UADE

TP Diseño y Análisis de Algoritmos

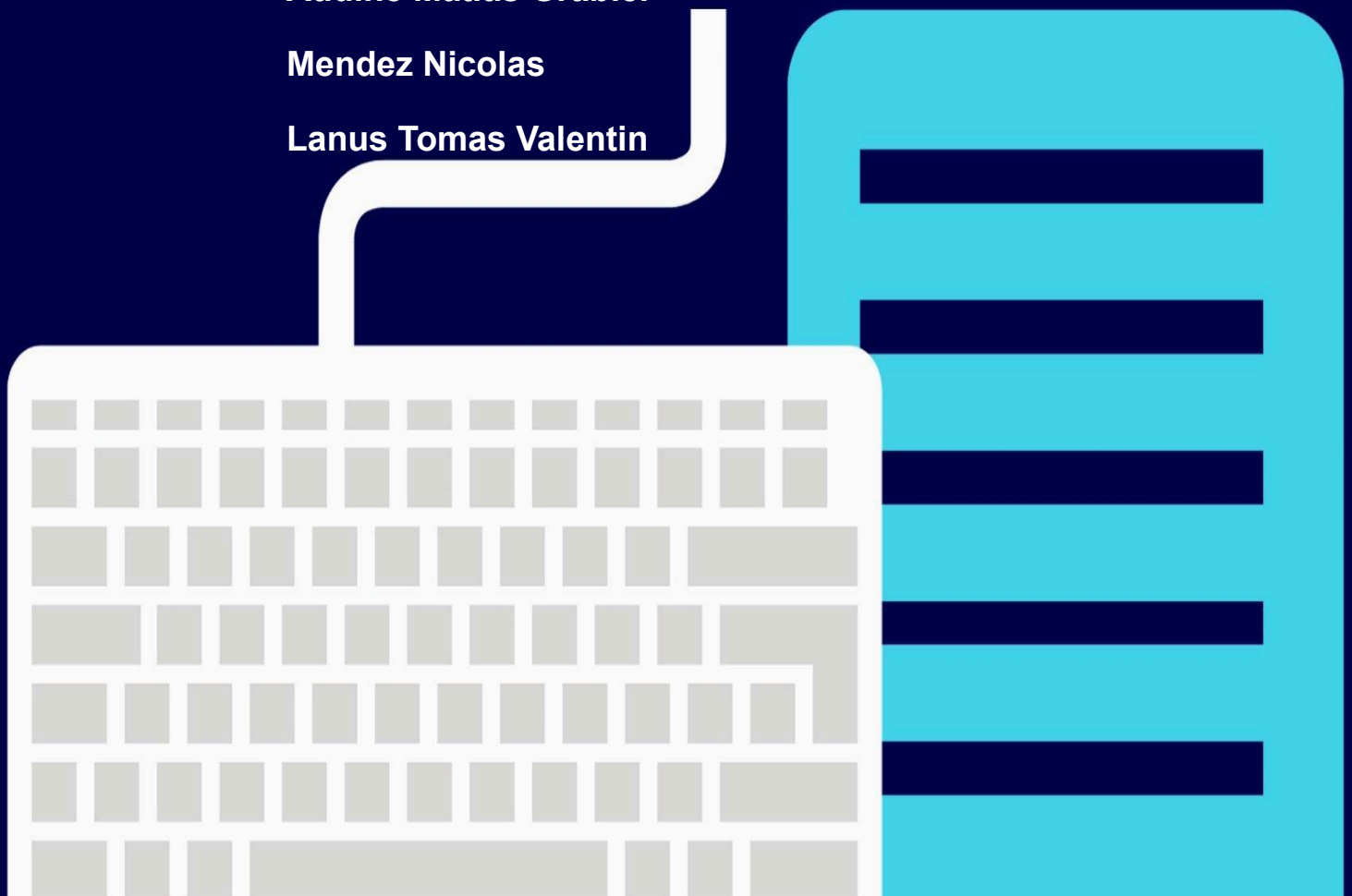
Profesor: Rodriguez Guillermo Horacio

Alumnos: Gomez Etcheber Geronimo

Audine Matias Grabiel

Mendez Nicolas

Lanus Tomas Valentin

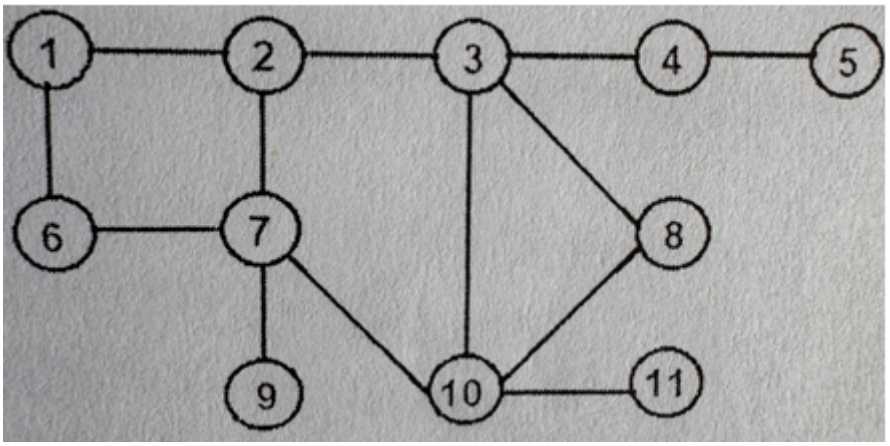


Enunciado:

El diámetro de un grafo **G = (V, E)** se define como el máximo de las excentricidades de todos sus vértices.

La excentricidad de un vértice **v** perteneciente a **V**, se define como la distancia máxima desde **v** a cualquier otro vértice siguiendo caminos de longitud mínima. Por ejemplo, el grafo de la figura tiene un diámetro 5.

Implemente un algoritmo que resuelva el problema de encontrar el diámetro de un grafo y calcule la complejidad temporal.



Informe del Proyecto: Cálculo del Diámetro de un Grafo

1. Justificación del Proyecto: El cálculo del diámetro de un grafo es un problema importante en ciencias de la computación y redes de telecomunicaciones. El diámetro de un grafo se puede entender como la máxima distancia entre todos sus vértices. Esta distancia máxima de un vértice se refiere a la longitud más larga desde dicho vértice hacia cualquier otro en el grafo, considerando únicamente los caminos más cortos. Calcular el diámetro permite evaluar la conectividad y eficiencia de una red, identificar puntos críticos y optimizar el rendimiento de la comunicación.

2. Diseño del Proyecto: El proyecto se estructura en torno a la implementación de un grafo no dirigido y un algoritmo de búsqueda en anchura (**BFS**) para calcular la excentricidad de cada vértice. La arquitectura del código es simple y se divide en los siguientes componentes:

- **Clase `Grafo`**: Representa el grafo con una lista de adyacencia.
- **Método `agregarArista`**: Agrega una conexión bidireccional entre dos vértices.

- **Método `BFS`**: Realiza una búsqueda en anchura para calcular la excentricidad de un vértice específico.
- **Método `calcularDiametro`**: Llama a `BFS` para cada vértice y encuentra la excentricidad máxima, es decir, el diámetro del grafo.
- **Método `main`**: Crea un objeto `Grafo`, agrega las aristas correspondientes y calcula el diámetro.

3. Método de búsqueda: La elección de la lista de adyacencia para representar el grafo (en este caso un grafo dinámico) es adecuada debido a la facilidad para acceder a los vecinos de cada vértice y la gran capacidad de eficiencia que tiene en el uso de memoria lo hace más eficiente en comparación con una matriz de adyacencia.

4. Pseudocódigo

```
Clase Grafo:
    Atributos:
        adjList: Lista de listas (representa la lista de adyacencia)
        numVertices: Número de vértices

    Método Constructor Grafo(numVertices):
        Inicializar numVertices
        Crear adjList con numVertices + 1 listas vacías

    Método agregarArista(origen, destino):
        Agregar destino a adjList[origen]
        Agregar origen a adjList[destino]

    Método BFS(v):
        Inicializar cola y distancias (con infinito)
        distancias[v] = 0
        Agregar v a la cola

        Mientras la cola no esté vacía:
            actual = cola.poll()
```

```
        Para cada vecino en adjList[actual]:
            Si distancias[vecino] es infinito:
                distancias[vecino] = distancias[actual] + 1
                Agregar vecino a la cola

    maxDistancia = 0
    Para cada distancia en distancias:
        Si distancia es válida:
            maxDistancia = max(maxDistancia, distancia)

    Retornar maxDistancia

Método calcularDiametro():
    maxExcentricidad = 0
    Para cada vértice i:
        excentricidad = BFS(i)
        maxExcentricidad = max(maxExcentricidad, excentricidad)

    Retornar maxExcentricidad

Método main():
    Crear grafo con 11 vértices
    Agregar aristas al grafo
    diametro = calcularDiametro()
    Imprimir "El diámetro del grafo es: " + diametro
```

5. Cálculo de la Complejidad Temporal: El algoritmo **BFS** utilizado para calcular la excentricidad de un vértice tiene una complejidad temporal de **O(V+E)**, donde **V** es el número de vértices y **E** es el número de aristas en el grafo. Dado que se realiza una búsqueda desde cada vértice para calcular el diámetro, la complejidad total del método `calcularDiametro` es **O(V · (V+E))**.

En resumen:

- Complejidad del método `BFS`: **O(V+E)**
- Complejidad del método `calcularDiametro`: **O(V · (V+E))**

Análisis de Complejidad de `BFS(grafo, v)`

1. **Inicialización**:
    - Inicializar la cola `Q` y el array `distancias` toma O(V), donde V es el número de vértices.
2. **Proceso de BFS**:
    - El bucle `while` recorre cada vértice una sola vez y se ejecuta hasta que todos los vértices alcanzables desde `v` se visiten. Esto toma O(V).
    - Para cada vértice `actual`, se revisan sus vecinos en la lista de adyacencia. El número total de veces que se revisan los vecinos en todas las iteraciones es igual al número de aristas E. Esto se debe a que cada arista se revisa una vez en cada dirección (una vez desde cada uno de sus extremos), lo que tiene una complejidad de O(E).

Complejidad del método `BFS`: O(V+E).

Análisis de Complejidad de `calcularDiametro(grafo, numVertices)`

1. El método `calcularDiametro` llama a `BFS` desde cada vértice una vez. Si hay V vértices en el grafo, el método `BFS` se ejecuta V veces.
2. Como cada ejecución de `BFS` tiene una complejidad de O(V+E), y `calcularDiametro` ejecuta `BFS` V veces, la complejidad total es:

$O(V) \cdot O(V+E) = O(V \cdot (V+E))$

Desglose Final:

- **Número de veces que se ejecuta `BFS`**: V veces.
- **Complejidad de cada `BFS`**: O(V+E).

Por lo tanto, la complejidad de `calcularDiametro` es:

$O(V \cdot (V+E))$

Esto significa que el tiempo de ejecución depende del producto del número de vértices y la suma del número de vértices y aristas.

6. Conclusiones El proyecto proporciona una solución efectiva para encontrar el diámetro de un grafo no dirigido mediante un enfoque simple basado en BFS, a su vez se beneficia de la estructura de la lista de adyacencia, ya que permite acceder directamente a todos los vecinos de un vértice. Aunque el enfoque es adecuado para grafos de tamaño moderado, puede ser ineficiente para grafos muy grandes debido a la complejidad de O(V^2) en casos densos. Para mejorar la eficiencia en grafos grandes, podrían explorarse algoritmos optimizados como el de Floyd-Warshall (Aunque haya que adaptarlo para un grafo no dirigido) o técnicas heurísticas.