PCS3438 - Algoritmos e seus pseudocódigos

September 11, 2019

Este documento contém as rotinas adotas, em pseudocódigo, para os algoritmos referenciados na disciplina. Utilize-os como referência para solucionar exercícios.

Note que é usado **borda** para fronteira e **explorado** para visitados, seguindo a notação do livro AIMA traduzido.

1 Estratégias de Busca sem Informação

Algorithm 1 Busca em Largura (problema)

```
nó ← um nó com ESTADO = problema.ESTADO-INICIAL
borda ← uma fila FIFO com nó como elemento único
explorado ← ∅
while borda ≠ ∅ do
  nó ← POP(borda) {escolhe o nó primeiro da fila}
  explorado ← explorado ∪ {nó.ESTADO}
  if problema.TESTE-DE-OBJETIVO(nó.ESTADO) then
    return SOLUÇÃO(nó)
  end if
  for cada ação em problema.AÇÕES(nó.ESTADO) do
    filho ← NÓ-FILHO(problema, nó, ação)
    if filho.ESTADO não está no explorado nem na borda then
      borda ← INSIRA(filho, borda)
    end if
  end for
end while
return erro
```

Algorithm 2 Busca de Custo Uniforme (problema)

```
1: nó ← um nó com ESTADO = problema.ESTADO-INICIAL
2: nó.CUSTO-DE-CAMINHO ← 0
3: borda ← fila de prioridade ordenada pelo CUSTO-DE-CAMINHO, com nó
   como elemento único
4: explorado ← ∅
5: while borda ≠ ∅ do
6:   nó ← POP(borda) {escolhe o nó de menor custo na borda}
7:   explorado ← explorado ∪ {nó.ESTADO}
8:   if problema.TESTE-OBJETIVO(nó.ESTADO) then
9:     return SOLUÇÃO(nó)
10:  end if
11:  for cada ação em problema.AÇÕES(nó.ESTADO) do
12:    filho ← NÓ-FILHO(problema, nó, ação) {atualiza também o
       filho.CUSTO-DE-CAMINHO}
13:    if filho.ESTADO não está na borda nem no explorado then
14:      borda ← INSIRA(filho, borda) {coloca na fila em ordem}
15:    else
16:      if filho.ESTADO é um nó da borda com maior nó.CUSTO-DE-
         CAMINHO then
17:        eliminar aquele nó da borda
18:        borda ← INSIRA(filho, borda)
19:      end if
20:    end if
21:  end for
22: end while
23: return erro
```

Algorithm 3 BPL(problema, limite):Busca em Profundidade Limitada

```
return BPL-R(CRIAR-NÓ(problema.ESTADO-INICIAL),problema,limite)
```

Algorithm 4 BPL-R(nó, problema, limite): Busca em Profundidade Limitada Recursiva

```
1: if problema.TESTE-OBJETIVO(nó.ESTADO) then
2:    return SOLUÇÃO(nó)
3: end if
4: if limite = 0 then
5:    return corte
6: end if
7: CORTE-OCORREU ← FALSO
8: for cada ação em problema.AÇÕES(nó.ESTADO) do
9:    filho ← NÓ-FILHO(problema, nó, ação)
10:   resultado ← BPL-R(filho, problema, limite - 1)
11:   if resultado = corte then
12:      CORTE-OCORREU ← VERDADEIRO
13:   else
14:      if resultado ≠ erro then
15:         return resultado
16:      end if
17:   end if
18: end for
19: if CORTE-OCORREU then
20:   return corte
21: else
22:   return erro
23: end if
```

Algorithm 5 BAI(problema): Busca por Aprofundamento Iterativo

```
1: for limite = 0 até ∞ do
2:    resultado ← BPL(problema, limite)
3:    if resultado ≠ corte then
4:       return resultado
5:    end if
6: end for
```

2 Busca Informada

O nó.CUSTO-OBJETIVO é o valor estimado pela heurística para atingir o objetivo a partir do nó. Na meta, o CUSTO-OBJETIVO é nulo.

Algorithm 6 Busca Gulosa pela Melhor Escolha (problema): *Greedy Best-First Search*

```
1: nó ← um nó com ESTADO = problema.ESTADO-INICIAL
2: nó.CUSTO-OBJETIVO ← heurística(nó.ESTADO)
3: borda ← fila de prioridade ordenada por CUSTO-OBJETIVO, com nó como
   único elemento
4: explorado ← ∅
5: while borda ≠ ∅ do
6:   nó ← POP(borda) {escolhe o nó de menor CUSTO-OBJETIVO na borda}
7:   explorado ← explorado ∪ nó.ESTADO
8:   if problema.TESTE-OBJETIVO(nó.ESTADO) then
9:     return SOLUÇÃO(nó)
10:  end if
11:  for cada ação em problema.AÇÕES(nó.ESTADO) do
12:    filho ← NÓ-FILHO(problema, nó, ação)
13:    if filho.ESTADO não está na borda nem no explorado then
14:      borda ← INSIRA(filho, borda) {insira na borda, na ordem}
15:    else
16:      if filho.ESTADO é um nó da borda com maior CUSTO-OBJETIVO
         then
17:        eliminar aquele nó da borda
18:        borda ← INSIRA(filho, borda)
19:      end if
20:    end if
21:  end for
22: end while
23: return erro
```

Algorithm 7 Busca A*(problema)

```
1: nó ← um nó com ESTADO = problema.ESTADO-INICIAL
2: nó.CUSTO-DE-CAMINHO = 0
3: nó.CUSTO-OBJETIVO ← heurística(nó.ESTADO) {o CUSTO-OBJETIVO é dado pela heurística}
4: borda ← fila de prioridade ordenada por (CUSTO-OBJETIVO + CUSTO-DE-CAMINHO) com nó como único elemento
5: explorado ← ∅
6: while borda ≠ ∅ do
7:   nó ← POP(borda) {escolhe o nó de menor custo na borda}
8:   explorado ← explorado ∪ nó.ESTADO
9:   if problema.TESTE-OBJETIVO(nó.ESTADO) then
10:    return SOLUÇÃO(nó)
11:  end if
12:  for cada ação em problema.AÇÕES(nó.ESTADO) do
13:    filho ← NÓ-FILHO(problema, nó, ação)
14:    if filho.ESTADO não está na borda nem no explorado then
15:      borda ← INSIRA(filho, borda)
16:    else
17:      if filho.ESTADO é um nó da borda com maior (CUSTO-DE-CAMINHO+CUSTO-OBJETIVO) then
18:        eliminar aquele nó da borda
19:        borda ← INSIRA(filho, borda)
20:      end if
21:    end if
22:  end for
23: end while
24: return erro
```