Lista de exercícios 1 (Busca)

Q1: Busca em árvores Trees. Quantos nós há na árvore de busca completa para o grafo do espaço de estados dado? O estado inicial é S. Você pode achar útil desenhar a árvore de busca em um pedaço de papel.

Grafo direcionado com arestas: S-C, S-A, S-B, C-B, C-G, B-A, A-G

Q2: DFS. Considere uma busca em profundidade no grafo abaixo, onde S é o estado inicial e G é o estado objetivo. Suponha que empates sejam resolvidos alfabeticamente (portanto, um plano parcial S->X->A seria expandido antes de S->X->B e S->A->Z seria expandido antes de S->B->A). Você pode achar útil executar a busca em um pedaço de papel.

Qual o caminho final retornado pela busca em profundidade?

Grafo (direcionado): S-B, S-A, B-A, A-C, A-G

Q3: BFS. Considere uma busca em grafos por largura no grafo abaixo, onde S é o estado inicial e G é o estado objetivo. Suponha que empates sejam resolvidos alfabeticamente (portanto, um plano parcial S->X->A seria expandido antes de S->X->B e S->A->Z seria expandido antes de S->B->A). Você pode achar útil executar a busca em um pedaço de papel.

Qual o caminho final retornado pela busca em largura?

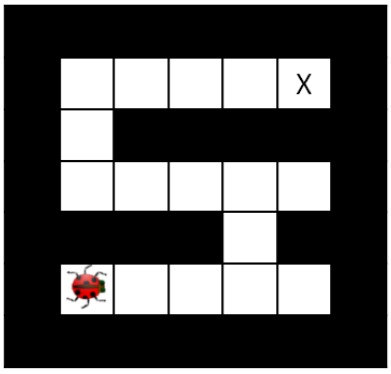
Grafo (direcionado): S-A, S-C, S-B, A-B, A-C, B-G

Q4: Considere o grafo abaixo. Os arcos são rotulados com seus pesos. Suponha que empates sejam resolvidos alfabeticamente (portanto, um plano parcial S->X->A seria expandido antes de S->X->B e S->A->Z seria expandido antes de S->B->A).

Arestas (direcionadas): S-A-2, S-B-1, A-B-1, A-C-3, A-D-1, B-D-5, C-G-7, D-G-4, B-G-10

Em que ordem os estados são expandidos por UCS (uniform cost search)? Qual o caminho final retornado por UCS?

As próximas questões compartilham uma configuração comum. Você controla um ou mais insetos em um ambiente retangular, semelhante a um labirinto, com dimensões M x N, conforme mostrado na figura abaixo.



A cada passo de tempo, um inseto pode (a) mover-se para uma célula adjacente se essa célula estiver atualmente livre, ou (b) permanecer em sua posição atual. As células podem ser bloqueadas por paredes, mas o mapa é conhecido. A otimalidade está sempre em termos de passos de tempo; todas as ações têm custo 1, independentemente do número de insetos em movimento ou para onde eles se movem.

Para cada uma das perguntas, você deve responder para uma instância geral do problema, e não apenas para os mapas de exemplo mostrados.

Q5: Você controla um único inseto, conforme mostrado no labirinto acima, que deve alcançar um local de destino designado X, também conhecido como a colmeia. Não há outros insetos se movendo pelo ambiente.

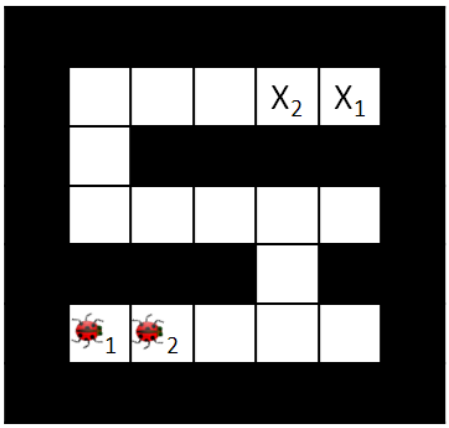
Qual das seguintes é uma representação mínima e correta do espaço de estados?

a) Um inteiro codificando a distância de Manhattan até a colmeia.
b) Uma tupla (x, y) que codifica as coordenadas x e y do inseto.
c) Uma tupla (x, y, d) codificando as coordenadas \(x\) e \(y\) do inseto, assim como a distância de Manhattan até a colmeia.
d) Nao pode ser representado como um problema de busca.

Qual o tamanho do espaço de estados?

a) MN
b) (MN)^2
c) 2^(MN)
d) M^N
e) N^M
f) max(N, M)

Q6: Você controla k insetos, cada um com um local de destino específico X_k. Nenhum dois insetos podem ocupar a mesma célula. A cada passo de tempo, todos os insetos se movem simultaneamente para uma célula atualmente livre (ou permanecem no lugar); insetos adjacentes não podem trocar de posição em um único passo de tempo.



Qual das seguintes é a representação mínima e correta do espaço de estados?

a) k tuplas (x_1, y_1), … , (x_k, y_k) codificando as coordenadas de cada inseto
b) k tuplas (x_1, y_1), … , (x_k, y_k) codificando as coordenadas de cada inseto além de k variáveis booleanas indicando se cada inseto está ao lado de outro inseto
c) k tuplas (x_1, y_1), … , (x_k, y_k) codificando as coordenadas de cada inseto, além de MN variáveis booleanas indicando quais células estão atualmente ocupadas por um inseto
d) MN variáveis booleanas codificando se um inseto está ou não em cada célula

Qual o tamanho do espaço de estado?

Opções: MN, 2^(MN), kMN, (MN)^k, (MN)^k 2^k, (MN)^k 2^(MN), 2^k MN

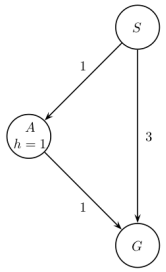
Q7: Considere o algoritmo de busca abaixo.

```
function GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Com a implementação acima, um nó que atinge um estado objetivo pode permanecer na fronteira enquanto o algoritmo continua a busca por um caminho que atinja um estado objetivo. Vamos considerar alterar o algoritmo testando se um nó atinge um estado objetivo ao inseri-lo na fronteira. Concretamente, adicionamos a linha de código destacada abaixo:

```
function EARLY-GOAL-CHECKING-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            for child-node in EXPAND(STATE[node], problem) do
                if GOAL-TEST(problem, STATE[child-node]) then return child-node
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Agora, produzimos um algoritmo de busca em grafos que pode encontrar uma solução mais rapidamente. No entanto, ao fazer isso, podemos ter afetado algumas propriedades do algoritmo. Para explorar as possíveis diferenças, considere o gráfico de exemplo abaixo.



Q7.1. Se utilizar o EARLY-GOAL-CHECKING-GRAPH-SEARCH com uma estratégia de expansão de nós de Custo Uniforme, qual caminho, se houver algum, o algoritmo retornará?

Q7.2. Se utilizar o EARLY-GOAL-CHECKING-GRAPH-SEARCH com uma estratégia de expansão de nós do tipo A*, qual caminho, se houver algum, o algoritmo retornará?

Q7.3. Assuma que você execute o EARLY-GOAL-CHECKING-GRAPH-SEARCH com a estratégia de expansão de nós de *Custo Uniforme*, selecione todas as afirmações que são verdadeiras.

a) A função EXPAND pode ser chamada no máximo 1 vez para cada estado
b) O algoritmo é completo
c) O algoritmo vai retornar uma solução ótima

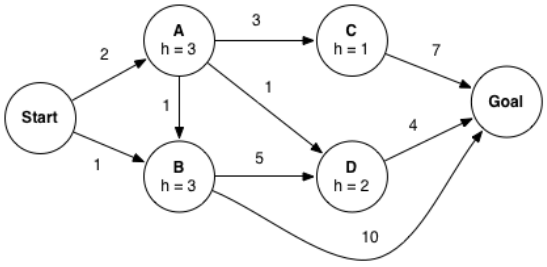
Q7.4. Assuma que você execute o EARLY-GOAL-CHECKING-GRAPH-SEARCH com a estratégia de expansão de nós A* e uma heurística consistente, selecione todas as afirmações que são verdadeiras.

a) A função EXPAND pode ser chamada no máximo 1 vez para cada estado
b) O algoritmo é completo
c) O algoritmo vai retornar uma solução ótima

Q8: Usando o GRAPH-SEARCH (ver código na questão anterior), quando um nó é expandido, ele é adicionado ao conjunto fechado. Isso significa que, mesmo que um nó seja adicionado à fronteira várias vezes, ele não será expandido mais de uma vez. Considere uma versão alternativa do GRAPH-SEARCH, chamada LOOKAHEAD-GRAPH-SEARCH, que economiza memória utilizando um "conjunto de fronteira-fechado", mantendo o controle de quais estados já estiveram na fronteira e adicionando um nó filho à fronteira somente se o estado desse nó filho não tiver sido adicionado a ela em algum momento. Veja o código a seguir.

```
function LOOKAHEAD-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    fringe-closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    add INITIAL-STATE[problem] to fringe-closed
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        for child-node in EXPAND(node, problem) do
            if STATE[child-node] is not in fringe-closed then
                add STATE[child-node] to fringe-closed
                fringe ← INSERT(child-node, fringe)
        end
    end
```

Agora, produzimos um algoritmo de busca em grafos mais eficiente em termos de memória. No entanto, ao fazer isso, podemos ter afetado algumas propriedades do algoritmo. Para explorar as possíveis diferenças, considere o gráfico de exemplo abaixo.



Q8.1. Se utilizar o LOOKAHEAD-GRAPH-SEARCH com uma estratégia de expansão de nós do tipo A*, qual caminho esse algoritmo retornará?

Q8.2. Assuma que você execute o LOOKAHEAD-GRAPH-SEARCH com a estratégia de expansão de nós do tipo A* e uma heurística consistente, selecione todas as afirmações que são verdadeiras.

a) A função EXPAND pode ser chamada no máximo 1 vez para cada estado
b) O algoritmo é completo
c) O algoritmo vai retornar uma solução ótima

Q9: Lembre-se de que um dicionário, também conhecido como hashmap, funciona da seguinte maneira:

```
dict ← an empty dictionary
dict["key"] ← "value"
print dict["key"]
→ "value"
```

Inserir um par chave-valor em um dicionário quando a chave ainda não está presente no dicionário adiciona o par ao dicionário:

```
dict["key"] ← "new value"
print dict["key"]
→ "new value"
```

Vimos que, para que a busca em grafos A* seja garantidamente ótima, a heurística precisa ser consistente. Nesta questão, exploramos um novo procedimento de busca utilizando um dicionário para o conjunto fechado, chamado A*-graph-search-with-Cost-Sensitive-Closed-Set (A*-CSCS).

```
function A*-CSCS-GRAPH-SEARCH(problem, fringe, strategy) return a solution, or failure
    closed ← an empty dictionary
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe, strategy)
        if GOAL-TEST(problem, STATE[node]) then return node
        if STATE[node] is not in closed or COST[node] < closed[STATE[node]] then
            closed[STATE[node]] ← COST[node]
            for child-node in EXPAND(node, problem) do
                fringe ← INSERT(child-node, fringe)
            end
    end
```

Em vez de apenas inserir o último estado de um nó no conjunto fechado, agora armazenamos o último estado emparelhado com o custo do nó. Sempre que o CSCS considera expandir um nó, ele verifica o conjunto fechado. Somente se o último estado não for uma chave no conjunto fechado, ou se o custo do nó for menor do que o custo associado ao estado no conjunto fechado, o nó será expandido.

Q9.1. Para a busca em grafos regular, quais das seguintes afirmações são verdadeiras?

a) Se h é admissível, então busca em grafo A* encontra uma solução ótima
b) Se h é consistente, então busca em grafo A* encontra uma solução ótima

Q9.2. Selecione o que for verdadeiro sobre A*-CSCS:
a) Se h é admissível, então A*-CSCS encontra uma solução ótima
b) Se h é consistente, então A*-CSCS encontra uma solução ótima
c) Se h é admissível, então A*-CSCS vai expandir no máximo o mesmo número de nós que *busca em* ***árvore*** A*
d) Se h é consistente, então A*-CSCS vai expandir no máximo o mesmo número de nós que *busca em* ***árvore*** A*
e) Se h é admissível, então A*-CSCS vai expandir no máximo o mesmo número de nós que *busca em* ***grafo*** A*
f) Se h é consistente, então A*-CSCS vai expandir no máximo o mesmo número de nós que *busca em* ***grafo*** A*