Introdução à Inteligência Artificial
MAC 5739 - MAC 415

2006

Exercício Programa 1
Busca

Data de Divulgação: 26 de agosto

Data de Entrega: 26 de setembro

1 Objetivo

Implementar os algoritmos de busca em Inteligência Artificial.

1.1 O jogo: régua-puzzle

Considere um jogo onde 2N blocos são alinhados em uma régua com 2N+1 posições. Existem N blocos brancos (B), N blocos azuis (A) e uma posição vazia. Uma régua pode ser especificada como sendo um vetor R que varia de [1 .. 2N+1].

B	A	-	A	B

O objetivo do jogo é colocar todos os blocos brancos do lado esquerdo dos blocos azuis, ou seja, para todo i, se R[i] = B então R[j] ≠ A, para todo 0 < j < i.

Uma solução ótima para esse jogo é aquela encontrada com o menor número de movimentos.

1.2 Movimentação de blocos:

Definimos a distância entre duas posições i e j, sendo $0 < i < j \leq 2N+1$, é j – i.

Blocos podem pular para a posição vazia quando a posição vazia estiver distante de no **máximo N** casas da posição do bloco. Desta maneira, existem no máximo 2N movimentos legais (no caso do vazio estar exatamente no meio da régua). O custo de um pulo é igual à distância entre a posição do bloco e a posição vazia. Por exemplo, considere o seguinte caso com N=2.

Estado inicial:

B	A	-	A	B

Existem quatro estados sucessores para o estado inicial:

B	-	A	A	B

B	A	A	-	B

-	A	B	A	B

B	A	B	A	-

Existem quatro estados meta possíveis:

B	B	A	-	A

B	B	-	A	A

B	-	B	A	A

-	B	B	A	A

2 Exercício programa

Implementar um programa capaz de resolver o jogo descrito acima, usando as seguintes estratégias de busca:

- Busca em Largura (*Breadth First Search - BrFS*)
- Busca em Profundidade (*Depth First Search - DFS*) (com passo=1)
- Busca de Menor Custo (*Uniform Cost Search - UCS*)
- Busca em Profundidade Iterativa (*Iterative Deepening Search - IDS*)
- Busca A*
- Busca IDA*

Seu programa deve considerar como entrada o nome de um arquivo contendo um jogo régua-puzzle (veja a seção *Formato de Entrada*) e um parâmetro que especifica o tipo de busca (*BrFS*, *DFS, UCS, IDS, A** e *IDA**). O programa deve imprimir:

- caminho da solução (todos os nós da raiz até o nó meta, inclusive)
- a profundidade do estado meta (a solução)
- o custo da solução

- o número total de nós gerados e o número de nós visitados (nós gerados que já foram testados e para os quais, se possível, foram gerados seus sucessores)
- o valor médio do fator de ramificação da árvore de busca (*branching factor*) (cálculado através da razão entre o número total de nós sucessores gerados e o número de nós visitados)

Por exemplo, se o jogo do exemplo acima esta no arquivo `puz1.txt`, e se programa de busca se chama `search`, então rodando `search puz1.txt BrFS` produziria a seguinte saida:

número de nós visitados: 5
número de nós gerados: 20
profundidade da meta: 3
custo da solução: 5
fator de ramificação médio: 4

caminho da solução:

```
BA-AB
BABA-
BAB-A
B-BAA
```

3 Como evitar o processamento redundante na busca:

Durante a busca, um mesmo estado pode ser visitado muitas vezes. Para melhorar a eficiência da busca é preciso evitar o processamento de nós repetidos. Se um nó já foi visitado então ele não deve ser incluido na lista de nós a serem visitados. Você deve usar uma tabela de hashing para manter registro dos estados já visitados.

4 Função successor de estado

Para facilitar a correção dos EPs, a geração dos nós sucessores deve obedecer uma ordem fixa:

1. Movimentar blocos da esquerda da posição vazia antes de movimentar os blocos da direita
2. Movimentar os blocos mais próximos da posição vazia antes dos blocos mais distantes.
3. A ordem de inserção dos filhos de um nó na lista de nós a serem processados deve obedecer a ordem de geração.
4. No caso da lista de prioridade, nós de mesmo custo devem ser ordenados segundo a sua ordem cronológica, ou seja, nós gerados mais recentemente devem ser ordenados antes do que os nós gerados anteriormente.

5 A função heurística para o régua-puzzle

Você pode construir uma mais do que uma função heurística para comparar os resultados.

6 Formato de Entrada

Arquivos contendo régua-puzzles, consistem de duas linhas: a primeira corresponde ao números de blocos brancos (igual ao número de blocos azuis). Se a primeira linha contém o número N, então o puzzle consiste de N blocos brancos, N blocos azuis e 2N+1 posições. A segunda linha mostra o estado inicial, representando um bloco branco por **B**, um bloco azul por **A** e um espaço branco por **-**.

Exemplo de um puzzle:

2

BA-AB

7 Linguagem de programação

Você poderá usar as seguintes linguagens de programação: C, C++ ou Java. A entrega será eletrônica através do sistema Moodle.

8 Exemplos de puzzles para testar os algoritmos

Testes usando Puzzles Pequenos

```
*****************************************
2
-ABAB
*****************************************
2
ABBA-
*****************************************
2
-AABB
*****************************************
```

Crie três arquivos textos contendo os três puzzles acima com os seguintes nomes de arquivos respectivamente: `puz1, puz2, e puz3`

Testes usando Puzzles Grandes

```
*****************************************
6
ABBBBBBAAAAA-
*****************************************
6
AAAAAA-BBBBBB
*****************************************
5
ABABABABAB-
*****************************************
```

Crie três arquivos textos contendo os três puzzles acima com os seguintes nomes de arquivos respectivamente: `puz4, puz5, e puz6.`

9 Algumas perguntas que você deve ser capaz de responder:

- *Qual é a importância de se evitar o processamento de nós repetidos para (1) a busca em largura e (2) a busca em profundidade*?
- *Porque a busca UCS para o jogo régua-puzzle garante encontrar a solução ótima (de menor custo)*?
- *Na busca em profundidade iterativa, que cuidados devemos ter ao se evitar o processamento de nós repetidos*?
- *Porque a busca em profundidade iterativa, apesar de consumir menos memória do que a busca em largura, ainda garante encontrar a solução ótima*?
- *A heurística que você construiu para o jogo régua-puzzle garante encontrar a solução ótima? Por que (prove)?*

Bonus (1.0): Propor casos de teste para o jogo régua-puzzle que evidenciem as propriedades estudadas das diferentes estratégias de busca e/ou das diferentes heurísticas que você elaborar.

10 Arquivos que deverão ser entregues

1. Cada aluno deverá entrgar um único arquivo fonte, chamado ***search***, contendo um cabeçalho especificando nome e número USP do aluno.
2. Um arquivo pdf chamado **`relatorio.pdf`** entitulado **ANÁLISE DOS MÉTODOS DE BUSCA**, onde você deverá fazer uma análise dos diferentes métodos de busca aplicados ao régua-puzzle. Nesta análise você deverá responder as perguntas da seção 6 e incluir gráficos para comparar as diferentes características dos métodos de busca implementados, por exemplo, você poderá comparar o número de nós expandidos, espaço de memória necessário para cada uma das buscas, etc. As instruções sobre os casos de teste e maiores detalhes sobre o relatório serão dadas até o dia 12 de setembro.

11 Instruções gerais:

Faça o seu programa bem estruturado, bem documentado e **sózinho**. Tenha o cuidado de implementar os algoritmos de busca de maneira independente do domínio de aplicação, ou seja, de forma que eles possam ser usados em outros jogos ou problemas sem que você tenha que mexer nos algoritmos de busca, mas somente na descrição de estado, funções geradoras de estados e teste de estado meta.