Ayudantía 10

Carlos Lagos - carlos.lagosc@usm.cl

Introducción a los Grafos

- **Grafo**: Conjunto de nodos (vértices) y aristas (enlaces) que conectan los nodos.
- **Tipos de Grafos**:
  - **No Dirigidos**: Las aristas no tienen dirección.
  - **Dirigidos**: Las aristas tienen dirección.
  - **Ponderados**: Las aristas tienen un peso o costo asociado.

Matriz de Adyacencia

- **Descripción**: Matriz 2D donde cada celda (i, j) indica si hay una arista entre los nodos i y j.
- **Ventajas**: Acceso rápido para verificar la existencia de una arista.
- **Desventajas**: Requiere O(V^2) espacio.

```
   0  1  2  3
0 [0, 1, 0, 0]
1 [1, 0, 1, 0]
2 [0, 1, 0, 1]
3 [0, 0, 1, 0]
```

Lista de Adyacencia

- **Descripción**: Lista donde cada entrada i contiene una lista de nodos adyacentes a i.
- **Ventajas**: Más eficiente en términos de espacio, especialmente para grafos dispersos.
- **Desventajas**: Acceso más lento para verificar la existencia de una arista.

```
0: [1]
1: [0, 2]
2: [1, 3]
3: [2]
```

BFS (Breadth-First Search)

- **Descripción**: Explora los nodos nivel por nivel.
- **Complejidad**: $O(V + E)$

BFS (Breadth-First Search)

```
void BFS(tGrafo& G, tVertice v) {
    tCola Q; tVertice w, z;
    Q.enqueue(v);
    G.setMark(v, VISITADO);
    while (Q.size() != 0) {
        z = Q.frontValue(); Q.dequeue();
        visitar(z);
        for (w = G.first(z); w < G.nVertex(); w = G.next(z, w))
        if (G.getMark(w) == NOVISITADO) {
            G.setMark(w, VISITADO);
            Q.enqueue(w);
        }
    }
}
```

DFS (Depth-First Search)

- **Descripción**: Explora los nodos en profundidad antes de retroceder.
- **Complejidad**: O(V + E)

```
void DFS(tGrafo& G, tVertice v) {
    tVertice w;
    visitar(G, v);
    G.setMark(v, VISITADO);
    for (w = G.first(v); w < G.nVertex(); w = G.next(v, w))
        if (G.getMark(w) == NOVISITADO)
            DFS(G, w);
}
```

Dijkstra

- **Descripción**: Encuentra el camino más corto desde un nodo fuente a todos los demás nodos en un grafo ponderado.
- **Complejidad**: $O(V^2)$ sin cola de prioridad
- **Complejidad**: $O((V+E)\log(V))$ con cola de prioridad

Pasos del Algoritmo de Dijkstra

1. **Inicialización**:
    - Asigna una distancia infinita a todos los nodos excepto al nodo fuente, cuya distancia se asigna a 0.
    - Crea una estructura para almacenar nodos pendientes (puede ser una cola de prioridad o un conjunto de nodos).
2. **Selección del Nodo Actual**:
    - Extrae o selecciona el nodo con la distancia mínima de la estructura de nodos pendientes.
3. **Actualización de Distancias**:
    - Para cada nodo adyacente al nodo actual, calcula la distancia posible a través del nodo actual.
    - Si esta nueva distancia es menor que la distancia actualmente conocida, actualiza la distancia.
    - Si se actualiza la distancia, también actualiza la estructura de nodos pendientes con la nueva distancia.
4. **Repetición**:
    - Repite los pasos 2 y 3 hasta que todos los nodos hayan sido procesados (la estructura de nodos pendientes esté vacía).

Dijkstra

```
void Dijkstra(tGrafo& G, int *D, tVertice s) {
    tVertice v, w;
    int i;
    D[s] = 0;
    for (i = 0; i < G.nVertex(); i++) {
        v = minVertex(G, D);
        if (D[v] == INFINITO)
            return;
        G.setMark(v, VISITADO);
        for (w = G.first(v); w < G.nVertex(); w = G.next(v, w))
        if (D[w] > (D[v] + G.weight(v, w)))
            D[w] = D[v] + G.weight(v, w);
    }
}
```

Algoritmo de Prim

- **Descripción**: Encuentra el Árbol Recubridor Mínimo (MST) de un grafo ponderado seleccionando iterativamente la arista de menor peso que conecta un vértice del MST con uno fuera de él.
- **Pasos**:
  1. Elegir un vértice inicial y marcarlo como visitado.
  2. Iterar hasta que todos los vértices estén incluidos en el MST:
     - Añadir la arista más pequeña que conecte un vértice del MST con uno no visitado.
     - Marcar el vértice conectado como visitado.
- **Complejidad**: O((V + E) log V) con cola de prioridad

Algoritmo de Kruskal

- **Descripción**: Encuentra el Árbol Recubridor Mínimo (MST) de un grafo ponderado seleccionando iterativamente las aristas más pequeñas que no forman ciclos.
- **Pasos**:
  1. Ordenar todas las aristas en orden no decreciente según su peso.
  2. Inicializar un bosque (conjunto de árboles) donde cada vértice es un árbol separado.
  3. Iterar sobre las aristas ordenadas y añadir una arista al MST si no forma un ciclo con las aristas previamente seleccionadas.
- **Complejidad**: $O(|E| \log |E|)$

Ejercicios

Preguntas

- DFS permite encontrar el camino más corto en un grafo no dirigido en que todos los arcos tienen el mismo peso.
- La lista de adyacencia siempre ocupará menos espacio que la matriz de adyacencia.
- La complejidad de obtener el elemento del tope desde un heap es O(log n).

Preguntas

- Un heap es un árbol binario que al ser casi-lleno se puede implementar con un arreglo.
- Prim y Kruskal pueden entregar un árbol recubridor mínimo distinto.
- El tiempo de ejecución $O((|V| + |E|) \log |V|)$ de Dijkstra se logra utilizando un max-Heap y la representación de lista de adyacencia.

Instagram

Dado un digrafo G que representa a la red social Instagram. Donde los vértices representan a los usuarios, y las aristas (vi, vj) indican que el usuario vi sigue al usuario vj. Implemente la función Influencers en C++ (y utilizando el TDA tGrafo visto en clases), que recibe como parámetro el grafo G y un entero m. La función retorna a los usuarios (vértices) que tienen al menos m seguidores. Si el grafo G tiene |V| vértices y |E| aristas, el tiempo de ejecución de su algoritmo debe ser O(|V| + |E|). Argumente cuál representación de grafos (de las vistas en clases) usaría para lograr el tiempo indicado.