João Victor Ottoni Garcia e Rafael Carvalho Avidago Geraldo

A Busca por Caminhos Mágicos

Brasil - São João del Rei/MG

2024

João Victor Ottoni Garcia e Rafael Carvalho Avidago Geraldo

A Busca por Caminhos Mágicos

Trabalho de Projeto e Análise de Algoritmos relacionado ao desenvolvimento de programa que encontre os k menores caminhos de um grafo direcionado.

Universidade Federal De São João del Rei – UFSJ

Faculdade de Ciência da Computação

Orientador: Leonardo Chaves Dutra da Rocha

Brasil - São João del Rei/MG

2024

João Victor Ottoni Garcia e Rafael Carvalho Avidago Geraldo

A Busca por Caminhos Mágicos / João Victor Ottoni Garcia e Rafael Carvalho Avidago Geraldo. – Brasil - São João del Rei/MG, 2024-

9p. : il. (algumas color.) ; 30 cm.

Orientador: Leonardo Chaves Dutra da Rocha

Trabalho Acadêmico – Universidade Federal De São João del Rei – UFSJ
Faculdade de Ciência da Computação , 2024.

1. Menores Caminhos. 2. Heap. 2. Dijkstra3. I. Leonardo Rocha. II. UFSJ. III. Ciência da Computação. IV. A Busca por Caminhos Mágicos

Sumário

1 Introdução

No desafio chamado “A Busca por Caminhos Mágicos”, é proposto o desenvolvimento de um algoritmo que aborda questões complexas de otimização e criação de grafos. O programa deve ser capaz de identificar os (k) caminhos mais curtos entre Mysthollow e Luminae. Além disso, deve-se considerar um conjunto de parâmetros e restrições para o programa. O foco do algoritmo decorre na necessidade de determinar as rotas ótimas dentro de um grafo.

O desafio reside em elaborar uma solução eficiente que possa cumprir um requisito específico, a possibilidade de caminhos que incluem múltiplas visitas a determinados nós, aumentando de maneira exponencial a complexidade da tarefa.

Solicita-se um algoritmo que calcule os k menores caminhos entre nós interligados por arestas ponderadas. Posto isso, têm-se um arquivo de entrada, no qual possui três parâmetros, sendo eles, respectivamente, o número de cidades (vértices), voos (arestas) e o parâmetro (k). Nas linhas seguintes são dados os nós de origem, destinos e custos conforme os valores previamente inseridos. Como resultado, são gerados os (k) caminhos mais curtos que satisfazem a condição do algoritmo, na qual deve ser inscrito no arquivo de saída.

Posteriormente, para que se conclua as especificações do trabalho utilizou-se as funções getrusage e gettimeofday para quantificar o tempo de execução, tempo de CPU utilizado e memória máxima residente, com base nos arquivos de entrada.

2 Conceitos Abordados No Trabalho

2.1 Grafo e Lista de Adjacência

Para resolver o desafio dos k menores caminhos, foi empregado um grafo, um sistema formado por nodos (vértices) e arestas que ligam duplas de nodos. Na situação do desafio, uma lista de adjacência foi empregada para representar o grafo na memória do computador. Para cada nó na estrutura do grafo, é guardada uma lista contendo todos os nós aos quais ele está conectado. Em um grafo ponderado, cada elemento da lista contém o peso da aresta conectando-os, permitindo a representação do custo de cada trajeto.

A estrutura escolhida para representar o grafo e eficiente para representar grafos esparsos, onde o número de arestas e significamente menor que o número máximo possível de arestas. As listas de adjacência oferecem uma maneira rápida de acessar as arestas ligadas a um determinado vértice, o que é crucial para algoritmos que analisam o grafo.

2.2 Fila de Prioridade e Heap

Na execução do programa, a fila de prioridade é crucial no algoritmo de dijkstra, pois é responsável por manipular e consultar os vértices do grafo de maneira eficaz, baseando-se nos custos das arestas. A fila de prioridade garante que os vértices com custos de caminho mais baixos sejam sempre processados primeiro. No código, a estrutura é feita usando um heap, que organiza os elementos de forma que o vértice com menor custo do caminho acumulado fica sempre no topo da fila. Juntos, a fila de prioridade e o heap gerenciam diversas rotas possíveis para cada vértice, possibilitando que o algoritmo de Dijkstra escolha de forma eficaz o próximo vértice com o menor custo acumulado.

2.3 Algoritmo de Dijkstra Modificado

O método de Dijkstra é utilizado para encontrar a rota mais curta de um nó inicial para um destino em um grafo com pesos não negativos nas arestas. No começo, o vértice de partida é definido com uma distância de zero, enquanto os demais vértices possuem uma distância infinita atribuída a eles. Ao utilizar a fila de prioridade, o algoritmo escolhe o próximo vértice não visitado com a menor distância acumulada, ajustando a distância dos vértices vizinhos conforme necessário, e isso continua até que todos os vértices sejam processados.

No projeto, o algoritmo foi alterado para identificar os K caminhos mais curtos de um ponto inicial para um ponto final no grafo. Isso é alcançado mantendo uma lista de k custos menores para cada vértice, em vez de apenas uma distância mínima. Quando um novo caminho mais curto é encontrado, ele é comparado e adicionado de forma ordenada à lista dos k menores caminhos, garantindo que apenas as melhores rotas sejam consideradas até então.

3 IMPLEMENTAÇÃO

Neste estudo, criamos uma versão alterada do algoritmo de Dijkstra para encontrar os k menores trajetos em um gráfico com peso, uma função necessária para usos que requerem opções de rotas efetivas além da rota mais curta usual. Para avaliar a eficácia da implementação, iremos dedicar parte desta seção à avaliação teórica do desempenho do algoritmo.

3.1 Pseudocódigo

```
INICIO:
1. para i de 0 até n faça
2.   para j de 0 até k faça
3.     distancias[i][j] = INFINITO;
4. distancias[1][0] = 0;

5. INICIALIZA(filaPrioridade, 10);
6. INSERE(filaPrioridade, (No){1, 0});

7. enquanto NÃO ESTÁ VAZIA(filaPrioridade) faça
8.   atual = POP(filaPrioridade);
9.   para cada aresta de listaAdj[atual.vertice] faça
10.     adjacente = aresta.destino;
11.     novoCusto = atual.custo + aresta.peso;
12.     se novoCusto < distancias[adjacente][k-1] faça
13.       INSERENOVOCUSTO(distancias[adjacente], k, novoCusto);
14.       INSERE(filaPrioridade, (No){adjacente, novoCusto});
15. LIBERA(filaPrioridade);
16. retorna;
FIM
```

3.2 Analise Teórica do algoritmo

Durante a implementação do algoritmo para encontrar os k menores caminhos em um grafo ponderado, realizaremos uma análise teórica para calcular a complexidade assintótica do algoritmo. O foco desta análise é nas principais atividades que influenciam diretamente no desempenho do algoritmo, incluindo desde a leitura e criação do grafo até a execução do algoritmo de Dijkstra. A análise do gráfico e a formação das listas de adjacência têm uma complexidade O(E), indicando um processamento linear de cada aresta. Essa parte e essencial para a representacao eficiente do grafo e define a fundacao para as operacoes subsequentes.

Depois de ser iniciado, cada vértice é inserido em uma fila de prioridade, com uma complexidade inicial de O(V). No entanto, a etapa mais complexa em termos de dificuldade é o procedimento de relaxamento de bordas. Por meio do heap que implementa a fila de prioridade, o algoritmo efetivamente seleciona o próximo vértice, levando à complexidade de O(E log V). O procedimento não só leva a uma escolha eficaz do próximo vértice, mas também viabiliza a atualização das menores distâncias de maneira organizada. Adicionar a capacidade de identificar os k menores caminhos acrescenta uma nova perspectiva à análise, embora a complexidade desse recurso seja tratada pela lógica de relaxamento e gerenciamento da fila de prioridade. A característica desse ajuste, embora amplie a extensão do algoritmo inicial, não muda a complexidade geral, mantendo-se nos parâmetros estabelecidos pelo termo O(ElogV).

Dessa forma, podemos afirmar que o principal fator de influência na complexidade assintótica do algoritmo de Dijkstra modificado é O(ElogV), baseando-se na operação de relaxamento das arestas e na performance da fila de prioridade. Este estudo ressalta a capacidade do algoritmo em diferentes cenários, fornecendo informações sobre seu desempenho teórico em situações reais e criando uma plataforma para futuras melhorias práticas e otimizações.

4 RESULTADOS E DISCUSSÕES

Nesta seção, mostramos os resultados alcançados ao usar uma versão adaptada do algoritmo de Dijkstra para encontrar os k caminhos mais curtos em um gráfo com pesos. Estes resultados são essenciais para a avaliação do desempenho do algoritmo em várias situações de uso. A avaliação é feita com base em vários testes de performance, com ênfase no tempo de execução, na utilização da CPU e na memória utilizada.

4.1 Avaliação de desempenho

Nos testes feitos, e possivel analisar que o tempo de execucao do algoritmo aumenta conforme o numero de caminhos a serem encontrados e aumentado. Esta variação é atribuída principalmente ao aumento do valor de K . A variação mencionada confirma que, apesar da eficiência do algoritmo, a complexidade de encontrar diferentes caminhos afeta diretamente o tempo de conclusão da tarefa. Quanto mais alto for o valor de k.

Contudo, é necessário que o algoritmo se esforce para manter e atualizar a lista dos caminhos mais eficazes, resultando em um aumento no tempo de processamento. Em relação ao uso de memória, notamos que a memória máxima residente variou de 3480 kilobytes a 3860 kilobytes. Essa medida tende a crescer à medida que o número de nós no gráfico aumenta, o que está de acordo com as expectativas para algoritmos que lidam com estruturas de dados complexas, como os grafos. O incremento na utilização de memória é afetado não somente pelas dimensões do grafo, mas também pelo valor de k, já que o algoritmo necessita de recursos extras para armazenar e administrar os k menores caminhos de cada vértice.

4.2 Conclusão

A eficácia do algoritmo modificado de Dijkstra em encontrar os k menores caminhos foi comprovada pelos testes realizados, demonstrando um bom desempenho tanto em tempo de execução quanto em eficiência de uso de memória. Estes resultados sustentam a possibilidade do algoritmo ser utilizado em situações do mundo real, proporcionando uma plataforma sólida para pesquisas futuras que tenham como objetivo explorar e superar os desafios encontrados em cenários de alta exigência computacional.