UNIVERSIDAD ADOLFO IBÁÑEZ
FACULTAD DE INGENIERÍA Y CIENCIAS

Ayudantía 8: Árboles en C

Profesores: Sebastián Sáez, Diego Ramos

Ayudantes: Diego Duhalde, Benjamín Wiedmaier, Fernando Zamora

PREGUNTAS

1. Defina qué es un grafo, diferenciando *nodos* y *aristas*. Además, explique la diferencia entre grafo dirigido y no dirigido, y dé un ejemplo de aplicación real para cada uno.

2. Dado el siguiente grafo G:

$$N = \{A, B, C, D, E, F\}$$
$$E = \{\{A, B\}, \{B, C\}, \{C, A\}, \{C, D\}, \{D, E\}, \{D, B\}, \{D, F\}, \{E, A\}, \{E, F\}\}$$

   a) Dibújelo y señale claramente
      - Un ciclo de G.
      - Un árbol de G.

   b) Explique la principal diferencia entre un ciclo y un árbol.

   c) Proponga al menos una aplicación práctica en la que sea importante detectar ciclos o extraer árboles de un grafo, y justifique su elección.

3. Compare brevemente (en tiempo y espacio) las dos principales representaciones de grafos: matriz de adyacencia y lista de adyacencia.

4. Explique que es un árbol de expansión mínimo y mencione cuáles algoritmos existen para determinarlo.

5. a) Explique brevemente el algoritmo de Kruskal y mencione cómo asegura que el árbol generado es de expansión mínima. Además, escriba en C un código que implemente Kruskal para un grafo no dirigido.

   b) Haga lo mismo con el algoritmo de Prim.

   c) ¿Cuáles podrían ser las ventajas y desventajas de usar Kruskal versus Prim en la práctica? Considere aspectos como la estructura de datos utilizada, la eficiencia en distintos tipos de grafos y la facilidad de implementación.

6. a) Explique qué es el camino más corto y cómo el algoritmo de Dijkstra permite encontrarlo en un grafo con pesos no negativos. Además, escriba en C un código que implemente el algoritmo de Dijkstra para hallar la distancia mínima desde un nodo origen a todos los demás nodos.

   b) ¿Por qué no se pueden ocupar pesos negativos?

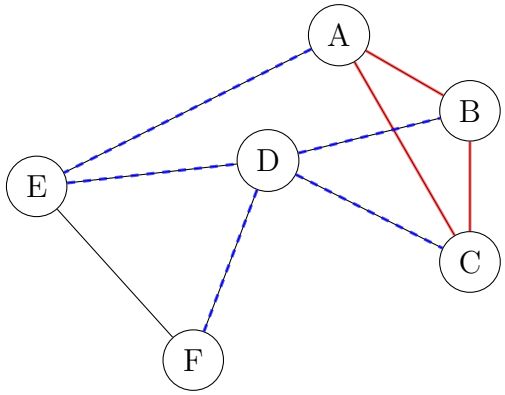
RESPUESTAS

1. Un grafo es una estructura matemática compuesta por un conjunto de nodos (o vértices) y un conjunto de aristas (o enlaces) que conectan pares de nodos. Los nodos representan entidades o puntos, mientras que las aristas representan las conexiones o relaciones entre ellos.

   La diferencia principal entre grafo dirigido y no dirigido es la orientación de las aristas:

   - **Grafo dirigido:** Las aristas tienen una dirección, es decir, van de un nodo origen a un nodo destino. Ejemplo de aplicación: redes de flujo de tráfico, donde las calles pueden ser unidireccionales.
   - **Grafo no dirigido:** Las aristas no tienen dirección, por lo que la relación entre los nodos es bidireccional. Ejemplo de aplicación: redes sociales, donde la amistad es mutua.

2. a) El grafo G se puede dibujar de la siguiente manera:



   donde, el ciclo está resaltado en rojo (aristas gruesas), mientras que un árbol está resaltado en azul (aristas gruesas y punteadas).

   b) La principal diferencia es que un ciclo es un camino cerrado en el grafo donde se puede volver al nodo inicial sin repetir aristas ni nodos (excepto el inicial/final), mientras que un árbol es un subgrafo conexo y acíclico (no tiene ciclos).

   c) Una aplicación práctica donde es importante detectar ciclos es en la detección de dependencias circulares en sistemas de paquetes de software (para evitar bucles de dependencia). Extraer árboles es fundamental, por ejemplo, en el diseño de redes eléctricas o de comunicaciones, donde se busca conectar todos los puntos con el menor costo posible y sin ciclos (árbol de expansión mínima).

3. **Matriz de adyacencia:**

   - **Tiempo:** Permite verificar si existe una arista entre dos nodos en tiempo $O(1)$, pero recorrer todos los vecinos de un nodo toma $O(n)$.
   - **Espacio:** Requiere $O(n^2)$ espacio, donde n es el número de nodos, ya que almacena todas las posibles conexiones.
   - **Ventaja:** Es eficiente para grafos densos.

- **Desventaja:** Consume mucha memoria en grafos dispersos.

Lista de adyacencia:

- **Tiempo:** Verificar la existencia de una arista puede tomar hasta $O(\deg(v))$, donde $\deg(v)$ es el grado del nodo, y recorrer todos los vecinos también tiene complejidad $O(\deg(v))$.
- **Espacio:** Requiere $O(n+m)$ espacio, donde m es el número de aristas, siendo mucho más eficiente para grafos dispersos.
- **Ventaja:** Ahorra memoria en grafos con pocas aristas.
- **Desventaja:** Menos eficiente para verificar la existencia de una arista específica.

4. Un árbol de expansión mínima (MST, por sus siglas en inglés) es un subgrafo de un grafo no dirigido, conexo y ponderado, que conecta todos los nodos con el menor peso total posible y sin formar ciclos (es decir, es un árbol). El MST contiene exactamente $n-1$ aristas si el grafo tiene n nodos.

   Los algoritmos más conocidos para encontrar un árbol de expansión mínima son:

   - **Kruskal:** Selecciona las aristas de menor peso y las agrega al árbol si no forman un ciclo, hasta conectar todos los nodos.
   - **Prim:** Comienza desde un nodo y va agregando la arista de menor peso que conecta un nodo del árbol a uno fuera de él, hasta incluir todos los nodos.

5. a) **Algoritmo de Kruskal:** Este algoritmo ordena todas las aristas del grafo por peso creciente y las va agregando al árbol de expansión si no forman un ciclo, utilizando una estructura de conjuntos disjuntos (Union-Find) para detectar ciclos. Así, siempre se agregan las aristas más baratas posibles sin cerrar ciclos, garantizando el mínimo costo total. La implementación del código la puede encontrar en la rama de ayudantías del repositorio del curso. Link.

   b) **Algoritmo de Prim:** Parte desde un nodo arbitrario y, en cada paso, agrega la arista de menor peso que conecta un nodo ya incluido con uno que aún no está en el árbol. Utiliza una estructura como un heap para seleccionar rápidamente la arista mínima. Lo mismo acá.

   c) **Ventajas y desventajas:** Ambos algoritmos tienen ventajas y desventajas según el tipo de grafo y la estructura de datos utilizada. Kruskal suele ser más eficiente en grafos dispersos y cuando las aristas se presentan como lista, además de ser sencillo de implementar con la estructura Union-Find; sin embargo, requiere ordenar todas las aristas, lo que puede ser costoso en grafos muy densos. Por otro lado, Prim es más eficiente en grafos densos y cuando se utiliza una matriz de adyacencia, especialmente si se emplea un heap para seleccionar la arista mínima; no obstante, puede ser menos eficiente en grafos dispersos si no se elige la estructura adecuada.

6. a) El **camino más corto** entre dos nodos en un grafo es la secuencia de aristas que conecta ambos nodos con el menor peso total posible. El **algoritmo de Dijkstra** permite encontrar el camino más corto desde un nodo origen a todos los demás en un grafo con pesos no negativos, utilizando una estructura de prioridad (como un heap)

para seleccionar el nodo con la distancia mínima conocida en cada paso y actualizando las distancias a sus vecinos. El código lo puede encontrar acá.

b) Dijkstra no funciona correctamente con pesos negativos porque podría pasar por un nodo y luego encontrar un camino más corto a ese nodo usando una arista negativa, pero el algoritmo ya habría marcado ese nodo como visitado y no actualizaría su distancia, produciendo resultados incorrectos.