Trabalho Prático 2 - Proteção da Capital

Luiz Felipe Gondim Araujo - 2023028188

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

luizfelipegondimaraujo@ufmg.br

1 Introdução

Esta documentação apresenta a solução desenvolvida para proteger a capital com o menor número possível de soldados, utilizando um mapa modelado como um grafo com capacidade, onde cada célula acessível foi dividida em dois nós (entrada e saída) conectados por uma aresta com capacidade igual à resistência da célula. As bordas do mapa foram ligadas a uma fonte, e a capital ao nó de destino. Com essa estrutura, foi aplicado o algoritmo de Edmonds-Karp para calcular o fluxo máximo de invasores, que corresponde ao número mínimo de tropas necessárias para impedir qualquer acesso à capital.

2 Implementação

O programa foi desenvolvido em linguagem C++, utilizando o compilador G++ da GNU Compiler Collection. Em relação ao ambiente de desenvolvimento, consistiu em uma máquina rodando o sistema operacional Linux Mint 21, equipada com 16 GB de memória RAM. A escolha do Linux Mint foi pensada visando a execução do código na avaliação efetuada pelos monitores. Além disso, ainda foram usados o valgrind e o gdb para capturar possíveis erros e vazamentos em relação ao uso de memória.

3 Modelagem

O problema foi modelado como um grafo direcionado com capacidades, onde cada célula acessível do mapa é representada por dois vértices distintos (um vértice de entrada e um de saída). A separação entre entrada e saída permite controlar o fluxo máximo que pode atravessar cada célula, de acordo com sua capacidade definida no grid.

Cada célula do mapa é conectada de sua entrada para sua saída com uma aresta de capacidade igual ao valor da célula, exceto células com valor 0 (montanhas), que são desconsideradas por serem inacessíveis. A partir da saída de cada célula, são criadas arestas de capacidade infinita para as entradas de suas células vizinhas acessíveis (acima, abaixo, esquerda e direita), permitindo o deslocamento livre entre elas, respeitando as restrições de capacidade locais.

Células localizadas na borda do mapa são conectadas diretamente ao nó fonte com arestas de capacidade infinita, representando possíveis pontos de invasão. A capital, por sua vez, é conectada de sua saída ao nó final também com uma aresta de capacidade infinita. O grafo foi representado

por meio de uma lista de adjacências com estrutura de arestas reversas, possibilitando a construção do grafo residual. Além disso, foi utilizado o algoritmo de Edmonds-Karp, uma implementação do método de fluxo máximo baseada em busca em largura (BFS).

4 Solução

Para resolver o problema de defesa da capital, foi desenvolvida uma estrutura baseada na classe `Map`, que centraliza toda a lógica de construção do grafo e aplicação do algoritmo de fluxo. A seguir, detalhamos as etapas principais da solução.

4.1 Leitura da Entrada de Dados

O método `readInput` foi responsável por interpretar e armazenar a matriz enviada como entrada do programa - contendo as capacidades defensivas de cada célula -, além de ler as coordenadas da capital. Enquanto a construção do grafo especificado na modelagem e utilizado no algoritmo de fluxo máximo é feita no método `minTroopsToProtectCapital`.

4.2 Aplicação do Algoritmo de Fluxo Máximo

Para calcular a quantidade mínima de tropas necessárias para invadir a capital, foi utilizado o algoritmo de Edmonds-Karp. Este algoritmo é uma implementação do método de Ford-Fulkerson que utiliza busca em largura (BFS) para encontrar caminhos aumentantes no grafo residual. Sabendo disso, a cada iteração, um caminho da fonte ao sorvedouro (capital) é encontrado, o fluxo é aumentado de acordo com a menor capacidade no caminho, e o grafo residual é atualizado. O processo continua até que não existam mais caminhos aumentantes entre a fonte e o sorvedouro. O valor total do fluxo encontrado ao final corresponde à resposta do problema: a força mínima necessária para vencer a defesa da capital.

5 Instruções de Compilação e Execução

Para executar o software, efetue os seguintes passos:

- Acesse a raiz do projeto;
- Utilizando um terminal, execute o comando 'make' para compilar o código;
- Ainda na raiz do projeto, digite: ./tp2.out;

6 Análise de Complexidade

A análise de complexidade será feita, principalmente, sobre os métodos `readInput`, `edmondsKarp` e `minTroopsToProtectCapital` da classe Map, uma vez que estas permitem uma visão geral da complexidade que o sistema carrega. Os demais métodos possuem custo constante, não impactando significamente a aplicação.

6.1 readInput

Este método realiza a leitura da matriz do terreno e da posição da capital. Ele percorre todas as posições da matriz de dimensão $N \times M$, o que resulta em uma complexidade de tempo $O(N \cdot M)$.

Enquanto a leitura da posição da capital ocorre em tempo constante. Portanto, o custo total do método é linear em relação ao tamanho da matriz.

6.2 edmondsKarp

O método `edmondsKarp` implementa o algoritmo de Edmonds-Karp para cálculo do fluxo máximo. Sua complexidade é conhecida como $O(V \cdot E^2)$, onde V é o número de vértices e E o número de arestas. No contexto do problema, cada célula válida do terreno gera dois vértices (entrada e saída), e as conexões entre células geram arestas adicionais. Assim, o número de vértices é da ordem de $O(N \cdot M)$ e o número de arestas também é proporcional a $O(N \cdot M)$, considerando até 4 vizinhos por célula. Logo, a complexidade do algoritmo nesse cenário é $O((N \cdot M) \cdot (N \cdot M)^2) = O(N^3 \cdot M^3)$ no pior caso.

6.3 minTroopsToProtectCapital

Este método é responsável por construir o grafo a partir da matriz de entrada e chamar o algoritmo de fluxo máximo. A construção percorre todas as células da matriz uma vez, e para cada célula válida (isto é, diferente de 0), adiciona arestas entre os nós de entrada e saída, com seus vizinhos acessíveis, e eventualmente com a fonte ou o sorvedouro. Como cada célula pode ter no máximo quatro vizinhos, o número de operações para adicionar arestas é limitado por uma constante multiplicada pelo número de células. Portanto, a construção do grafo ocorre em tempo $O(N \cdot M)$. Após essa etapa, o método chama `edmondsKarp`, cuja complexidade domina. Assim, a complexidade total de `minTroopsToProtectCapital` é $O(N^3 \cdot M^3)$.

6.4 Análise geral

Com base nas análises individuais, observa-se que a maior parte do custo computacional da aplicação está concentrada na execução do algoritmo de Edmonds-Karp, utilizado para calcular o fluxo máximo no grafo construído a partir da matriz do terreno. Embora as etapas de leitura e construção do grafo tenham complexidade linear em relação ao tamanho da matriz ($O(N \cdot M)$), a execução do fluxo máximo impõe uma complexidade cúbica em ambos os parâmetros, resultando em $O(N^3 \cdot M^3)$. Portanto, a complexidade geral da aplicação é dominada por este fator, tornando o algoritmo potencialmente custoso para mapas de grande dimensão. Ainda assim, essa abordagem garante uma solução exata e eficiente dentro de limites razoáveis de entrada.

7 Considerações Finais

A solução desenvolvida neste trabalho demonstrou a aplicabilidade de técnicas de teoria dos grafos na modelagem e resolução de problemas de defesa territorial, representando com precisão o mapa como uma estrutura direcionada com capacidades. O uso do algoritmo de Edmonds-Karp permitiu determinar, de forma exata, a quantidade mínima de tropas necessárias para proteger a capital, através do cálculo do fluxo máximo de invasores possíveis.

Durante o desenvolvimento, foi possível observar como a construção cuidadosa do grafo influencia diretamente na eficiência da solução. A separação entre nós de entrada e saída por célula, bem como o uso de arestas com capacidades específicas e infinitas, permitiu controlar com precisão os limites de acesso entre posições do terreno. Esse tipo de modelagem, embora abstrata, demonstrou ser eficiente para simular cenários reais.

Por fim, durante a implementação da solução, houveram importantes desafios a serem superados, por exemplo, a interpretação do problema como um grafo de fluxo, em especial à forma de representar corretamente as restrições de movimentação entre as células. Além disso, foi desafiador entender também qual seria a melhor forma de aplicar o algoritmo de fluxo máximo de modo a garantir a coerência entre o modelo e o enunciado. No entanto, superar essas barreiras contribuiu significativamente para a consolidação do conhecimento sobre algoritmos de fluxo, grafos com capacidades e construção de soluções robustas para problemas complexos.

Referências

[1] HACKEREARTH. Breadth First Search. Disponível em: `https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/`. Acesso em: 19 maio 2025.

[2] USP. Fluxo Máximo em Redes. Disponível em: `https://www.ime.usp.br/~pf/otimizacao-combinatoria/aulas/max-flow.html`. Acesso em: 19 maio 2025.