Milena Lucas dos Santos e Vivian Miwa Fugihara

Algoritmos de Busca - Relatório Técnico de Inteligência Artificial

FOZ DO IGUAÇU
16 de fevereiro de 2022

Sumário

Lista de ilustrações

1 Introdução

Para o presente trabalho, foi proposto que no mínimo dois algoritmos fossem implementados, um algoritmo deve gerar a melhor solução para o problema de encontrar caminhos entre uma posição inicial e final. O outro algoritmo deve apresentar a pior solução para o mesmo problema.

O objetivo desse trabalho é ter um conhecimento geral de algoritmos de busca e identificar quais algoritmos trazem o resultado esperado para o problema. Para este trabalho, foram implementados três algoritmos, que são: Busca em Profundidade (Depth-First Search - DFS), Busca em Largura (Breadth-First Search - BFS) e Bellman-Ford.

Esses três algoritmos foram implementados na linguagem C++, pois essa linguagem está mais familiarizada pelos autores e por terem conhecimento de implementação de grafos na mesma linguagem. A estrutura de dados de grafos representada no trabalho são duas classes, chamadas de *NodeGraph* e *Graph*, onde na classe *NodeGraph* estão as informações de vértices e pesos e na classe *Graph* está o vetor de listas de *NodeGraph*.

2 Busca em Profundidade e Largura

A busca em profundidade é usado para buscar itens dentro das estruturas de dados grafos ou árvores. Sua característica fundamental é passar por todos os vértices filhos ao vértice raiz o mais profundo possível para somente depois retroceder [1].

As arestas do grafo estão representadas por meio de uma lista de adjacência e o algoritmo é implementado de forma recursiva. A lógica básica implica em primeiro criar uma pilha e empilhar o vértice inicial no topo. Enquanto a pilha não estiver vazia, percorra os vértices adjacentes do vértice inicial. Caso um vértice não seja o vértice buscado, e ainda não tenha sido visitado, empilhe-o.

Como justificativa para a escolha desse algoritmo, a busca em profundidade pode ser usado para encontrar saídas de labirintos ou encontrar um caminho entre dois pontos em um software de GPS [1], no entanto, podemos atravessar mais arestas para alcançar um vértice de destino de uma origem. Além disso, o algoritmo não é completo e nem ótimo para grafos muito profundos ou infinitos.

A busca em largura é um algoritmo utilizado para percorrer ou buscar itens dentro das estruturas de dados grafos ou árvores. A principal característica é que a busca sempre ocorre nos filhos ou vértices mais próximos ao vértice pelo qual a busca foi iniciada [2].

As arestas do grafo estão representadas por meio de uma lista de adjacência e o algoritmo é implementado de forma iterativa. A lógica básica implica em primeiro criar uma fila e enfileirar o vértice inicial. Enquanto a fila não estiver vazia, percorra os vértices adjacentes do vértice inicial. Caso um vértice não seja o vértice buscado, e ainda não tenha sido visitado, adicione-o à fila.

Como justificativa para a escolha desse algoritmo, a busca em largura é aplicado em soluções de coleta de lixo (*garbage collection*), serialização/deserialização de árvores binárias, entre outros [2]. Suas aplicações não estão relacionadas ao problema do presente trabalho, é completa se fator de ramificação é finito e ótima, se custos dos passos iguais. Nesse caso, largura pode ser completa, mas não ótima, pois custos das arestas podem ser diferentes.

Ambos algoritmos representam a pior solução para o problema de caminho de personagens em jogos. Com a implementação dos algoritmos e métricas de desempenho, podemos ver mais claramente a diferença em ambos algoritmos.

3 Bellman-Ford

O algoritmo de Bellman-Ford foi criado por Richard Bellman e Lester Ford, e é geralmente utilizado em casos que se deseja buscar o melhor caminho. Seu funcionamento se dá superestimando o comprimento do caminho do vértice de origem para os demais vértices. Feito isso, ele utiliza a técnica de relaxamento, ou seja, faz sucessivas deduções das distâncias, encontrando caminhos mais curtos que os anteriores até achar o menor de todos.

O algoritmo calcula o caminho mais curto de um vértice em um grafo para todos os outros vértices em questão, mas, neste caso, foram utilizados apenas um vértice de origem e um de destino, além dos demais. Além disso, ele vai para cada aresta em cada iteração, em vez de examinar apenas os vizinhos diretos de um vértice, como no algoritmo de Dijkstra [3].

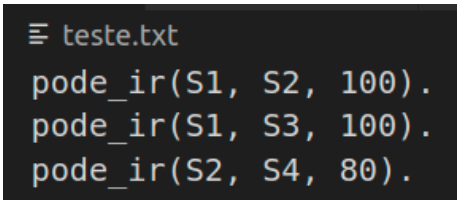
Bellman-Ford é um algoritmo para grafos orientados, motivo pelo qual foi feita a sua escolha, além do fato de considerar pesos para cada aresta, que podem ser tantos

negativos, quanto positivos.

4 Métricas de avaliação e desempenho

O arquivo de entrada está de acordo com o padrão apresentado na Figura 1.

Figura 1 – Arquivo de entrada.



Após ler o arquivo, é possível verificar se o arquivo foi lido corretamente na opção "Mostrar grafo". Em seguida, podemos executar os algoritmos de busca.

Para o cálculo do desempenho foi implementado a classe *ScopeTimer*, onde é utilizado a biblioteca *Chrono* [4]. Essa biblioteca tem acesso a alguns relógios diferentes na máquina, cada um deles com finalidades e características diferentes, onde é possível calcular um escopo. Essa biblioteca utiliza o *Wall Time* como medida. O *Wall Time* pode ser descrito como o tempo que você pode medir com um cronômetro, supondo que você consiga iniciá-lo e pará-lo exatamente nos pontos de execução desejados [5].

Outras medidas consideradas são a distancia percorrida entre a sala inicial e final e o número de sala visitadas pelo o algoritmo.

5 Resultados

O mapa/labirinto (grafo) utilizado para gerar os resultados e as avaliações do mesmo está representado na Figura 2.

Com a execução de ambos algoritmos, na posição inicial: s0 e final: s18. Ao encerrar o programa, é gerado um arquivo chamado *relatorio.txt*, com o caminho percorrido pelo algoritmo, o número de salas visitadas e o tempo de execução. Na Figura 3, é apresentado os resultados da execução de ambos algoritmos no arquivo gerado pelo programa.

Para a melhor solução, foi escolhido o algoritmo de Bellmanford, na Figura 4 apresentado o resultado com os mesmos parâmetros anteriores.

Figura 2 – Grafo para a execução de testes.

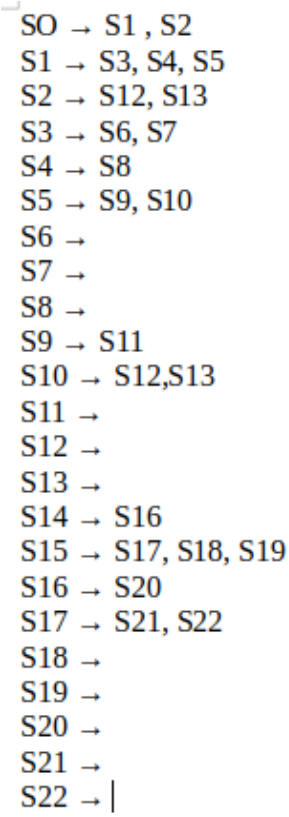


Figura 3 – Resultado gerado das buscas de largura e profundidade, com s0 e s18.

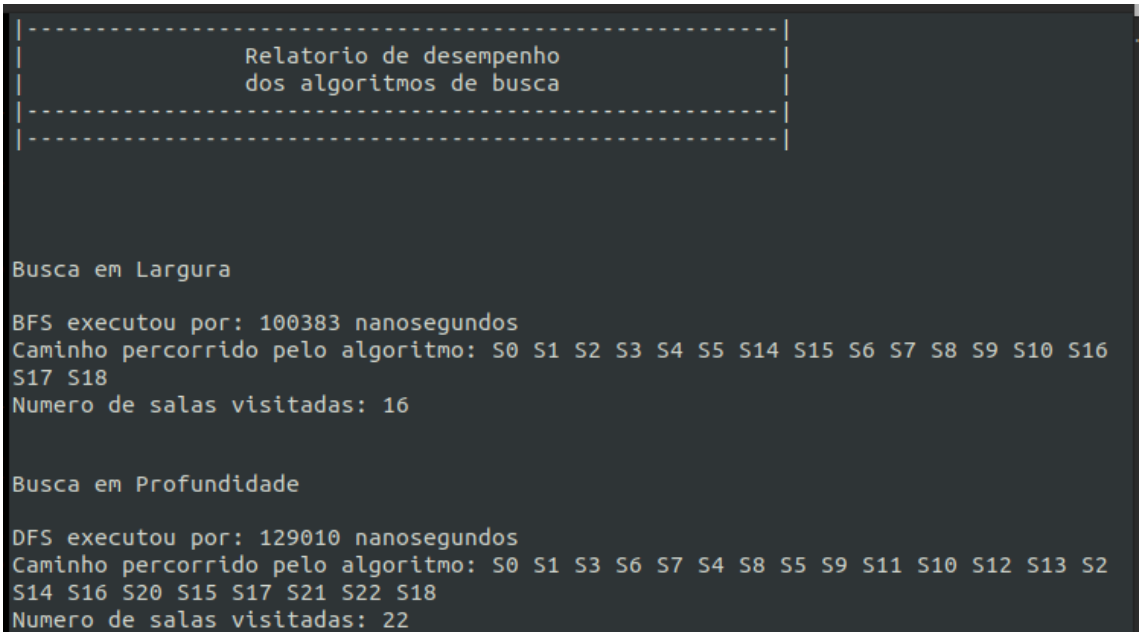
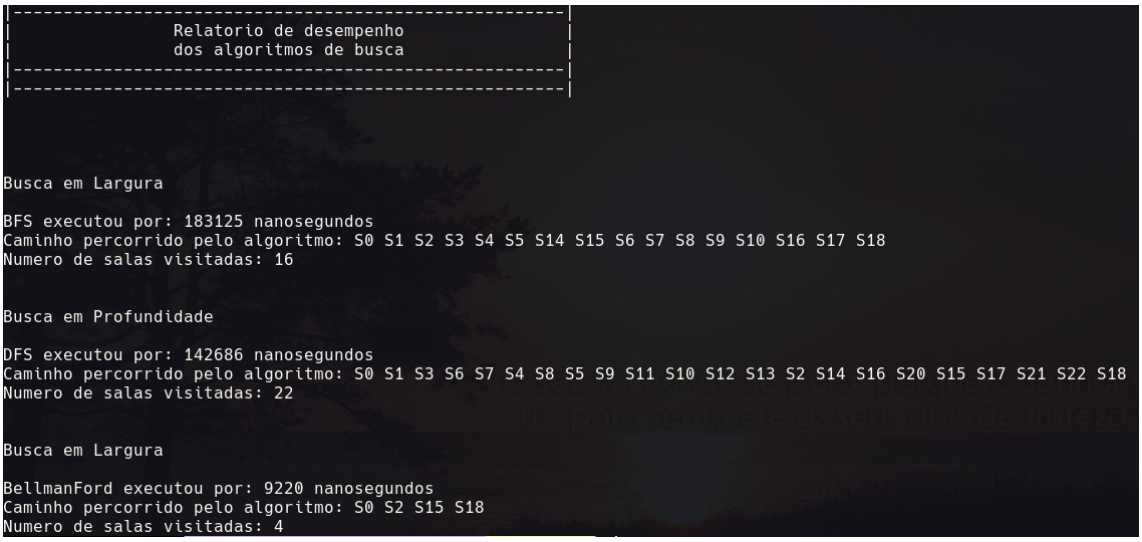


Figura 4 – Resultado gerado pelo algoritmo de Bellmanfor, com s0 e s18.



6 Conclusão

Podemos concluir que, apesar da busca em profundidade ter uma aplicação mais voltada para o problema do presente trabalho, o fato de não ser completo e ótimo e sempre buscar o vértice mais profundo, faz com que visite mais salas e, consequentemente, o tempo de execução dure mais.

Já o algoritmo de Bellman-Ford foi escolhido justamente por ser um algoritmo usado para buscar caminhos mínimos em um grafo incluindo arestas de pesos negativos, e por ser especificamente orientado.

Sendo assim, podemos considerar que o algoritmo de busca em Profundidade é a pior solução para este trabalho e o algoritmo de Bellman-Ford é a melhor solução.

Referências

1 PANTUZA, G. **Busca em profundidade**. 2017. Gustavo Pantuza - Blog sobre Ciência da Computação. Disponível em: <https://blog.pantuza.com/artigos/busca-em-profundidade>. Acesso em: 25 fev 2022. Citado na página 2.

2 PANTUZA, G. **Busca em Largura**. 2017. Gustavo Pantuza - Blog sobre Ciência da Computação. Disponível em: <https://blog.pantuza.com/artigos/busca-em-largura>. Acesso em: 25 fev 2022. Citado na página 2.

3 BELLMAN Ford. 2021. Disponível em: <https://pt.wiki-base.com/7779284-bellman-fords-algorithm>. Citado na página 2.

4 REFERENCE, C. **Date and time utilities**. [S.l.], 2011. Disponível em: <https://en.cppreference.com/w/cpp/chrono>. Acesso em: 14 fev 2022. Citado na página 3.

5 FERREIRA, C. **8 Ways to Measure Execution Time in C/C++**. [S.l.], 2020. Disponível em: <https://levelup.gitconnected.com/8-ways-to-measure-execution-time-in-c-c-48634458d0f9>. Acesso em: 14 fev 2022. Citado na página 3.