Comparação de Algoritmos de Busca: Dijkstra, A* e Dijkstra Bidirecional

Luryan Delevati Dorneles

16 de abril de 2025

Resumo

Este trabalho apresenta a implementação e comparação de três algoritmos de busca (Dijkstra, A* e Dijkstra Bidirecional) para encontrar o caminho mais curto entre dois pontos em um grafo modelado a partir de coordenadas conhecidas. O grafo foi construído com base em um raio de conexão e incluiu uma penalidade populacional para priorizar rotas por cidades menos populosas. Os resultados mostram as diferenças de desempenho entre os algoritmos e destacam os desafios e limitações também dos modelos utilizados.

1 Introdução

O trabalho foi desenvolvido com finalidade de implementar e comparar três algoritmos de busca para encontrar o caminho mais curto entre dois pontos dentro de um grafo conexo. O grafo foi modelado a partir das coordenadas conhecidas de cidades do estado de Alagoas, onde as conexões entre elas dependem de um raio de distância. Além do raio, foi adicionada uma penalidade baseada na população das cidades para influenciar na escolha das rotas, sendo considerado o critério de desempate em situações de tomada de decisão entre nós que os métodos irá aceitar.

2 Metodologia

A implementação foi realizada em Python, utilizando estruturas de dados simples como listas, dicionários e a biblioteca ‘heapq‘ para filas de prioridade. Abaixo, explico os passos principais:

2.1 Coleta e Tratamento de Dados

Após obter os dados obtidos através portal de Dados Abertos do governo, estes foram tratados e reestruturados em um CSV, após ajustes de codificação, projeção de coordenadas e remoção de ”trash”content, a tabela foi transformada em um JSON com todas as cidades e informações de relevância para o problema. Esse passo é indispensável para garantir dados limpos e no sistema de referência(coordenadas) correto.

2.2 Modelagem do Grafo

O grafo foi construído a partir de um arquivo JSON contendo informações sobre as cidades, como coordenadas geográficas e população. As conexões entre as cidades foram determinadas com base na distância euclidiana, considerando um raio máximo r. Cada aresta do grafo possui um peso igual à distância entre as cidades, com um fator adicional de penalidade proporcional à população da cidade de destino.

2.3 Sobre os Algoritmos Implementados

Os algoritmos foram implementados com base nas seguintes idéias principais:

- **Dijkstra**: Expande a busca de forma uniforme a partir de um nó inicial, procura encontrar o caminho mais curto com pesos não-negativos. Foi eficiente para encontrar a solução ótima, mas explorou nós desnecessários por não ter direcionamento na busca.
- **A***: Similar ao Dijkstra, mas ele utiliza uma heurística (distância em linha reta até o destino) assim ele consegue priorizar os nós que mais prometem bons resultados. Quando a heurística é aceitável, também garante o caminho mais curto, e foi mais rápido que Dijkstra por explorar menos nós desnecessários.
- **Dijkstra Bidirecional**: Realiza duas buscas simultâneas, uma indo e uma vindo. Ou seja, uma parte da origem e outra parte do destino, assim o espaço de busca é "menor". É particularmente útil em grafos grandes quando o destino é conhecido, pois pode encontrar um ponto de encontro antes de explorar a totalidade do grafo.

2.4 Penalidade Populacional

Para incluir a penalidade populacional, o custo de cada aresta foi ajustado com base na população da cidade de destino. O custo total de uma aresta foi calculado desta forma:

$$\text{Custo Total} = \text{Distância} + (\text{População} \times \text{Fator de Penalidade})$$

Essa implementação permitiu que os algoritmos priorizem as rotas que passem por cidades menos populosas quando as distâncias são semelhantes.

3 Resultados

Os testes foram realizados em um grafo com 101 cidades, utilizando um raio fixo de $r = 0.22$ (aproximadamente 24.42 km). Esse valor de 'r' foi pre-definido na main como constante no código principal do pipeline. A Tabela 1 apresenta os resultados para uma rota específica, comparando os algoritmos em termos de distância total, número de nós visitados e tempo de execução.

3.1 Análise dos Resultados

- **Distância Total**: Todos os algoritmos encontraram a mesma rota ótima, com pequenas diferenças devido a arredondamentos.

Tabela 1: Comparação entre os algoritmos

Algoritmo	**Distância (km)**	**Nós**	**Tempo (s)**	**Mem. (KB)**
Dijkstra	114.24	44	0.0023	12.16
A*	114.24	9	0.0017	13.79
Bidirecional	113.85	35	0.0214	17.73

- **Nós Visitados**: O A* foi o mais eficiente, visitando menos nós graças à heurística. O Dijkstra Bidirecional também reduziu o número de nós visitados em comparação ao Dijkstra padrão.
- **Tempo de Execução**: O A* foi o mais rápido, seguido pelo Dijkstra. O Dijkstra Bidirecional teve um desempenho inferior devido ao maior consumo de memória e à necessidade de gerenciar duas buscas simultâneas.
- **Uso de Memória**: O uso de memória foi medido para cada algoritmo, com o Dijkstra Bidirecional apresentando o maior consumo (17.73 KB para 102 nós). Esses valores indicam que, embora o uso de memória seja linear, outros fatores, como tempo de execução, podem limitar a escalabilidade antes de atingir o limite de memória.

4 Desafios e Limitações

4.1 Desafios

- **Eficiência na Busca de Vizinhos**: A busca por vizinhos foi implementada de forma ingênua, verificando todas as cidades para determinar quais estão dentro do raio r. Isso resultou em uma complexidade $O(N)$ para cada nó, o que pode ser ineficiente para grafos maiores.
- **Modelagem Simples**: O uso de distância euclidiana e raio fixo simplifica o problema, mas não reflete a realidade de estradas e obstáculos geográficos.

4.2 Limitações

- **Dados Simples**: As informações se limitam a população e localização, sem considerar estradas, custo financeiro ou fatores de tráfego.
- **Escalabilidade**: Em grafos muito grandes, tanto a etapa de procura de vizinhos quanto o consumo de memória podem se tornar problemas.

5 Conclusão

Os algoritmos implementados conseguiram encontrar o caminho mais curto entre dois pontos no grafo, com o A* se destacando em termos de eficiência. A inclusão da penalidade populacional adicionou uma dimensão interessante ao problema, permitindo priorizar rotas por cidades menos populosas.

Futuras melhorias incluem o uso de dados reais de estradas, a implementação de estruturas de dados mais eficientes para consultas espaciais e a consideração de outros fatores de custo, como tempo de viagem e tráfego.