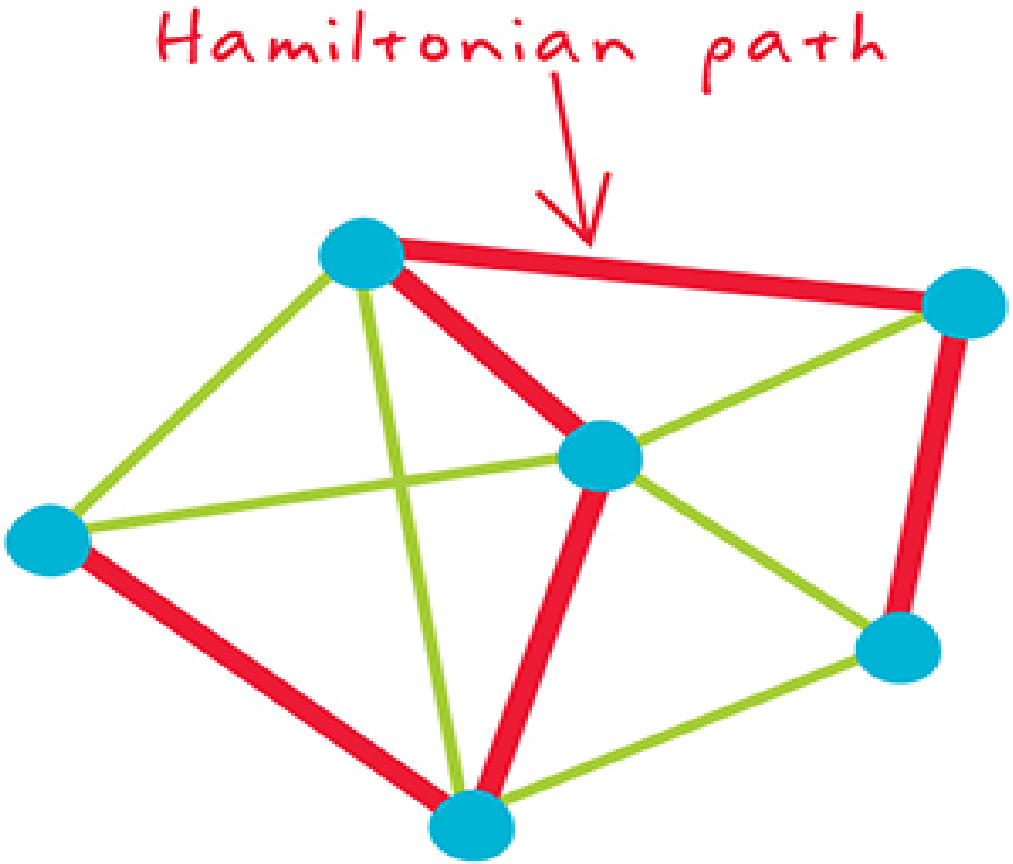


Figura 2: Caminho Hamiltoniano – Há um caminho que passe por todas cidades uma única vez?

Caminho Hamiltoniano II

- **Diferente** do ciclo Euleriano, o caminho Hamiltoniano, origem e destino são diferentes
- Todos os nós precisam ser visitados uma única vez sem repetição
- Num grafo pode haver muitos caminhos Hamiltonianos, mas, pode não existir nenhum!
- Ao contrário do ciclo Euleriano, este problema, computacionalmente é *difícil* de resolver!
- Vamos usar o caminho Hamiltoniano como exemplo, para construir um algoritmo ingênuo, mas que funciona bem!

Problemas, Estados, Grafos e Árvores de Buscas I

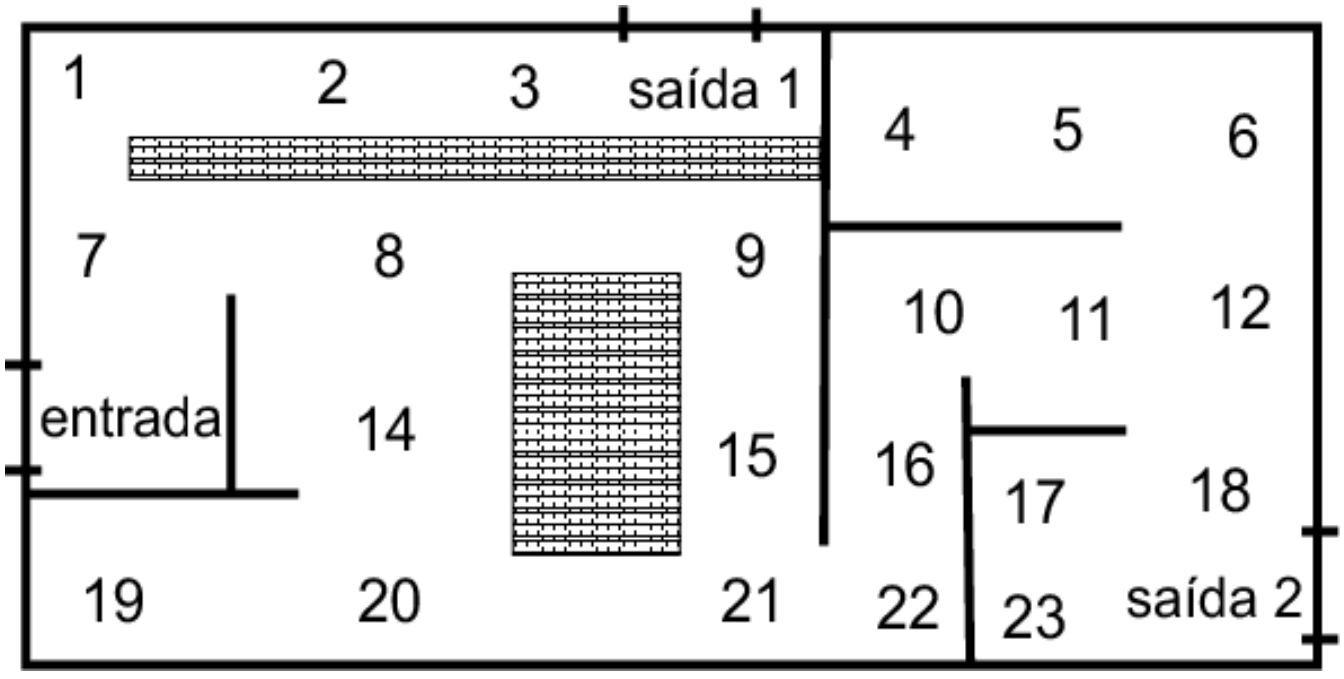
Contextualizando estes termos:

- Em geral, problemas podem ser vistos como *fotografias instantâneas* de uma situação, isto é, **um estado discreto**
- Uma *sucessão* destes estados, compõem *um caminho* de um estado *i* ao estado *j*
- Assim, estes *estados* são representados pelos *nós dos grafos*, e a ligação entre estes, são resultados de *uma ação*, mudança ou evolução do problema
- Há um estado particular chamado *inicial*, vários outros de estados *intermediários*, e outros estados *finais*
- Se o problema tiver várias soluções, o mesmo apresenta vários caminhos do estado inicial ao final.

Problemas, Estados, Grafos e Árvores de Buscas II

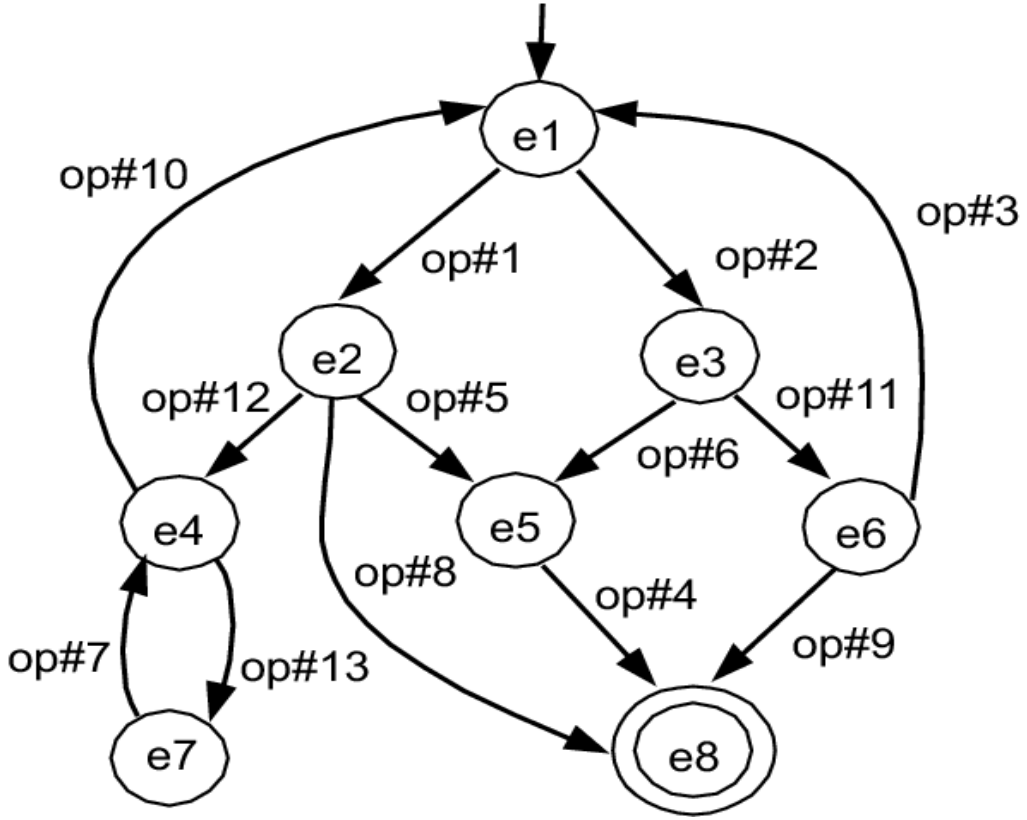
- Assim uma sucessão ou transição válida entre estados, é conhecido como uma *solução* ou *prova* do problema
- Essencialmente vamos varrer uma estrutura entre estados ou nós, de modo sistemático até encontrarmos uma solução aceitável/desejável.
- Logo, vamos empregar alguns conceitos da teoria dos grafos, em modelar problemas e resolvê-los por um esquema de busca computacional

Problema do Robô no Labirinto



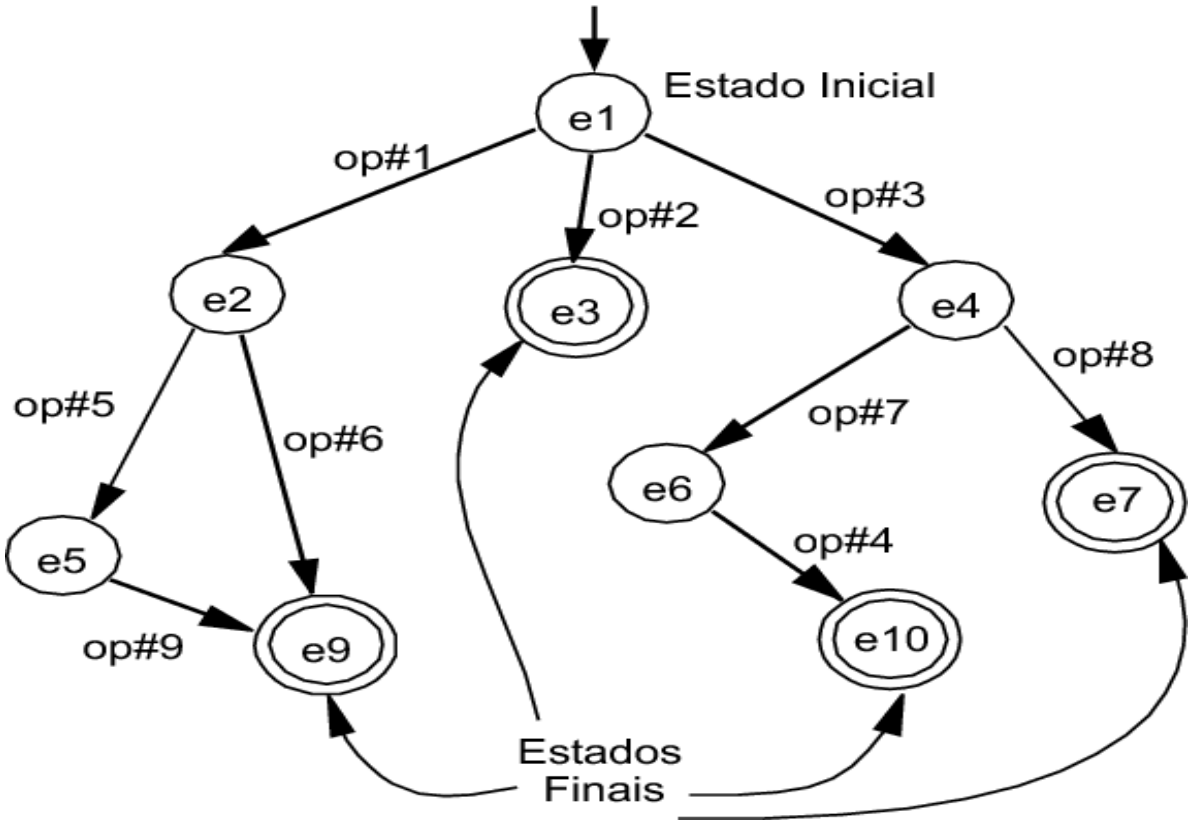
Imagine qualquer problema: uma busca na WEB, atomicidade de transações de BD, máquina de café, carro autônomo, etc

Problemas de Grafos se Transformam em Árvores de Buscas



Resumindo, os problemas são modelados em estruturas complexas, tais como grafos, mas o processo de solução se mantém: **realizar uma busca, tal como uma estrutura de uma árvore (listas)** !

Problemas de Grafos se Transformam em Árvores de Buscas



Assim, a idéia é percorrer os **ramos destas árvores**, de maneira sistemática. Um **caminho** da raiz há um nó terminal de interesse, este é uma solução do problema. Logo, tudo se resume em manipular muitas **listas**!

Pseudo-código já em Picat

```
resolve(P) =>
      inicio(Start),
      busca(Start,[Start],Qsol),
      imprime_saida(Qsol,P).

busca(S,P,P) ?=>  objetivo(S).     % objetivo alcancado : FIN
busca(S,Visited,P) =>
     proximo_estado(S,Nxt),        % gera um proximo estado
     estado_seguro(Nxt),           % verifica se este estado
     sem_loop(Nxt,Visited),        % verifica se está em loop
     busca(Nxt,[Nxt|Visited],P).   % continue a busca recursi
```

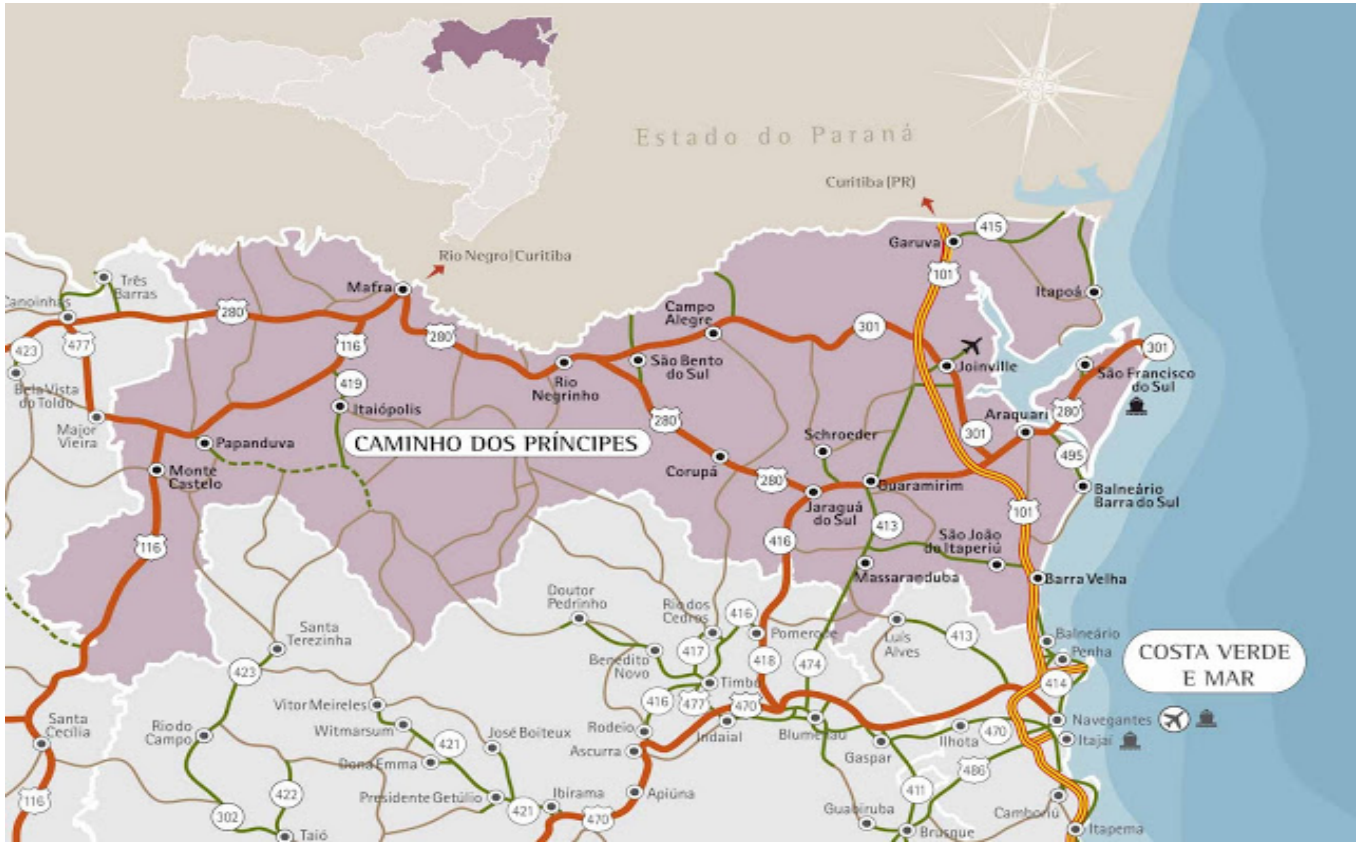
```
sem_loop(Nxt,Visited) :-
      \+member(Nxt,Visited).

proximo_estado(S,Nxt) =>   < fill in here >.
estado_seguro(Nxt) =>     < fill in here >.
sem_loop(Nxt,Visited) =>  < fill in here >.

inicio(...).
objetivo(...).
```

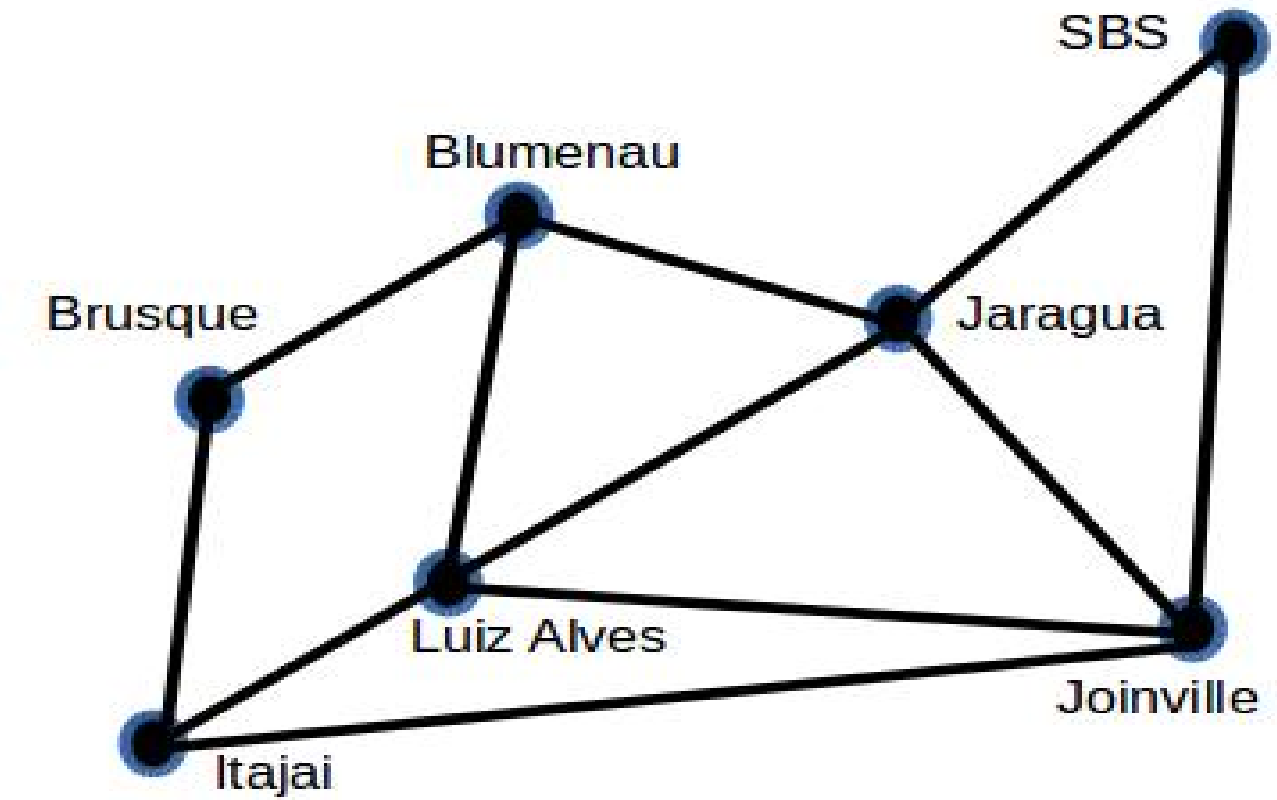
Vamos reescrever este pseudo-código em um problema!

Caminho Hamiltoniano Aplicado



Seja um viajante que sai cedo de Joinville, e chegar a noite em Blumenau, passando por algumas destas cidades uma única vez!

Cidades Escolhidas pelo Viajante



Modelagem do Problema – *O nosso viajante do Vale do Itajaí* I

- Em nosso problema temos 7 cidades pré-escolhidas
- A lista de cidades são:

```
index(-) %% lista de todas cidade do mapa
as_cidades( [ brusque, blumenau, itajai, luiz_alves,
              jaragua, sao_bento, joinville ] ).
```

- Duas cidades em particular:

```
index(-)
destino( blumenau ).

index(-)
origem( joinville ).
```

- As estradas transitáveis entre as cidades definem o nosso mapa, consequentemente um grafo entre cidades:

Modelagem do Problema – *O nosso viajante do Vale do Itajaí* II

```
%% MAPA da região
index(-,-)
arco(joinville, sao_bento)   .
arco(joinville, itajai)      .
arco(joinville, jaragua)      .
arco(joinville, luiz_alves)      .
arco(jaragua, sao_bento)     .
arco(jaragua, blumenau)      .
arco(jaragua, luiz_alves)    .
arco(itajai, luiz_alves)     .
arco(blumenau, luiz_alves)     .
arco(blumenau, itajai)       .
arco(brusque, itajai)        .
arco(brusque, blumenau)      .
```

- As estradas entre as cidades são bidirecionais. Se há estrada para ir da cidade X a cidade Y então na outra direção é verdadeiro. Em regras isto é escrito por:

Modelagem do Problema – *O nosso viajante do Vale do Itajaí* III

```
/* BI-DIRECIONALIDADE DOS ARCOS */
move_no(X,Y) ?=> arco(X,Y).
move_no(X,Y)  => arco(Y,X).
```

- Claro, este problema é pequeno e construindo o grafo dá para constatar que existe mais uma solução para o nosso viajante
- Para resolver este problema vamos utilizar uma **busca em profundidade**
- Esta **busca em profundidade** (do inglês. *depth first search – DFS*), encontra-se inserida no contexto *buscas em geral*, visto anteriormente.

O *Miolo* ou Núcleo da Busca

```
busca_DFS ( [ No_corrente | Caminho] , L_sol) ?=>
      destino(No_final),        %%% condicao de parada 1
      No_corrente == No_final,
      L_sol = [ No_corrente | Caminho ],
      as_cidades(L_Todas_Cidades),    %%% condicao de parada 2
      %% TODAS CIDADES FORAM VISITADAS
      length (L_sol) ==  length(L_Todas_Cidades),
      write(L_sol),
      printf(" \n UMA SOLUCAO ....: OK\n ==>").

busca_DFS (  [NoH | Caminho], Solucao) =>
      %%% explorar um novo movimento ou um novo noh
      move_no(NoH , Novo_NoH),
      %% testar se este novo noh nao foi visitado ainda
      %% ou novo_NOH eh permitido
      not( member(Novo_NoH, [NoH|Caminho]) ),
      busca_DFS(  [Novo_NoH , NoH | Caminho ] , Solucao).
```

O Código Completo

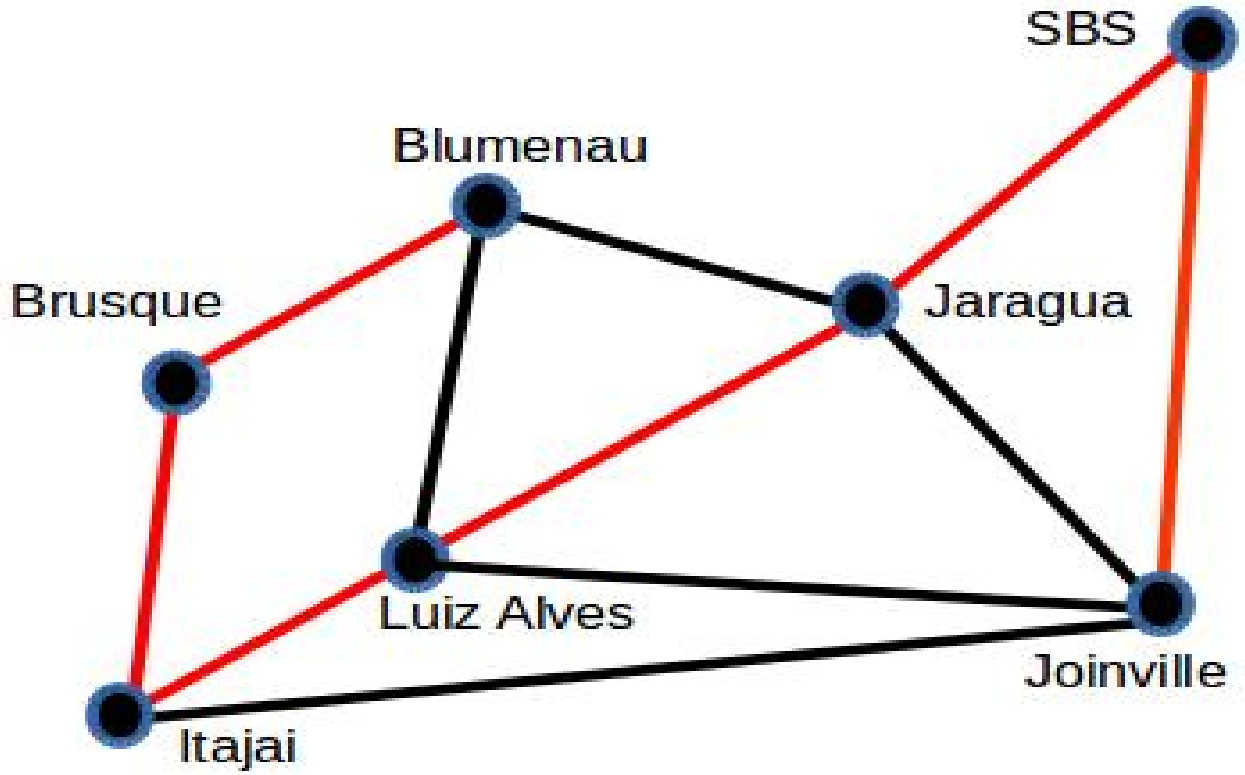
- Acompanhar as explicações do código de: `https://github.com/claudiosa/CCS/blob/master/picat/hamiltoniano_DFS.pi`
- Muitos elementos da linguagem neste código
- Confira a execução

Saída

```
$ picat hamiltoniano_DFS.pi
.............................
[blumenau,brusque,itajai,luiz_alves,jaragua,sao_bento,joinville]
 UMA SOLUCAO ....: OK
 ==>joinville : sao_bento : jaragua : luiz_alves : itajai : brusque
 : blumenau :
 Cidade Inicial: joinville
 Cidade Final: blumenau
 Total de cidades visitadas: 7
 CPU time 0.000000 em SEGUNDOS
 OVERALL PICAT CPU time 0.013000 em SEGUNDOS
 Backtrackings total 0
 ========================================
 [ccs@gerzat picat]$
```

Existe uma função chamada **findall**, cujo valor de retorno é uma lista com todas soluções possíveis de um dado predicado!

Uma Solução



Concluindo Buscas

- A recursividade na modelagem das buscas, define **uma metodologia** de se *programar em lógica* e resolver problemas

Concluindo Buscas

- A recursividade na modelagem das buscas, define **uma metodologia** de se *programar em lógica* e resolver problemas
- A área de buscas é ampla e apresenta muitas variações. Apresentamos nesta seção um **núcleo mágico**, que pode ser utilizado em muitas outras estrategias–métodos de buscas.

Concluindo Buscas

- A recursividade na modelagem das buscas, define **uma metodologia** de se *programar em lógica* e resolver problemas
- A área de buscas é ampla e apresenta muitas variações. Apresentamos nesta seção um **núcleo mágico**, que pode ser utilizado em muitas outras estrategias–métodos de buscas.
- Praticamente todos os métodos de buscas fazem o uso extensivo das **listas** em problemas complexos

Concluindo Buscas

- A recursividade na modelagem das buscas, define **uma metodologia** de se *programar em lógica* e resolver problemas
- A área de buscas é ampla e apresenta muitas variações. Apresentamos nesta seção um **núcleo mágico**, que pode ser utilizado em muitas outras estrategias–métodos de buscas.
- Praticamente todos os métodos de buscas fazem o uso extensivo das **listas** em problemas complexos
- Aos problemas complexos, há outras técnicas de programação para resolvê-los.

Concluindo Buscas

- A recursividade na modelagem das buscas, define **uma metodologia** de se *programar em lógica* e resolver problemas
- A área de buscas é ampla e apresenta muitas variações. Apresentamos nesta seção um **núcleo mágico**, que pode ser utilizado em muitas outras estrategias–métodos de buscas.
- Praticamente todos os métodos de buscas fazem o uso extensivo das **listas** em problemas complexos
- Aos problemas complexos, há outras técnicas de programação para resolvê-los.
- Assunto das próximas seções: PD, Planejamento e CP