Universidade Federal do Paraná
Departamento de Informática

Algoritmos de Busca Heurística (Parte 1)

Alexandre I. Direne
E-mail: alexd@inf.ufpr.br
Web: http://www.inf.ufpr.br/~alexd

BIBLIOGRAFIA RECOMENDADA

1- Artificial Intelligence: A Modern Approach. Stuart Russell e Peter Norvig. Second Edition, Prentice Hall, 2003.

2- Programming in Prolog. William F. Clocksin and C.S. Mellish. Springer-Verlag, 1987.

3- Guilherme Bittencourt. Inteligência Artificial: Ferramentas e Teorias. Terceira Edição, Editora da UFSC, 2006 (ISBN: 85-328-0138-2).

4- Elaine Rich e Kevin Knight, Artificial Intelligence, Second Edition, McGraw Hill, 1993.

5- Patrick H. Winston, Artificial Intelligence, Second Edition, Addison-Wesley, 1993.

PÁGINAS RECOMENDADAS

http://www.cs.dartmouth.edu/~brd/Teaching/AI/Lectures/Summaries/search.html

http://www.decom.ufop.br/prof/guarda/CIC250/index.htm

http://aima.cs.berkeley.edu/

http://aima.cs.berkeley.edu/newchap05.pdf

SOFTWARE RECOMENDADOS

http://www.cs.bham.ac.uk/research/poplog/freepoplog.html

http://www.swi-prolog.org

Algoritmos de Busca

Características:

1. Algoritmos de Busca são técnicas de Inteligência Artificial aplicadas a problemas de alta complexidade teórica que não são resolvidos com técnicas de programação convencionais, principalmente as de natureza puramente numérica;
2. A "complexidade" de um problema está diretamente relacionada ao tamanho do seu "Espaço de Busca" correspondente.

Hipótese Simplificadoras (Redução de Problemas do Mundo Real):

1. O conhecimento do domínio específico pode ser representado em Estados de Busca, formalmente definíveis por meio de variáveis de memória;
2. O processo de solução de um problema pode ser reduzido a um Algoritmo de Busca Heurística, cujo Espaço de Busca é formado por transformações sucessivas de Estados em uma certa ordem de geração e percurso.

Conseqüências:

1. Redução da explosão combinatória de possibilidades de Busca;
2. O trabalho humano se restringe à atuação empírica de identificar e formalizar: (a) representações de estados; (b) parâmetros Heurísticos; (c) operações de transformações atômica; (d) combinadores de transformações que atinjam a solução com tempos e tamanhos de memória aceitáveis.

Exemplos de Sub-Problemas, Espaços e Problemas e Algoritmos de Busca

1. Definição precisa do sub-problema;
2. Análise do problema;
3. Isolamento e representação do conhecimento de um Estado de Busca;
4. Escolha das técnicas "apropriadas" de Busca Heurística.

Exemplo: Um programa para o jogo de Xadrez entre humanos e maquinas.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)
(1)	X1	Y1	Z1	RE1	RA1	Z1	Y1	X1
(2)	o1	o1	o1	o1	o1	o1	o1	o1
(3)		?						
(4)								
(5)								
(6)								
(7)	o2	o2	o2	o2	o2	o2	o2	o2
(8)	X2	Y2	Z2	RE2	RA2	Z2	Y2	X2

Elementos Envolvidos no Jogo de Xadrez:

1. Estado Inicial: (X1,a,1) **V** (Y1,b,1) **V** ... **V** (vazia,d,4) **V** ... **V** (X2,h,8)
2. Estado(s) Final(is) = Estado Meta = Estado Solução: Regras (operações) e/ou Fatos (variáveis) que definem todas as condições possíveis de Solução/Vitória.
3. Espaço de Busca (ou Espaço de Solução de Sub-Problema): Grafos que representam a plicação sucessiva e cumulativa de operações atômicas sobre o Estado Inicial, até incluir o Estado Final em seu conjunto de nodos. Por exemplo, se em um sub-problema sempre são aplicáveis duas operações em 20 movimentos, temos:

   $2 \times 2 \times 2 \times \ldots \times 2 = 2^{20}$ *= mais de 1 milhão.*

   Em Xadrez, existem aproximadamente 10^{120} posições possíveis no tabuleiro !!!

4. Regras (Operações) lícitas de transformação atômica de um estado para outro.

Exemplo de Regra ou Operação de transformação atômica:

REGRA k: SE (Peao_Branco, b,2) **&**

(vazia,b,3) **&**

(vazia,b,4)

ENTÃO

MOVER(Peao_Branco,b,4)

FIM-REGRA

5. Função Heurística: de redução da Explosão Combinatória do Espaço de Busca: Um sub-Espaço de Busca "relevante" e "processável" para a estado corrente.

O Domínio dos Recipientes de Água

São dados 2 Recipientes:

- Recipiente-1 (capacidade 4 litros);
- Recipiente-2 (capacidade 3 litros);
- Os recipientes não tem marcas de medidas.

Problema: Colocar exatamente 2 litros no recipiente-1.

Elementos formais envolvidos:

1. Representação de um Estado de Busca qualquer: par ordenado de inteiros não negativos.
2. Estado de Final: (2 , QUALQUER);
3. Espaço de Busca: Espaço Cartesiano composto pelo conjunto de pares ordenados de inteiros (x,y) tal que x pertence a {0, 1, 2, 3, 4} e y pertence a {0, 1, 2, 3};
4. Estado Inicial: (0 , 0);

Resumo dos passos de uma solução do sub-problema em foco:

Recip - 1 (4 Litros)	Recip - 2 (3 Litros)	Regra
0	0	2
0	3	9
3	0	2
3	3	7
4	2	5
0	2	$^{9}/_{11}$
2	0	-

Conjunto possível de regras (operações) de transformação atômica:

Regra	Estado Inicial	Condicional de Regra	Estado Final
1	(x,y)	x < 4	(4,y)
2	(x,y)	y < 3	(x,3)
3	(x,y)	x > 0	(x-d,y)
4	(x,y)	y > 0	(x,y-d)
5	(x,y)	x > 0	(0,y)
6	(x,y)	y > 0	(x,0)
7	(x,y)	x+y ≥ 4 & y > 0	(4,y-(4-x))
8	(x,y)	x+y ≥ 3 & x > 0	(x-(3-y),3))
9	(x,y)	x+y ≤ 4 & y > 0	(x+y,0)
10	(x,y)	x+y ≤ 3 & x > 0	(0,x+y)
11	(0,2)	<VERDADE>	(2,0)
12	(2,y)	<VERDADE>	(0,y)

O Donínio do Jogo da Velha

SOLUÇÃO 1 :

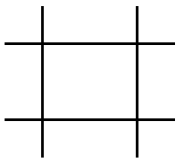
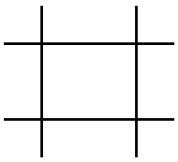
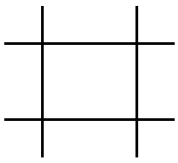
1	2	3
4	5	6
7	8	9

→

1	2	3	4	5	6	7	8	9

onde: 0 → vazio
1 → x
2 → 0

Estado de Busca = *Espaço de Busca: Todos os tabuleiros possíveis* = 3^9 = *19.683*

1 2 ... 19.683

Algoritmo de Busca:

1. Visualizar o tabuleiro corrente em BASE-3 (valores 0,1,2) e converter para BASE-10;
2. Usar o número calculado como índice de entrada no Espaço de Busca;
3. O tabuleiro selecionado representa a próxima jogada plausível.

Comentários: É muito eficiente, porém há várias desvantagens:

1. Muita memória para armazenar as combinações de tabuleiros;
2. Alguém deve despender enormes esforços manuais para ORGANIZAR o espaço de tabuleiros;
3. Espaço de tabuleiros pode conter ERROS de criação;
4. Se ampliarmos as dimensões do tabuleiro, o algoritmo não funciona.

SOLUÇÃO 2 :

1	2	3
4	5	6
7	8	9

$\rightarrow$

1	2	3	4	5	6	7	8	9

onde: v $\rightarrow$ vazio
x $\rightarrow$ X
o $\rightarrow$ O

Estado de Busca: Apenas 1 (um) tabuleiro !

I =

1	2	3	4	5	6	7	8	9

Algoritmo de Busca:

1. Retorna o número do quadrado vencedor se o jogador atual tiver condições de ganhar. Caso contrário, retorna o número do quadrado para o movimento vitorioso do oponente se esse tiver a chance de ganhar no próximo movimento. Caso contrário, retorna 5 se o quadrado central estiver em branco. Caso contrário, retorna qualquer quadrado em branco que não seja de canto;
2. Efetua movimento no quadrado N (parâmetro de retorno), ajustando posição para "x" (X) se a jogada for impar e para "o" (O) se a jogada for par.

Comentários: Não é tão eficiente como o primeiro mas tem vantagens:

1. Requer pouco espaço;
2. Fácil de entender a estratégia.

Desvantagens:

1. Parece apenas se defender pois não usa nenhuma ferramenta de nível tático para gerir memória avançada;
2. Também não generalizável para 3 dimensões.

SOLUÇÃO 3 :

Estado de Busca:

1. *Apenas 1 (um) tabuleiro.*

1 2	3	4	5	6	7	8	9

2. Número representando uma estimativa (heurística) que o tabuleiro tem de levar à vitoria.

Algoritmo de Busca (MINIMAX):

1. Verifique próximas jogadas diretamente atingíveis a partir do tabuleiro corrente se a altura máxima de busca não tiver sido alcançada, caso contrário, retorne o a estimativa (heurítica) do tabuleiro corrente;
2. Caso uma se trate de posição de vitória, dê a ela a mais alta estimativa possível e retorne este valor;
3. Caso contrário, considere todos os movimentos que o oponente possa fazer em seguida. Assuma que o oponente fará a pior jogada contra a máquina. Ative recursivamente a expansão de estados;
4. A próxima jogada plausível é o do tabuleiro com a mais alta estimativa.

É ineficiente do ponto de vista de tempo de pesquisa pois cria sub-árvores de jogadas também para o oponente como forma de planejamento, mas tem vantagens:

1. É mais genérico que os outros;
2. Pode ser usado até para outros jogos, o que não seria possível com os outros dois algoritmos vistos.

Principais Algoritmos de Busca Heurística:

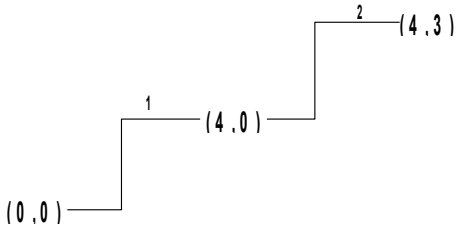
* Busca em **Grafos OU** (Gerar e Testar, Subida de Encosta, Melhor Escolha, Satisfação de Restrições, Análise Intermediária, outros);
* Busca em **Grafos E-OU** (MINIMAX por Corte Alfa-Beta, MINIMAX SSS*, outros que se prestam à chamada Busca Adversarista).

Busca em Profundidade

Descrição do algoritmo:

- Se estado ATUAL é o estado solução (ESTADO-META) então retorne sucesso;
- Caso contrário, repetir até sucesso (ou "fracasso") :
    - Gere estado E como SUCESSOR do ATUAL;
    - Chamar Recursivamente este procedimento utilizando E como se fosse ATUAL;
    - Se houver sucesso, indicar para retorno Senão, continuar a repetição controlada por (2).

Aplicação no exemplo dos recipientes de água:

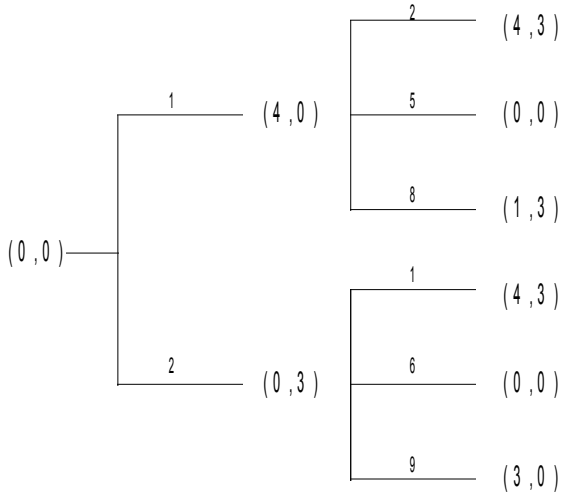


Busca em Amplitude

Descrição do algoritmo:

- Criar uma lista de nodos (LISTA-NODOS) com estado INICIAL;
- Até ser encontrado o estado solução (ESTADO-META), repetir:
    - E = Primeiro elemento de LISTA-NODOS;
    - LISTA-NODO perde seu primeiro elemento;
    - Repetir para cada REGRA de pré-condição verdadeira quando aplicada a E:
        - Gerar novo estado;
        - Se estado gerado é ESTADO-META retorne sucesso;
        - Caso contrário, acrescente o estado gerado no final de LISTA-NODO.

Aplicação no exemplo dos recipientes de água:

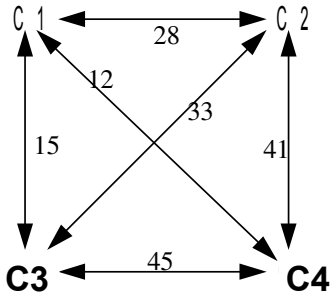


Comparação entre Busca em Profundidade x Amplitude

1. A busca em profundidade requer menos memória pois apenas o estado corrente é armazenado;
2. Se houver qualquer grau de ordenação de estados traduzido na ordem das regras de transformação atômica, a busca em profundidade pode encontrar a solução sem examinar todo o espaço de busca. Na busca em amplitude, nenhum estado do nível **n+1** da árvore de busca é visitado antes que todos do nível **n** o sejam;
3. A busca em profundidade pode seguir caminhos infrutíferos durante longo tempo, ou mesmo por um tempo infinito, o que não ocorre com a busca em aplitude;
4. Se houver mais de uma solução, então a uma solução *ótima* será encontrada pela busca em amplitude (busca em amplitude encontra soluções mais curtas antes das mais longas).

Mais Problemas Clássicos

O problema do Caixeiro Viajante (cidades C1, C2, ... , Cn):



Para 4 Cidades temos 3! = 6 Combinações:

C1	C2	C3	C4
C1	C2	C4	C3
C1	C3	C2	C4

… …

Para 11 cidades temos 10! (mais de 3.000.000)
Para 25 cidades, só com HEURISTICA

O problema do Quadrado Mágico:

6	7	2	Compor 3 Linhas, 3 Colunas e 2 Diagonais;
			Com dígitos de 1 a 9;
1	5	9	Cuja soma sempre resulta em 15.
			(3 x 3)! = 9! = 362.880 cofigurações diferentes
8	3	4	

O problema dos Ladrilhos Deslizantes.

6	8	7	– Análogo a um problema de ordenação
3	5	1	– Existem 40.320 configurações diferentes
4	2	-	– Aplicamos alguma estratégia –

Dependendo da configuração do estado atual (corrente corrente) :

6	8	7
3	5	1
4	2	-

Nesta configuração, apenas 2 (dois) movimentos são são possíveis: deslizar o ladrilho 2 para a direita **OU** o 1 para baixo.

6	8	7
3	5	1
4	-	2

6	8	7
3	5	-
4	2	1

Depois de ter deslizado o -2- para a direita, 3 (três) movimentos adicionais são possíveis: deslizar o 2 **DE VOLTA** para a esquerda, deslizar o -4- para a direita ou deslizar o -5- para baixo.

6	8	7
3	5	1
4	2	-

6	8	7
3	5	1
-	4	2

6	8	7
3	-	1
4	5	2

A META é encontrar "UMA" seqüência de movimentos que nos permita atingir a configuração seguinte:

1	2	3
4	5	6
7	8	-

Porém, alguns domínios de problemas não são facilmente redutíveis. Exemplos:

* Calcular: 200 ! - 200 ! ?
* Existe um número primo MAIOR que 100.000.000 ?
* O incêndio na ilha.
* A partilha de um terreno.
* *Vida ou morte.*

Um Domínio com Problemas cujas buscas podem não ter solução

O problema das Torres de altura pré-determinada utilizando blocos de tamanhos variados.

É dado um conjunto de números e o PROBLEMA é achar um (ou mais) subconjuntos cuja cardinalidade seja igual a um dado número de referencia.

Conjunto	Referencia
{1 2 3 4}	8
{1 4 7 9}	3
{2 2 2 2 2 2}	9
{6 5 7 4}	9
{1 3 5 2 8}	18

Além da busca poder ser Heurística, alguns fatores de natureza Epistemológica podem ser aplicados ao algoritmo base da busca de maneira a tentar identificar impossibilidade de solução. Em qualquer passo na seqüência de solução de um problema temos:

- C (conjunto de números Correntes)
- D (conjunto de números Disponíveis)

} UM ESTADO.

Achar uma seqüência de estados que leve a um próximo estado válido, partindo do seguinte estado inicial:

C: Conjunto Vazio

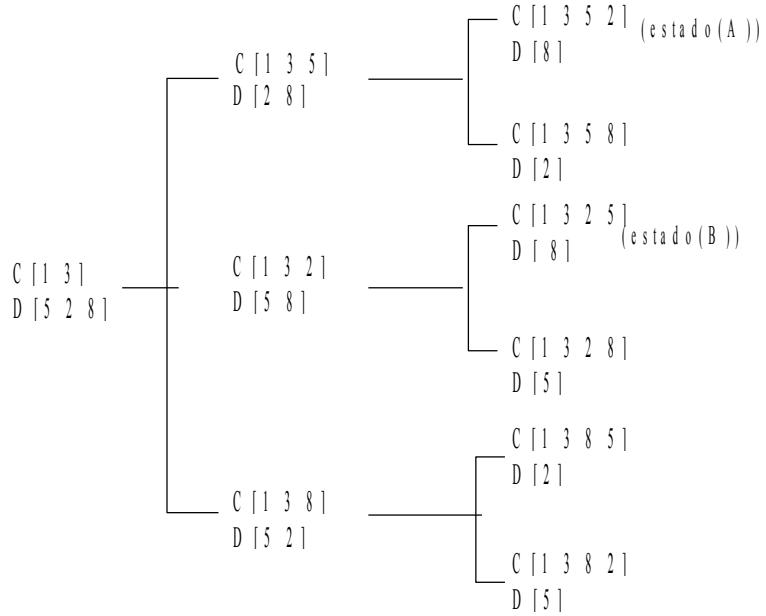
D: Conjunto Dado

A partir dele, a busca deve levar a um estado final:

C: Conjunto cuja soma de elementos é igual ao Número de Referencia

D: Um conjunto qualquer

Diagrama com 2 passos do processo de solução, a partir do seguinte ESTADO:



Algoritmo Geral de Busca Heurística (Combinação+Ordenação)

Funções Necessárias/Típicas:

Função-1: eh_estado_final(E) **RETORNA** <verdadeiro>/<falso>

Função-2: adjacentes(E) **RETORNA** [E1 , E2 , ... , En]

Função-3: adjacente(E) **RETORNA** RETROATIVAMENTE Ei [E1, E2 , ... , En]

Função-4: heuristica(E) **RETORNA** <valor numérico representando estimativa de proximidade do Ef>

*Definição da função **busca**:*

```
FUNÇÃO busca(E_i);
        vars E_c, E_d, Caminho, Caminho_Estendido, Lista_Prior;
        E_c = E_i;
        Caminho = [ ];
        Lista_Prior = [ ];
        ENQUANTO não  eh_estado_final(E_c) faça
                PARA_CADA  E_d  em  adjacentes(E_c) faça
                        SE não pertence_a(E_d, Caminho) ENTÃO
                                Caminho_Estendido = [ ^^Caminho   ^E_c   ^E_d ];
                                Lista_Prior = insere(Caminho_Estendido, Lista_Prior);
                        FIM-SE;
                FIM-PARA_CADA;
                Lista_Prior   UNIFICADA_COM   [  [ ??Caminho  ?E_c ]    ??Lista_Prior  ];
        FIM-ENQUANTO;
        RETORNA(  [ ^^Caminho  ^Ec]  );
FIM-FUNÇÃO;
```

*Definição da função **insere**:*

```
        FUNÇÃO insere(Caminho, Lista_Caminhos);
                vars Cabeça, Cauda;
                SE  Lista_Caminhos  eh_UNIFICÁVEL_COM   [ ?Cabeça ??Cauda ]  ENTÃO
                        SE  melhor(Caminho, Cabeça)  então
                                Nova_Lista = [ ^Caminho   ^Cabeça   ^^Cauda ];
                        SENÃO
                                Cauda = insere(Caminho, Cauda);
                                Nova_Lista = [^Cabeça ^^Cauda];
                        FIM-SE
                Senão
                        Nova_Lista = [ ^Caminho ];
                FIM-Se
                RETORNA( Nova_Lista );
        FIM-FUNÇÃO;
```

Observação: O procedimento "adjacentes" retorna uma lista de estados imediatamente atingíveis a partir do estado argumento, de acordo com as operações atômicas aplicáveis a ele (estado argumento). Note que um ou mais dos estados produzidos já podem existir no Caminho em expansão

No problema clássico de ordenação em vetores de 4 números, 3 alterações atômicas sempre se aplicam. Por exemplo:

adjacentes([3 4 2 1]) **RETORNA** [[3 4 1 2] [3 2 4 1] [4 3 2 1]]

Busca em Profundidade

search([3 4 2 1]);

[Considering state number 1 Estimate for this state is 5 Number of untried paths is 0 History and state is]

[[3 4 2 1]]

[Considering state number 2 Estimate for this state is 6 Number of untried paths is 2 History and state is]

[[3 4 2 1] [4 3 2 1]]

[Considering state number 3 Estimate for this state is 5 Number of untried paths is 3 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1]]

[Considering state number 4 Estimate for this state is 4 Number of untried paths is 4 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1]]

[Considering state number 5 Estimate for this state is 3 Number of untried paths is 5 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1]]

[Considering state number 6 Estimate for this state is 4 Number of untried paths is 6 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1] [3 2 4 1]]

[Considering state number 7 Estimate for this state is 3 Number of untried paths is 6 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1] [3 2 4 1] [3 2 1 4]]

[Considering state number 8 Estimate for this state is 2 Number of untried paths is 7 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1] [3 2 4 1] [3 2 1 4] [2 3 1 4]]

[Considering state number 9 Estimate for this state is 1 Number of untried paths is 7 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1] [3 2 4 1] [3 2 1 4] [2 3 1 4] [2 1 3 4]]

[Solution has been found after considering 9 states The length of the solution is 10 Maximum memory requirements were 9 Search strategy was depth Estimation method was perfect The solution itself is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1] [3 2 4 1] [3 2 1 4] [2 3 1 4] [2 1 3 4] [1 2 3 4]]

Busca em Amplitude

search([3 4 2 1]);

[Considering state number 1 Estimate for this state is 5 Number of untried paths is 0 History and state is]

[[3 4 2 1]]

[Considering state number 2 Estimate for this state is 4 Number of untried paths is 2 History and state is]

[[3 4 2 1] [3 4 1 2]]

[Considering state number 3 Estimate for this state is 4 Number of untried paths is 3 History and state is]

[[3 4 2 1] [3 2 4 1]]

[Considering state number 4 Estimate for this state is 6 Number of untried paths is 4 History and state is]

[[3 4 2 1] [4 3 2 1]]

[Considering state number 5 Estimate for this state is 3 Number of untried paths is 5 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2]]

[Considering state number 6 Estimate for this state is 5 Number of untried paths is 6 History and state is]

[[3 4 2 1] [3 4 1 2] [4 3 1 2]]

[Considering state number 7 Estimate for this state is 3 Number of untried paths is 7 History and state is]

[[3 4 2 1] [3 2 4 1] [3 2 1 4]]

[Considering state number 8 Estimate for this state is 3 Number of untried paths is 8 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1]]

[Considering state number 9 Estimate for this state is 5 Number of untried paths is 9 History and state is]

[[3 4 2 1] [4 3 2 1] [4 3 1 2]]

[Considering state number 10 Estimate for this state is 5 Number of untried paths is 10 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1]]

[Considering state number 11 Estimate for this state is 2 Number of untried paths is 11 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4]]

. . . OMITIDAS APROXIMADAMENTE 200 LINHAS DE TENTATIVAS . . .

[Considering state number 43 Estimate for this state is 3 Number of untried paths is 39 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 4 1 3]]

[Considering state number 44 Estimate for this state is 3 Number of untried paths is 40 History and state is]

[[3 4 2 1] [4 3 2 1] [4 2 3 1] [2 4 3 1] [2 3 4 1]]

[Considering state number 45 Estimate for this state is 4 Number of untried paths is 41 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4] [3 2 1 4] [3 2 4 1]]

[Considering state number 46 Estimate for this state is 2 Number of untried paths is 41 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4] [3 2 1 4] [2 3 1 4]]

[Considering state number 47 Estimate for this state is 2 Number of untried paths is 42 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4] [1 3 2 4] [1 3 4 2]]

[Solution has been found after considering 47 states The length of the solution is 6 Maximum memory requirements were 43 Search strategy was breadth Estimation method was perfect The solution itself is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4] [1 3 2 4] [1 2 3 4]]

Busca Gulosa (Greedy)

search([3 4 2 1]);

[Considering state number 1 Estimate for this state is 5 Number of untried paths is 0 History and state is]

[[3 4 2 1]]

[Considering state number 2 Estimate for this state is 4 Number of untried paths is 2 History and state is]

[[3 4 2 1] [3 2 4 1]]

[Considering state number 3 Estimate for this state is 3 Number of untried paths is 3 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1]]

[Considering state number 4 Estimate for this state is 2 Number of untried paths is 4 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4]]

[Considering state number 5 Estimate for this state is 1 Number of untried paths is 5 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4] [2 1 3 4]]

[Solution has been found after considering 5 states The length of the solution is 6 Maximum memory requirements were 7 Search strategy was best Estimation method was perfect The solution itself is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4] [2 1 3 4] [1 2 3 4]]

*Busca A**

search([3 4 2 1]);

[Considering state number 1 Estimate for this state is 5 Number of untried paths is 0 History and state is]

[[3 4 2 1]]

[Considering state number 2 Estimate for this state is 4 Number of untried paths is 2 History and state is]

[[3 4 2 1] [3 4 1 2]]

[Considering state number 3 Estimate for this state is 4 Number of untried paths is 3 History and state is]

[[3 4 2 1] [3 2 4 1]]

[Considering state number 4 Estimate for this state is 3 Number of untried paths is 4 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2]]

[Considering state number 5 Estimate for this state is 3 Number of untried paths is 5 History and state is]

[[3 4 2 1] [3 2 4 1] [3 2 1 4]]

[Considering state number 6 Estimate for this state is 3 Number of untried paths is 6 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1]]

[Considering state number 7 Estimate for this state is 2 Number of untried paths is 7 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4]]

[Considering state number 8 Estimate for this state is 2 Number of untried paths is 8 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [1 3 4 2]]

[Considering state number 9 Estimate for this state is 2 Number of untried paths is 9 History and state is]

[[3 4 2 1] [3 2 4 1] [3 2 1 4] [3 1 2 4]]

[Considering state number 10 Estimate for this state is 2 Number of untried paths is 10 History and state is]

[[3 4 2 1] [3 2 4 1] [3 2 1 4] [2 3 1 4]]

[Considering state number 11 Estimate for this state is 2 Number of untried paths is 11 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4]]

[Considering state number 12 Estimate for this state is 1 Number of untried paths is 12 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4] [1 3 2 4]]

[Considering state number 13 Estimate for this state is 1 Number of untried paths is 13 History and state is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [1 3 4 2] [1 3 2 4]]

[Considering state number 14 Estimate for this state is 1 Number of untried paths is 14 History and state is]

[[3 4 2 1] [3 2 4 1] [3 2 1 4] [3 1 2 4] [1 3 2 4]]

[Considering state number 15 Estimate for this state is 1 Number of untried paths is 15 History and state is]

[[3 4 2 1] [3 2 4 1] [3 2 1 4] [2 3 1 4] [2 1 3 4]]

[Considering state number 16 Estimate for this state is 1 Number of untried paths is 16 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4] [2 1 3 4]]

[Solution has been found after considering 16 states The length of the solution is 6 Maximum memory requirements were 18 Search strategy was astar Estimation method was perfect The solution itself is]

[[3 4 2 1] [3 4 1 2] [3 1 4 2] [3 1 2 4] [1 3 2 4] [1 2 3 4]]

Busca de Custo Uniforme

search([3 4 2 1]);

[Considering state number 1 Estimate for this state is 5 Number of untried paths is 0 History and state is]

[[3 4 2 1]]

[Considering state number 2 Estimate for this state is 4 Number of untried paths is 2 History and state is]

[[3 4 2 1] [3 2 4 1]]

[Considering state number 3 Estimate for this state is 3 Number of untried paths is 3 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1]]

[Considering state number 4 Estimate for this state is 2 Number of untried paths is 4 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4]]

[Considering state number 5 Estimate for this state is 1 Number of untried paths is 5 History and state is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4] [2 1 3 4]]

[Solution has been found after considering 5 states The length of the solution is 6 Maximum memory requirements were 7 Search strategy was hill Estimation method was perfect The solution itself is]

[[3 4 2 1] [3 2 4 1] [2 3 4 1] [2 3 1 4] [2 1 3 4] [1 2 3 4]]

O Mundo de Blocos

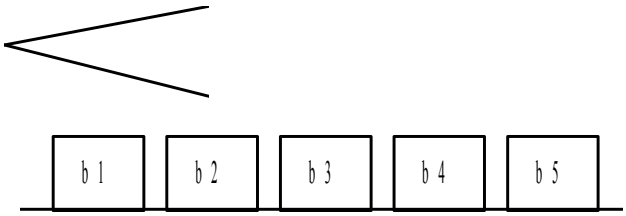
BASE DE REGRAS:

```
[[pegar ?X da mesa]
        [[braco_vazio] [topo_vazio ?X] [na_mesa ?X]]
        [[braco_vazio] [na_mesa ?X]]
        [[segurando ?X]]]

[[colocar ?X sobre mesa]
        [[segurando ?X]]
        [[segurando ?X]]
        [[na_mesa ?X] [braco_vazio]]]

[[tirar ?X de ?Y]
        [[braco_vazio] [?X sobre ?Y] [topo_vazio ?X]]
        [[braco_vazio] [?X sobre ?Y]]
        [[segurando ?X] [topo_vazio ?Y]]]

[[apoiar ?X sobre ?Y]
        [[segurando ?X] [topo_vazio ?Y]]
        [[segurando ?X] [topo_vazio ?Y]]
        [[braco_vazio] [?X sobre ?Y]]]
```

Cada regra é composta de:

1. NOME da AÇÃO da regra;
2. PRECONDIÇÕES da AÇÃO da regra (verificadas contra a BASE DE FATOS);
3. ASSERTIVAS ajustadas como FALSAS pela AÇÃO da regra (na BASE DE FATOS);
4. ASSERTIVAS ajustadas como VERDADE pela AÇÃO da regra (na BASE DE FATOS).

BASE DE FATOS:

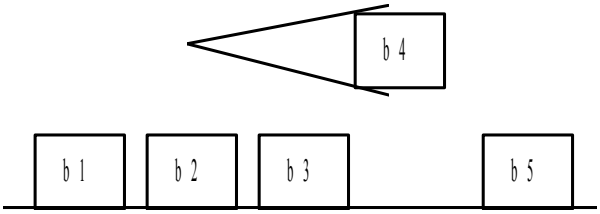



```
[  [na_mesa b1]
   [na_mesa b2]
   [na_mesa b3]
   [na_mesa b4]
   [na_mesa b5]
   [topo_vazio b1]
   [topo_vazio b2]
   [topo_vazio b3]
   [topo_vazio b4]
   [topo_vazio b5]
   [braco_vazio]     ]
```

Comparação A x Análise Intermediária*

BASE DE FATOS (ESTADO INICIAL): A mesma acima.

ESTADO META:



BASE DE FATOS DO ESTADO META:

[[na_mesa b1] [na_mesa b2] [na_mesa b3] [na_mesa b5] [topo_vazio b1]

[topo_vazio b2] [topo_vazio b3] [topo_vazio b4] [topo_vazio b5] [segurando b4]]

*Algoritmo A**

[SEGURANDO B4]
pegar b1 da mesa
pegar b2 da mesa
pegar b3 da mesa
PEGAR B4 DA MESA
pegar b5 da mesa

COMPLETE PLAN IS:
[pegar b4 da mesa]

Analise Intermediária

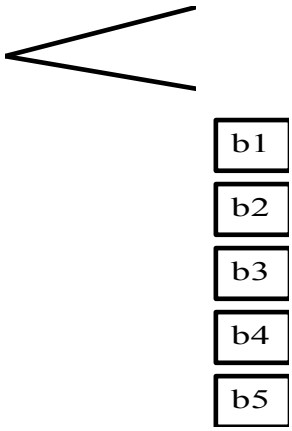
achieve [segurando b4]
reduce [segurando b4]
perform [pegar b4 da mesa]

COMPLETE PLAN IS:
[pegar b4 da mesa]

Em outra situação

BASE DE FATOS DO ESTADO INICIAL: A mesma anterior.

ESTADO META:



BASE DE FATOS DO ESTADO META:

[[b1 sobre b2] [b2 sobre b3] [b3 sobre b4] [b4 sobre b5] [na_mesa b5] [topo_vazio b1] [braco_vazio]]

Análise Intermediária

achieve [b1 sobre b2] [b2 sobre b3] [b3 sobre b4] [b4 sobre b5]
- reduce [b4 sobre b5]
  - achieve [segurando b4] [topo_vazio b5]
    - reduce [segurando b4]
      - perform [pegar b4 da mesa]
  - perform [apoiar b4 sobre b5]
- reduce [b3 sobre b4]
  - achieve [segurando b3] [topo_vazio b4]
    - reduce [segurando b3]
      - perform [pegar b3 da mesa]
  - perform [apoiar b3 sobre b4]
- reduce [b2 sobre b3]
  - achieve [segurando b2] [topo_vazio b3]
    - reduce [segurando b2]
      - perform [pegar b2 da mesa]
  - perform [apoiar b2 sobre b3]
- reduce [b1 sobre b2]
  - achieve [segurando b1] [topo_vazio b2]
    - reduce [segurando b1]
      - perform [pegar b1 da mesa]
  - perform [apoiar b1 sobre b2]

COMPLETE PLAN IS:
- [pegar b4 da mesa]
- [apoiar b4 sobre b5]
- [pegar b3 da mesa]
- [apoiar b3 sobre b4]
- [pegar b2 da mesa]
- [apoiar b2 sobre b3]
- [pegar b1 da mesa]
- [apoiar b1 sobre b2]

*Algoritmo A**

[B1 SOBRE B2] [B2 SOBRE B3] [B3 SOBRE B4] [B4 SOBRE B5]

pegar b1 da mesa
- apoiar b1 sobre b2
  - pegar b3 da mesa
  - pegar b4 da mesa
  - pegar b5 da mesa
- apoiar b1 sobre b3
- apoiar b1 sobre b4
- apoiar b1 sobre b5

pegar b2 da mesa
- apoiar b2 sobre b1
- apoiar b2 sobre b3
  - pegar b1 da mesa
  - pegar b4 da mesa
  - pegar b5 da mesa
- apoiar b2 sobre b4
- apoiar b2 sobre b5

pegar b3 da mesa
- apoiar b3 sobre b1
- apoiar b3 sobre b2
- apoiar b3 sobre b4
  - pegar b1 da mesa
  - pegar b2 da mesa
  - pegar b5 da mesa
- apoiar b3 sobre b5

PEGAR B4 DA MESA
- apoiar b4 sobre b1
- apoiar b4 sobre b2
- apoiar b4 sobre b3
- APOIAR B4 SOBRE B5
  - pegar b1 da mesa
    - apoiar b1 sobre b2
      - pegar b3 da mesa
      - tirar b4 de b5
    - apoiar b1 sobre b3
    - apoiar b1 sobre b4
  - pegar b2 da mesa
    - apoiar b2 sobre b1
    - apoiar b2 sobre b3
      - pegar b1 da mesa
      - tirar b4 de b5
    - apoiar b2 sobre b4
  - PEGAR B3 DA MESA
    - apoiar b3 sobre b1
    - apoiar b3 sobre b2
    - APOIAR B3 SOBRE B4
      - pegar b1 da mesa
        - apoiar b1 sobre b2
          - tirar b3 de b4
        - apoiar b1 sobre b3
      - PEGAR B2 DA MESA
        - apoiar b2 sobre b1
        - APOIAR B2 SOBRE B3
          - PEGAR B1 DA MESA
            - APOIAR B1 SOBRE B2
              - pegar b5 da mesa

COMPLETE PLAN IS:
- [pegar b4 da mesa]
- [apoiar b4 sobre b5]
- [pegar b3 da mesa]
- [apoiar b3 sobre b4]
- [pegar b2 da mesa]
- [apoiar b2 sobre b3]
- [pegar b1 da mesa]
- [apoiar b1 sobre b2]

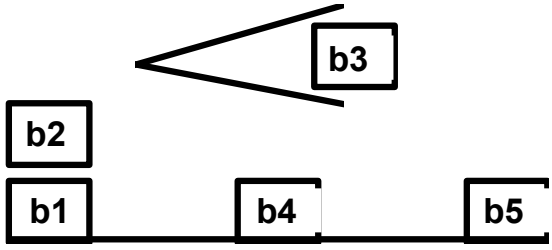
Em outra situação

BASE DE FATOS DO ESTADO INICIAL:

[[na_mesa b1] [b2 sobre b1] [topo_vazio b2] [segurando b3] [topo_vazio b3] [na_mesa b4] [topo_vazio b4]

[na_mesa b5] [topo_vazio b5]]

ESTADO INICIAL:



BASE DE FATOS DO ESTADO META: A mesma anterior.

*Algoritmo A**

COMPLETE PLAN IS:
[colocar b3 sobre mesa]
[pegar b4 da mesa]
[apoiar b4 sobre b5]
[pegar b3 da mesa]
[apoiar b3 sobre b4]
[tirar b2 de b1]
[apoiar b2 sobre b3]
[pegar b1 da mesa]
[apoiar b1 sobre b2]

Analise Intermediária

COMPLETE PLAN IS:
[colocar b3 sobre mesa]
[tirar b2 de b1]
[colocar b2 sobre mesa]
[pegar b1 da mesa]
[apoiar b1 sobre b2]
[tirar b1 de b2]
[colocar b1 sobre mesa]
[pegar b2 da mesa]
[apoiar b2 sobre b3]
[tirar b2 de b3]
[colocar b2 sobre mesa]
[pegar b3 da mesa]
[apoiar b3 sobre b4]
[tirar b3 de b4]
[colocar b3 sobre mesa]
[pegar b4 da mesa]
[apoiar b4 sobre b5]
[pegar b3 da mesa]
[apoiar b3 sobre b4]
[pegar b2 da mesa]
[apoiar b2 sobre b3]
[pegar b1 da mesa]
[apoiar b1 sobre b2]