MA3705. Algoritmos Combinatoriales 2020.
Profesor: José Soto
Escriba(s): Tomás Banduc y Yeniffer Muñoz.
Fecha: 11 de septiembre de 2020.

Cátedra 4

Resumen de cátedra:

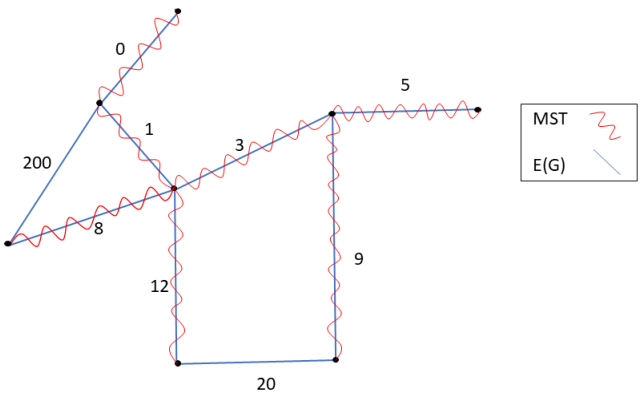
En la presente cátedra se estudia el problema del bosque generador de peso mínimo, y se prueban un teorema y corolario utilizados en la construcción de dos algoritmos que solucionan el problema de un MST (Prim y Borůvka). También, se introducen nociones y conceptos básicos asociados a la complejidad temporal de un algoritmo a partir del modelo RAM de computación. Finalmente, se hace justificación al uso de notación asintótica y se hace mención del criterio de acotamiento polinomial en la complejidad de un algoritmo para la determinación de su eficiencia.

1. Problema del árbol/bosque generador de peso mínimo (MST)

1.1. Bosque generador de costo mínimo:

Dado $G = (V, E)$ grafo, y $c : E \longrightarrow \mathbb{R}$ función de costo, se plantea el **Problema del bosque generador de costo mínimo**, que consiste en encontrar $F \subseteq E$ bosque generador de costo $c(F) = \sum_{e \in F} c(e)$ mínimo del grafo.

Si G es conexo, el problema se llama árbol generador/cobertor de costo mínimo, el cuál ya fue estudiado en la clase anterior (MST: *minimum spanning tree*).



.

Figura 1: Ejemplo ilustrado de bosque generador de costo mínimo.

1.2. Teorema de arista mínima de un corte:

Teorema 1. *Sean $G = (V, E)$ grafo conexo, $c : E \longrightarrow \mathbb{R}$ función de costo y $OPT \subseteq E$ un MST. Sea $\emptyset \subset U \subset V$. Si e es una arista de menor costo de $\delta(U)$, entonces e se puede introducir a OPT intercambiando una arista de $\delta(U) \cap OPT$; es decir:*

$$\exists f \in \delta(U) \cap OPT : OPT - f + e \text{ es un } MST$$

Demostración: Sea $e \in \delta(U)$ tal que $e := argmin\{c(f) : f \in \delta(U)\}$

Caso 1: Si $e \in OPT$, basta tomar $f := e$, y es directo que $f \in \delta(U) \cap OPT$. Luego, $OPT - e + e = OPT$, que es MST.

Caso 2: Si $e \notin OPT$. Sean $u \in U$ y $v \notin U$ tales que $e = uv$. Como OPT es árbol y $e \notin OPT$, entonces $OPT + e$ tiene un ciclo $C(OPT, e)$ (resultado visto en la cátedra anterior), que además es único (de no ser así, se contradice la no-ciclicidad de OPT).

Notemos que $(C(OPT, e) - e) \cap \delta(U) \neq \emptyset$, esto pues $C(OPT, e) - e$ es camino de u a v, y $v \notin U$, por lo que existe un primer momento (arista) en que $C(OPT, e) - e$ sale de U.

Con lo anterior, sea $f \in C(OPT, e) - e \cap \delta(U)$. Dado que $e \notin OPT$ y $f \in C(OPT, e) - e$, entonces $OPT + e - f$ es árbol generador (propiedad de intercambio).

Falta ver que $T := OPT + e - f$ es efectivamente MST:

Como $c(T) = c(OPT) + c(e) - c(f)$ y, recordando que $e = argmin\{c(f) : f \in \delta(U)\}$, se tiene que el costo de e es menor o igual a cualquier arista del corte de U, en particular $c(e) \leq c(f)$. Por lo tanto, $c(T) \leq c(OPT)$ y, en consecuencia, T es MST.

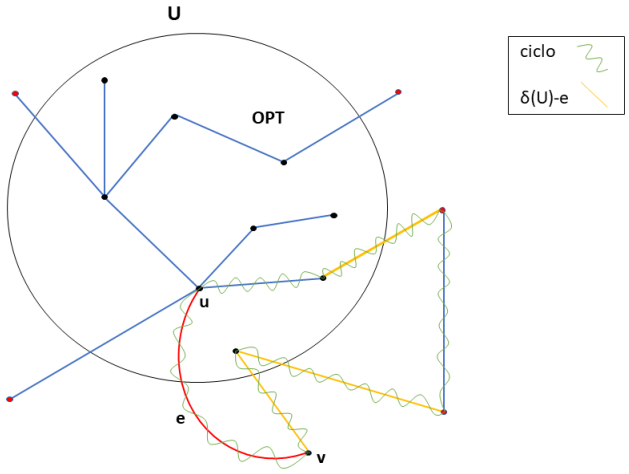


Figura 2: Ilustración de demostración Teorema 1.

Definición 1 (Óptimo parcial)**.** Sea $G = (V, E)$ grafo, se dice que $F \subseteq E$ es un óptimo parcial si existe MST óptimo OPT con $F \subseteq OPT$.

Corolario 1. *Sean $G = (V, E)$ grafo conexo y $c : E \longrightarrow \mathbb{R}$ función de costo. Sean $F \subseteq E$ óptimo parcial y $U \subseteq V$ tal que $\delta(U) \cap F = \emptyset$. Si e es arista de menor costo de $\delta(U)$, entonces $F + e$ es óptimo parcial.*

Demostración:

Sean $G = (V, E)$ grafo, $F \subseteq E$ óptimo parcial y $U \subseteq V$ tal que $\delta(U) \cap F = \emptyset$. Sea OPT el MST que satisface $F \subseteq OPT$. Sea $e \in \delta(U)$ la arista de menor costo de $\delta(U)$. Evaluemos por casos:

Caso 1: Si $e \in OPT$, es directo que $F + e \subseteq OPT$, y se tiene que $F + e$ es óptimo parcial.

Caso 2: Si $e \notin OPT$, por el *Teorema 1*, existe $f \in \delta(U) \cap OPT$ tal que $OPT + e - f$ es MST. Notemos que, como $F \subseteq OPT$ entonces $F + e \subseteq OPT + e$.

Además, $f \notin F$, pues el corte de U no intersecta a F. Luego, $F + e \subseteq OPT + e - f$ y se concluye que $F + e$ es óptimo parcial.

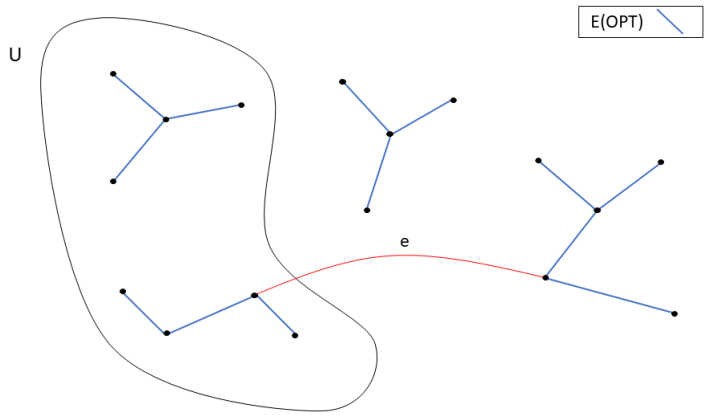


Figura 3: Ejemplo ilustrado de Caso 2, demostración de *Corolario 1.*

1.3. Algoritmos basados en los resultados anteriores:

1.3.1. Algoritmo de Prim (Prim 1957 - Jarník 1930):

Entrada: $G = (V, E)$ conexo , $r \in V$
Elegir $r \in V$;
U $\leftarrow$ {r}
F $\leftarrow$ $\emptyset$

Mientras $\delta(U) \neq \emptyset$ **hacer**:
Sea $e = uv \in \delta(U), u \in U, v \notin U$
tal que e es la arista de menor peso en $\delta(U)$;
U $\leftarrow$ U+v
F $\leftarrow$ F+e
Fin

Devolver $T = (U, F)$

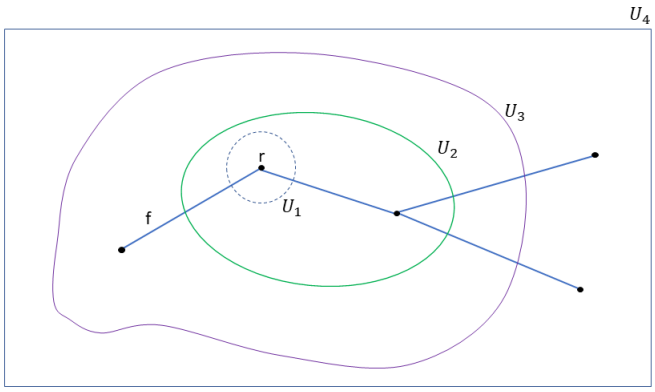


Figura 4: Ilustración ejemplificadora del Algoritmo de Prim, donde $U = U_i$ es el estado de U en la i-ésima iteración.

Correctitud: En virtud del corolario anterior, al encontrarse en cada instante un corte $\delta(U)$ que no intersecta a F, agregándose la arista de menor peso de $\delta(U)$ se genera un nuevo óptimo parcial $F + e$ para cada iteración.

Dado que V es finito y en cada paso se agrega un vértice de V no contenido en U, existe un instante k en que $U = V$, lo que implica $\delta(U) = \emptyset$, por lo que el algoritmo termina. Además, no se detiene antes de llegar al óptimo global pues, dada la conexidad de U, siempre que $U \subset V$, para $v \in V \backslash U$ existirá algún vértice $u \in U$ y una arista asociada $e = uv$ que saldrá de U, con lo que $\delta(U) \neq \emptyset$.

1.3.2. Algoritmo de Borůvka (Borůvka 1926):

Entrada: $G = (V, E)$ conexo
F $\leftarrow \emptyset$

Repetir
Calcular componentes conexas de (V, F)
Calcular para cada componente U la arista $e_U \in \delta(U)$ de menor peso*
Agregar todas las aristas e_U a F
Hasta que cc(V,F)=1

Devolver T=(V,E)

---

* Rompiendo empates de manera consistente
(siguiendo la misma regla siempre, por ejemplo, eligiendo la arista de menor nombre)

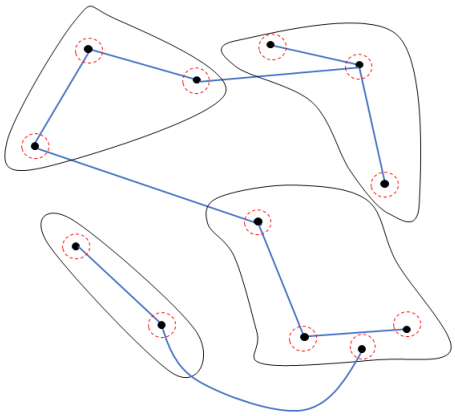


Figura 5: Ilustración del algoritmo de Borůvka.

Observación: El empleo de un criterio consistente para romper empates permite evitar la generación de ciclos en el árbol cobertor. La demostración de esta propiedad es no trivial.

Correctitud: El alcance de un óptimo parcial (y óptimo global con el fin del algoritmo), viene justificada nuevamente por el corolario anterior, y se asegura su término por la conexidad del grafo.

2. Complejidad temporal de un algoritmo

2.1. Modelo de computación:

Usaremos el modelo RAM (*random access memory*), que básicamente asume lo siguiente (muy simplificado):

Modelo RAM:

- **Conjunto de registros** de números naturales de exactamente w bits cada uno, indexados por números naturales.
- **Procesador** capaz de hacer operaciones básicas:
  - Leer/escribir/copiar el valor de un registro (a un valor fijo o al de otro registro).
  - Tomar dos registros a y b y escribir en un tercero $a+b, a-b, a \cdot b$ ó a/b (división entera), operaciones que se supondrá que demoran lo mismo.
  - Comparar dos registros mediante $>, <$ ó $=$.

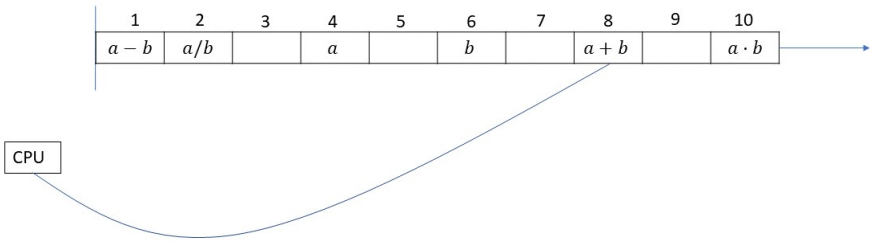


Figura 6: Ilustración de Modelo RAM.

Combinando las **operaciones básicas** mencionadas del modelo, se pueden crear **operaciones intermedias**, tales como:

- Estructuras de control (*if-else-then*; *while*; *for*; etc.)
- Operaciones sobre variables y arreglos (crear variables, acceder a vectores, etc.)
- Lectura/escritura en estructuras de datos simples (listas enlazadas, colas, pilas, etc.)

Suponemos que dichas operaciones intermedias toman un número acotado por una constante universal de operaciones básicas. Para este modelo, las operaciones intermedias tardan $O(1)$ operaciones básicas.

2.2. Formas de medir el tamaño de la entrada:

En optimización combinatorial, todo algoritmo toma como entrada una secuencia de números.

- La cantidad de números se denotará por N.
- El número total de bits se denotará por B.
- Además, si la entrada es un grafo G, usaremos $n := |V(G)|$, $m := |E(G)|$.

Normalmente estos valores están relacionados entre sí.

Proposición 1 (Codificación de un número)**.** *Un número $K \in \mathbb{N}$, en binario, se codifica usando $O(\log K)$ bits. Luego, si un algoritmo recibe N números, y el número más grande tiene largo L, entonces $B = O(N \log L)$.*

Observación: Cuando L es fijo, $O(N \log L) = O(N)$.

2.3. Formas de codificar un grafo:

2.3.1. Matriz de adyacencia:

Es una matriz $M \in \{0,1\}^{V \times V}$ con $M_{uv} = 1$, si $uv \in E$, y $M_{uv} = 0$, si $uv \notin E$.

Para un grafo de n vértices, $N = \Theta(n^2)$ y $B = \Theta(n^2)$.

Ventaja: El tiempo que tarda en determinar si una arista pertenece a un grafo corresponde con el costo de una operación básica $O(1)$.

Desventaja: A pesar de poder tenerse una cantidad pequeña de aristas en relación al número n de vértices, debe entregarse de igual forma n^2 datos para codificar en la matriz.

2.3.2. Lista de incidencia:

Es un arreglo de listas, una para cada vértice. Para un grafo $G = (V, E)$, la lista de $v \in V$ contiene un puntero (nombre) a cada vértice en $N(v)$ (vecinos de v).

Para un grafo de n vértices y m aristas, su lista de incidencia recibe un total de $n + 2m$ números. Además, para una numeración de 1 a n de los vértices del grafo, se tiene que el número de mayor tamaño tiene largo n. Luego, $N = \Theta(n+m)$ y $B = \Theta((n+m)\log n)$.

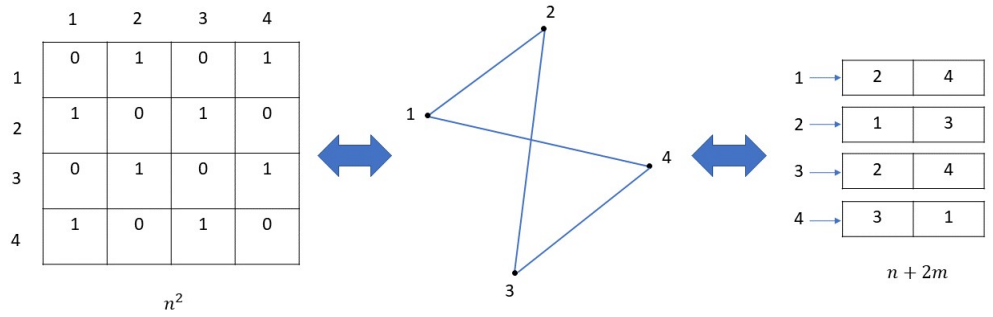


Figura 7: Ejemplo de un grafo codificado en una matriz de adyacencia y en una lista de incidencia.

2.4. Complejidad de un algoritmo:

Definición 2 (Tiempo del algoritmo)**.** Sea ALG un algoritmo. Se define el tiempo del algoritmo como el número de operaciones básicas que realiza ALG sobre su ENTRADA, antes de terminar. Se denota $Tiempo(ALG, ENTRADA)$.

Observación: Tal como indica su notación, el tiempo del algoritmo dependerá de la $ENTRADA$, empleando más (o menos) operaciones básicas según la estructura a evaluar.

Definición 3 (Función de complejidad)**.** Sea ALG un algoritmo. Se define su función de complejidad (peor-caso) como:

$$T(x) := \text{máx}\{Tiempo(ALG, ENTRADA) : \text{tamaño de ENTRADA=x}\}$$

Por ejemplo, la complejidad de un algoritmo podría ser $50B^2$, ó $20N^5 + 2^N$; y la complejidad de un algoritmo en grafos podría ser $6mn^2$.

En general, resulta complicado calcular con exactitud la complejidad de un algoritmo. Es por esto que se trabaja con notación asintótica. Así, para los ejemplos anteriores, las complejidades respectivas de los algoritmos se denotan $O(B^2)$, $O(2^N)$ y $O(mn^2)$, donde la información de interés para la complejidad de cada algoritmo corresponde con su acotamiento superior.

2.5. Notación asintótica y eficiencia:

Nos interesa el crecimiento de la complejidad a medida que el tamaño de la entrada aumenta, así que todo lo hacemos en notación asintótica.

Desde un punto de vista teórico, un algoritmo se considera eficiente cuando **su complejidad está acotada por un polinomio**.

¿Por qué polinomios?

El acotamiento por un polinomio determina un comportamiento deseable con respecto a la ejecución del algoritmo para un aumento en el tamaño de la entrada, amplificando el tiempo del algoritmo a lo sumo por una constante.

Por ejemplo, supongamos que se tiene un algoritmo de complejidad $T(N) = O(N^{10})$. Entonces, si se ejecuta con una entrada de tamaño $2N$, su complejidad viene dada por $T(2N) \sim 2^{10}N^{10} = O(1) \cdot O(N^{10})$.

Por otro lado, supongamos que se tiene un algoritmo de complejidad $T(N) = O(2^N)$. Luego, de duplicarse el tamaño de la entrada, el tiempo que tarda en correr pasa a ser del orden $T(2N) \sim (2^N)^2 = O(2^N)^2$, es decir, el algoritmo se demora en el peor de los casos el cuadrado de lo que tardaba antes.

2.6. Algoritmos polinomiales y fuertemente polinomiales:

Definición 4 (Algoritmo polinomial/débilmente polinomial)**.** Es uno de complejidad $O(B^k)$ para algún $k \in \mathbb{N}$ fijo; es decir, es polinomial en el número de bits de la entrada.

Definición 5 (Algoritmo fuertemente polinomial)**.** Es uno de complejidad $O(N^k)$ para algún $k \in \mathbb{N}$ fijo; es decir, es polinomial en el número de datos de la entrada.

Observación 1: Si un algoritmo es fuertemente polinomial, entonces es débilmente polinomial.

Observación 2: Si no nos importa el exponente, podemos escribir $B^{O(1)}$ y $N^{O(1)}$.