PICAT: Uma Linguagem de Programação Multiparadigma

Miguel Alfredo Nunes, Jeferson L. R. Souza, Claudio Cesar de Sá

miguel.nunes@edu.udesc.br

jeferson.souza@udesc.br

claudio.sa@udesc.br

Departamento de Ciência da Computação
Centro de Ciências e Tecnológias
Universidade do Estado de Santa Catarina

8 de abril de 2019

Contribuições

- Alexandre Gonçalves;
- João Herique Faes Battisti;
- Paulo Victor de Aguiar;
- Rogério Eduardo da Silva;
- Hakan Kjellerstrand – (`http://www.hakank.org/picat/`)
- Neng-Fa Zhou – (`http://www.picat-lang.org/`)
- Outros anônimos que auxiliaram na produção deste documento;

Sumário I

Buscas

- Requisito: conceitos de listas e recursividade dominados!
-
- Essencialmente vamos varrer uma estrutura de estados ou nós, de modo sistemático até encontrarmos uma solução aceitável.
- Em geral os problemas se apresentam como uma conexão complexa tipo um *grafo*, e a varredura sob este grafo é sistemática no formato sob uma *árvore de busca*
- Computar sob listas é o esquema aqui utilizado

Ciclo Euleriano I

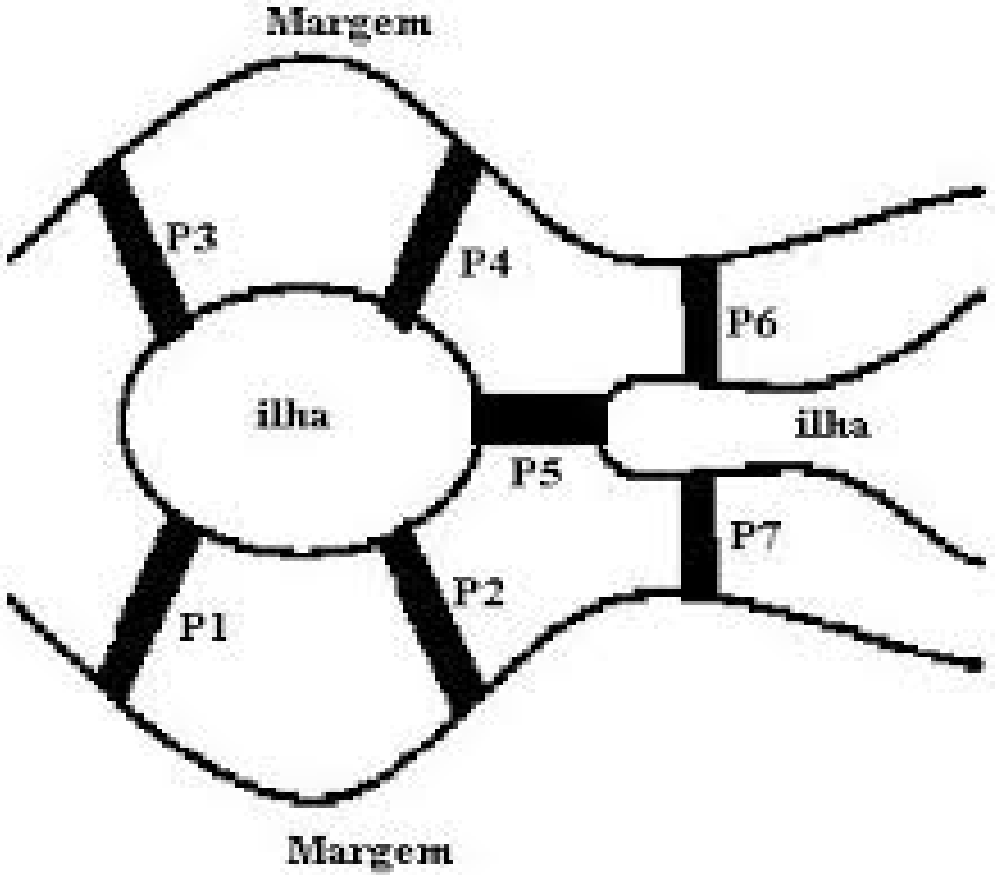

Figura 1: Ciclo Euleriano – Problema das Pontes de Königsberg

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, computacionalmente é fácil de resolver!

Caminho Hamiltoniano I

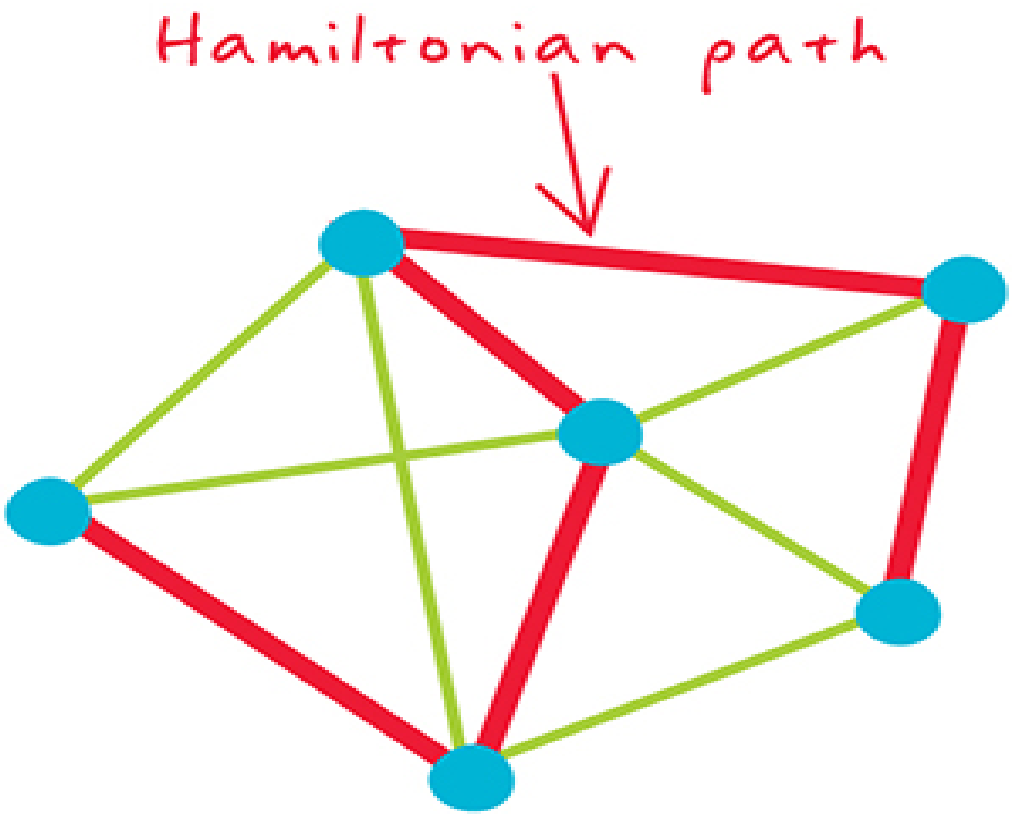

Figura 2: Caminho Hamiltoniano – Há um caminho que passe por todas cidades uma única vez?

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!
- Mas é este que vamos usar como exemplo, com um algoritmo bem ingênuo.

Problemas, Estados, Grafos e Árvores de Buscas

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*, e algum ou vários outros, os estados *finais*
- Se o problema tiver várias soluções, o mesmo apresenta vários caminhos do estado inicial há vários finais.
- Assim uma sucessão ou transição válida entre estados, é conhecido como uma solução ou instância do problema
- Logo, vamos empregar alguns conceitos de grafos, em modelar

Problemas de Grafos se Transformam em Árvores de Buscas

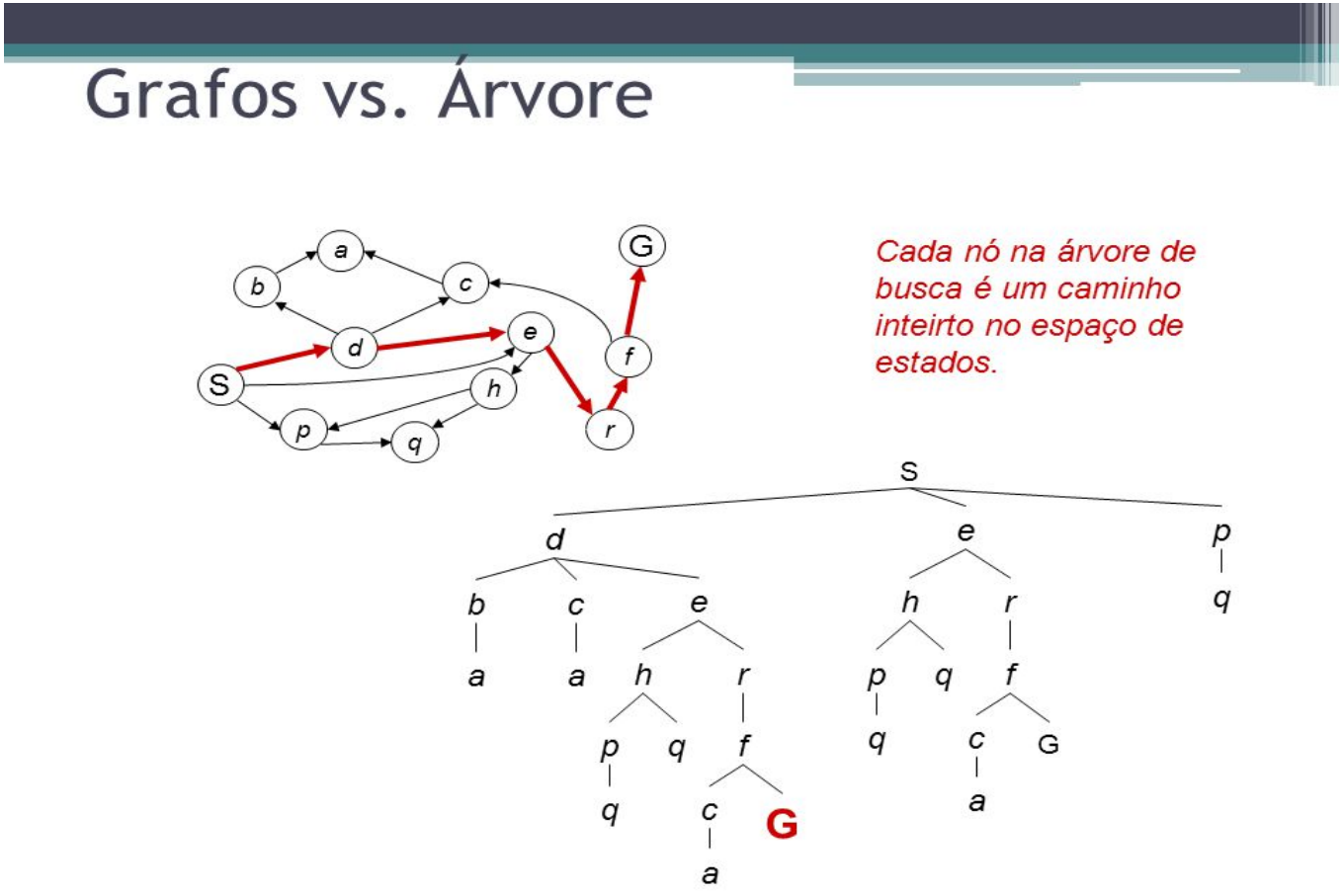

Figura 3: Refazer esta figura

Resumindo, os problemas são modelados em estruturas complexas, tais como grafos, mas o processo de solução se mantém: realizar

Núcleo Geral de Buscas I

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
```

Núcleo Geral de Buscas II

```
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

Resumindo

- Picat é jovem (nascida em 2013);
- Uma evolução ao Prolog após seus mais de 40 anos de existência e sucesso!
- Sua sintaxe é moderna;
- Código aberto, multi-plataforma, e repleta de possibilidades;
- Uso para fins diversos;
- Muitas bibliotecas específicas prontas: CP, SAT, Planner, etc;
-
- .