Estruturas de Dados e Algoritmos II

Licenciatura em Engenharia Informática

2º Trabalho

Palm Island

André Gonçalves, 58392 | André Zhan, 58762 | Mooshak: g209

Departamento de Informática
Universidade de Évora
Abril 2025

Conteúdo

1 Problema

O problema *Palm Island Neighbours* consiste em determinar o maior caminho mínimo entre dois habitantes de uma ilha com ligações bidirecionais. Este problema traduz-se no cálculo do diâmetro de uma árvore.

1.1 Leitura dos Dados

O problema fornece como entrada o número de habitantes n, seguido de $n-1$ pares de inteiros que representam conexões bidirecionais entre os habitantes. Cada conexão representa uma rua que pode ser percorrida. Essas conexões formam uma árvore (não há ciclos e todos os habitantes estão conectados).

1.2 Solução para o Problema

Como a estrutura da ilha é uma árvore, o problema reduz-se a encontrar o **diâmetro da árvore**. Para isso, utilizamos duas buscas em profundidade (DFS):

1. Executa-se a primeira DFS a partir de um nó arbitrário, encontrando o nó mais distante (`farthestNode`).
2. Em seguida, uma segunda DFS é iniciada a partir desse `farthestNode`, retornando a maior distância possível: o diâmetro.

1.3 Função Recursiva

A função recursiva que define a DFS modificada para encontrar o diâmetro da árvore pode ser representada da seguinte forma:

$$dfsVisit(u) = \begin{cases} \text{Atualiza } d[u], \text{ cor para cinza} \\ \text{Para cada } v \in adjacents[u] \text{ tal que cor[v]} = \text{branco:} \\ \quad \text{distance[v]} \leftarrow \text{distance[u]} + 1 \\ \quad \text{dfsVisit(v)} \\ \quad \text{Se distance[v]} > \text{maxDistance, atualiza maxDistance e farthestNode} \\ \text{Atualiza } f[u], \text{ cor para preto} \end{cases}$$

Onde:

- `dfsVisit(u)` representa a visita ao nó u e suas ramificações.
- `distance[v]` armazena a distância acumulada desde o nó de origem até v.

- `maxDistance` e `farthestNode` são atualizados sempre que um novo caminho mais longo é encontrado.
- A cor dos nós segue o padrão DFS: branco (não visitado), cinza (em processamento), preto (finalizado).
- Os tempos de descoberta ($d[u]$) e finalização ($f[u]$) são registrados para cada nó.

1.4 Cálculo Iterativo

Algorithm 1: dfsModificada(*startNode*)

Input: startNode
Output: farthestNode

```
Inicializa:
    for u ← 0 to nodes - 1 do
        cor[u] ← BRANCO;
        p[u] ← -1;
        dist[u] ← 0;
    tempo ← 0;
    maxDistância ← 0;
    farthestNode ← startNode;
dfsVisit(startNode);
return farthestNode;
```

Algorithm 2: dfsVisit(u)

```
Input: u
Início da visita ao nó u:
    tempo ← tempo + 1;
    d[u] ← tempo;
    cor[u] ← CINZA;
    foreach v em adjacents[u] do
        if cor[v] = BRANCO then
            p[v] ← u;
            dist[v] ← dist[u] + 1;
            dfsVisit(v);
            if dist[v] > maxDistância then
                maxDistância ← dist[v];
                farthestNode ← v;
    cor[u] ← PRETO;
    tempo ← tempo + 1;
    f[u] ← tempo;
```

Diferenças para a DFS original

A versão original da DFS era genérica e percorria todos os nós: **Modificações:**

- Adição das variáveis globais `maxDistance` e `farthestNode`.
- DFS modificada para iniciar num nó específico.
- Durante a visita, atualiza-se a maior distância.
- A função `dfsVisit` passou a receber o array de distância e atualiza `maxDistance` e `farthestNode` ao encontrar um novo caminho mais longo.

Listing 1: dfs Modificada

```java
public static int dfs(int startNode) {
    Colour[] colour = new Colour[nodes];
    int[] distance = new int[nodes]; // Distance from the start node
    int[] p = new int[nodes]; // Predecessor
    int[] d = new int[nodes]; // Discovery time
    int[] f = new int[nodes]; // Finish time

    int[] time = new int[1]; // DFS time
```

```java
        // Initialize all nodes as unvisited
        for (int u = 0; u < nodes; u++) {
            colour[u] = Colour.WHITE;
            p[u] = -1; // No predecessor
            distance[u] = 0; // Distance from the start node
        }

        // Reset global variables before starting DFS
        time[0] = 0;
        maxDistance = 0;
        farthestNode = startNode;

        // Start DFS from the given start node
        dfsVisit(startNode, colour, p, d, f, time, distance);

        return farthestNode; // Return the farthest node found
    }
```

Listing 2: dfsVisit Modificada

```java
private static void dfsVisit(int u, Colour[] colour, int[] p, int[] d, int
        time[0]++;
        d[u] = time[0]; // Discovery of U
        colour[u] = Colour.GREY;

        for (int v : adjacents[u]) {
            if (colour[v] == Colour.WHITE) {
                p[v] = u; // U is the predecessor of V
                distance[v] = distance[u] + 1; // Calculate distance from
                dfsVisit(v, colour, p, d, f, time, distance);

                // Track the farthest node and its distance
                if (distance[v] > maxDistance) {
                    maxDistance = distance[v];
                    farthestNode = v;
                }
            }
        }
```

```
        colour[u] = Colour.BLACK; // Finished processing U
        time[0]++;
        f[u] = time[0];
    }
```

2 Análise de Complexidade

2.1 Complexidade Temporal

A complexidade temporal do programa pode ser analisada separando as principais operações:

1. **Construção do grafo:** A inserção das ligações nos arrays de adjacência é feita em tempo constante por aresta, totalizando:

$$O(N)$$

2. **Execução das duas DFS:** Cada DFS visita todos os nós e arestas uma vez:

$$O(N) \text{ por DFS } \Rightarrow O(2N) = O(N)$$

3. **Inicializações e leitura de dados:** Os arrays são inicializados em tempo linear:

$$O(N)$$

Complexidade Temporal Global: $O(N)$

2.2 Complexidade Espacial

A complexidade espacial do programa depende das seguintes estruturas:

- Lista de adjacência: $O(N)$
- Arrays auxiliares (cor, distância, predecessores, etc.): $O(N)$
- Stack da recursão (no pior caso): $O(N)$

Complexidade Espacial Total: $O(N)$

2.3 Conclusão

O programa apresenta uma complexidade temporal e espacial de $O(N)$, sendo altamente eficiente para resolver problemas de diâmetro de árvores com milhares ou mesmo milhões de nós.

3 Referências

Referências

[1] Slides das aulas teóricas.