Aplicação de A* para Saída de Labirinto

Gabriel Rodrigues de Sousa, Vinícius Andrade R. Miranda

Instituto de Educação Superior de Brasília (IESB)

(rsgabriel.pnt@gmail.com), (viniciusarmarm@gmail.com)

1. Introdução

2. Objetivo Geral

O objetivo geral do projeto é: utilizar as propriedades do algoritmo A* para realizar saída de labirintos, visando sempre o método mais performático possível.

3. Objetivo Específico

Com uma análise mais aprofundada do objetivo geral, podemos destacar os seguintes tópicos:

Explicar o conceito e a aplicação do Algoritmo A* (estrela);

Apresentar os casos de testes do algoritmo desenvolvido;

Detalhar a progressão geral do algoritmo escolhido;

Avaliar o desempenho em meio aos diferentes tipos de labirintos (tamanho, complexidade).

4. Referencial Teórico

4.1. BFS (Breadth-First Search)

Segundo o livro, A Linguagem da Programação Go: "Na teoria dos grafos, busca em largura (ou busca em amplitude, também conhecido em inglês por Breadth-First Search - BFS) é um algoritmo de busca em grafos utilizado para realizar uma busca ou travessia num grafo e estrutura de dados do tipo árvore. Intuitivamente, você começa pelo vértice raiz e explora todos os vértices vizinhos. Então, para cada um desses vértices mais próximos, exploramos os seus vértices vizinhos inexplorados e assim por diante, até que ele encontre o alvo da busca".

Mais especificamente, a Busca em Largura segue uma sequência de passos para que seja possível a análise do grafo como um todo. Inicialmente, para controlar a progressão da busca, podemos definir três tipos de cores representando a situação de

cada vértice, no qual todos os vértices inicialmente se tornam brancos para representar que nenhum deles foi descoberto ainda. Em segundo lugar, a cor cinza remete que o vértice foi descoberto, além disso, é considerado como uma " fronteira " entre vértices descobertos (pretos) e não descobertos (brancos). E por último, a cor preta significa que o vértice já foi descoberto.

Após a definição das funções de cada cor, o BFS irá percorrer uma árvore (grafo), definindo o vértice de origem. Após isso, ele determinará o vértice raiz como "u" e começará seu trajeto definindo para cada adjacente de "u" um vértice "v" representando seu parentesco (U = pai; V = filho), contudo, deve-se lembrar que o vértice descoberto tem no máximo um pai. Logo em seguida, atribuímos à um dos vértices filho à condição de pai, e assim repete-se o processo até que o grafo seja completamente mapeado.

4.2. Busca Heurística

Métodos Heurísticos são algoritmos exploratórios que almejam a resolução de problemas. Muitas vezes não realizam a solução mais performática/simples, sendo assim classificados como "busca cega". Segundo um trabalho realizado pelo Instituto Federal de Santa Catarina (IFSC): "Uma solução ótima de um problema nem sempre é o alvo dos métodos heurísticos, uma vez que, tendo como ponto de partida uma solução viável, baseiam-se em sucessivas aproximações direcionadas a um ponto ótimo".

Contudo, para garantir que a busca heurística promova o melhor caso possível para o programa deve-se utilizar a seguinte expressão: f(n) = g(n) + h(n), onde g(n) representa a distância de n ao nó principal e o h(n) representa a distância estimada de n ao nó final.

4.3. Algoritmo de Dijkstra

De acordo com um trabalho realizado por alunos de Ciência da Computação da Universidade do Cruzeiro do Sul, os autores afirmam que: "O algoritmo considera um conjunto S de menores caminhos, iniciado com um vértice inicial I. A cada passo do algoritmo busca-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a I e adiciona-o a S e, então, repetindo os passos até que todos os vértices alcançáveis por I estejam em S. Arestas que ligam vértices já pertencentes a S são desconsideradas".

De certa forma, o algoritmo de Dijkstra possui certa semelhança ao BFS, já que seu conjunto S corresponde exatamente ao conjunto de vértices descobertas (pretas) na Breadth-First Search, porém, podemos afirmar que este algoritmo não usa especificamente o BFS como método de busca. Entretanto, o fato do algoritmo de Dijkstra verificar todas as possibilidades do grafo, o torna um algoritmo menos performático para realizar a saída de um labirinto.

4.3. Algoritmo A*

O algoritmo A* é provavelmente um dos Best-First com o melhor método para realizar a saída de um labirinto já que dispõe da combinação heurística da Busca em Largura (BFS) e da formalidade do Algoritmo de Dijkstra. O método de busca pelo A* pode ser considerado completo, eficiente e ótimo, já que não há outro algoritmo capaz de garantir uma expansão menor de nós do que o próprio A*, o tornando então um dos algoritmos de busca mais rápidos atualmente.

Para uma explicação clara de como funciona o Algoritmo Estrela, pense na seguinte ocasião: Uma pessoa está na cidade A e pretende ir o mais rápido possível para a cidade D, entretanto, ela precisa passar por uma das seguintes cidades: B e C (de acordo com a figura 1). Como ela decidirá qual será o caminho mais curto? Como citado anteriormente (tópico 4.2), considere que f(n) = g(n) + h(n) será a expressão utilizada para resolver nosso problema e teremos os seguintes dados: O valor Heurístico de A = 5, B = 20, C = 10, D = 2. A distância do ponto A para o ponto B são 15 unidades, e de A para C são de 10 unidades. Com essas informações, vamos ao primeiro passo, que é calcular qual a menor distância entre A->B ou A->C.

$$
\begin{aligned}
&f(n) = g(n) + h(n) \\
&\text{A}\rightarrow\text{B}: f(n) = 15 + 20 \quad \rightarrow \quad f(n) = 35 \\
&\text{A}\rightarrow\text{C}: f(n) = 10 + 10 \quad \rightarrow \quad f(n) = 20
\end{aligned}
$$

Primeiramente verificamos que de A para B nós temos f(n) = 35 e de A para C temos f(n) = 20. Como podemos analisar que a segunda opção possui um valor menor, nosso nodo passa a ser o C e seguimos adiante com nossa verificação. Visto que o único nó adjacente de C é o D (nosso objetivo de chegada), aplicamos novamente a função para obter o custo final do trajeto.

$$
\begin{aligned}
&f(n) = g(n) + h(n) \\
&\text{C}\rightarrow\text{D}: f(n) = 30 + 2 \quad \rightarrow \quad f(n) = 32
\end{aligned}
$$

Em relação ao resultado do nosso segundo passo, vimos que de C->D: f(n) = 32, já que somamos a distância de A->C(10) + C->D(20) + Valor Heurístico de D (2). Como chegamos no nosso objetivo após percorrer todo o grafo, concluímos que nosso Caminho Solução será de: A->C->D com o custo de 32. Por conseguinte, esse será o caminho mais curto a ser percorrido.

Após a análise da questão proposta, podemos ressaltar as seguintes detalhes que comprovam a eficiência do algoritmo, como exemplo:

Desempenho de tempo será exponencial para a resolução do problema, entretanto, o uso de boas funções heurísticas diminuem drasticamente esse custo.

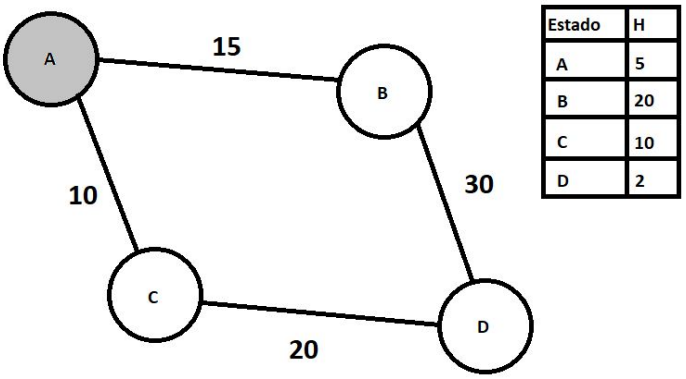
O custo de memória será expressado como: O (b^d), onde irá guardar todos os nós expandidos na memória, possibilitando então o backtracking.

O A* irá expandir os nós somente com f(n) <= f*, onde f* é o custo do caminho ótimo (f é crescente).

4.4. Python

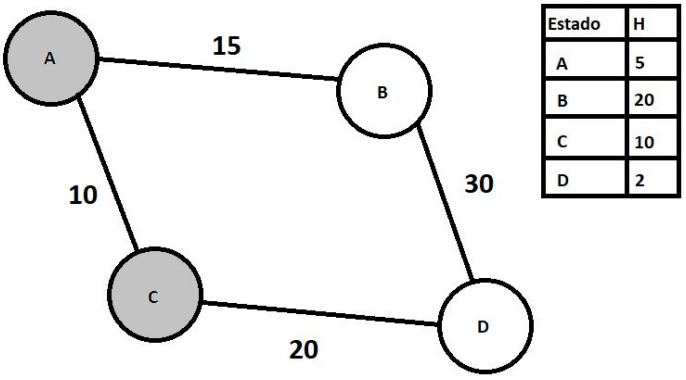
5. Imagens

Representação 4.3



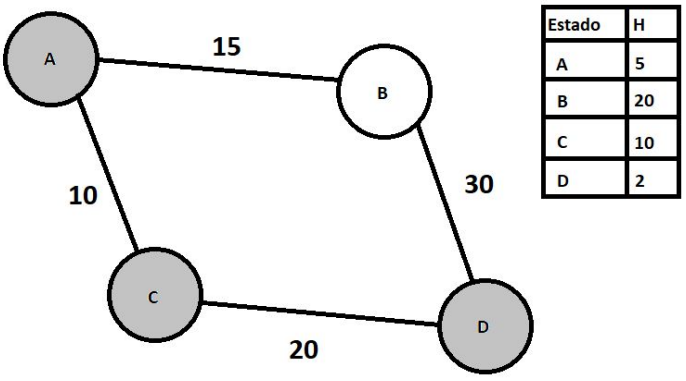
Estado	H
A	5
B	20
C	10
D	2

Figura 1



Estado	H
A	5
B	20
C	10
D	2

Figura 1.1



Estado	H
A	5
B	20
C	10
D	2

Figura 1.2

6. Referências