Algoritmo de Busca em Largura e Algoritmo A* para Problema Labirinto

Iesley Bezerra dos Santos[1], Wladimir Araújo Tavares[1]

[1]Curso Ciência da Computação - Universidade Federal do Ceará (UFC)
Campus de Quixadá , Quixadá , Ceará , Brazil , CEP 63900-000

iesleybezerra@alu.ufc.br, wladimirufc@gmail.com

Abstract. *This paper presents the time results obtained by the A * and Width Search algorithms in solving the Maze Game problem. It describes both algorithms, and explains how both are applied to solve the Maze Problem and analyzes the times obtained by both.*

Resumo. *Este artigo apresenta os resultados de tempo obtidas pelos algoritmos A* e Busca em Largura na solução do problema Jogo do labirinto.Descreve os dois algoritmos, e explica como ambos são aplicados a resolução do Problema Labirinto e analisa os tempos obtidos por ambos.*

Palavras Chaves: *Algoritmo A estrela, Busca em Largura, Problema do Labirinto*

1. Introdução

Em muitos problemas computacionais, a solução do problema consiste em uma sequência de ações realizadas por um agente que levam de estado inicial para um estado alvo. Temos vários exemplos de problemas deste tipo: encontrar uma saída em um labirinto, resolver o problema dos missionários e canibais, entre outros. Essa sequência de ações é encontrada utilizando uma estratégia de busca através do espaço de busca do problema. Além disso, queremos ter algumas garantias com relação a estratégia de busca utilizada como completude, otimalidade, complexidade de tempo e espaço.

De maneira geral, podemos dividir as estratégias de busca em duas classes: busca não informada e busca informada. Neste artigo, propomos a utilização de um método de busca não informada (Algoritmo de Busca em Largura) e um método de busca informada (Algoritmo A*) para a resolução de uma variante de problema de Labirinto. O algoritmo de busca em largura tem a garantia de ser ótimo, completo e a complexidade de tempo e espaço no pior caso $O(b^d)$, onde b é o fator de ramificação e d é a profundidade da busca. Já o algoritmo A* garante ser completo, ótimo, complexidade de tempo e espaço no pior caso $O(b^d)$. Porém, A* é provado ser otimamente eficiente, ou seja, nenhum outro algoritmo ótimo expande menos nós que o A*.

O algoritmo A* foi publicado por Peter Hart, Nils Nilsson e Bertram Raphael em 1968. O algoritmo A* pode ser visto como uma extensão do algoritmo de caminho mínimo proposto por Dijkstra em 1959. Atualmente, o A* possui diversas variantes

como busca em profundidade iterativa A* (IDA*), Novo Bidirecional A*(NBA*) e Any Time Repairing A* (ARA*).

Na seção 2, apresentaremos a variante de problema de Labirinto que será resolvida. Na seção 3, apresentaremos a solução do problema utilizando a Busca em Largura. Na seção 4, discutiremos a solução utilizando o Algoritmo A*. Na seção 5, realizamos um breve experimento computacional comparando as duas soluções.

2. Variante do Problema de Labirinto.

Nessa variante do problema Labirinto, temos um quadriculado de células quadradas com *N* linhas e *M* colunas, onde cada célula contém um valor associado que varia de 0 a 9, um agente pode mover para células adjacentes desde que o valor da célula destino seja menor ou maior em uma unidade do que o valor da célula de origem, ou permanecer na célula, de modo que ao fim de cada ação temos um turno e o valor das células aumenta em uma unidade, de modo que ao chegar no valor 9 , o valor da célula no próximo turno vai para o valor 0. A solução do problema consiste em encontrar uma sequência de ações que levem o agente da célula (1,1) até a célula (*N,M*) com a menor quantidade de turnos possível. Para isso, o problema foi modelado como um grafo de estados, onde cada estado pode ser representado pela posição do agente no quadriculado e a quantidade de turnos até o agente chegar naquela posição.

O Algoritmo de busca genérico no grafo de estado pode ser apresentado da seguinte maneira:

```
ALGORITMO DE BUSCA GENÉRICO(PROBLEMA, FRONTEIRA)
nodes ← FRONTEIRA ( MAKE_NODE(INITIAL_STATE[PROBLEMA]))
repita
        Se nodes está vazio então retorne falha
        node ← REMOVE_FRONTEIRA(nodes)
        Se GOAL_TEST[problem] == STATE[node] então retorne node
        nodes ← ADICIONE_FRONTEIRA(problem, EXPAND(nodes))
fim
```

Observe o que muda entre as diversos algoritmos de busca usando o algoritmo de busca genérico será o tratamento da fronteira do problema.

3. Busca em Largura

O algoritmo de busca em largura, é muito utilizado para percorrer grafos, e pode ser adaptado para diversos fins, o seu funcionamento se dá de forma iterativa, onde

temos uma fila que guarda os estados que estão na fronteira da busca. Inicialmente colocamos o estado inicial na fila, e enquanto a fila não estiver vazia, retiramos um estado da fila, examinando os seus vizinhos, e colocando os que ainda não foram processados; dessa forma conseguimos percorrer todo o grafo sem entrar em ciclos.

Para o problema Jogo do Labirinto, na solução com o algoritmo de busca em largura, a fila guarda uma estrutura que armazena informações, como a posição da célula e a quantidade de turnos para se chegar até ela partindo-se da posição (1,1). Então, inicialmente colocamos a posição (1,1) com 0 turnos, tendo em vista que partimos dessa célula, e vamos colocando seus vizinhos de acordo com as regras de deslocamento no grafo, e calculando seus valores de turno de acordo com uma função de custo que calcula a quantidade mínima de turnos para que seja possível se deslocar entre a célula destino e a célula origem onde as duas são adjacentes. Para cada célula que alcançamos em determinado estado, adicionamos ela a fila desde que ela ainda não tenha entrado, ou o custo para ter chegado até ela no estado em questão que já se tinha obtido, for maior que o custo que acabamos de conseguir para alcançá-la, e neste caso, atualizamos o valor para a célula e estado, e a adicionamos novamente na fila. Para saber o menor valor para se atingir a célula (N,M) basta verificar se a célula atual que vamos expandir é a célula (N,M) e então pegar o seu valor de turno, e atualizar como solução caso seja o menor encontrado até então.

4. Algoritmo A*

O A*, é um algoritmo muito utilizado em IA para jogos, pois para qualquer grafo ele encontrará um caminho ótimo entre os estados inicial e final, caso exista um caminho entre esses estados. Seu funcionamento se assemelha em alguns aspectos a outros algoritmos de busca em grafos, mas seu diferencial está na heurística que usa para escolher o próximo nó a ser expandido entre os candidatos, para isso ele escolhe o nó com menor valor F(n) = H(n) + G(n) onde o G(n) é o custo para chegar até uma configuraç ao n, partindo da origem; e H(n) é o valor de uma função heurística que "subestima" o custo real para se chegar até o destino partindo de uma configuração n. O algoritmo A* mantém uma lista com os nós candidatos a serem expandidos.

O algoritmo é realizado da seguinte maneira:

1 - Coloca-se o nó origem na lista de candidatos

2 - Enquanto a lista de candidatos não estiver vazia, pega se o nó com menor custo F(n), e para todos os seus nós vizinhos, se eles ainda não tiverem sido verificados ou o custo para chegar até eles for menor que o custo que já se tinha, atualiza-se o custo; e adiciona o nó na lista de candidatos.

3 - O processo 2 se repete ate se atingir o nó destino

Para o problema Jogo do Labirinto, cada célula é um nó, e é levado em consideração o seu valor associado. A função heurística H(n), calcula a chamada distância de Manhattan, partindo-se do nó atual, até a origem, essa heurística garante

que o valor calculado é sempre o menor valor possível, e a heurística é válida pois não haverá valor possível menor que o calculado.

5. Conclusões

Para poder avaliar o desempenho dos algoritmos, foram realizados 10 testes com N = 100 e M = 100, em uma máquina com processador Intel Core i5-8250u, 8GB ram e sistema operacional Windows 10; e foram registrados os tempos de execução de cada um dos algoritmos, conforme tabela abaixo:

	A estrela	Busca em Largura
0	0.103	0.620
1	0.154	0.571
2	0.150	0.493
3	0.230	0.586
4	0.183	0.602
5	0.193	0.482
6	0.190	0.501
7	0.055	0.601
8	0.116	0.567
9	0.153	0.572

Conforme o que pode ser visto na tabela acima o algoritmo A* obteve os menores tempos de execução para todas os casos de testes, a média do Algoritmo A* foi de 0.1527 segundos e a média do Algoritmo Busca em Largura foi de 0.5595 segundos, e portanto podemos concluir que o Algoritmo A* é mais rápido que o Busca em Largura para o problema Jogo do Labirinto.

6. Referencias

Costa, Ernesto , Inteligência Artificial Fundamenstos e Aplicações

Jogo_do_Labirinto.Disponivel_em:<https://www.urionlinejudge.com.br/judge/pt/problems/view/2332>. Acessado em: 03. nov. 2019