ADLER GUILHERME FURTADO FARIA
2023028501

TP 2 ALG I

UFMG Belo Horizonte, 2024

INTRODUÇÃO

O problema abordado neste trabalho consiste em determinar a quantidade mínima de tropas necessárias para proteger a capital de um reino contra possíveis invasões. O território é representado por um mapa bidimensional, no qual cada célula corresponde a uma posição que pode ser atravessada por inimigos ou bloqueada por montanhas, além de exigir uma certa quantidade de soldados para ser defendida. O objetivo é impedir que qualquer invasor, partindo de fora do mapa, consiga alcançar a capital, garantindo assim sua segurança com o menor número possível de soldados.

Para isso, o problema foi modelado como um grafo, onde cada posição do mapa corresponde a um vértice com restrições de capacidade, representando o custo de defesa daquela posição. As conexões entre posições vizinhas são modeladas como arestas, permitindo o deslocamento dos invasores caso não sejam devidamente bloqueadas. A partir dessa modelagem, o problema se traduzir calcular o corte mínimo que separe a capital das bordas do mapa (de onde vêm os invasores), de forma que a soma dos custos das posições defendidas seja a menor possível, ou seja, de forma que o menor número de soldados seja usado para defender a capital.

Diante dessa abordagem, optei por utilizar o algoritmo de fluxo máximo, que é eficiente para encontrar o corte mínimo em grafos com múltiplos caminhos possíveis.

MODELAGEM

O mapa foi modelado como um grafo direcionado, no qual cada posição do mapa que não representa uma montanha corresponde a um conjunto de dois vértices: um vértice de entrada e um de saída, ou seja, cada posição do mapa é representada por um nó de entrada e um nó de saída. Entre esses nós, é criada uma aresta com capacidade igual número de soldados necessários para defender aquela posição, isso foi feito para “remover” o peso dos nós e colocar nas arestas, para tratar o problema utilizando algoritmos que já conhecemos.

Já as conexões entre as posições vizinhas são representadas por arestas de capacidade infinita (pois elas não importam), que ligam o nó de saída de uma célula ao nó de entrada de sua vizinha.

Para representar os pontos de onde os invasores podem partir (as bordas do mapa), é criado um nó de origem, que se conecta a todas as posições de borda que não são montanha, utilizando arestas de capacidade infinita, simulando a entrada dos invasores no reino; ou seja, utilizamos as bordas do mapa como parâmetro para a entrada dos invasores.

Por fim, é criado um nó de destino, que seria a capital. Esse nó recebe uma aresta de capacidade infinita partindo do nó de entrada da posição da capital. Assim, todo fluxo que chegar à capital deve passar por essa aresta.

Dessa forma, o problema se torna calcular o fluxo máximo do nó de origem (borda) até o nó de destino (capital).

Além disso este projeto foi desenvolvido em C++ e estruturado de forma a manter o código organizado, modular e de fácil compreensão

SOLUÇÃO

A solução proposta envolve a leitura dos dados do mapa do reino, a construção de um grafo que modela tanto os custos de defesa quanto as possíveis rotas dos invasores, e a aplicação de um algoritmo de fluxo máximo para calcular o número mínimo de soldados necessários para proteger a capital.

O programa é composto por três arquivos principais. O arquivo main.cpp é responsável pela leitura dos dados de entrada, construção do grafo e execução da lógica principal do problema. O arquivo grafo.hpp contém a definição da estrutura de grafo, incluindo suas arestas, listas de adjacência e a implementação do algoritmo de fluxo máximo. Já o arquivo auxs.hpp oferece funções auxiliares, como a verificação de se uma posição está dentro dos limites do mapa, além de constantes auxiliares para os deslocamentos no mapa.

O funcionamento do programa começa com a leitura das dimensões do mapa, seguida pela matriz que representa cada posição do território. Nesta matriz, valores iguais a zero representam montanhas, as quais não precisam de nenhum soldado para defendê-las, enquanto valores positivos indicam posições que devem ser defendidas e a quantidade necessária de soldados para isso. Por fim, são lidas as coordenadas da capital, que é o ponto a ser defendido.

Assim, com o grafo devidamente modelado (como explicado anteriormente), a solução do problema se reduz ao cálculo do fluxo máximo entre o ponto de origem do invasor (as bordas) e o ponto a ser defendido (a capital). Esse fluxo máximo representa exatamente o custo do corte mínimo, ou seja, o menor conjunto de posições que precisam ser protegidas, somando seus respectivos custos, de forma a impedir que qualquer invasor, partindo das bordas, consiga alcançar a capital.

Como visto em sala, o algoritmo de Ford Fulkerson (que foi o principal algoritmo visto para resolver problemas de fluxo máximo e corte mínimo) depende diretamente do valor dos fluxos, que neste VPL eram muito altos. Assim, após pesquisar, fiz algumas melhorias no algoritmo, para evitar lentidão, que se basearam em construir o grafo a partir de níveis, utilizando um BFS como uma função auxiliar, e, assim, encontrar os caminhos válidos a partir de um DFS, que vai reduzindo as capacidades das arestas até que nenhum fluxo adicional possa ser enviado (fluxo máximo).

Quando não é mais possível enviar fluxo dentro do grafo de níveis atual, o algoritmo executa uma nova BFS para atualizar os níveis e repete o processo de DFS. Isso se repete até que a BFS não consiga mais encontrar um caminho da origem até a capital, indicando que não há mais fluxo possível a ser enviado. O valor do corte mínimo, então, seria exatamente o somatório de todo esse fluxo que corresponde à quantidade mínima de soldados necessários para garantir que a capital permaneça protegida.

Em resumo, a solução é feita a partir de algoritmos clássicos de fluxo máximo e algumas funções auxiliares, aplicadas de maneira eficiente. A partir dessa abordagem, é possível garantir a segurança da capital utilizando a menor quantidade de recursos possível, como solicitado no enunciado do problema.

ANÁLISE DE COMPLEXIDADE

Na primeira etapa, é realizada a construção de um grafo a partir do mapa, em que cada posição válida (com valor diferente de zero, ou seja, que não são montanhas) é convertida em dois vértices (entrada e saída), com arestas entre eles e também entre vizinhos adjacentes. Como o mapa tem dimensão $n \times m$, essa etapa percorre todas as células e adiciona um número fixo de arestas por célula (no máximo 5). Assim, a complexidade dessa etapa é $O(n \times m)$.

Na segunda etapa, é feita a modelagem do caminho das bordas (origem) até a capital, que envolve no máximo $O(n + m)$ posições (as bordas), o que não afeta a complexidade geral do algoritmo.

Na terceira e principal etapa, é executado o algoritmo de fluxo máximo de Ford Fulkerson, com funções auxiliares. Ele realiza múltiplas buscas em largura (BFS) para construir níveis e buscas em profundidade (DFS) para encontrar o fluxo máximo. No pior caso, visto que é realizada uma BFS para construir o grafo de níveis, com custo $O(E)$, seguida por múltiplas DFS para aumentar o fluxo, também com custo $O(E)$, a complexidade do algoritmo é de $O(V^2 \times E)$, onde V é o número de vértices e E o número de arestas. Entretanto, como cada célula do mapa gera dois vértices, o grafo tem $O(n \times m)$ vértices e arestas, resultando em uma complexidade total de $O((n \times m)^3)$ para essa etapa, no pior caso.

Assim, a etapa mais custosa do algoritmo é o cálculo do fluxo máximo, com complexidade cúbica no pior caso. As demais etapas são lineares em relação ao número de células. Portanto, a complexidade total do algoritmo é $O((n \times m)^3)$.

CONSIDERAÇÕES FINAIS

Este trabalho prático me fez sair do padrão e pensar de forma diferente para resolver o problema. Ao invés de usar um algoritmo de Ford-Fulkerson tradicional, precisei implementar funções auxiliares para lidar com o fluxo de forma mais eficiente, o que expandiu meu entendimento sobre algoritmos de fluxo máximo. Achei muito interessante também a ideia de dividir cada vértice em dois (entrada e saída) para poder representar pesos nas posições do mapa, algo que eu nunca teria pensado por conta própria, mas que fez muito sentido após estudar mais a fundo. Apesar da dificuldade inicial, isso me motivou a pesquisar, aprender e, com isso, entender melhor como modelar problemas reais em grafos. De forma geral, o trabalho foi desafiador, mas extremamente enriquecedor, pois me ajudou a consolidar conceitos de grafos, algoritmos de fluxo e modelagem de problemas de forma criativa e eficiente.

REFERÊNCIAS

Slides da disciplina.

1. HACKEREARTH
HACKEREARTH. *Maximum Flow*. Disponível em: https://www.hackerearth.com/practice/algorithms/graphs/maximum-flow/tutorial/. Acesso em: 29 mai. 2025.

2. USP – Instituto de Matemática e Estatística
FUKUMOTO, P. F. *Grafos com pesos*. Instituto de Matemática e Estatística – USP. Disponível em: https://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/weightedgraphs.html. Acesso em: 28 mai. 2025.