Trabalho Prático 1

Ítalo Dell'Areti

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

1. Introdução

O problema proposto a ser resolvido consiste em desenvolver um sistema de segurança para o império de Archadia, que necessita organizar suas defesas militares em diferentes estados. Cada estado é representado por uma estrutura contendo uma capital, centros urbanos e estradas de mão única conectando estes centros. Sendo assim, o sistema deve resolver três problemas principais:

1. Identificar a capital ideal que permita ao exército alcançar todos os centros urbanos pelo menor caminho possível.
2. Determinar a quantidade e localização de batalhões secundários necessários para garantir que as tropas possam retornar e se reagrupar.
3. Estabelecer rotas de patrulhamento que permitam cobrir todas as estradas possíveis e retornar ao ponto de origem.

Ao final, devemos computar estes problemas e retornar os resultados para o imperador. Portanto, com base nas características e propriedades grafos e seus componentes, tipos de busca e ordenação, podemos implementar este problema de forma a respeitar todas as nuances da problemática. Tendo em vista disso, este documento possui como objetivo explicitar o funcionamento desta implementação.

2. Modelagem

O problema foi modelado utilizando grafos, de forma onde:

- Vértices (V): Representam os centros urbanos, sem peso.
- Arestas direcionadas (E): Representam as estradas de mão única (direcionado) e não possuem peso (A "distância" entre dois centros urbanos é medida pelo número mínimo de estradas que precisam ser percorridas).
- Grafo G(V,E): Representa um estado completo.
- Lista de adjacência: Estrutura escolhida para representar o grafo, por ser eficiente.

Principais algoritmos utilizados:

- Busca em Largura (BFS): Para encontrar menores caminhos e a capital.

- Algoritmo de Kosaraju: Para identificar componentes fortemente conectados para os batalhões.
- Algoritmo para encontrar ciclos Eulerianos: Para determinar rotas de patrulhamento.

3. Solução

3.1 Determinação da Capital

A solução para encontrar a capital ideal se baseia no conceito de alcançabilidade e minimização de distâncias em um grafo direcionado. Para cada centro urbano candidato, realizamos uma Busca em Largura (BFS) que:

1. Calcula o menor número de estradas necessárias para alcançar todos os outros centros urbanos.
2. Verifica se todos os outros centros são alcançáveis.
3. Determina a maior distância necessária para alcançar qualquer centro.

A capital escolhida será o centro urbano que:

- Consegue alcançar todos os outros centros urbanos.
- Tem a menor distância máxima até o centro mais distante.

A BFS é usada porque, em um grafo não ponderado, ela garante encontrar os caminhos com menor número de arestas (estradas). Ao expandir em "ondas" a partir do centro inicial, ela naturalmente encontra primeiro os caminhos mais curtos.

3.2 Localização dos Batalhões

Para determinar onde posicionar os batalhões secundários, utilizamos o algoritmo de Kosaraju para identificar os Componentes Fortemente Conectados (CFCs) do grafo. Este processo é necessário porque:

1. Um CFC representa um grupo de centros urbanos onde é possível ir e voltar entre quaisquer dois centros do grupo.
2. O componente que contém a capital não precisa de batalhão adicional.
3. Cada outro componente precisa de exatamente um batalhão.

O processo completo envolve:

1. Identificar todos os CFCs usando Kosaraju.
2. Excluir o componente que contém a capital.
3. Para cada componente restante:
    - Escolher o centro urbano mais próximo da capital como local do batalhão
    - A proximidade é medida pelo número de estradas do caminho mais curto da capital até o centro

Os batalhões são posicionados nos centros mais próximos da capital para minimizar o tempo de comunicação e deslocamento entre a capital e os batalhões secundários.

3.3 Rotas de Patrulhamento

O problema das rotas de patrulhamento é essencialmente uma busca por ciclos eulerianos em subgrafos. Para cada batalhão (incluindo o principal na capital):

1. Identificamos o subgrafo alcançável a partir do batalhão:
    - Este subgrafo contém apenas os centros e estradas que podem ser alcançados a partir do batalhão e que permitem retorno.
2. Verificamos se existe um ciclo euleriano neste subgrafo:
    - Todas as estradas devem ser percorridas exatamente uma vez.
    - O caminho deve começar e terminar no batalhão.
    - Cada vértice deve ter grau de entrada igual ao grau de saída.
3. Se existir um ciclo euleriano:
    - Utilizamos o algoritmo de Hierholzer para encontrar o ciclo.
    - O ciclo representa uma rota de patrulhamento válida.
    - O algoritmo construtivo garante que encontraremos um ciclo se ele existir.

O processo é repetido para cada batalhão, pois cada um pode ter seu próprio conjunto de estradas patrulháveis em sua área de influência. Uma rota só é considerada válida se:

- Começa e termina no mesmo batalhão.
- Usa cada estrada exatamente uma vez.
- Mantém a conectividade (não se divide em subciclos desconectados).

Esta abordagem garante que encontremos as possíveis rotas de patrulhamento que satisfazem os requisitos do problema, permitindo uma cobertura eficiente do território sob responsabilidade de cada batalhão.

4. Análise de Complexidade:

4.1 Estruturas de Dados

- Lista de adjacência: O(|V| + |E|) de espaço

4.2 Funções Principais

calcula_capital:

- Tempo: O(|V| * (|V| + |E|)) - BFS para cada vértice
- Espaço: O(|V|) - vetor de distâncias

calcula_batalhao:

- Tempo: O(|V| + |E|) - Kosaraju + O(|V| * log|V|) ordenação
- Espaço: O(|V| + |E|) - estruturas do Kosaraju

calcula_patrulhamento:

- Tempo: O(|V| * (|V| + |E|)) - verificação de ciclos para cada batalhão
- Espaço: O(|V| + |E|) - subgrafos e caminhos

4.3 Funções Auxiliares

busca_em_largura:

- Tempo: O(|V| + |E|)
- Espaço: O(|V|)

transpor_grafo:

- Tempo: O(|V| + |E|)
- Espaço: O(|V| + |E|)

dfs1 e dfs2:

- Tempo: O(|V| + |E|)
- Espaço: O(|V|)

insert_ordenado_por_distancia:

- Tempo: O(|V|)
- Espaço: O(1)

e_alcancavel:

- Tempo: O(|V| + |E|)
- Espaço: O(|V|)

verifica_graus_euler:

- Tempo: O(|V| + |E|)
- Espaço: O(|V|)

encontrar_ciclo_euler:

- Tempo: O(|E|)
- Espaço: O(|E|)

obter_ciclo_euler:

- Tempo: O(|E|)
- Espaço: O(|E|)

dfs_componente_local:

- Tempo: O(|V| + |E|)
- Espaço: O(|V|)

5. Considerações Finais

O desenvolvimento deste trabalho proporcionou uma experiência prática valiosa na aplicação de algoritmos em grafos para resolver problemas complexos.

A parte mais fácil foi a implementação de calcular a capital, por ser um conceito básico de BFS e cálculo de distâncias mínimas e a parte de determinar os batalhões foi um pouco mais complicada, mas ambos algoritmos foram inúmeras vezes vistos e debatidos em sala de aula.

A maior dificuldade encontrada foi referente a parte 3, por se tratar de um algoritmo ainda não visto em aula e não possuir muitas informações de como implementá-lo corretamente. A priori, estava implementando **obter_ciclo_euler** para encontrar todos os ciclos e por necessidade a complexidade ficava muito alta mas todos os outputs funcionam corretamente.

Após verificar que esta complexidade era inviável, realizei a modificação para que a maior complexidade seria O(|V| * (|V| + |E|)), onde |V| é o número de vértices e |E| é o número de arestas. Mas esta mudança acabou afetando alguns casos para os quais não posso garantir o comportamento da parte 3.

Além disso, utilizei o programa Valgrind para verificar se existia algum vazamento de memória e compilei o programa para demonstrar seu funcionamento.

Vídeo com a implementação e teste do Valgrind:

https://drive.google.com/file/d/1sqy5GoWGN3WZbKklrLlKaqA_58JrgJQx/view?usp=sharing

6. Referências

Geeks for geeks: Depth First Search or DFS for a Graph

Geeks for geeks : Breadth First Search or BFS for a Graph

USP IME: Algoritmo de Kosaraju-Sharir para componentes fortes

Geeks for geeks: Strongly Connected Components

Geeks for geeks : Euler Circuit in a Directed Graph

Geeks for geeks : Hierholzer's Algorithm for directed graph

CORMEN, T. H. et al. Introduction to Algorithms. 3rd ed. MIT Press, 2009.

KLEINBERG, J.; TARDOS, E. Algorithm Design. Pearson, 2005.