Solução de Problemas e Busca

Agentes de Resolução de Problemas

- Forma restrita de agente geral:

```
function AGENTE-SIMPLES-RESOLUCAO-PROBLEMAS(p) returns acao
  inputs: p, percepcao
  static: s, uma sequencia de acoes, comeca vazio
          estado, alguma descricao do estado do mundo
          g, objetivo, comeca vazio
          problem, a formulacao do problema
  estado ← ATUALIZA-ESTADO(estado, p)
  se s esta vazio entao
    g ← FORMULA-OBJETIVO(estado)
    problema ← FORMULA-PROBLEMA(estado, g)
    s ← BUSCA(problema)
  acao ← RECOMMENDACAO(s, estado)
  s ← RESTO(s, estado)
return acao
```

Isto é em solução de problemas offline.
A solução de problemas online envolve agir sem conhecimento do problema e da solução.

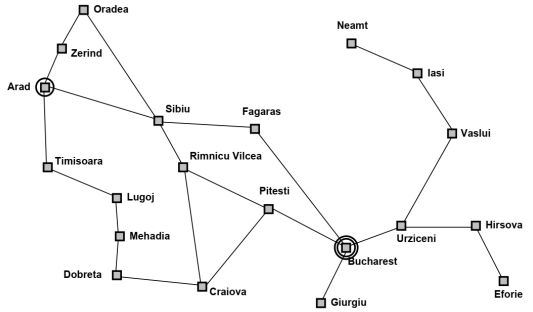
Exemplo: Romênia

De férias na Romênia; atualmente em Arad.
O voo parte amanhã de Bucareste

Formular meta:
estar em Bucareste

Formular o problema:
estados: várias cidades
operadores: dirigir entre cidades

Encontrar solução:
sequência de cidades, por exemplo, Arad, Sibiu, Fagaras, Bucareste



Tipos de Problemas

Determinístico, acessível ⇨ *problema do estado único*
Determinístico, inacessível ⇨ *problema dos múltiplos estados*

Não determinístico, inacessível ⇨ *problema de contingência*
devem usar sensores durante a execução
solução é uma *árvore* ou **política** em
muitas vezes se **entrelaça** busca, execução

Espaço de estado não reconhecido ⇨ *problemas de exploração* ("online")

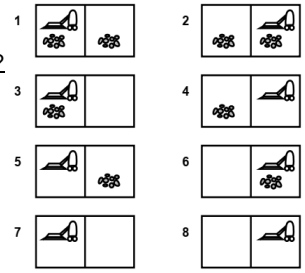
Exemplo: Mundo do Aspirador

Estado único, começa em #5. Solução?

Estado múltiplo, começa em {1, 2, 3, 4, 5, 6, 7, 8}
por exemplo.; *Direita* vai para {2, 4, 6, 8}. Solução?

Contingência, começa em #5
Lei de Murphy: *Sugar* (ação)
pode sujar seu carpete limpo

Sensoriamento local: sujeira, apenas localização.
Solução??



Formulação de problemas de Estado Único

- Um *problema* é definido por quatro itens:
  estado inicial por exemplo "em Arad"
  operadores (ou função sucessora S(x))
  por exemplo.,
  Arad => Zerind, Arad => Sibiu, etc...
  teste de objetivo, pode ser
  explicito por exemplo., x= "Em Bucareste"
  implícito por exemplo., *SemSujeira(x)*
  custo do caminho (aditivo)
  por exemplo., soma de distâncias, número de operadores executados, etc.

Uma *solução* é a sequência de operadores levando dos estados iniciais a um estado de objetivo

Selecione um espaço de estados

O mundo real é absurdamente complexo, o espaço de estado deve ser *abstraído* para a solução de problemas

(Abstração) estado = conjunto de estados reais

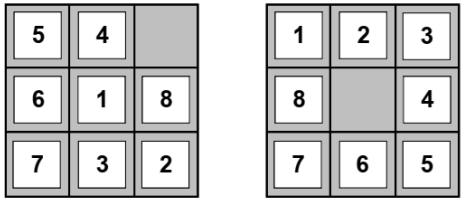
(Abstração) operador = combinação complexa de ações reais, por exemplo, "Arad => Zerind" representa um conjunto complexo de rotas possíveis, desvios, paradas de descanso, etc.

Para uma realização garantida, qualquer estado real "em Arad" deve chegar a *algum* estado real "em Zerind".

(Abstração) solução = conjunto de caminhos reais que são soluções em três mundos reais

Cada ação abstrata deve ser "mais fácil" do que o problema original!

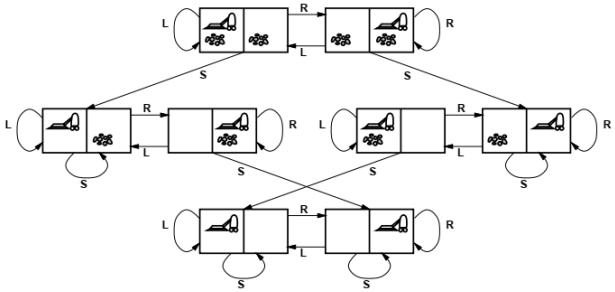
Exemplo: O enigma de 8



estados?? localizações inteiras das telhas(ignorar posições intermediárias)
operadores?? mover-se para a esquerda, direita, para cima, para baixo (ignorar os limites, etc.)
teste de objetivo?? Estado objetivo (dado)
custo do caminho?? 1 por movimento

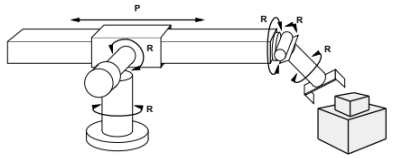
[Nota: a solução ótima da família *n*- Enigma é NP-difícil]

Exemplo: Gráfico dos estados do mundo do aspirador



estados?? sujeira inteira e localização de robôs (ignorar *quantidades* de sujeira)
operadores?? *Direita, Esquerda, Sugar*
teste de objetivo?? sem sujeira
custo do caminho?? 1 por operação

Exemplo: Montagem de robô



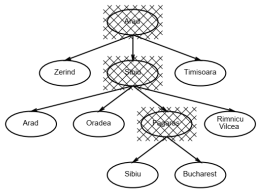
estados?? coordenadas de valor real do
robô unem ângulos partes do objeto a ser montado
operadores?? movimentos contínuos de juntas robotizadas
teste de objetivo?? montagem completa, *sem robô incluído!*
custo do caminho?? Tempo para executar

Implementação do Algoritmo de Busca Geral

```
function BUSCA-GERAL( estrategia, problema) returns solucao, ou falha
   inicializa a busca da arvore usando o estado inicial do problema
   laco faca
      se nao tem candidatos para expansao entao return falha
      escolha um no folha para expensao de acordo com a estrategia
      se o no tem um estado de objetivo entao
         return a solucao correspondente
      senao expanda o no e adiciona os nos resultantes na arvore de busca
end
```

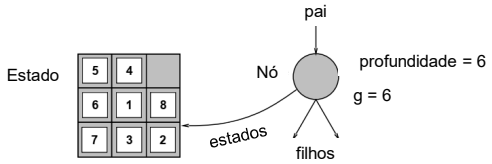
```
function BUSCA-GERAL( problema, FN-FILA) returns a solucao, ou falha
   nos ← CRIAR-FILA(CRIAR-NO(ESTADO-INICIAL[problema]))
   loop do
      se nos esta vazio entao return falha
         node ← REMOVER-FRENTE(nos)
      se TESTE-OBJETIVO[problema] aplicado ao ESTADO(no) esta certo
         entao return node
      nos ← FN-FILA(nos, EXPANDE(no, OPERADORES[problema]))
end
```

exploração simulada e offline do espaço dos estados gerando sucessores de estados já explorados (também conhecido como estados *em expansão*)



Implementação cont.: Estados vs. Nós

Um *estado* é uma (representação de)uma configuração física.
Um *nó* é uma estrutura de dados que constitui parte de uma árvore de busca que inclui *pais, filhos, profundidade, custo g(x) caminho.*
Os estados não têm pais, filhos, profundidade, ou custo do caminho!



A função EXPANDIR cria novos nós, preenchendo os diversos campos e utilizando os OPERADORES (ou SUCESSORFn) do problema para criar os estados correspondentes.

Estratégias de Busca

Uma estratégia é definida pela escolha da *ordem de expansão do nó*

As estratégias são avaliadas nas seguintes dimensões:
- completude - sempre encontra uma solução se ela existe?
- complexidade de tempo - número de nós gerados/expandidos
- complexidade de espaço - número máximo de nós na memória
- otimalidade - sempre encontra uma solução de menor custo?

A complexidade de tempo e espaço é medida em termos de
- b--fator máximo de ramificação da árvore de busca
- d--profundidade da solução de menor custo
- m--profundidade máxima do espaço de estado (pode ser ∞)

Estratégias de Busca Desinformada

Estratégias *desinformadas* utilizam apenas as informações disponíveis na definição do problema

Busca em Largura
Busca de custo uniforme
Busca em profundidade
Pesquisa de profundidade limitada
Busca intencional de aprofundamento

Busca em Profundidade Iterativa

Busca em Profundidade limitada = igual a busca em profundidade mas com uma profundidade limitada / Implementação: Os nós em profundidade l não têm sucessores

```
function BUSCA-APROFUNDAMENTO-ITERATIVO( problema) returns
   uma sequencia de solucoes
   inputs: problema, um problema
  para profundidade ← 0 ate ∞ faca
      resultado ← BUSCA-PROFUNDIDADE-LIMITADA( problema, profundidade)
      se resultado ≠ corte entao return resultado
end
```