MA3705. Algoritmos Combinatoriales 2024
Profesor: José Soto
Auxiliar: Domingo Ruiz, Luciano Villarroel

Ingeniería Matemática
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS UNIVERSIDAD DE CHILE

Pauta Auxiliar 1

P1. Sea $G = (V, E)$ un grafo

(a) Demuestre que $\sum_{v \in V} d(v) = 2|E|$

Solución : Daremos 2 soluciones.
Idea: Cada arista aporta exactamente 2 a la suma, 1 por cada vértice de la arista. Utilizando esta idea, demostraremos (a) por inducción en $|E|$, es decir, probaremos que se tiene (a), para todo grafo con vértices V y cantidad de aristas $0 \leq |E| \leq \binom{n}{2}$.
Caso Base : $|E| = 0$. En este caso, $E = \emptyset$, por lo que $\forall v \in V$, $d(v) = |\{e \in E : v \in e\}| = 0$. Ahora $\sum_{v \in V} d(v) = 0 = 2|E|$.

Paso Inductivo : Sea $0 < m \leq \binom{n}{2}$, suponemos que se cumple (a) para $|E| = m - 1$ (HI) y probaremos que se cumple (a) para $|E| = m$.
Suponemos $|E| = m$. Sea $e = \{u, w\} \in E$, consideramos el grafo $G' = (V, E')$, con $E' = E \setminus \{e\}$. $|E'| = |E| - 1 = m - 1$, por (HI), $\sum_{v \in V} d_{G'}(v) = 2|E'| = 2(m-1) = 2m-2$.
Ahora, como $E' = E \setminus \{\{u, w\}\}$, $\forall v \in V \setminus \{u, w\}$, $d_G(v) = d_{G'}(v)$ y para $v \in \{u, w\}$, $d_G(v) - 1 = d_{G'}(v)$.
Ahora, $2m-2 = \sum_{v \in V} d_{G'}(v) = \sum_{v \in V \setminus \{u,w\}} d_{G'}(v) + d_{G'}(u) + d_{G'}(w) = \sum_{v \in V \setminus \{u,w\}} d_G(v) + d_G(u) - 1 + d_G(w) - 1 = \sum_{v \in V} d_G(v) - 2$.
Ahora, $\sum_{v \in V} d_G(v) = 2m = 2|E|$.
Por inducción concluimos (a).

Solución alternativa :
Primero notamos que para $v \in V$, $d(v) = |\{e \in E : v \in e\}| = \sum_{e \in E} \mathbb{1}_{v \in e}$. Donde $\mathbb{1}_{v \in e} = \begin{cases} 1 & \text{si } v \in e \\ 0 & \text{si } v \notin e \end{cases}$
Ahora $\sum_{v \in V} d(v) = \sum_{v \in V} \sum_{e \in E} \mathbb{1}_{v \in e} = \sum_{e \in E} \underbrace{\sum_{v \in V} \mathbb{1}_{v \in e}}_{=2} = \sum_{e \in E} 2 = 2|E|$

(b) Demuestre que todo vértice tiene grado par ssi todo corte tiene tamaño par

Solución : Primero recordaremos la definición de corte. $F \subseteq E$ es un corte si existe $W \subseteq V$ tal que $F = E(W, V \setminus W) = \{e \in E : e \cap W \neq \emptyset \wedge e \cap (V \setminus W) \neq \emptyset\}$. Es decir F son las aristas con un extremo en W y otro extremo en $V \setminus W$. Anotamos $F = \delta(W)$.
Continuamos con la demostración:
($\Longleftarrow$) Sea $v \in V$, notamos que $d(v) = |\{e \in E : v \in e\}| = |\delta(\{v\})|$, de lo que concluimos que $d(v)$ es par.
($\Longrightarrow$) Sea $W \subseteq V$. La idea de esta demostración es que $\sum_{w \in W} d(w) = 2|\{e \in E : e \subseteq W\}| + |\delta(W)|$, pues en la suma cada arista contenida en W aporta 2 y cada arista en el corte de W aporta 1. Para probar esto, utilizaremos la parte (a) y una demostración similar solución alternativa realizada en (a).
Notamos que $\sum_{w \in W} d(w) = \sum_{w \in W} |\{e \in E : w \in e\}| = \sum_{w \in W}(|\{e \in E : w \in e \wedge e \subseteq W\}| + |\{e \in E : w \in e \wedge e \not\subseteq W\}|) = \sum_{w \in W} |\{e \in E : w \in e \wedge e \subseteq W\}| + \sum_{w \in W} |\{e \in E : w \in e \wedge e \not\subseteq W\}|$
Analizaremos ambos términos por separado.
$\sum_{w \in W} |\{e \in E : w \in e \wedge e \not\subseteq W\}| = \sum_{w \in W} \sum_{e \in E} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W} = \sum_{w \in W}(\sum_{e \in \delta(W)} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W} + \sum_{e \in E \setminus \delta(W)} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W}) = \sum_{w \in W} \sum_{e \in \delta(W)} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W} + \sum_{w \in W} \sum_{e \in E \setminus \delta(W)} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W} = \sum_{e \in \delta(W)} \underbrace{\sum_{w \in W} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W}}_{=1} + \sum_{e \in E \setminus \delta(W)} \underbrace{\sum_{w \in W} \mathbb{1}_{w \in e} \cdot \mathbb{1}_{e \not\subseteq W}}_{=0}$
$= \sum_{e \in \delta(W)} 1 + \sum_{e \in E \setminus \delta(W)} 0 = |\delta(W)|$.

Veamos el otro término. Definiendo $G' = (V', E')$, con $V' = W$ y $E' = \{e \in E : e \subseteq W\}$, tenemos que para $w \in W$, $|\{e \in E : w \in e \wedge e \subseteq W\}| = d_{G'}(w)$, por lo que utilizando la parte (a), tenemos que $\sum_{w \in W} |\{e \in E : w \in e \wedge e \subseteq W\}| = \sum_{w \in V'} d_{G'}(w) = 2|E'|$.
Juntando esto con lo anterior, tenemos que $\sum_{w \in W} d(w) = 2|E'| + |\delta(W)|$. Como todo vértice de G tiene grado par, tenemos que $\sum_{w \in W} d(w)$ es par, de lo que concluimos que $|\delta(W)|$ es par, pues $|\delta(W)| = \sum_{w \in W} d(w) - 2|E'|$.

(c) **[Propuesto]** Demuestre que la cantidad de vértices de grado impar de G es par

(d) **[Propuesto]** Demuestre que si G no tiene ciclos y tiene al menos una arista, entonces tiene al menos 2 vértices de grado 1

P2. Sea $D = (V, A)$ un digrafo. Para $v \in V$, definimos $d^+(v) = |\{a = (u, w) \in A \,|\, u = v\}|$, $d^-(v) = |\{a = (u, w) \in A \,|\, w = v\}|$. Para $W \subseteq V$, definimos $d^+(W) = |\{a = (u, v) \in A \,|\, u \in W, v \notin W\}|$ y $d^-(W) = |\{a = (u, v) \in A \,|\, u \notin W, v \in W\}|$.

(a) Demuestre que $\sum_{v \in V} d^+(v) = \sum_{v \in V} d^-(v) = |A|$

(b) Demuestre que si $d^+(v) = d^-(v) \, \forall v \in V$, entonces $d^+(W) = d^-(W) \, \forall W \subseteq V$

Solución : Propuesto (muy similar a la P1).

P3. Sea $G = (V, E)$ un grafo. Demuestre que si $\forall v \in V$, $d(v) \geq k \geq 2$, entonces G tiene un ciclo de largo al menos $k + 1$

Solución : Sea $P = x_0 x_1 ... x_n$ el camino de largo ($|E(P)|$) máximo en G, el cual existe porque la cantidad de caminos es finita. Notamos que $\{v \in V \,|\, x_n v \in E\} \subseteq V(P)$, pues sino, P no sería de largo máximo. Ahora, definimos $i = min\{j \in \mathbb{N} \,|\, x_j x_n \in E\}$. Como $d(x_n) \geq k \geq 2$, $i < n - 1$. Ahora, $C = x_i x_{i+1} ... x_n x_i$ define un ciclo de largo al menos $k + 1$, pues $\{v \in V \,|\, x_n v \in E\} \subseteq V(C)$, y $|\{v \in V \,|\, x_n v \in E\}| = d(x_n) \geq k$. De esto tenemos que $|V(C)| \geq k + 1$, de lo que concluimos que el ciclo C tiene largo al menos $k + 1$

P4. Sea $G = (V, E)$ un grafo conexo. Demuestre que G tiene un paseo euleriano ssi $\forall v \in V \, d(v)$ es par.

Primero definiremos paseo euleriano. Un paseo euleriano es un paseo $v_0 e_1 v_1 ... e_k v_k$ que pasa por cada arista de E exactamente una vez y que cumple $v_0 = v_k$.
Solución : ($\Longrightarrow$) Sea $v \in V$. Si recorremos el paseo euleriano de G partiendo desde un vértice distinto de v, cada vez que el paseo pasa por v ocupa dos aristas incidentes en v. Sea k la cantidad de veces que el paseo pasa por v. Como el paseo pasa por cada arista exactamente una vez, $d(v) = 2k$.
($\Longleftarrow$) Sea $W = v_0 e_1 v_1 e_2 ... e_k v_k$ un paseo en G que no repite aristas maximal (con la mayor cantidad de aristas). Notamos que $v_0 = v_k$, pues si no, $|E(W) \cap E(v_k)|$ sería impar y por hipótesis ($d(v_k)$ par), hay una arista incidente en v_k que no está en W, lo que contradice la maximalidad de W, pues podríamos extender el paseo. Ahora veremos que W pasa por todas las aristas por contradicción. Suponemos que $\exists e = uv \notin E(W)$.
Si $\{u, v\} \cap V(W) \neq \emptyset$, SPG $u \in V(W)$, por lo que $u = v_i$, con $0 \leq i \leq k$. Ahora el paseo $v_i e_{i+1} ... e_k v_k e_1 v_1 ... e_i v_i e v$ no repite aristas y contiene una arista más que W, lo que es una contradicción.
Si $u, v \notin V(W)$, por conexidad $\exists e' = u'v'$ tal que $u' \in V(W)$ y $v' \notin V(W)$. Ahora, $u' = v_i$ con $0 \leq i \leq k$ y el paseo $v_i e_{i+1} ... e_k v_k e_1 v_1 ... e_i v_i e' v'$ no repite aristas y contiene una arista más que W, lo que es una contradicción.
De esto concluimos que $E(W) = E$ y como W no repite aristas, concluimos que es un paseo euleriano.

P5. Analice la correctitud y complejidad del algoritmo de búsqueda binaria

Algorithm 1: Búsqueda Binaria

Input: Un arreglo ordenado $A[1 \dots n]$ y un valor x
Output: El índice del valor x en el arreglo A o -1 si x no está en el arreglo

```
1  BinarySearch(A, x):
2      L ← 1;
3      R ← n;
4      while L ≤ R do
5          m ← ⌊(L+R)/2⌋;
6          if A[m] = x then
7              return m;
8          else
9              if A[m] < x then
10                 L ← m + 1;
11             else
12                 R ← m − 1;
13             end
14         end
15     end
16     return −1;
```

Solución :
Primero probaremos que el algoritmo termina. Sea $T_k = R - L + 1$, donde los valores de R y L corresponden al comienzo de la k-ésima iteración del *while* [1].
Probaremos por inducción que:

$$T_k \leq \frac{n}{2^k} \qquad \text{al comienzo de toda iteración } k \text{ antes de que el algoritmo termine} \qquad (1)$$

Caso Base : $k = 0$. $T_0 = n - 1 + 1 = n \leq \frac{n}{2^0}$

Paso Inductivo : Suponemos que $T_k \leq \frac{n}{2^k}$ al comienzo de la iteración k (HI).
En esta iteración, en el algoritmo se calcula $m = \lfloor \frac{L+R}{2} \rfloor$, y hay 3 casos posibles.
Caso 1: $A[m] = x$. En este caso el algoritmo termina, por lo que no hay nada que probar.

Caso 2: $A[m] < x$. En este caso, el valor de L se actualiza por $m + 1$, y R se mantiene igual, por lo que $T_{k+1} = R - (m + 1) + 1 = R - m$. Ahora, $T_{k+1} = R - m = R - \lfloor \frac{L+R}{2} \rfloor = R + \lceil \frac{-L-R}{2} \rceil = \lceil R + \frac{-L-R}{2} \rceil = \lceil \frac{R-L}{2} \rceil = \lfloor \frac{R-L+1}{2} \rfloor = \lfloor \frac{T_k}{2} \rfloor \underbrace{\leq}_{HI} \lfloor \frac{n}{2^k} \cdot \frac{1}{2} \rfloor = \lfloor \frac{n}{2^{k+1}} \rfloor \leq \frac{n}{2^{k+1}}$.
Donde se utilizó que para un real x, y un entero m se tienen las siguientes propiedades:

1. $-\lfloor x \rfloor = \lceil -x \rceil$
2. $m + \lceil x \rceil = \lceil m + x \rceil$
3. $\lceil \frac{m}{2} \rceil = \lfloor \frac{m+1}{2} \rfloor$

Por completitud, probaremos esta última. Sea $m \in \mathbb{Z}$, si m es par, $m = 2k$, por lo que $\lceil \frac{m}{2} \rceil = \lceil k \rceil = k = k + \lfloor \frac{1}{2} \rfloor = \lfloor k + \frac{1}{2} \rfloor = \lfloor \frac{m+1}{2} \rfloor$. Si m es impar, $m = 2k + 1$, por lo que $\lceil \frac{m}{2} \rceil = \lceil k + \frac{1}{2} \rceil = k + 1 = \lfloor k + 1 \rfloor = \lfloor \frac{2(k+1)}{2} \rfloor = \lfloor \frac{m+1}{2} \rfloor$.

Caso 3: $x < A[m]$. En este caso, el valor de R se actualiza por $m - 1$, y L se mantiene igual, por lo que $T_{k+1} = m - 1 - L + 1 = m - L = \lfloor \frac{R+L}{2} \rfloor - L = \lfloor \frac{R+L}{2} - L \rfloor = \lfloor \frac{R-L}{2} \rfloor \leq \lfloor \frac{R-L+1}{2} \rfloor = \lfloor \frac{T_k}{2} \rfloor \underbrace{\leq}_{HI} \lfloor \frac{n}{2^k} \cdot \frac{1}{2} \rfloor = \lfloor \frac{n}{2^{k+1}} \rfloor \leq \frac{n}{2^{k+1}}$
Por inducción se concluye (1)

Ahora probaremos por contradicción que el algoritmo termina en a lo más $log_2(n) + 1$ iteraciones del *while*. Suponemos que el algoritmo no ha terminado luego de $log_2(n) + 1$ iteraciones del *while*. Esto implica que luego de $log_2(n) + 1$ iteraciones del *while*, $L \leq R$. Ahora, por (1), al comenzar la iteración $log_2(n) + 2$, se tiene que $R - L + 1 = T_{log_2(n)+1} \leq \frac{n}{2^{log_2(n)+1}} = \frac{n}{2n} = \frac{1}{2} < 1$ (recordando que contamos las iteraciones desde el 0). Esto implica que $R < L$ lo que es una contradicción.

[1] T_k representa el largo de la sección del arreglo en la que se "busca" a x, al comienzo de la iteración k. La idea es probar que en cada iteración del *while*, este largo se reduce al menos a la mitad.

Correctitud:

Ya sabiendo que el algoritmo termina, probaremos la correctitud. Para ello hay que ver 2 cosas; si $x \notin A$, el algoritmo retorna -1, y si $x \in A$, el algoritmo retorna m tal que $A[m] = x$.

Si $x \notin A$: En este caso, es imposible que el algoritmo termine en la linea 7, y como sabemos que el algoritmo termina, debe terminar retornando -1.

Si $x \in A$: Sea $i_x \in \mathbb{N}$ tal que $A[i_x] = x$. Probaremos que si $L \leq i_x \leq R$, al comienzo de una iteración, entonces el algoritmo termina entregando m tal que $A[m] = x$, o el algoritmo no termina y al comienzo de la siguiente iteración, se mantiene que $L \leq i_x \leq R$ $(\star)$.

Suponemos que $L \leq i_x \leq R$ $(*)$ al comienzo de una iteración.
Caso 1: $A[m] = x$. En este caso el algoritmo termina entregando m tal que $A[m] = x$.

Caso 2: $A[m] < x$. Como el arreglo está ordenado y $x \in A$, se tiene que $m+1 \leq i_x \underbrace{\leq}_{(*)} R$. En este caso, el valor de L se actualiza a $m + 1$, y R se mantiene igual, por lo que al comenzar la siguiente iteración se tiene que $L \leq i_x \leq R$. En particular, el algoritmo no terminó, pues $L \leq R$ al final de la iteración.

Caso 2: $x < A[m]$. Como el arreglo está ordenado y $x \in A$, se tiene que $L \underbrace{\leq}_{(*)} i_x \leq m-1$. En este caso, el valor de R se actualiza a $m - 1$, y L se mantiene igual, por lo que al comenzar la siguiente iteración se tiene que $L \leq i_x \leq R$. En particular, el algoritmo no terminó, pues $L \leq R$ al final de la iteración.
Con esto, ya probamos $(\star)$.
Ahora, como $x \in A$, al comienzo de la primera iteración, se tiene que $L = 1 \leq i_x \leq n = R$. Por $(\star)$, la única forma de que el algoritmo termine, es que retorne m tal que $A[m] = x$. Como ya probamos que el algoritmo termina, concluimos que termina de manera correcta.

Complejidad: Ahora, analizaremos la complejidad del algoritmo. Ya demostramos que el algoritmo termina en a lo más $log_2(n)+1$ iteraciones del *while*. Esto implica que el algoritmo realiza a lo más $O(log_2(n))$ operaciones básicas. Esto pues, antes del *while* se realizan 2 operaciones básicas y en cada iteración del *while* se realizan una cantidad constante de operaciones básicas.
Con esto concluimos que el algoritmo tiene complejidad $O(log(n))$