Trabalho Prático 01 - Algoritmos I

Lucas Rodrigues Pereira

Abril 2025

1 Introdução

Ana e Bernardo, dois amigos, foram juntos a um festival de música que contava com diversas barracas de comida. Ana decidiu experimentar os doces caseiros, enquanto Bernardo preferiu os sabores salgados. Entretanto, uma chuva repentina assolou o festival, transformando a experiência em uma grande correria por cobertura. Por sorte, dentro do festival havia espalhados diversos guardas-sóis que serviram como abrigo, Ana e Bernardo conseguiram ficar a salvos, e sob nenhuma circunstância estavam dispostos a se molhar. A partir dessa situação três problemas foram levantados.

Problema 1. Sabendo que somente é possível passar de um abrigo para outro, se e somente se, eles tem espaços em comum, ou seja, ao menos um ponto em comum. Determinar se Ana e Bernardo podem se encontrar e, caso possível, calcular o número mínimo de abrigos que eles devem atravessar para isso.

Problema 2. Considerando que as pessoas sempre andam buscando percorrer as menores distâncias. Determinar o maior número de abrigos que uma pessoa qualquer teria de percorrer para atravessar o festival após a chuva ter começado.

Problema 3. Determinar o número de abrigos críticos, ou seja, após o inicio da chuva, abrigos que se removidos tornariam alguns abrigos inacessíveis a partir de outros.

Tendo em mente as questões levantadas, é prático que o problema em si possa ser observado de forma que ferramentas conhecidas possam ser utilizadas. Para isso, o problema foi modelado como um Grafo.

2 Modelagem

A situação é modelada como um grafo não direcionado $G = (V, E)$, onde:

- V: conjunto de vértices, cada um representando um abrigo.
- E: conjunto de arestas, onde uma aresta (u, v) existe se os abrigo u e v têm interseção (i.e., a distância entre seus centros é $\leq$ soma dos raios).

De tal maneira, a partir de G reduzimos cada problema à:

Problema 1. Encontrar o caminho mais curto entre dois vértices em G, resolvido com BFS.

Problema 2. Calcular do diâmetro de G, que pode ser obtido a partir dos caminhos mais longos entre todos os pares de vértices.

Problema 3. Identificar vértices de articulação, que podem ser mapeados usando o Algoritmo de Tarjan.

Computacionalmente, o grafo foi representado como um vetor de listas, seguindo o conceito da representação por lista de adjacência.

Com cada problema reduzido a abstrações mais simples que nos munem de ferramentas algorítmicas poderosas, podemos avaliar a construção das soluções.

3 Solução

3.1 Problema 1

Seja v vértice de G, e $BFS(v)$ uma função que retorna um vetor de distâncias $D = [d_1, d_2, \ldots, d_{|V|}]$, onde d_i é a distância de v a v_i, $\forall v_i \in V$, e seja também $v_a, v_b, \in V$ os vértices iniciais de Ana e Bernardo. A solução obtida via BFS a partir de v_a segue os seguintes passos:

- Inicialize um vetor D com $D[v_i] = 0$, $\forall v_i \in V$.
- Execute BFS, atualizando $D[v]$ para cada vértice v visitado.
- Se v_b não for visitado, retorne -1. Caso contrário, retorne $D[v_b]$.

Demonstração. Como o BFS visita todos os vértices alcançáveis a partir de v_a em ordem de distância crescente, a primeira ocorrência de v_b na fila garante que $D[b]$ é a distância mínima. Se v_b não for visitado, o grafo é desconexo. □

3.1.1 Implementação

A para a resolução do problema, foi implementada a função $shortestPath(v_a, v_b)$, que implementa o BFS com otimização para interromper a busca quando v_b é encontrado, reduzindo tempo de execução em casos favoráveis.

3.2 Problema 2

O diâmetro de um dado grafo não ponderado e não orientado pode ser definido como:

Definição 1. *Seja $G = (V, E)$ um grafo não orientado e não ponderado. O diâmetro de G é definido como a maior distância mínima entre todos os pares de vértices distintos do grafo. Se G for desconexo, o diâmetro é o maior diâmetro entre suas componentes conexas.*

Tendo em mente a definição de diâmetro, e sendo G_i o i-ésimo sub-grafo conexo de G, podemos procurar uma solução da seguinte forma:

- Para cada G_i faça:
  - Execute BFS a partir de um vértice arbitrário v. Seja u o vértice mais distante de v.

  - Execute BFS a partir de u. Seja w o vértice mais distante de u.
  - Guarde a distância entre u e w.

- Retorne a maior distância encontrada.

Demonstração. Seja u o vértice mais distante de um vértice arbitrário v. Como o BFS encontra caminhos mais curtos, u deve ser um extremo do diâmetro. Executar BFS a partir de u encontra o vértice w mais distante, cuja distância é o diâmetro. Caso contrário, existiria um caminho mais longo não detectado, contradizendo a propriedade do BFS. □

3.2.1 Implementação

A implementação do problema foi feita em duas etapas. A primeira pela implementação da função $sizePaths(i)$, que recebe um índice de um vértice e executa o BFS a partir dele, retornando um vetor de distâncias para todos os vértices alcançados, se um vértice não é alcançado sua distância é 0, a distância de um vértice v para ele mesmo, na implementação, é sempre 1. A segunda através da função $greatestPath()$, que realiza as chamadas da função $sizePaths(i)$ para todos componentes conexos de G, além disso, após o calculo de todas as distâncias, retorna o maior comprimento encontrado.

3.3 Problema 3

Vértices de articulação são definidos como:

Definição 2. *Seja v vértice de G, e seja também $K(G)$, o número de componentes conexos em G. v é definido como vértice de articulação, se e somente se, $K(G - \{v\}) > K(G)$.*

Sabendo disso, podemos utilizar de algumas características de uma Árvore Geradora Mínima, obtida através do algoritmo de DFS, para inferir algumas condições suficientes para a classificação de um dado v como articulação ou não. Para isso definimos $time(v)$ como sendo o tempo em que v foi descoberto pelo DFS, e $low(v)$ como sendo o menor $time$ possível, considerando arestas de retorno, na sub-arvore onde v é raiz. Por fim, a solução é obtida da seguinte forma:

- Para cada G_i faça:
  - Execute DFS a partir de um vértice arbitrário.
  - Encontre a MST.
  - Compute todos os valores de $time$, e low.
  - Verifique para cada vértice v, a condição de articulação:
    * v é raiz da MST e tem dois ou mais filhos.
    * Seja v_c vértices filhos de v na MST, $\exists v_c : low(v_c) \geq time(v)$.
  - Guarde v, se v for um vértice de articulação.

- Retorne todos os vértices de articulação.

Demonstração. Se v é raiz da MST e tem dois ou mais filhos, sua remoção divide a árvore em componentes desconectadas, tornando-o um vértice de articulação. Caso contrário, se um filho v_e de v não puder alcançar ancestrais de v sem passar por v (i.e., $low(ve) \geq time(v)$, então v é um vértice de articulação. □

3.3.1 Implementação

A implementação do problema novamente se dividiu em duas etapas. A primeira a implementação da função $DFStree(i)$, que majoritariamente recebia um índice i para iniciar o DFS, e retornava as informações necessárias para a construção da MST. A segunda etapa consistiu da implementação da função $cutNodes()$, que realizava as chamadas da função $DFStree(i)$ para cada componente conexo de G e computava os valores de *time*, *low*, *childs* e *parents*, vetores respectivamente relacionados aos valores de *time* e *low* de cada vértice, o número de filhos e a referência para o pai de cada vértice dentro da MST. Por fim, a função verificava as satisfatibilidade das condições para vértice de corte para cada vértice, e retornava um map com os respectivos índices de cada vértice de corte encontrado.

4 Análise de Complexidade

4.1 Problema 1: Encontro de Ana e Bernardo

Função: `shortestPath(Node start, Node end)`
Tempo: $O(V + E)$.
A função implementa BFS, visitando cada vértice e aresta no máximo uma vez. A interrupção ao encontrar o nó destino não altera a complexidade assintótica.
Espaço: $O(V)$.
Utiliza vetores auxiliares (`visited` e `size_path`) e uma fila (`next_node`) proporcional ao número de vértices.

4.2 Problema 2: Diâmetro do Grafo

Funções: `greatestPath()` e `_sizePaths(long unsigned int idx)`
Tempo: $O(V(V + E))$.

- `_sizePaths`: Executa BFS ($O(V + E)$) para cada componente conexa.
- `greatestPath`: Itera sobre todos os vértices ($O(V)$) para cada componente, chamando `_sizePaths` duas vezes por componente (para encontrar os extremos do diâmetro).

Espaço: $O(V)$.
Armazena vetores temporários (`size_path_local`, `size_path_global`) e estruturas de fila/lista.

4.3 Problema 3: Abrigos Críticos

Funções: `cutNodes()` e `_DFStree(Node fnode, ...)`
Tempo: $O(V + E)$.

- `_DFStree`: Realiza uma DFS modificada ($O(V+E)$) para cada componente conexa.
- `cutNodes`: Processa vértices e arestas uma única vez durante a DFS, com operações adicionais $O(V)$ para calcular *low* e *time*.

Espaço: $O(V)$.
Utiliza vetores para *time*, *low*, *childs*, *parents*, e uma pilha implícita na DFS.

5 Considerações Finais

A implementação dos algoritmos seguiu rigorosamente a modelagem proposta, garantindo corretude e eficiência. As principais otimizações incluem:

- Interrupção antecipada do BFS no Problema 1 quando o alvo é encontrado.
- Cálculo do diâmetro com BFS duplo por componente conexa, evitando redundâncias.
- Identificação eficiente de vértices de articulação com `Tarjan`, utilizando estruturas auxiliares mínimas.

Como desafios, destacam-se a garantia de corretude para grafos desconexos e a validação dos valores de *low* e *time* no Problema 3. O uso de listas de adjacência mostrou-se ideal para grafos esparsos, típicos no contexto de abrigos com interseções localizadas.

6 Referências

- Cormen, T. H. *Introduction to Algorithms.* MIT Press, 2009.
- GeeksforGeeks. *Articulation Points (or Cut Vertices) in a Graph.* Disponível em: geeksforgeeks.org/articulation-points Acesso em: [19/04/2025].
- Kleinberg, J.; Tardos, É. *Algorithm Design.* Pearson, 2005.