Manual Técnico

Inteligência Artificial | Projeto 1 - Jogo do Cavalo | André Meseiro 202100225 e Pedro Anjos 202100230

1. Algoritmos Implementados - Completos, por partes, e devidamente comentados

1.1. Algoritmo de Procura em Largura - Breadth-First Search (BFS)

```
;; Algoritmo de procura em largura
(defun bfs (no-inicial objetivop funcao-sucessores no-existep operadores &optional
(nos-expandidos 0) (nos-gerados 0) abertos fechados (tempo-inicial (get-internal-
real-time)))
  "Implementação do algoritmo de procura em largura. Recebe o nó inicial, o
objetivo de pontuação, a função que gera os sucessores, a função que verifica se
um nó existe e os operadores. Retorna uma lista com os nós que compõem o caminho,
ou NIL."
  ; Gera a lista de nós sucessores, gerados pelo nó passado como argumento,
através dos operadores
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
        ; Lista de nós abertos juntamente com os nós fechados
        (t (let* ((sucessores-gerados (remove-if (lambda (suc) (funcall no-existep
suc fechados 'bfs)) (funcall funcao-sucessores no-inicial operadores 'bfs)))
                  ; Gera a lista de nós que são solução
                  (solucao (list (apply #'append (mapcar (lambda (suc) (cond
((funcall objetivop suc) suc))) sucessores-gerados))))
                  ; Gera a lista de nós abertos com os nós sucessores (que não
constam na lista de nós abertos) adicionados
                  (abertos-novo (abertos-bfs abertos sucessores-gerados)))
             (let ((nos-expandidos-novo (1+ nos-expandidos))
                   (nos-gerados-novo (+ nos-gerados (length sucessores-
gerados))))
               (cond
                 ; Verifica se o nó inicial é solução, se for retorna-o
                 ((funcall objetivop no-inicial) (list no-inicial nos-expandidos-
novo nos-gerados (penetrancia no-inicial nos-gerados) (ramificacao-media no-
inicial nos-gerados) (/ (- (get-internal-real-time) tempo-inicial) internal-time-
units-per-second)))
                 ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                 ((null abertos-novo) (list nil nos-expandidos-novo nos-gerados-
novo 0 0 (/ (- (get-internal-real-time) tempo-inicial) internal-time-units-per-
second)))
                 ; Verifica se a lista de nós solução não é nula, se não for
retorna o 1º nó da lista
                 ((not (null (car solucao))) (list (car solucao) nos-expandidos-
novo nos-gerados-novo (penetrancia (car solucao) nos-gerados-novo) (ramificacao-
media (car solucao) nos-gerados-novo) (/ (- (get-internal-real-time) tempo-
inicial) internal-time-units-per-second)))
                 ; Aplica recursividade para continuar a procurar
                 (t (bfs (car abertos-novo) objetivop funcao-sucessores no-
```

```
existep operadores nos-expandidos-novo nos-gerados-novo (cdr abertos-novo) (append
fechados (list no-inicial)) tempo-inicial))
                )
              )
            )
          )
  )
)
```

1.2. Algoritmo de Procura em Profundidade - Depth-First Search (DFS)

```
;; Algoritmo de procura em profundidade
(defun dfs (no-inicial objetivop funcao-sucessores no-existep operadores
profundidade-max &optional (nos-expandidos 0) (nos-gerados 0) abertos fechados
(tempo-inicial (get-internal-real-time)))
  "Implementação do algoritmo de procura em largura. Recebe o nó inicial, o
objetivo de pontuação, a função que gera os successores, a função que verifica se
um nó existe, os operadores e a profundidade máxima. Retorna uma lista com os nós
que compõem o caminho, ou NIL."
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
        ; Lista de nós abertos juntamente com os nós fechados
        (t (let* ((abertos-fechados (append abertos fechados))
                  ; Lista de nós sucessores gerados pelo nó passado como argumento
através dos operadores
                  (sucessores-gerados (remove-if (lambda (suc) (funcall no-existep
suc abertos-fechados 'dfs)) (funcall funcao-sucessores no-inicial operadores 'dfs
profundidade-max)))
                  ; Lista de nós que são solução
                  (solucao (list (apply #'append (mapcar (lambda (suc) (cond
((funcall objetivop suc) suc))) sucessores-gerados))))
                  ; Lista de nós abertos com as profundidades recalculadas
                  (abertos-recalculados (recalcular-profundidade sucessores-
gerados abertos))
                  ; Lista de nós abertos com os nós sucessores (que não constam na
lista de nós abertos e fechados) adicionados
                  (abertos-novo (abertos-dfs abertos-recalculados sucessores-
gerados))
                  ; Lista de nós fechados com as profundidades recalculadas
                  (fechados-recalculados (recalcular-profundidade sucessores-
gerados fechados)))
             (let ((nos-expandidos-novo (1+ nos-expandidos))
                   (nos-gerados-novo (+ nos-gerados (length sucessores-
gerados))))
               (cond
                 ; Verifica se o nó inicial é solução, se for retorna-o
                 ((funcall objetivop no-inicial) (list no-inicial nos-expandidos-
novo nos-gerados (penetrancia no-inicial nos-gerados) (ramificacao-media no-
inicial nos-gerados) (/ (- (get-internal-real-time) tempo-inicial) internal-time-
units-per-second)))
                 ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                 ((null abertos-novo) (list nil nos-expandidos-novo nos-gerados-
```

```
novo 0 0 (/ (- (get-internal-real-time) tempo-inicial) internal-time-units-per-
second)))
                  ; Verifica se a lista de nós solução não é nula, se não for
retorna o 1º nó da lista
                  ((not (null (car solucao))) (list (car solucao) nos-expandidos-
novo nos-gerados-novo (penetrancia (car solucao) nos-gerados-novo) (ramificacao-
media (car solucao) nos-gerados-novo) (/ (- (get-internal-real-time) tempo-
inicial) internal-time-units-per-second)))
                  ; Aplica recursividade para continuar a procurar
                  (t (dfs (car abertos-novo) objetivop funcao-sucessores no-
existep operadores profundidade-max nos-expandidos-novo nos-gerados-novo (cdr
abertos-novo) (append fechados-recalculados (list no-inicial)) tempo-inicial))
                )
              )
            )
          )
    )
)
```

1.3. Algoritmo de Procura A*

```
;; Algoritmo de procura A*
(defun aestrela (no-inicial objetivop funcao-sucessores no-existep funcao-
heuristica operadores &optional (nos-expandidos 0) (nos-gerados 0) abertos
fechados (tempo-inicial (get-internal-real-time)))
  "Implementação do algoritmo de procura A*. Recebe o nó inicial, o objetivo de
pontuação, a função que gera os nós sucessores, a função que verifica se um nó
existe, a função que calcula a heurística e os operadores. Retorna uma lista com
os nós que compõem o caminho, ou NIL."
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
        ; Lista de nós sucessores gerados pelo nó passado como argumento através
dos operadores
        (t (let* ((sucessores-gerados (remove-if (lambda (suc) (funcall no-existep
suc (append abertos fechados) 'aestrela)) (funcall funcao-sucessores no-inicial
operadores 'aestrela 0 funcao-heuristica)))
                  ; Lista de nós que são solução
                  (solucao (list (apply #'append (mapcar (lambda (suc) (cond
((funcall objetivop suc) suc))) sucessores-gerados))))
                  ; Lista de nós abertos com as profundidades recalculadas
                  (abertos-recalculados (recalcular-profundidade sucessores-
gerados abertos))
                  ; Lista de nós abertos com os nós sucessores (que não constam na
lista de nós abertos e fechados) adicionados
                  (abertos-novo (colocar-sucessores-em-abertos abertos-
recalculados sucessores-gerados))
                  ; Lista de nós fechados com as profundidades recalculadas
                  (fechados-recalculados (recalcular-profundidade sucessores-
gerados fechados)))
              (let ((nos-expandidos-novo (1+ nos-expandidos))
                    (nos-gerados-novo (+ nos-gerados (length sucessores-
gerados))))
```

```
                (cond
                  ; Verifica se o nó inicial é solução, se for retorna-o
                  ((funcall objetivop no-inicial) (list no-inicial nos-expandidos-
novo nos-gerados (penetrancia no-inicial nos-gerados) (ramificacao-media no-
inicial nos-gerados) (/ (- (get-internal-real-time) tempo-inicial) internal-time-
units-per-second)))
                  ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                  ((null abertos-novo) (list nil nos-expandidos-novo nos-gerados-
novo 0 0 (/ (- (get-internal-real-time) tempo-inicial) internal-time-units-per-
second)))
                  ; Verifica se a lista de nós solução não é nula, se não for
retorna o 1º nó da lista
                  ((not (null (car solucao))) (list (car solucao) nos-expandidos-
novo nos-gerados-novo (penetrancia (car solucao) nos-gerados-novo) (ramificacao-
media (car solucao) nos-gerados-novo) (/ (- (get-internal-real-time) tempo-
inicial) internal-time-units-per-second)))
                  ; Aplica recursividade para continuar a procurar
                  (t (aestrela (car abertos-novo) objetivop funcao-sucessores no-
existep funcao-heuristica operadores nos-expandidos-novo nos-gerados-novo (cdr
abertos-novo) (append fechados-recalculados (list no-inicial)) tempo-inicial))
                )
              )
            )
          )
  )
)
```

1.4. Algoritmo de Procura IDA* Completo

```
;; Algoritmo de procura IDA*
(defun idaestrela (no-inicial objetivop funcao-sucessores no-existep funcao-
heuristica operadores &optional (limiar 0) (nos-expandidos 0) (nos-gerados 0)
(tempo-inicial (get-internal-real-time)))
  "Implementação do algoritmo de procura IDA*. Recebe o nó inicial, o objetivo de
pontuação, a função que gera os nós sucessores, a função que verifica se um nó
existe, a função que calcula a heurística e os operadores. Retorna uma lista com
os nós que compõem o caminho, ou NIL."
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
        ; Lista de nós abertos juntamente com os nós fechados
        (t (idaestrela-loop no-inicial objetivop funcao-sucessores no-existep
funcao-heuristica operadores limiar nos-expandidos nos-gerados tempo-inicial))
  )
)

;; Algoritmo de procura IDA* (loop)
(defun idaestrela-loop (no-inicial objetivop funcao-sucessores no-existep funcao-
heuristica operadores limiar nos-expandidos nos-gerados tempo-inicial &optional
abertos fechados)
  "Função auxiliar do algoritmo de procura IDA* que é responsável por fazer o loop
de procura"
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
```

```
        ; Lista de nós sucessores gerados pelo nó passado como argumento através
dos operadores
        (t (let* ((sucessores-gerados (remove-if (lambda (suc) (funcall no-existep
suc (append abertos fechados) 'dfs)) (funcall funcao-sucessores no-inicial
operadores 'idaestrela 0 funcao-heuristica)))
                  ; Lista de nós que são solução
                  (solucao (list (apply #'append (mapcar (lambda (suc) (cond
((funcall objetivop suc) suc))) sucessores-gerados))))
                  ; Lista de nós abertos com as profundidades recalculadas
                  (abertos-recalculados (recalcular-profundidade sucessores-
gerados abertos))
                  ; Lista de nós abertos com os nós sucessores (que não constam na
lista de nós abertos e fechados) adicionados
                  (abertos-novo (abertos-dfs abertos-recalculados sucessores-
gerados))
                  ; Lista de abertos com custo menor que o limiar
                  (abertos-menores-limiar (remove-if (lambda (no) (> (no-custo no)
limiar)) abertos-novo))
                  ; Lista de nós fechados com as profundidades recalculadas
                  (fechados-recalculados (recalcular-profundidade sucessores-
gerados fechados)))
              (let ((nos-expandidos-novo (1+ nos-expandidos))
                    (nos-gerados-novo (+ nos-gerados (length sucessores-
gerados))))
                (cond
                  ; Verifica se o nó inicial é solução, se for retorna-o
                  ((funcall objetivop no-inicial) (list no-inicial nos-expandidos-
novo nos-gerados (penetrancia no-inicial nos-gerados) (ramificacao-media no-
inicial nos-gerados) (/ (- (get-internal-real-time) tempo-inicial) internal-time-
units-per-second)))
                  ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                  ((null abertos-menores-limiar) (idaestrela no-inicial objetivop
funcao-sucessores no-existep funcao-heuristica operadores (no-custo (no-menor-
custo abertos-novo)) nos-expandidos-novo nos-gerados-novo tempo-inicial))
                  ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                  ((null abertos-novo) (list nil nos-expandidos-novo nos-gerados-
novo 0 0 (/ (- (get-internal-real-time) tempo-inicial) internal-time-units-per-
second)))
                  ; Verifica se a lista de nós solução não é nula, se não for
retorna o 1º nó da lista
                  ((not (null (car solucao))) (list (car solucao) nos-expandidos-
novo nos-gerados-novo (penetrancia (car solucao) nos-gerados-novo) (ramificacao-
media (car solucao) nos-gerados-novo) (/ (- (get-internal-real-time) tempo-
inicial) internal-time-units-per-second)))
                  ; Aplica recursividade para continuar a procurar
                  (t (idaestrela-loop (car abertos-novo) objetivop funcao-
sucessores no-existep funcao-heuristica operadores limiar nos-expandidos-novo nos-
gerados-novo tempo-inicial (cdr abertos-novo) (append fechados-recalculados (list
no-inicial))))
                )
              )
            )
        )
```

```
  )
)
```

1.4.1. Função Principal do Algoritmo IDA*

```
;; Algoritmo de procura IDA*
(defun idaestrela (no-inicial objetivop funcao-sucessores no-existep funcao-
heuristica operadores &optional (limiar 0) (nos-expandidos 0) (nos-gerados 0)
(tempo-inicial (get-internal-real-time)))
  "Implementação do algoritmo de procura IDA*. Recebe o nó inicial, o objetivo de
pontuação, a função que gera os nós sucessores, a função que verifica se um nó
existe, a função que calcula a heurística e os operadores. Retorna uma lista com
os nós que compõem o caminho, ou NIL."
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
        ; Lista de nós abertos juntamente com os nós fechados
        (t (idaestrela-loop no-inicial objetivop funcao-sucessores no-existep
funcao-heuristica operadores limiar nos-expandidos nos-gerados tempo-inicial))
  )
)
```

1.4.2. Função Loop (Auxiliar) do Algoritmo IDA*

```
;; Algoritmo de procura IDA* (loop)
(defun idaestrela-loop (no-inicial objetivop funcao-sucessores no-existep funcao-
heuristica operadores limiar nos-expandidos nos-gerados tempo-inicial &optional
abertos fechados)
  "Função auxiliar do algoritmo de procura IDA* que é responsável por fazer o loop
de procura"
  (cond ((null no-inicial) (error "Nó inicial não pode ser nulo"))
        ; Lista de nós sucessores gerados pelo nó passado como argumento através
dos operadores
        (t (let* ((sucessores-gerados (remove-if (lambda (suc) (funcall no-existep
suc (append abertos fechados) 'dfs)) (funcall funcao-sucessores no-inicial
operadores 'idaestrela 0 funcao-heuristica)))
                  ; Lista de nós que são solução
                  (solucao (list (apply #'append (mapcar (lambda (suc) (cond
((funcall objetivop suc) suc))) sucessores-gerados))))
                  ; Lista de nós abertos com as profundidades recalculadas
                  (abertos-recalculados (recalcular-profundidade sucessores-
gerados abertos))
                  ; Lista de nós abertos com os nós sucessores (que não constam na
lista de nós abertos e fechados) adicionados
                  (abertos-novo (abertos-dfs abertos-recalculados sucessores-
gerados))
                  ; Lista de abertos com custo menor que o limiar
                  (abertos-menores-limiar (remove-if (lambda (no) (> (no-custo no)
limiar)) abertos-novo))
                  ; Lista de nós fechados com as profundidades recalculadas
                  (fechados-recalculados (recalcular-profundidade sucessores-
```

```
gerados fechados)))
              (let ((nos-expandidos-novo (1+ nos-expandidos))
                    (nos-gerados-novo (+ nos-gerados (length sucessores-
gerados))))
                (cond
                  ; Verifica se o nó inicial é solução, se for retorna-o
                  ((funcall objetivop no-inicial) (list no-inicial nos-expandidos-
novo nos-gerados (penetrancia no-inicial nos-gerados) (ramificacao-media no-
inicial nos-gerados) (/ (- (get-internal-real-time) tempo-inicial) internal-time-
units-per-second)))
                  ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                  ((null abertos-menores-limiar) (idaestrela no-inicial objetivop
funcao-sucessores no-existep funcao-heuristica operadores (no-custo (no-menor-
custo abertos-novo)) nos-expandidos-novo nos-gerados-novo tempo-inicial))
                  ; Verifica se a lista de nós abertos é nula, se for retorna NIL
                  ((null abertos-novo) (list nil nos-expandidos-novo nos-gerados-
novo 0 0 (/ (- (get-internal-real-time) tempo-inicial) internal-time-units-per-
second)))
                  ; Verifica se a lista de nós solução não é nula, se não for
retorna o 1º nó da lista
                  ((not (null (car solucao))) (list (car solucao) nos-expandidos-
novo nos-gerados-novo (penetrancia (car solucao) nos-gerados-novo) (ramificacao-
media (car solucao) nos-gerados-novo) (/ (- (get-internal-real-time) tempo-
inicial) internal-time-units-per-second)))
                  ; Aplica recursividade para continuar a procurar
                  (t (idaestrela-loop (car abertos-novo) objetivop funcao-
sucessores no-existep funcao-heuristica operadores limiar nos-expandidos-novo nos-
gerados-novo tempo-inicial (cdr abertos-novo) (append fechados-recalculados (list
no-inicial))))
                )
              )
            )
          )
  )
)
```

2. Descrição dos objetos que compõem o projeto, incluindo dados e procedimentos

Dados

- Nó - Objeto base do projeto; Consiste numa lista que contém o tabuleiro, o custo e o nó pai (antecessor); Neste contexto, é o estado atual do jogo;
- Operador - Consiste numa operação a aplicar a um nó; Neste contexto está relacionado com uma operação intermédia necessária para chegar ao estado pretendido, associado à jogada efetuada;
- Sucessores - Consiste numa lista que contém os nós sucessores de um determinado nó, após a aplicação de um operador; Neste contexto, diz respeito a todas as casas disponíveis para a jogada seguinte;
- Lista de abertos - Consiste numa lista que contém os nós que ainda não foram explorados;

- Lista de fechados - Consiste numa lista que contém os nós que já foram explorados;
- Nó solução - Consiste no nó que contém a lista com os nós que compõem a solução (caminho desde o nó inicial até ao nó solução) do problema.

Procedimentos

- Aplicar operador aos nós sucessores do nó atual, que se encontram na lista de abertos, expandindo-os;
- Colocar nó expandido na lista de fechados, verificando se algum dos nós que se encontram nessa lista é solução;
- Repetir os dois passos acima até à lista de abertos estar vazia;
- O programa termina quando encontrar a solução, ou quando a lista de abertos estiver vazia, encontrando ou não a solução.

3. Limitações

No que diz respeito às limitações, a principal que foi encontrada foi o facto do IDE utilizado, LispWorks, na versão grátis, ter um limite de memória definido para os programas.

A outra limitação foi o facto de não ter sido possível implementar uma segunda heurística, suficientemente boa para resolver o problema E.

4. Análise critica dos resultados das execuções do programa, transparecendo a compreensão das limitações do projeto

Devido às limitações mencionadas no ponto anterior, os algoritmos BFS e DFS não conseguem encontrar solução para os problemas E e F. Pelos mesmos motivos, ambos os algoritmos A* e IDA* não conseguem encontrar solução para o problema E, devido às heurísticas utilizadas não serem boas o suficiente.

Estas duas situações devem-se ao facto dos algoritmos gerarem demasiados nós, ultrapassando o limite de memória disponível e dando "crash" ao programa.

5. Análise comparativa do conjunto de execuções do programa para cada algoritmo e cada problema, permitindo verificar o desempenho de cada algoritmo e das heurísticas

Algoritmo BFS

	Encontrou solução? (Sim/Não e porquê)	**Nós expandidos**	**Nós gerados**	**Penetrância**	**Fator de Ramificação Média**	**Tempo de Execução**
Problema A	Sim	6	8	0.375 (37.5 %)	1.578 (157.8%)	0.004000 seg.
Problema B	Sim	43	46	0.174 (17.4%)	1.389 (138.9%)	0.021000 seg.

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema C	Sim	32	39	0.154 (15.4%)	1.574 (157.4%)	0.015000 seg.
Problema D	Sim	267	279	0.043 (4.3%)	1.452 (145.2%)	0.198000 seg.
Problema E	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível	-	-	-	-	-
Problema F	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível	-	-	-	-	-

Algoritmo DFS

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema A	Sim	6	7	0.429 (42.9%)	1.488 (148.8%)	0.003000 seg.
Problema B	Sim	10	14	0.714 (71.4%)	1.060 (106.0%)	0.007000 seg.
Problema C	Sim	8	16	0.375 (37.5%)	1.289 (128.9%)	0.008000 seg.
Problema D	Sim	36	46	0.283 (28.3%)	1.170 (117.0%)	0.040000 seg.
Problema E	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível	-	-	-	-	-
Problema F	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível	-	-	-	-	-

Heurísticas

- Heurística Base - Privilegia visitar as casas com o maior número de pontos.
    - Segue a fórmula h(x) = o(x)/m(x), em que:
        - x é o tabuleiro;
        - m(x) é a média por casa dos pontos que constam no tabuleiro x;
        - o(x) é o número de pontos que faltam para atingir o valor definido como objetivo.
- Heurística Implementada - Privilegia visitar as casas com o maior número de pontos, excluindo as casas que se encontram ao redor do cavalo.
    - Segue a fórmula h(x) = o(x)/(msr(x)/n(x))
        - x é o tabuleiro;
        - msr(x) é a média por casa dos pontos que constam no tabuleiro x, excluindo as casas que se encontram ao redor do cavalo;
        - n(x) é o número de jogadas que o cavalo pode fazer no tabuleiro x.

Algoritmo A*

A*: Resultados obtidos com a Heurística Base

	Encontrou solução? (Sim/Não e porquê)	**Nós expandidos**	**Nós gerados**	**Penetrância**	**Fator de Ramificação Média**	**Tempo de Execução**
Problema A	Sim	3	5	0.600 (60.0%)	1.278 (127.8%)	0.002000 seg.
Problema B	Sim	32	39	0.205 (20.5%)	1.352 (135.2%)	0.033000 seg.
Problema C	Sim	32	39	0.154 (15.4%)	1.574 (157.4%)	0.030000 seg.
Problema D	Sim	73	98	0.122 (12.2%)	1.302 (130.2%)	1.132000 seg.
Problema E	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível. Devido ao problema ter poucos pontos, requer um caminho muito específico, que apenas é atingível com uma heurística boa	-	-	-	-	-

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema F	Sim	30	120	0.233 (23.3%)	1.089 (108.9%)	0.284000 seg.

A*: Resultados obtidos com a Heurística Implementada

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema A	Sim	5	7	0.429 (42.9%)	1.488 (148.8%)	0.009000 seg.
Problema B	Sim	17	21	0.381 (38.1%)	1.213 (121.3%)	0.026000 seg.
Problema C	Sim	18	22	0.273 (27.3%)	1.388 (138.8%)	0.029000 seg.
Problema D	Sim	64	76	0.171 (17.1%)	1.234 (123.4%)	0.154000 seg.
Problema E	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível. Devido ao problema ter poucos pontos, requer um caminho muito específico, que apenas é atingível com uma heurística boa	-	-	-	-	-
Problema F	Sim	70	172	0.233 (23.3%)	1.062 (106.2%)	1.092000 seg.

Algoritmo IDA*

IDA*: Resultados obtidos com a Heurística Base

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema A	Sim	7	10	0.300 (30.0%)	1.737 (173.7%)	0.005000 seg.
Problema B	Sim	11	19	0.526 (52.6%)	1.114 (111.4%)	0.011000 seg.
Problema C	Sim	9	21	0.286 (28.6%)	1.373 (137.3%)	0.015000 seg.
Problema D	Sim	37	56	0.232 (23.2%)	1.195 (119.5%)	0.048000 seg.
Problema E	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível. Devido ao problema ter poucos pontos, requer um caminho muito específico, que apenas é atingível com uma heurística boa	-	-	-	-	-
Problema F	Sim	64	159	0.302 (30.2%)	1.043 (104.3%)	0.816000 seg.

IDA*: Resultados obtidos com a Heurística Implementada

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema A	Sim	7	10	0.300 (30.0%)	1.737 (173.7%)	0.010000 seg.
Problema B	Sim	18	26	0.385 (38.5%)	1.169 (116.9%)	0.029000 seg.
Problema C	Sim	9	21	0.286 (28.6%)	1.373 (137.3%)	0.023000 seg.
Problema D	Sim	37	56	0.232 (23.2%)	1.195 (119.5%)	0.088000 seg.

	Encontrou solução? (Sim/Não e porquê)	Nós expandidos	Nós gerados	Penetrância	Fator de Ramificação Média	Tempo de Execução
Problema E	Não, uma vez que gera demasiados nós, ultrapassando o limite de memória disponível. Devido ao problema ter poucos pontos, requer um caminho muito específico, que apenas é atingível com uma heurística boa	-	-	-	-	-
Problema F	Sim	40	130	0.292 (29.2%)	1.056 (105.6%)	0.594000 seg.

6. Lista dos requisitos do projeto (listados no enunciado) que não foram implementados

- Implementação do Algoritmo RBFS;
- Implementação do Algoritmo SMA*;