UNIVERSIDADE FEDERAL DO PIAUÍ – UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS – PICOS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

VANDIRLEYA BARBOSA DA COSTA

RELATÓRIO DE ESTRUTURAS DE DADOS

Picos – PI, 2023.

RELATÓRIO DE ESTRUTURAS DE DADOS - GRAFOS

Relatório com requisito para aprovação e apresentado à disciplina de Estruturas de Dados II, no curso Bacharelado em Sistemas de Informação pela Universidade Federal do Piauí, ministrada pela Prof. Juliana Carvalho no semestre letivo 2023.1.

Picos – PI, 2023.

Resumo

Este trabalho aborda a resolução do clássico problema da Torre de Hanói e a aplicação de conceitos de grafos orientados. São utilizados os algoritmos de Dijkstra e Bellman-Ford para analisar diferentes abordagens de resolução. Em seguida é feita uma comparação de busca entre os algoritmos.

1. Introdução

Os grafos são estruturas matemáticas que representam relações entre objetos. Eles consistem em nós, também chamados de vértices, que são conectados por arestas. Os grafos têm uma ampla gama de aplicações em ciência da computação, engenharia, redes, logística e muito mais. Eles permitem modelar interações complexas entre elementos de um sistema, tornando-os uma ferramenta essencial na resolução de problemas do mundo real. Dentro do campo da teoria dos grafos, surgem vários problemas que requerem encontrar caminhos mais curtos ou custos mínimos entre dois nós.

O algoritmo de Dijkstra é usado para encontrar o caminho mais curto entre um nó de origem e todos os outros nós em um grafo com pesos não negativos nas arestas. O algoritmo começa marcando a distância do nó de origem para ele mesmo como zero e todas as outras distâncias como infinito. Ele então explora os nós adjacentes, atualizando as distâncias se encontrar um caminho mais curto. O algoritmo continua expandindo as distâncias até que todos os nós tenham sido processados, garantindo que a distância final para cada nó seja o caminho mais curto da origem.

O algoritmo de Bellman-Ford lida com grafos que podem conter arestas com pesos negativos, o que é uma restrição que o algoritmo de Dijkstra não pode lidar. Ele também pode identificar ciclos de peso negativo. O algoritmo funciona relaxando repetidamente todas as arestas no grafo V-1 vezes, onde V é o número de nós no grafo. Durante cada iteração, o algoritmo tenta melhorar as estimativas de distância para cada nó, considerando todas as arestas. A ideia é iterar sobre todas as arestas várias vezes para gradualmente melhorar as estimativas de distância até que não haja mais mudanças significativas. A execução dos algoritmos foi realizada em um notebook Samsung com as seguintes especificações:

Modelo	Lenovo 82MF
Processador	AMD Ryzen 5 5500U with Radeon Graphics (12 GPUs), ~2.1GHz
Memória RAM	8.192MB RAM
Placa de vídeo	Vega 7
Sistema Operacional	Pop!_OS

2. Problemas Apresentado

O primeiro problema é o clássico da Torre de Hanói que envolve mover n discos entre três pinos, visando minimizar os movimentos. Cada movimento consiste em transferir um disco para um pino diferente, mantendo a ordem dos discos. Para representar as configurações, é usado um grafo, onde os vértices são as configurações possíveis e as arestas ligam configurações alcançáveis por movimento válido. No caso de 4 discos, podemos criar um grafo usando matriz de adjacência, com arestas valoradas em 1. O Algoritmo de Dijkstra é aplicado para encontrar o menor caminho entre uma configuração inicial e a configuração final. O tempo gasto na busca da solução é registrado. O problema explora a otimização em movimentos de discos dentro de limitações específicas. Em seguida o mesmo problema é resolvido com o algoritmo de Ford-Moore-Bellman.

O segundo problema envolve um grafo direcionado com arestas valoradas, onde os valores representam a confiabilidade de canais de comunicação. Esses valores indicam a probabilidade de que o canal entre dois vértices não falhe, sendo independentes entre as arestas. O objetivo é desenvolver um programa eficiente para encontrar o caminho de maior confiabilidade entre dois vértices. Isso requer determinar a rota que maximize a probabilidade de sucesso ao longo do caminho. O problema é relevante em redes de comunicação e pode ser resolvido aplicando técnicas como o Algoritmo de Dijkstra, modificando-o para avaliar a confiabilidade das arestas em vez de suas distâncias.

3. Seções Específicas

Para o melhor entendimento das seções a seguir, é importante conhecer as estruturas de dados utilizadas na resolução do problema. A seguir, uma breve apresentação sobre os tipos de algoritmos utilizados e grafos:

3.1 Grafos

Um grafo é uma estrutura matemática usada para representar relações entre objetos. Ele consiste em um conjunto de elementos chamados "nós" ou "vértices"

conectados por linhas chamadas "arestas". Grafos são amplamente aplicados em ciência da computação, matemática, engenharia e diversas outras áreas para modelar interações complexas entre elementos de um sistema. Existem dois tipos principais de grafos: direcionados e não direcionados. Em um grafo não direcionado, as arestas não têm uma direção específica e representam relações bidirecionais. Em um grafo direcionado, as arestas possuem uma direção, indicando uma relação unidirecional entre vértices. Os grafos podem ser utilizados para resolver uma variedade de problemas, como modelar redes de computadores, rotas em mapas, relacionamentos sociais, programação de tarefas e muito mais. Eles são uma ferramenta poderosa para analisar sistemas complexos e encontrar soluções eficientes para diversos desafios do mundo real.

3.2 Algoritmo de Dijkstra

O algoritmo de Dijkstra é um método utilizado em teoria dos grafos para encontrar o caminho mais curto entre um nó de origem e todos os outros nós em um grafo ponderado, onde as arestas têm pesos não negativos. Ele é especialmente eficiente em encontrar os menores caminhos em grafos com características específicas. O algoritmo começa marcando a distância do nó de origem como zero e todas as outras distâncias como infinito. Crie uma estrutura de dados como uma fila de prioridade para manter os nós a serem processados. Depois retira o nó com a menor distância da estrutura de dados. Este é o nó mais próximo da origem que ainda não foi processado. Em seguida, para cada nó vizinho do nó atual, calcule a distância total através do nó atual até o vizinho.

Se essa distância calculada for menor do que a distância atual do vizinho, atualize a distância e o nó anterior, para reconstruir o caminho posteriormente. Por fim, continua selecionando e relaxando nós até que todos os nós tenham sido processados ou até que o nó de destino seja alcançado. O algoritmo de Dijkstra garante que, uma vez que um nó tenha sido marcado como processado, sua distância é a menor possível. À medida que novos nós são processados e suas distâncias são ajustadas, o algoritmo garante que as estimativas de distância sejam atualizadas de maneira correta. No entanto, ele só funciona corretamente com arestas de pesos não negativos e pode não ser adequado para grafos com pesos negativos ou ciclos negativos.

3.3 Algoritmo de Ford-Moore-Bellman

O algoritmo de Bellman-Ford é uma técnica utilizada para encontrar os caminhos mais curtos em um grafo ponderado, podendo lidar com grafos que possuam arestas de peso negativo. Ele é um método versátil que pode ser aplicado mesmo em situações onde outras abordagens, como o algoritmo de Dijkstra, não seriam eficazes devido à presença de pesos negativos. O algoritmo começa marcando a distância do nó de origem como zero e todas as outras distâncias como infinito. O relaxamento de arestas repete o processo V-1 vezes, onde V é o número de nós no grafo. Para cada aresta (u, v) com peso w, verifique se a distância

estimada para o nó v através de u é menor do que a distância atual de v. Se for, atualize a distância de v.

A verificação de ciclo negativo das V-1 iterações, é necessário verificar se existem ciclos de peso negativo no grafo. Para cada aresta (u, v) com peso w, verifique novamente se a distância estimada para o nó v através de u é menor do que a distância atual de v. Se essa verificação for verdadeira, isso indica a existência de um ciclo negativo. O algoritmo de Bellman-Ford é mais lento do que o algoritmo de Dijkstra, pois realiza um número fixo de iterações para relaxar todas as arestas. No entanto, sua capacidade de lidar com pesos negativos e detectar ciclos negativos é uma vantagem crucial em cenários onde esses elementos estão presentes.

4. Estrutura

Esta seção apresenta a estrutura e organização das funções utilizadas nos códigos-fonte do algoritmo de Dijkstra e Ford-Moore-Bellman para resolver o problema proposto. A fim de facilitar o entendimento, são especificados os parâmetros e as funcionalidades das funções presentes em cada uma.

4.1 Torre do Hanói

1. *LerVertices(Vertice Vertices[]):* Lê informações dos vértices (configurações) de um arquivo CSV e armazena-os em uma estrutura de dados. Recebe como parâmetro um array de estruturas do tipo "Vertice".

2. *ImprimirVertices(Vertice Vertices[]):* Imprime as informações dos vértices lidos, incluindo número, quantidade de pinos e arestas. Recebe como parâmetro o mesmo array de estruturas.

3. *InesrirMatriz(int Matriz[][NumVertices], Vertice Vertices[]):* Cria uma matriz de adjacência a partir das informações dos vértices, indicando as conexões entre eles. Recebe como parâmetros a matriz de adjacência e o array de estruturas.

4. *Dijkstra(int Matriz[][NumVertices], int src, int resultado):* Implementa o algoritmo de Dijkstra para encontrar o caminho mais curto entre os vértices de origem (src) e destino (resultado). Retorna o vetor de distâncias.

5. *BellmanFord(int Matriz[][NumVertices], int src, int resultado):* Implementa o algoritmo de Bellman-Ford para encontrar o caminho mais curto entre os vértices de origem (src) e destino (resultado). Retorna o vetor de distâncias.

6. *imprimirMenu():* Imprime um menu de opções para o usuário escolher entre ler/visualizar vértices, calcular o menor caminho usando Dijkstra ou Bellman-Ford, ou sair do programa.

7. *main():* Função principal que executa o menu interativo, permitindo ao usuário escolher as ações a serem realizadas. Chama as funções acima conforme a escolha do usuário.

4.2 Grafo orientado

1. *Struct Aresta*: Armazena as informações sobre uma aresta, o vértice de destino e a confiabilidade associada à aresta.
2. *Struct Grafo:* Armazena um array bidimensional de Arestas representando as conexões entre vértices em um grafo. Mantém o número total de vértices no grafo.
3. *Função bellmanFord(Grafo *grafo, int verticeInicial, float dist[], int verticeAnterior[]):* Implementa o algoritmo de Bellman-Ford para encontrar o caminho mais confiável entre dois vértices no grafo. Recebe o grafo, o vértice inicial, um array para armazenar as distâncias até os vértices e um array para rastrear os vértices anteriores ao longo do caminho. Realiza a inicialização das distâncias e vértices anteriores. E depois Rraliza o relaxamento das arestas repetidamente, atualizando as distâncias e vértices anteriores conforme necessário.
4. *Função imprimirCaminho(int verticeAnterior[], int vertice):* Imprime o caminho de vértices de um vértice de destino até o vértice inicial, usando o array verticeAnterior.
5. *Função main():* Cria um Grafo. Lê o número de vértices e arestas do usuário e inicializa as informações do grafo. Lê as arestas (vértice atual, vértice de destino e confiabilidade) e as armazena no grafo. Lê os vértices inicial e final para calcular o caminho mais confiável usando o algoritmo de Bellman-Ford. Chama a função bellmanFord() e, se encontrar um caminho, imprime o caminho e a confiabilidade resultante; caso contrário, informa que não há caminho confiável.

5. Resultados da execução dos programas

Nessa seção serão apresentados os resultados obtidos nas comparações de tempo para a operação de busca para solução do problema com os algoritmos de Dijkstra e Ford-Moore-Bellman.

5.1 Dijkstra X Ford-Moore-Bellman.

A análise comparativa de busca entre os dois algoritmos foi realizada comparando a busca de um determinado caminho da torre de Hanói. A princípio foi realizada apenas uma busca em cada algoritmo e o tempo em ambos não

apresentou diferença, foram realizados 50000 mil testes em cada algoritmo. Dessa forma, foi possível obter um tempo médio de busca com mais precisão. No algoritmo de Dijkstra, o tempo médio para cada busca foi de 0,036 milissegundos, já no algoritmo de Ford-Moore-Bellman o tempo médio de busca foi de 1,213817 milissegundos. Ambos os algoritmos apresentaram pequenos tempos de buscas, entretanto é possível notar que o tempo de busca do algoritmo de Dijkstra é superior ao de Ford-Moore-Bellman.

Algoritmo	**Tempo de busca**
Dijkstra	0.036000 (ms)
Ford-Moore-Bellman	1.213817 (ms)

6 Conclusão

Através da busca por caminhos em configurações da Torre de Hanói, realizamos uma análise comparativa entre os dois algoritmos. A realização inicial de uma busca em cada algoritmo revelou tempos de busca semelhantes, o que nos levou a conduzir um conjunto mais abrangente de testes, totalizando 50.000 iterações em cada algoritmo. A partir disso, obtivemos tempos médios de busca mais precisos. Os resultados mostraram que ambos os algoritmos demonstraram eficiência, com tempos de busca relativamente curtos. Contudo, destacamos que o algoritmo de Dijkstra apresentou um tempo médio de busca de 0,036 milissegundos, enquanto o algoritmo de Ford-Moore-Bellman registrou um tempo médio de busca de 1,213817 milissegundos. Essa discrepância nos tempos de busca entre os algoritmos sugere que o algoritmo de Dijkstra é mais eficiente nesse contexto específico, pelo menos para a busca de caminhos em configurações da Torre de Hanói. No entanto, é importante ressaltar que o desempenho de algoritmos pode variar em diferentes situações e tamanhos de entrada. A resolução do grafo orientado segue quase a mesma estrutura, entretanto calculamos a probabilidade de cada vértice, ao invés de cada caminho.