Universidade Estadual do Oeste do Paraná – UNIOESTE

Centro de Engenharias e Ciências Exatas – CECE

Disciplina de Inteligência Artificial

Docentes: Huei D. Lee e Newton Spolaôr

Isabela Pimentel Loebel

Nickolas Crema

Relatório Projeto 2:

Algoritmo de Busca

FOZ DO IGUAÇU

2024

Sumário

1. A* – Melhor solução

O algoritmo A* (A-Estrela) é a técnica de busca mais usada atualmente, sendo um algoritmo de busca informada, ou busca de menor caminho, o que significa que é formulado em termos de grafos ponderados: partindo de um nó inicial específico de um grafo, ele visa encontrar um caminho para um determinado nó objetivo com o menor custo. Por muitos considerado como uma extensão do algoritmo do caminho mais curto de *Dijkstra.*

Assim como os autores Hart, Nilsson e Raphael (1968) apresentam, afirma-se que o algoritmo A*, ao contrário de outras técnicas de travessia, é um algoritmo inteligente, o que o separa dos outros algoritmos convencionais. Em cada etapa da execução o algoritmo escolhe o nó de acordo com um valor *f()* que é um parâmetro igual à soma de dois outros parâmetros *g()* e *h()*. Em cada etapa, ele escolhe o nó com o *f()* mais baixo e processa esse nó.

Define-se *g()* e *h()* como:

- g: Custo de movimento para se deslocar do ponto inicial até um determinado nó, seguindo o caminho gerado para chegar lá.
- h: Denominada como função heurística, é o custo estimado de movimento para se deslocar daquele nó até o nó destino.

Certamente *g()* é facilmente calculável, enquanto *h()* não é tão simples assim. Existem dois métodos para se obter o valor de *h()*, o primeiro que é determinar seu valor exato (o que pode ser bem demorado), ou obter uma aproximação de seu valor, essa aproximação pode ser feita das seguintes maneiras:

- Distância de Manhattan: É o total dos valores absolutos das diferenças entre as coordenadas X e Y dos nós atuais e nó alvo. Dada pela fórmula:

  $$h = |nó\_atual.X - nó\_destino.X| + |nó\_atual.Y - nó\_destino.Y|$$

  É utilizado este método heurístico quando só é possível mover-se em quatro direções – para cima, esquerda, direita e para baixo.
- Distância Diagonal: O maior valor absoluto das diferenças entre as coordenadas X e Y do nó atual e do nó alvo. Representado pela fórmula:

  $$dX = |nó\_atual.X - nó\_destino.X|$$
  $$dY = |nó\_atual.Y - nó\_destino.Y|$$
  $$h = D * (dX + dY) + (D2 - 2 * D) * min(dX, dY)$$

  Onde D é o comprimento de cada nó e D2 é a diagonal

  Este método é utilizado quando é permitido mover-se apenas em oito direções, como os movimentos do Rei no Xadrez.

- Distância Euclidiana: É a distância entre o nó objetivo e o nó atual. Representado pela fórmula da distância:

  $$h = \sqrt[2]{(nó\_atual.X - nó\_destino.X)^2 + (nó\_atual.Y - nó\_destino.Y)^2}$$

  É usado este método heurístico quando é permitido se mover em qualquer direção de escolha.

2. Dijkstra – Segunda melhor solução

O algoritmo de Dijkstra é um algoritmo de busca em grafos que encontra o caminho mais curto entre um nó inicial e todos os outros nós em um grafo com pesos não negativos. Ele foi proposto por Edsger W. Dijkstra em 1956 e publicado três anos depois, e é amplamente utilizado em várias aplicações, como roteamento de redes, sistemas de navegação e otimização de caminhos em sistemas de transporte.

O algoritmo de Dijkstra marca inicialmente a distância do ponto inicial a todos os outros nós do grafo com o infinito (a distância entre ponto inicial para ponto inicial é marcada com 0), com objetivo de identificar que os nós ainda não foram visitados.

Para a primeira iteração, o nó atual será o ponto inicial e a distância até ele será zero. Para iterações subsequentes, o nó atual será o nó não visitado mais próximo do ponto inicial, a partir do nó atual, atualiza a distância para cada nó não visitado que seja diretamente conectado a ele.

Isso é feito determinando a soma da distância entre um nó não visitado e o valor do nó atual e, em seguida, renomeando o nó não visitado com esse valor se for menor que o valor atual do nó não visitado. Com isso, o nó é renomeado se o caminho até ele através do nó atual for mais curto que os caminhos anteriormente conhecidos.

Sendo assim, o algoritmo pode ser estruturado como os seguintes passos:

1. Inicialização: Comece definindo o nó inicial como o nó de origem. Marque a distância do nó inicial para si mesmo como 0 e todas as outras distâncias como infinito (ou uma grande constante, dependendo da implementação). Inicialize uma lista de nós não visitados contendo todos os nós do grafo.
2. Iteração: Enquanto houver nós não visitados na lista, continue o processo de iteração. Selecione o nó não visitado com a menor distância conhecida (ou seja, o nó mais próximo do nó inicial) como o próximo nó a ser visitado.
3. Atualização das distâncias: Para cada nó vizinho do nó selecionado na etapa anterior, calcule a distância acumulada até esse nó vizinho através do nó selecionado. Se essa distância for menor do que a distância atualmente conhecida para o nó vizinho, atualize a distância com o novo valor calculado.
4. Marcação como visitado: Após atualizar as distâncias de todos os vizinhos do nó selecionado, marque-o como visitado e remova-o da lista de nós não visitados.
5. Condição de parada: Repita os passos 2 a 4 até que todos os nós tenham sido visitados. O algoritmo termina quando todos os nós foram visitados ou quando a distância para o nó destino não muda mais (em casos em que apenas o caminho mais curto até um determinado nó é necessário).
6. Recuperação do caminho mais curto: Após a conclusão do algoritmo, o caminho mais curto do nó inicial para qualquer outro nó no grafo pode ser recuperado seguindo as arestas e os nós marcados durante o processo.

Por fim, ele garante encontrar o caminho mais curto de um nó inicial para todos os outros nós em um grafo direcionado ou não direcionado com pesos não negativos. No entanto, ele não é adequado para grafos com pesos negativos ou para encontrar o caminho mais curto em grafos com ciclos de peso negativo, já que a presença de tais ciclos pode resultar em loops infinitos durante a execução do algoritmo.

3. DFS – Pior solução

A pesquisa em profundidade (DFS) constitui um algoritmo busca não informada, o que significa que expande e examina de maneira sistemática todos os vértices de um grafo, seja ele direcionado ou não-direcionado. Em termos mais precisos, o algoritmo conduz uma busca exaustiva no grafo, percorrendo todas as arestas e vértices presentes. Portanto, o algoritmo deve assegurar que nenhum vértice ou aresta seja visitado mais de uma vez. Para atender a esse requisito, emprega-se uma estrutura de dados, que mantém a ordem de chegada dos vértices.

Utilizado para percorrer ou buscar informações em uma estrutura de dados do tipo grafo ou árvore. Ele explora tão longe quanto possível ao longo de um ramo antes de retroceder (*backtrack*). Pode ser estruturado seguindo os passos:

1. Inicialização: Escolha de um nó inicial para começar a busca. Marcação desse nó como visitado.
2. Visitação dos vizinhos: Para o nó atual, é feita a escolha de um dos vizinhos ainda não visitados. Se não houver vizinhos não visitados, deve-se retroceder para o nó anterior. Se houver um vizinho não visitado, deve mover-se para esse vizinho e marcá-lo como visitado.
3. Recursão: Repetição do processo para o novo nó visitado. Ou seja, visita-se um dos vizinhos não visitados e marque-o como visitado. Continue esse processo até que não haja mais vizinhos não visitados.
4. Retrocesso (*Backtrack*): Quando não houver mais vizinhos não visitados para o nó atual, retrocede-se para o nó anterior e repete o passo 2 até que todos os nós tenham sido visitados.
5. Conclusão: O algoritmo termina quando todos os nós alcançáveis a partir do nó inicial foram visitados.

O resultado de uma busca em profundidade de um grafo pode ser convenientemente descrito em termos de uma árvore geradora mínima dos vértices alcançados durante a busca. Entretanto, o algoritmo DFS não garante que o caminho encontrado seja o mais curto entre o nó inicial e o nó objetivo. Ele pode encontrar um caminho que não seja o mais curto, dependendo da ordem em que os nós são visitados, sendo assim, é mais adequado para problemas onde encontrar uma solução é mais importante do que encontrar a solução ótima.

4. BFS – Segunda pior solução

O algoritmo BFS (*Breadth-First Search*), ou busca em largura, é um algoritmo utilizado para percorrer ou buscar informações em uma estrutura de dados do tipo grafo. Ao contrário do DFS, que explora em profundidade, o BFS explora os vértices de um grafo em "camadas", começando pelo vértice inicial e visitando todos os seus vizinhos antes de avançar para os vizinhos dos vizinhos.

Seguindo a declaração de Berge (1985), o algoritmo segue os seguintes passos:

1. Inicialização: Comece escolhendo um vértice inicial para iniciar a busca. Marque esse vértice como visitado e insira-o em uma fila.
2. Exploração de vizinhos: Enquanto a fila não estiver vazia, remova um vértice da fila. Para cada vértice removido, examine todos os seus vizinhos não visitados. Marque cada vizinho não visitado como visitado e insira-o na fila.
3. Repetição: Repita o passo 2 até que não haja mais vértices na fila para serem explorados.
4. Conclusão: O algoritmo termina quando todos os vértices alcançáveis a partir do vértice inicial foram visitados.

O funcionamento do algoritmo BFS pode ser descrito em termos de um processo iterativo, onde cada iteração representa uma nova "camada" de nós visitados a partir do nó inicial. Durante cada iteração, os nós são visitados na ordem em que foram descobertos, utilizando uma estrutura de dados de fila (*queue*) para garantir que o primeiro nó descoberto seja o primeiro a ser explorado. Isso garante que o algoritmo visite todos os nós em um mesmo nível antes de prosseguir para o próximo nível, o que pode ser útil para encontrar o caminho mais curto entre dois vértices em um grafo não ponderado.

Entretanto, algoritmo BFS pode exigir uma quantidade significativa de memória, especialmente em grafos grandes. Isso ocorre porque ele mantém uma fila de nós a serem explorados, o que pode resultar em uma grande quantidade de armazenamento necessário para grafos com muitos nós. Essencialmente, o BFS é mais adequado para grafos não ponderados, onde todas as arestas têm o mesmo custo. Em grafos ponderados, onde as arestas têm diferentes custos, o BFS pode não encontrar o caminho mais curto entre dois vértices.

Berge (1985) ainda afirma que é importante notar que o BFS também pode ser utilizado para encontrar componentes conectados em um grafo não direcionado, bem como para a detecção de ciclos em um grafo direcionado, entre outras aplicações.

5. Validação e testes

Para realizar tanto a implementação quanto a validação e testes dos algoritmos foi desenvolvido um grafo georreferenciado na cidade de Foz do Iguaçu, que abrange parte dos bairros Conjunto Libra e Campos do Iguaçu, podendo ser visualizado na Figura 1.

Para o desenvolvimento do grafo foi utilizado uma malha de linhas georreferenciadas (conjunto de dados geográficos do tipo linha) que representam as ruas da cidade de Foz do Iguaçu, onde a partir do uso do *software* ArcGIS Pro e de ferramentas de geoprocessamento foi selecionado uma pequena parcela da malha, na qual se situam os bairros citados, em cima dessa seleção foi criado um ponto georreferenciado (dado geográfico do tipo ponto) em cada intersecção das linhas e em cada final de linha.

Os dados geográficos resultantes dessas operações foram armazenados em um banco de dados geográfico, posteriormente carregados na memória via script Python e escritos em um arquivo texto no formato adequado.

Figura 1 – Grafo georreferenciado nos bairros Campos do Iguaçu e Conjunto Libra.

O georreferenciamento do grafo surgiu com o intuito de se obter uma heurística adequada para a resolução do problema. A função heurística utilizada foi obtida a partir do cálculo da distância euclidiana entre todos os pontos do grafo e o ponto final que se deseja alcançar. Definida pelas funções descritas nos segmentos de código a seguir:

```
def pointSqrDistance(current, goal):
    """Realiza a soma da diferença dos pontos ao quadrado"""
    return (current.X - goal.X)**2 + (current.Y - goal.Y)**2

def heuristic(current, goal):
    """
    Calcula a distância euclidiana entre os dois pontos com
    base em suas coordenadas X e Y, dadas pela propriedade point()
    """
    return pointSqrDistance(current.point(), goal.point())**0.5
```

A função denominada "*heuristic*" foi executada para todos os pontos do grafo em cima de *goal* (variável que armazena o destino), tal ato pode ser visualizado no segmento de código a seguir:

```
for vertice in graph.vertices:
    heuristic(vertice, goal)
```

Para a solução do problema disposto, acordou-se que as métricas seriam a quantidade de nós explorados para encontrar a solução e o tamanho da solução encontrada, visto que a quantidade de nós explorados se refere à proficiência do algoritmo de oferecer a solução mais eficiente, permitindo a busca pelo caminho mais curto com menor poder de processamento. Quanto ao tamanho da solução encontrada, se refere ao tamanho do caminho encontrado, visto que o algoritmo DFS é incapaz de obter o menor caminho entre dois pontos e que o algoritmo BFS define o menor caminho em número de arestas.

Com base nessas métricas estabelecidas, o algoritmo A* foi implementado como a principal abordagem para otimização, enquanto o algoritmo de Dijkstra foi considerado como uma alternativa. Isso se deve ao fato de que o algoritmo A* demonstra uma eficácia superior, especialmente quando aplicado a distâncias reais, beneficiando-se de uma heurística mais precisa e, consequentemente, oferecendo uma solução mais eficiente em termos de tempo computacional. Embora o algoritmo de Dijkstra também forneça uma solução ótima, ele é superado pelo A* em termos de eficiência computacional quando se tem uma heurística adequada.

Os algoritmos DFS e BFS foram implementados e propostos como alternativas para a solução menos eficiente. O algoritmo DFS é sugerido como a opção principal devido à sua incapacidade de determinar o menor caminho. Este algoritmo, muitas vezes, segue um caminho no nível mais profundo até encontrar o caminho desejado, o que pode resultar em soluções menos satisfatórias. Por outro lado, o algoritmo BFS é sugerido como uma alternativa devido à sua tendência de encontrar o menor caminho em termos de número de arestas. No entanto, ele não leva em consideração os pesos das arestas, o que pode levar a soluções que têm um menor número de arestas, mas não necessariamente uma menor distância.

6. Referências

RAVIKIRAN, A. S. A* Algorithm Concepts and Implementation. 2024. Disponível em: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm>. Acesso em: 29 de jan de 2024.

STANFORD THEORY. Introduction to A*. 2009. Disponível em: <https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>. Acesso em: 29 de jan de 2024.

GEEKS FOR GEEKS. A* Search Algorithm. 2024. Disponível em: <https://www.geeksforgeeks.org/a-search-algorithm/>. Acesso em: 29 de jan de 2024.

EVEN, S. Graph algorithms. Rockville: Computer Science Press, 1979. (Computer Software Engineering Series).

BERGE, C. Graphs. 2nd ed. Amsterdam: NorthHolland, 1985. 413 p. (North-Holland Mathematical Library, v. 6, pt. 1).

HART, P. E., NILSSON, N. J., & RAPHAEL, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2).

Edsger W. Dijkstra. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik, 1(1), 269-271.

FEOFILOFF P. Busca em Profundidade (DFS). (2019). IME USP. Disponível em: <https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/dfs.html>. Acesso em: 29 de jan de 2024.

GEEKS FOR GEEKS. What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm. (2024). Disponível em: <https://www.geeksforgeeks.org/introduction-to-dijkstras-shortest-path-algorithm/>. Acesso em: 29 de jan de 2024.

WIKIPEDIA. Breadth-first Search. (2024). Disponível em: <https://en.wikipedia.org/wiki/Breadth-first_search>. Acesso em: 29 de jan de 2024.

AERO ENGENHARIA. O que é: Distância Euclidiana. (2024). Disponível em: <https://aeroengenharia.com/glossario/o-que-e-distancia-euclidiana/>. Acesso em: 29 de jan de 2024.

LEE H. D. ESTRATÉGIAS DE BUSCA NÃO INFORMADA (Exaustiva ou Cega). (2024). Slides de aula. Acesso em: 29 de jan de 2024.

LEE H. D. ESTRATÉGIAS DE BUSCA INFORMADA (Parte 1). (2024). Slides de aula. Acesso em: 29 de jan de 2024.

LEE H. D. ESTRATÉGIAS DE BUSCA INFORMADA (Parte 2). (2024). Slides de aula. Acesso em: 29 de jan de 2024.