Universidad Nacional Autónoma de México

Facultad de Ciencias

Tarea 09

Presentan

- Nelson Osmar Garcia Villa - 322190357
- Valeria Camacho Hernández - 322007273
- Mauricio Casillas Álvarez - 322196342

Asignatura

Gráficas y Juegos 2025-2

Profesor

César Hernández Cruz

Ayudante

Iñaki Cornejo de la Mora

Fecha

Viernes 30 de mayo del 2025

Tarea 09

1. Sea G una gráfica conexa y e una arista de G que no sea un lazo. Exhiba una biyección entre el conjunto de árboles generadores de G que contienen a e y el conjunto de árboles generadores de G/e (considere que G/e puede tener multiaristas).

   Respuesta:

   (a) De árboles en G que contienen e a árboles en G/e

   Sea T un árbol generador de G tal que $e \in T$. Un árbol generador, es aquella gráfica que tiene todos los vértices de G, usando el mínimo de aristas sin generar ciclos y que sea conexa.

   Supongamos que existe una arista $e \in T$, que conecta a u y v.

   Luego al ser una arista $e = uv$, como lo dice la característica del ejercicio, tenemos que quitar la arista e, pero al ser un árbol T generador de G hace que se vuelva inconexo el árbol. Entonces hacemos que los vértices u y v se vuelvan un nuevo vértice denominado w.

   Implica que el árbol T se convierte en el conjunto $T' = T \setminus \{e\}$, teniendo ahora $n-1$ vértices $(n-1)-1 = n-2$ aristas, y que siga siendo conexo y sin ciclos, ya que eliminar e no desconecta la gráfica.

   $\therefore T'$ es un árbol generador del grafo G/e.

   (b) De árboles en G/e a árboles en G que contienen e

   Sea T' un árbol generador de G/e.

   El cual, por la demostración anterior, G/e contiene un vértice w que representa a los vértices u y v de la gráfica original G.

   Hacemos que u y v vuelvan a conectar sus respectivas aristas, es decir aquellas aristas que eran los extremos de w. Y agregamos e de $e = uv$, es decir, volvemos a separar w en los vértices originales u y v.

   Añadimos la arista $e = uv$ al conjunto de aristas de T'.

   Implica que el nuevo conjunto tiene $n-2+1 = n-1$ aristas, que se completen todos los vértices y que siga siendo conexo, ya que u y v están conectados directamente por e, y el resto del grafo ya estaba conectado en T'. Además, no se forma ningún ciclo, ya que sólo añadimos una arista.

   $\therefore T = T' \cup \{e\}$ es un árbol generador de G que contiene a e.

2. Sea T un árbol óptimo en una gráfica conexa ponderada (G, w) (con pesos positivos), y sean x y y vértices adyacentes en T. Demuestre que la trayectoria $xTy = xy$ es una xy-trayectoria de peso mínimo en G.

   Respuesta:

   Demostración

   Sea T un árbol generador óptimo (de peso mínimo) de la gráfica conexa ponderada G con función de peso w, donde $w(e) > 0$ para toda arista $e \in E(G)$. Supongamos que x e y son vértices adyacentes en T, es decir, la arista xy pertenece a T.

   Por definición, en un árbol generador óptimo, para cualquier par de vértices u y v, la trayectoria única uTv en T es una trayectoria de peso mínimo entre u y v en G. Esta es una propiedad fundamental de los árboles generadores óptimos.

   En particular, para los vértices x e y adyacentes en T, la trayectoria xTy consiste únicamente en la arista xy. Por la propiedad mencionada, esta trayectoria debe ser de peso mínimo en G. Es decir:

   $$w(xTy) = w(xy) \leq w(P) \quad \text{para toda } xy\text{-trayectoria } P \text{ en } G$$

   Por lo tanto, la trayectoria $xTy = xy$ es efectivamente una xy-trayectoria de peso mínimo en G.

3. Demuestre que si todos los pesos de una gráfica ponderada G son distintos, entonces G tiene un único árbol óptimo.

   Respuesta:

   Consideremos el algoritmo de Kruskal aplicado a la gráfica $G = (V, E)$. Este algoritmo construye un árbol generador mínimo seleccionando aristas en orden creciente de peso, siempre que no formen un ciclo con las aristas previamente seleccionadas.

   Luego sabemos, por meción del enunciado, que todos los pesos de las aristas son distintos. Esto implica que para cada par de aristas $e_1, e_2 \in E$, se cumple $w(e_1) \neq w(e_2)$, por lo tanto, el conjunto de aristas puede ordenarse de forma creciente según su peso.

   Así, el algoritmo de Kruskal en cada iteración hay una única arista de menor peso que se puede agregar sin formar un ciclo. Por lo tanto, el conjunto de aristas seleccionadas está completamente determinado, y el árbol generador mínimo resultante es único.

   Supongamos, por contradicción, que existen dos árboles generadores mínimos distintos T_1 y T_2 con el mismo peso total. Entonces, si ejecutamos el algoritmo de Kruskal sobre G, debería producir tanto T_1 como T_2. Sin embargo no se puede llevar acabo, ya que el algoritmo solo puede devolver un único conjunto de aristas cuando los pesos son todos distintos. Esto contradice nuestra suposición de que existían dos árboles generadores distintos distintos.

4. Demuestre que el problema de encontrar un árbol generador de peso máximo en una gráfica conexa puede resolverse eligiendo iterativamente una arista de peso máximo, con la condición de que la subgráfica resultante siga siendo un bosque. (Proponga un algoritmo y demuestre que es correcto.)

   Respuesta:

Algoritmo Propuesto: Kruskal Modificado para Máximo

Algorithm 1 Algoritmo para árbol generador de peso máximo

Require: Gráfica conexa ponderada $G = (V, E)$ con función de peso $w : E \to \mathbb{R}$
Ensure: Árbol generador de peso máximo T

```
Ordenar las aristas E en orden no creciente según w
T ← (V, ∅)                                          ▷ Inicializar bosque trivial
for cada arista e = {u, v} en el orden no creciente do
    if u y v están en componentes distintas en T then
        T ← T ∪ {e}
    end if
    if T tiene |V| − 1 aristas then break

        return T
```

Demostración de Correctitud

Un *árbol generador de peso máximo* (MST) es un árbol generador cuyo peso total es máximo entre todos los posibles árboles generadores de G.

[Propiedad de la arista segura para máximo] Sea $G = (V, E)$ una gráfica conexa ponderada, y sea $\mathcal{F}$ un bosque parcial de algún MST de G. Si e es una arista de peso máximo que conecta dos componentes distintas de $\mathcal{F}$, entonces e es segura para $\mathcal{F}$ (existe un MST que contiene a $\mathcal{F} \cup \{e\}$).

Sea T^* un MST que no contiene a e. Al añadir e a T^*, se crea un ciclo C. Como e conecta dos componentes de $\mathcal{F}$, existe al menos otra arista $e' \in C$ que también conecta estas componentes. Por la elección de e, $w(e) \geq w(e')$.

Si $w(e) > w(e')$, reemplazar e' por e en T^* produciría un árbol generador de mayor peso, contradiciendo la maximalidad de T^*. Por tanto, $w(e) = w(e')$, y $T^* \cup \{e\} \setminus \{e'\}$ es otro MST que contiene a e.

El algoritmo propuesto encuentra correctamente un árbol generador de peso máximo.

La demostración sigue por inducción:

(a) Invarianza: En cada paso, el bosque T es subgráfico de algún MST. Esto se cumple inicialmente ($T = (V, \emptyset)$) y el Lema garantiza que se preserva.

(b) Terminación: El algoritmo termina cuando T tiene $|V| - 1$ aristas, pues no se agregan aristas que formen ciclos. Como G es conexa, siempre se alcanza este tamaño.

(c) Optimalidad: Al final, T es árbol generador (conexo y sin ciclos) y, por el Lema, es de peso máximo.

Complejidad Temporal

- Ordenar aristas: $O(|E| \log |E|)$
- Verificación de ciclos (con Union-Find): $O(|E| \, \alpha(|V|))$, donde α es la función inversa de Ackermann
- Total: $O(|E| \log |E|)$

Ejemplo Sea G con vértices $\{a, b, c\}$ y aristas:

- ab con peso 3
- bc con peso 5
- ac con peso 4

Ejecución del algoritmo:

(a) Ordena aristas: $[bc, ac, ab]$

(b) Agrega bc ($T = \{bc\}$)

(c) Agrega ac ($T = \{bc, ac\}$)

(d) Ignora ab (formaría ciclo abc)

El árbol generador de peso máximo es $\{bc, ac\}$ con peso total 9.

5. Escriba una versión del algoritmo BFS para digráficas. Utilice esta versión de BFS dirigida para describir un algoritmo que encuentre un ciclo dirigido de longitud mínima en una digráfica. Su versión dirigida de BFS debe de correr en tiempo $\mathcal{O}(|V| + |E|)$, y el algoritmo para encontrar el ciclo dirigido más corto debe correr en tiempo a lo más $\mathcal{O}(|V|^2 + |V||E|)$.

   Respuesta:

   - Algoritmo BFS para digráficas.

   Proponemos la siguiente modificación al algoritmo BFS para gráficas dirigidas en el que se sigue a las aristas salientes desde cada vértice, visita cada vértice una vez (cuando no está "coloreado"), y además calcula niveles (ℓ), tiempos de descubrimiento (t), y padre (p).

   Input: Una digráfica G con un vértice distinguido r.

   Output: Funciones de parentesco p, nivel ℓ, y tiempo de exploración t.

   1. $Q \leftarrow [\ ]$; Inicializamos la cola vacía donde iremos agregando los vértices por procesar.

   2. $i \leftarrow 0$; Iniciamos un contador para el tiempo de cada visita.

   3. $i \leftarrow i+1$; Incrementamos el contador antes de usarlo por primera vez para que $t(r)$ comience en 1.

   4. colorear a r de negro ; Marcamos el vértice inicial r como visitado.

   5. añadir r al final de Q ; Ponemos el vértice inicial r en la cola.

   6. $t(r) \leftarrow i; p(r) \leftarrow \emptyset; \ell(r) \leftarrow 0$; Inicializamos sus atributos.

   7. while $Q \neq [\ ]$ do

   8. $x \leftarrow$ primer elemento de Q ; Sacamos la cabeza de la cola.

   9. for each y vecino saliente de x do ; Trabajamos con las aristas que salen de x. Cada vez que encontramos un nuevo vértice y, hacemos lo siguiente:

   10. if y no está coloreado then ; Revisamos si el vecino y aún no ha sido visitado. Como en BFS queremos visitar cada vértice una sola vez, esto evita que repitamos vértices. Si ya fue coloreado (es decir, ya fue descubierto antes), lo ignoramos.

   11. $i \leftarrow i + 1$; Aumentamos el contador

   12. colorear y de negro ; Marcamos a y como visitado.

   13. añadir y al final de Q ; Lo agregamos a la cola para procesarlo más adelante.

   14. $t(y) \leftarrow i$; Registramos cuándo lo descubrimos (el tiempo).

   15. $p(y) \leftarrow x$; Guardamos de cuál vértice venimos, osea establecemos que x es el padre de y.

   16. $\ell(y) \leftarrow \ell(x)+1$; Guardamos cuántas aristas hay desde r hasta y (el nivel).

17. eliminar x de Q ; Eliminamos al vértice después de que ya procesamos sus adyacencias.

18. return (p, ℓ, t)

Sobre su complejidad, la inicialización de estructuras cuesta $\mathcal{O}(1)$ por vértice, por lo que el costo total de inicialización es $\mathcal{O}(|V|)$. Luego, el procesamiento de vértices a través de la cola se hace una sola vez por cada vértice (cada vértice entra y sale de la cola una vez), lo cual también cuesta $\mathcal{O}(|V|)$. Para explorar los vecinos salientes, se procesa cada arista saliente una sola vez, lo que tiene un costo total de $\mathcal{O}(|E|)$. Por lo tanto, su complejidad es $\mathcal{O}(|V + E|)$.

- Algortimo BFS para encontrar un ciclo dirigido de longitud miníma en digráficas.

Proponemos la siguiente modificación al algoritmo anterior para encontrar un ciclo dirigido de longitud mínima en una digráfica.

Como el BFS original está diseñado para explorar sin repetir vértices, esto impide detectar ciclos. Sin embargo, si podemos registrar cuándo volvemos a un vértice ya descubierto, podemos identificar un ciclo por reconstrucción.

Observando las líneas 10 a 16 del algoritmo, notamos que solo se hace algo con el vecino y si aún no ha sido coloreado, pues si y ya fue descubierto, simplemente se ignora. Esto significa que estamos ignorando aristas que podrían formar parte de un ciclo. En BFS original, eso se ignora porque solo se procesan vértices no visitados.

Para solucionar esto, nos quedamos con las líneas 1 a 8, y modificamos lo que sucede en la exploración de vecinos (línea 9). Así, cuando estemos en un vértice x y encontramos un vecino saliente y, si y ya fue visitado (ya está coloreado) y además y no es el padre de x, entonces el ciclo se podría cerrar.

En ese caso, guardamos los dos vértices (x, y) que define esta arista que puede cerrar un ciclo. Luego podemos reconstruir el camino desde y hasta x usando la función de padres p, y así obtener el ciclo completo cerrándolo con la arista (x, y).

Para reconstruir el ciclo, lo que queremos es encontrar el camino que va desde el vértice y hasta el vértice x siguiendo la función de padres p, y luego cerrar el ciclo regresando de x a y, dado que detectamos la arista dirigida (x, y) como posible cierre de un ciclo.

Para ello, inicializamos una lista vacía llamada $camino = [\,]$. Luego, nos posicionamos en x y recorremos hacia atrás usando la función de padres p, agregando cada vértice al principio de la lista. Continuamos mientras $x \neq y$, es decir, hasta que alcancemos el vértice y.

Una vez alcanzado y, lo agregamos también a la lista (si aún no lo habíamos hecho), y finalmente añadimos y una vez más al final para cerrar el ciclo y cerramos el ciclo añadiendo la arista (x, y) al final del camino reconstruido desde y hasta x, obteniendo el ciclo $(y, ..., x, y)$, el cual comienza y termina en el mismo vértice y, y fue generado

al encontrar una arista hacia un vértice previamente visitado que no era el padre en BFS, lo cual garantiza que es un ciclo.

Ahora, sabemos que es el mínimo porque BFS explora primero los caminos más cortos en términos de número de aristas desde un vértice distinguido r. Así, hasta ahora sabemos que el ciclo es mínimo respecto a r desde el cual empezamos BFS. Sin embargo, también podemos considerar que puede haber otro ciclo más corto en otra parte de la digráfica y que no involucre a r. Para estar seguros, entonces repetimos BFS desde cada vértice de V como punto de inicio, y en cada ejecución de BFS, guardamos el ciclo más corto detectado (si hay alguno), y al final nos quedamos con el más corto entre todos. Así garantizamos que encontramos el ciclo dirigido más corto en toda la gráfica.

Sobre su complejidad, La inicialización de estructuras cuesta $\mathcal{O}(1)$ por vértice, por lo que el costo total de inicialización es $\mathcal{O}(|V|)$. Luego, el procesamiento de vértices a través de la cola se hace una sola vez por cada vértice (cada vértice entra y sale de la cola una vez), lo cual también cuesta $\mathcal{O}(|V|)$. Para explorar los vecinos salientes, se procesa cada arista saliente una sola vez, lo que tiene un costo total de $\mathcal{O}(|E|)$. Así, tenemos que cuesta $\mathcal{O}(|V+E|)$. Para la reconstrucción del ciclo (cuando se encuentra) implica revisar quienes son los padres desde x hasta y, lo cual, en el peor caso, es hacerlo desde el último hasta la raíz r, lo que cuesta $|V|$ pasos. Como queremos encontrar el ciclo dirigido mínimo en toda la gráfica dirigida, hacemos BFS para todos los vértices y la comparación entre longitudes, lo cual se repite $|V|$ y $\mathcal{O}(1)$ veces, respectivamente. Por lo tanto, como se ejecuta BFS $\mathcal{O}(|V|+|E|)$ veces por cada vértice (osea $\mathcal{O}(|V|)$), su complejidad es $\mathcal{O}(|V| \cdot (|V|+|E|))$. ★