Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Algoritmos I

Trabalho Prático 1

Henrique Gomes dos Santos Medeiros
2021084986

Belo Horizonte
2024

Introdução

O problema abordado neste trabalho envolve a análise de um grafo representando cidades conectadas por estradas. O trabalho se trata da resolução de três problemas sobre isso:

1. Qual centro urbano (cidade) seria o melhor candidato a capital?
2. Quantos batalhões secundários são necessários nesse estado e quais são eles?
3. Quantas rotas de patrulhamento existem e quais são elas?

Para tal, foram utilizados algoritmos como BFS (breadth-first search) e o algoritmo de Kosaraju.

Modelagem

O problema foi modelado como um grafo dirigido e não ponderado onde:

- Os nós representam cidades.
- As arestas representam estradas unidirecionais entre as cidades.

Estruturas de Dados:

Para estruturação dos grafos foram utilizadas matrizes de adjacência e listas de adjacência. As matrizes foram responsáveis pela computação da maior parte do código, enquanto as listas foram mais úteis ao final dele.

Para as demais funções foram utilizados arrays e vetores.

Algoritmos Utilizados:

1. **BFS (Busca em Largura):** Calcula as distâncias entre as cidades para determinar a capital.
2. **Kosaraju:** Encontra os SCCs (strongly connected components) no grafo, ajudando a identificar os batalhões.
3. **DFS Modificada:** Determina as rotas de patrulha dentro de cada SCC.

Solução

Determinar a Capital

A cidade escolhida como capital é aquela que minimiza a soma das distâncias para todas as outras cidades.

Em grafos direcionados e ponderados, usaria-se o algoritmo de Dijkstra, mas não é esse tipo de grafo que estamos lidando. De outra forma, a DFS (busca em

profundidade) não é ideal porque não garante o menor caminho, apenas verifica a conectividade. Usaremos a BFS, busca em largura, que calcula os menores caminhos em grafos não ponderados.

Assim:

- Executamos a BFS a partir de cada cidade
- Calculamos a soma das distâncias para todas as cidades alcançáveis. Cidades não alcançáveis recebem um valor simbólico alto.
- A cidade com menor soma é definida como a capital.

Identificar os Batalhões

Os batalhões são as cidades que recebem tropas da capital mas não podem enviá-las de volta de modo algum. O algoritmo de Kosaraju é usado para encontrar os SCCs. Cada SCC, exceto o que contém a capital, representa um local para alocação de batalhões. Ainda dentro de cada SCC verificamos aquele que recebe tropas da capital primeiro, e o próprio uso da DFS no algoritmo de Kosaraju nos fornece isso.

Determinar Rotas de Patrulha

Uma rota de patrulha é um caminho que permite o envio de tropas a partir do batalhão e a possibilidade de retorno das mesmas. Patrulhar é mandar a tropa pelo máximo de vértices possíveis (o que é chamado de área de influência) com a possibilidade de volta. As SCCs que encontramos na etapa anterior se tratam do conjunto de cidades fortemente conectadas, e agora precisamos determinar as rotas de patrulha entre essas cidades.

Para cada SCC que contém mais de um nó

- Utiliza-se uma DFS para construir um circuito euleriano (quando possível).
- A última aresta que retorna ao ponto inicial é removida para evitar redundâncias

Pseudocódigo: Determinação da Capital

```
inicializar lista soma_distancias

para cada cidade i de 0 até n_cidades-1 faça:
    distâncias_i = BFS(grafo, i)  // roda BFS para a cidade i
    soma_distancias[i] = soma das distâncias de distâncias_i  // soma as
distâncias

capital_id = índice de menor valor em soma_distancias
imprimir cidades[capital_id]  // imprime o nome da cidade com a menor soma de
distâncias
```

Identificação de SCCs (Kosaraju)

```
SCCs = resultado do algoritmo Kosaraju(grafo, capital_id)  // encontra as
SCCs no grafo, excluindo a capital

imprimir número de SCCs - 1  // quantidade de batalhões (SCCs excluindo a
capital)

para cada componente conexo SCC em SCCs faça:
    se SCC[0] não for igual a capital_id:
        imprimir cidade[SCC[0]]  // imprime o nome da cidade do batalhão
```

Determinação das Patrulhas

```
inicializar quant_patrulhas = 0
inicializar lista de patrulhas vazia

copiar matriz do grafo para matriz

para cada componente conexo SCC em SCCs faça:
    se SCC tem apenas 1 nó, continuar para o próximo SCC

    incrementar quant_patrulhas

    inicializar patrulha_atual com o nó inicial de SCC (batalhão)

    linha_atual = SCC[0]  // começa no nó do batalhão

    para cada nó a em todas as cidades faça:
        se existe estrada de linha_atual para a:
            marcar a como visitada (remover aresta)
            adicionar a à patrulha_atual
            atualizar linha_atual para a
            reiniciar a busca (a = 0)

    remover o retorno ao batalhão da patrulha_atual

    adicionar patrulha_atual à lista de patrulhas

    resetar matriz para o próximo SCC

imprimir quant_patrulhas

para cada patrulha em patrulhas:
    imprimir cidades visitadas na patrulha
```

Análise de Complexidade

1. **BFS**:
   - **Complexidade**: A BFS percorre todos os nós e arestas do grafo.
   - **Tempo**: O tempo de execução é O(V + E), onde V é o número de vértices e E é o número de arestas.

2. **Determinação da capital**:
   - **Complexidade**: Executa o BFS para cada cidade.
   - **Tempo**: Como a BFS é O(V + E) e ocorre V vezes, a complexidade total é O(V * (V + E)).

3. **Kosaraju**:
   - **Passo 1**: DFS para determinar a ordem dos nós (complexidade O(V + E)).
   - **Passo 2**: Transposição do grafo (complexidade $O(V^2)$).
   - **Passo 3**: DFS novamente para encontrar as SCCs (complexidade O(V + E)).
   - **Tempo total**: A complexidade total de Kosaraju, implementado com uma matriz de adjacência, é $O(V^2 + E)$.

4. **Cálculo de patrulhas**:
   - **Complexidade**: Para cada componente conexo (SCC), há uma iteração sobre os nós e arestas do grafo.
   - **Tempo**: Para cada SCC com V' nós, a complexidade é O(V' * (V' + E)). No pior caso, todos os vértices pertencem a um único SCC, então a complexidade total para esse passo é $O(V^2 + E)$.

Complexidade total:
A complexidade total do código é dominada pelos passos mais pesados. Portanto, considerando as operações de BFS, Kosaraju e o cálculo de patrulhas, a complexidade total é $O(V^2 + E)$.

Quanto à complexidade de espaço, as complexidades de armazenamento das matrizes de adjacência e estruturas auxiliares são, respectivamente, $O(V^2)$ e O(V + E).

Considerações Finais

Esse trabalho destacou a aplicação prática de algoritmos de grafos em problemas reais. Apontar a capital foi simples, mas implementar Kosaraju e determinar as rotas de patrulha exigiu maior cuidado. O uso de estruturas e algoritmos conhecidos tornou a solução eficiente e escalável.

Comecei pensando em implementar como já havia feito outras vezes, utilizando structs que representavam nodes, mas pareceu extremamente imprático. Decidi utilizar matrizes de adjacência e funcionou muito bem. Quando se mostraram extremamente esparsas pensei na lista de adjacência, mas decidi continuar trabalhando com matrizes pois a consulta de uma aresta na matriz é direta pelos índices, O(1), enquanto na lista é O(n). Escolhi abrir mão de armazenamento em prol de melhor tempo de processamento.

A utilização da flag -Werror na compilação também apresentou um pequeno desafio, pois precisei utilizar formas de implementação menos comuns para atingir os objetivos.

Referências

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*.
Aulas e notas fornecidas pelo curso.