ADLER GUILHERME FURTADO FARIA
2023028501

TP 1 ALG I

UFMG Belo Horizonte, 2024

INTRODUÇÃO

O problema abordado neste trabalho consiste em garantir a movimentação segura de pessoas durante uma situação de emergência em um evento ao ar livre, representado por um ambiente com diversos abrigos circulares. Cada abrigo é definido por suas coordenadas e raio de cobertura, e pessoas só podem se deslocar dentro dessas áreas cobertas, sem sair para regiões descobertas.

O objetivo principal é modelar esse cenário como um grafo, onde cada abrigo corresponde a um vértice e existe uma aresta entre dois abrigos sempre que suas áreas se sobrepõem ou são tangentes. A partir dessa modelagem, o problema é dividido em três partes computacionais, sendo: determinar o número mínimo de abrigos que duas pessoas precisam atravessar para se encontrarem, considerando que só podem se mover por áreas cobertas; calcular, entre todos os pares de abrigos acessíveis, o maior número de abrigos que alguém precisa atravessar em um caminho ótimo e identificar os abrigos críticos, ou seja, aqueles cuja remoção comprometeria a conectividade do sistema, impedindo a movimentação entre regiões.

Assim, após a análise do problema, optei por utilizar algoritmos de busca em grafos, especificamente os algoritmos BFS e DFS, visto que acredito ser o mais adequado para essa solução.

MODELAGEM E SOLUÇÃO

Este projeto foi desenvolvido em C++ e estruturado de forma a manter o código organizado e fácil de entender. A ideia central gira em torno da leitura de dados sobre pessoas e abrigos, da construção de um grafo que conecta esses abrigos quando suas áreas de cobertura se sobrepõem, e da aplicação de algoritmos clássicos de grafos para resolver três desafios principais.

O código se baseia em três estruturas principais: Pessoa, Abrigo e Grafo. A classe Pessoa armazena as coordenadas de alguém no plano e tem métodos simples para acessar ou alterar essas informações. Já a classe Abrigo guarda a posição e o raio de cobertura de cada abrigo, oferecendo funções que verificam se uma pessoa está dentro de sua área. O grafo é montado conectando abrigos cujas áreas circulares se cruzam ou encostam, e é representado como uma matriz ou lista de adjacência. Apesar de boa parte da lógica estar na função principal, essas estruturas dão o suporte necessário para manter tudo mais limpo e organizado.

O programa começa lendo as coordenadas de duas pessoas (chamadas aqui de Ana e Bruno) e as informações de cada abrigo — posição e raio. Esses dados são armazenados em vetores de objetos. Depois, o grafo é montado: se dois abrigos tiverem áreas que se sobrepõem ou tocam, eles são conectados. A função responsável por isso, chamada *estaoConectados*, compara a distância entre os centros dos abrigos com a soma de seus raios. Em seguida, o programa verifica em quais abrigos Ana e Bruno estão localizados, usando a função *pessoaDentroDoAbrigo*, que basicamente checa se uma coordenada está dentro do raio de um abrigo.

Na Parte 1, o objetivo é descobrir o menor número de abrigos que Ana e Bruno precisam atravessar para se encontrar, sempre ficando dentro da rede de cobertura. Para isso, o programa usa uma busca em largura (BFS) começando nos abrigos onde Ana está, e tenta chegar a

qualquer abrigo onde Bruno esteja. O menor caminho encontrado entre esses abrigos é a resposta.

A Parte 2 busca encontrar o "diâmetro" das componentes conexas do grafo, ou seja, a maior distância entre dois abrigos que ainda estão conectados. Para fazer isso de forma eficiente, o programa faz duas BFS consecutivas: a primeira parte de um abrigo qualquer e encontra o mais distante dentro daquela componente; a segunda começa nesse ponto e mede até onde é possível chegar — esse valor representa o diâmetro daquela componente. Esse processo é repetido para todas as regiões conectadas, e o maior valor encontrado é o resultado da etapa.

Já na Parte 3, o foco é encontrar os chamados abrigos críticos — pontos da rede cuja remoção dividiria o grafo em partes desconectadas. Para isso, o programa usa uma variação do algoritmo de Tarjan. A função *encontrarCriticosDFS* faz uma busca em profundidade (DFS) e mantém dois vetores: um com o tempo em que cada abrigo foi descoberto, e outro com o menor tempo que pode ser alcançado a partir de seus descendentes. Com base nessas informações, o algoritmo identifica os pontos de articulação, ou seja, os abrigos que são essenciais para manter a rede conectada. No fim, esses abrigos críticos são listados de forma ordenada.

Em resumo, o projeto combina conceitos de geometria e grafos de forma direta e eficiente para modelar a rede de abrigos, responder a perguntas sobre caminhos possíveis, acessibilidade e vulnerabilidades da estrutura.

ANÁLISE DE COMPLEXIDADE

O algoritmo proposto resolve o problema em quatro etapas principais, com diferentes complexidades computacionais.

Na construção do grafo, um laço duplo percorre todos os pares de abrigos (complexidade $O(n^2)$), verificando interseções entre seus raios de cobertura para estabelecer conexões.

Em seguida, na Parte 1, uma BFS (Busca em Largura) calcula o caminho mínimo entre os abrigos de Ana e Bruno, garantindo a menor distância em um grafo não ponderado (complexidade $O(n + m)$).

A Parte 2 determina o diâmetro (distância máxima) dentro de cada componente conexa, realizando duas BFSs por componente e marcando vértices visitados, mantendo a complexidade em $O(n + m)$.

Por fim, a Parte 3 utiliza uma DFS modificada (algoritmo de Tarjan) para identificar abrigos críticos (pontos de articulação) – vértices cuja remoção desconecta o grafo – também com complexidade $O(n + m)$.

Assim, a etapa mais custosa é a construção do grafo ($O(n^2)$), enquanto as demais operações têm complexidade linear ($O(n + m)$). Assim, a complexidade total do algoritmo é $O(n^2)$.

CONSIDERAÇÕES FINAIS

Em comparação às outras experiencias que tive com trabalhso praticos, acredito que tive uma experiencia muito boa durante a realização desse trabalho, ele possuiu explicações bem diretas o que facilitou no entendimento do problema, sendo o mais dificil para mim o entendimetno dos algoritmos vistos em aula e conseguir decidir qual utilizar em qual parte do código, visto que cada algoritmo é unico e tem sua própria especialidade.

REFERENCIAS

TARJAN, R. E. Depth-first search and linear graph algorithms. SIAM Journal on Computing, v. 1, n. 2, p. 146–160, 1972.

Slides da disciplina.