1er Parcial AED3: Parte Domiciliaria

Manuel Panichelli
L.U. 72/18

October 12, 2020

Preámbulo

Teoremas, lemas y proposiciones citadas en los ejercicios. La numeración no necesariamente coincide con sus fuentes.

Grafos

Prop. 1. El grafo completo con n vértices, K_n, tiene cantidad de aristas

$$m_{K_n} = \frac{n(n-1)}{2}.$$

Lema 1. La unión de dos caminos simples distintos entre dos vértices contiene un circuito simple.

Arboles

Teorema 1 (Definiciones equivalentes de árbol)**.** Dado $G = (V, X)$ un grafo, las siguientes son equivalentes:

1. G es un árbol, un grafo conexo sin circuitos simples.
2. G es un grafo sin circuitos simples, y e arista tq $e \notin X$, el grafo $G + e$ tiene exactamente un circuito simple el cual contiene a e.
3. G es conexo, pero si se saca cualquier arista queda un grafo no conexo. Es decir, toda arista es puente.
4. G es un grafo sin circuitos simples y $m = n - 1$.
5. G es conexo y $m = n - 1$.

Lema 2. Sea $G = (V, X)$ conexo y $e \in X$.

$$G - e \text{ es conexo} \iff \text{e pertenece a un circuito simple de G.}$$

Teorema 2. Sean $T = (V, X_T)$ un AG de $G = (V, X)$, $e \in X \setminus X_T$. Luego $T + e - f$ con f una arista del único circuito de $G + e$ es AG de G.

Teorema 3. Todo arbol conexo tiene al menos un AG.

Resolución

Ejercicio 1

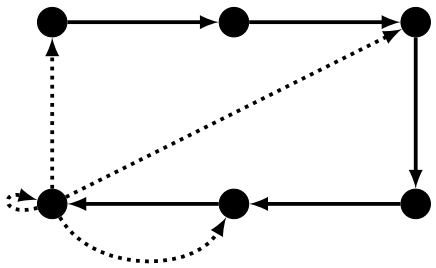
Digrafos con forma de ρ

Un digrafo con loops tiene ***forma de ρ*** *cuando todos sus vértices tienen grado de salida igual a 1.*

Antes de comenzar, observo que como el grado de salida de **todos** los vértices es 1, hay exactamente n arcos ($m = n$)

a) *Dibujar 4 digrafos conexos y no isomorfos entre sí que tengan 6 vértices y forma de ρ*

Para ello, voy a aprovechar que tienen $m = n$ arcos, y como $n-1$ es lo mínimo para que sea conexo, basta con ir cambiando el arco redundante para formar diferentes grafos.



Ejemplos de digrafos conexos con forma de ρ. Cada arco punteado representa un grafo diferente.

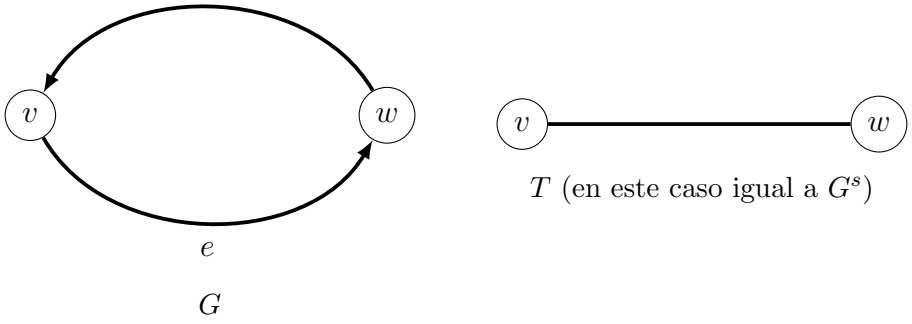
b) *Demostrar que si un digrafo es conexo (cuando su grafo subyacente lo es) y tiene forma de ρ entonces tiene un único ciclo.*

Dem. Sea G un digrafo conexo con forma de ρ. Su grafo subyacente también será conexo, y por Teo. 3 tendrá un AG $\boldsymbol{T}$, que por Teo. 1.5 tiene $m_T = n - 1$.

Como G tiene $m = n$ arcos, sean

- $\boldsymbol{G'}$ el sub-digrafo que resulta de convertir devuelta los ejes de T a los arcos de G.
- $\boldsymbol{e = (v, w)}$ el que está demás ($e \notin T, G'$)

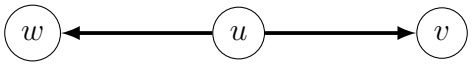
Ya que T es un árbol, no puede tener ciclos, y G' tendrá solo si se juntaron dos arcos de ida y vuelta en uno. Pero en ese caso, G tendría un ciclo, así que voy a suponer que no sucede y que G' no tiene ciclos.



Ejemplo de aristas que se combinan

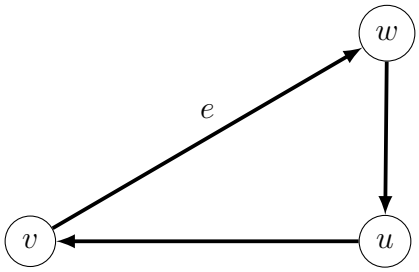
Como G' no tiene ciclos y si G los tiene, todos los ciclos tienen que contener a e. Hay dos casos,

- Si e resulta ser un *loop*, $e = (v, v)$, entonces ese es el ciclo.
- Si no, como $e = (v, w)$, e sale de v hacia w, y como el grafo es conexo tiene que haber arcos que unan w con v, para que en el subyacente haya camino y sea conexo. Además, el camino tiene que estar orientado de w a v, ya que si no lo estuviera, los nodos que lo conforman no tendrían grado de salida 1.



Ejemplo de digrafo conexo pero que no tiene camino orientado de w a v, $d_{OUT}(u) = 2$

Por lo tanto, se que hay un camino orientado P que va de w a v. Y con él puedo armar el ciclo $C : (v, w) + P_{wv}$.



Ejemplo de ciclo en un digrafo con forma de ρ

Ya se que tiene al menos un ciclo, y este contiene a e. Quiero ver que es único.

Supongo que no lo es, y que existen dos:

$$C_1 : e + P^1_{vw}$$
$$C_2 : e + P^2_{vw}$$

Tengo dos caminos distintos de v a w que al no contener a e, están también en T. En consecuencia su unión también: $P^1_{vw} \cup P^2_{vw} \in T$. Y por Lema. 1, esa unión contiene un circuito simple, que también está en T. Pero como T es un árbol, no tiene ciclos. Abs! Entonces el camino es único.

Concluyo que G tiene un único ciclo. □

c) *Diseñar un algoritmo para encontrar un ciclo de longitud máxima en un digrafo con forma de ρ (no necesariamente conexo)*

Para esto, hago un algoritmo iterativo que se fije partiendo de cada nodo si hay un ciclo, y se quede con el de longitud máxima. Supongo que cuento con las funciones

- $\boldsymbol{next(v)}$ que dado un vértice v, devuelve el único w tal que $(v, w) \in X$, el cual tiene que existir ya que G tiene forma de ρ.
- Funciones sobre secuencias a lo haskell, cuyas equivalencias en python serian

$$\boldsymbol{last(s)} = s[-1]$$
$$\boldsymbol{head(s)} = s[0]$$
$$\boldsymbol{init(s)} = s[1:]$$
$$\boldsymbol{length(s)} = len(s)$$
$$\boldsymbol{s + w} = \text{Concatenación}$$

Algorithm 1 Algoritmo para encontrar el ciclo con longitud maxima en un digrafo con forma de ρ. $O(n \times m)$

```
1:  function CicloMax(G)
2:      In: G = (V, X), un digrafo con forma de ρ
3:      Out: maxC un ciclo de longitud máxima.
4:      maxC ← ⟨⟩
5:      for v ∈ V do
6:          C ← ⟨⟩
7:          do
8:              C ← C + ⟨next(last(C))⟩          ▷ Extiendo el camino
9:          while last(C) ∉ init(C)              ▷ Mientras no se repita
10:         if last(C) ≠ first(C) then           ▷ Tenemos un subciclo?
11:             continue                         ▷ Se vera en otra iteración
12:         maxC ← argmax_{c∈{C,maxC}}(length(c))
13:     return maxC
```

Para cada vértice, se vé si hay un ciclo partiendo de él extendiendo el camino por el único siguiente. En caso de haberlo, se compara su longitud con el máximo. Hay

un chequeo extra para garantizar que no se cuenten erróneamente la longitud de posibles subciclos, ejemplificado en la figura 3. Se ve como al comenzar desde v, de no tener ese chequeo, contabilizaríamos a $v\ x\ w\ u\ x$ como un ciclo de longitud errónea.

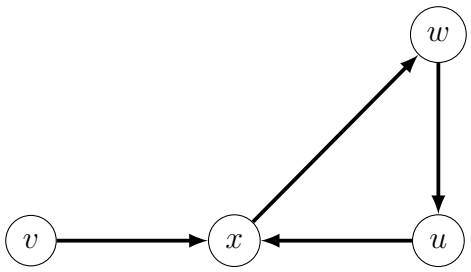


Figure 2: Ejemplo de subciclo en un digrafo con forma de ρ

La complejidad del algoritmo es $O(n \times m)$, ya que el do while interno se ejecuta a lo sumo m veces en total, y el for n veces. Pero como en los digrafos con forma de ρ $n = m$, es $O(n^2)$.

Selección de actividades

Período de tiempo circular $[0, T]$*, n actividades tal que la i-ésima comienza en el instante* s_i *y termina en* t_i*. Si* $s_i > t_i$*, la actividad contiene el instante* $0 = T$*. El objetivo es determinar la máxima razón* x/y *para una secuencia circular de actividades* $A_1, \ldots, A_x$ *que se realiza en y períodos completos cuando* A_{i+1} *se inicia lo antes posible una vez terminado* A_i*. Estrategia* ***golosa*** *para elegir actividad j si se eligió i: Tomarla como una actividad que no se solapa con i y cuyo tiempo de finalización es el primero desde* t_i*, en un recorrido del tiempo en el sentido de las agujas del reloj. Definir el digrafo de actividades D que tiene un vert i por cada actividad y tiene un arco* $i \to j$ *cuando j es la elección golosa que se toma si se elige i.*

Veamos los ejemplos del enunciado como digrafos y luego lo definimos formalmente

$$A_0 = [0, 6] \qquad A_4 = [11, 15]$$
$$A_1 = [4, 10] \qquad A_5 = [13, 18]$$
$$A_2 = [7, 12] \qquad A_6 = [16, 23]$$
$$A_3 = [8, 10] \qquad A_7 = [19, 2]$$

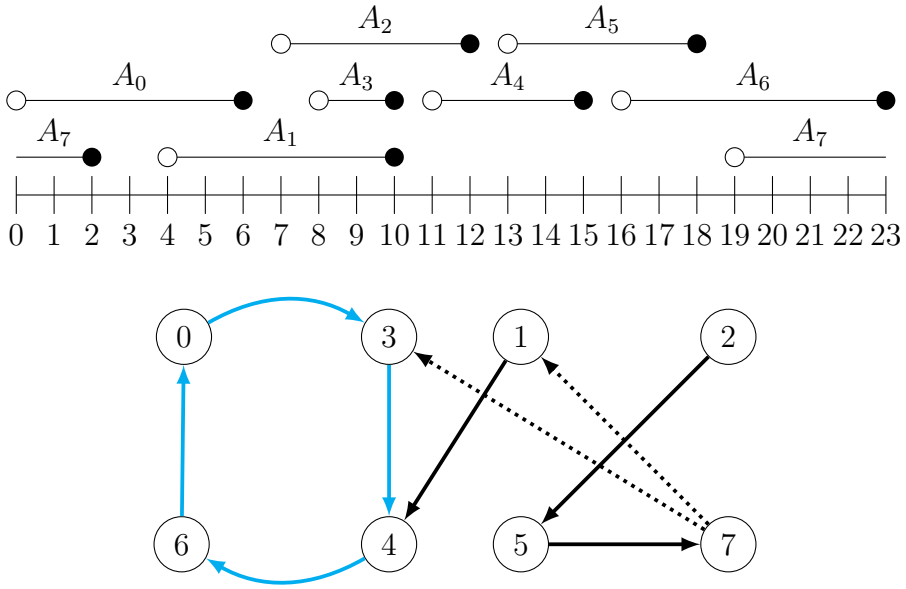


Figure 3: Ejemplo de un digrafo de actividades D, en cyan el ciclo máximo.

Definimos $D = (V, X)$ como

$$V = \{1, \ldots, x\}$$
$$X = \{(i, j) \mid i \in V, \text{noSeSolapan}(i, j) \wedge \text{terminaAntes}(i, j, k)\ \forall k\}$$

Donde

- noSeSolapan(i, j) es verdadero sii las actividades A_i y A_j no se solapan.
- terminaAntes(i, j, k) es verdadero si la actividad A_j tiene un tiempo de finalización anterior a A_k a partir del de A_i, en un recorrido del tiempo en el sentido de las agujas del reloj. En caso de empate, prefiere la de menor número.

d) Intuitivamente, como es un arbol de elecciones de un algoritmo goloso, tiene sentido que tenga forma de ρ ya que desde cada nodo se hará una sola elección que nunca se revise. Viendo la definición formal, efectivamente para cada vertice (actividad) elegimos una sola: la que termine primero. Por lo tanto, para cada i $\exists!\ j : (i, j) \in X$. Entonces el grado de salida de todo vertice i es 1, y el grafo D tiene forma de ρ.

e) *Demostrar que el ciclo máximo de D es una solución al problema de selección de actividades.*

opcional, no lo hago

f) *Dar un algoritmo para resolver el problema de selección de actividades, suponiendo que se describen usando un conjunto de pares $[s_i, t_i]$ con $0 \leq s_i, t_i \leq T, s_i \neq t_i$ y $T = 2n$*

Describo los pasos que tomaría el algoritmo en lenguaje coloquial. Este debe, dada una lista de actividades, devolver la máxima razón x/y para una secuencia circular de actividades $A_1, \ldots A_x$ que se realiza en y períodos completos.

1. Armo el digrafo D con las tareas como vértices, y para cada una aplico el algoritmo goloso para conseguir a cual hacerlo adyacente. La complejidad del algoritmo goloso es $O(n^2)$, pues para cada actividad se ven todas las demás para ver cual tiene el tiempo de finalización mas corto. Entonces la complejidad total de este paso es $O(n^3)$.
2. Como el ciclo máximo de D es la secuencia circular, y D es un digrafo con forma de ρ, puedo usar el algoritmo 1 para encontrarlo. Lo llamo C, y luego $x = length(C)$. Este paso se ejecuta en $O(n^2)$
3. Falta encontrar y, que es la cantidad de períodos que toma realizar las actividades del ciclo C. Para ello, recorro el ciclo, y si el inicio de la actividad i es anterior a la finalización de la $i-1$, incremento la cantidad de períodos. Como se recorre el ciclo que tiene a lo sumo $m = n$ aristas, este paso se ejecuta en $O(n^2)$.
4. Devuelvo la máxima razón: x/y.

Finalmente, la complejidad temporal del algoritmo será $O(n^3)$.

Ejercicio 2

Hilera con n recipientes con $r_1, \ldots, r_n \in \mathbb{N}$ unidades de agua, se quiere traspasar a dos baldes con capacidades $B_1, B_2 \in \mathbb{N}$. En cada paso se toma el primer recipiente de la hilera y se vuelva completamente en uno de los baldes. Se repite mientras algun balde tenga capacidad suficiente para albergar ***toda*** *el agua del recipiente. El objetivo es maximizar la cantidad de recipientes que se vuelcan.*

a) Formulo de forma recursiva una solución del problema con una función matemática $f(i, b)$ la cual tendrá la siguiente semántica: $f(i, b)$ es la máxima cantidad de recipientes que se pueden volcar suponiendo que ya se volcaron $r_1, ...r_{i-1}$ y que el balde con capacidad B_1 tiene b de capacidad restante. Observo que esta información es suficiente para inferir la capacidad restante del balde B_2, dada por

$$b_2 = B_2 - \overbrace{\underbrace{\sum_{j=1}^{i-1} r_j}_{\substack{\text{total} \\ \text{volcado}}} - \underbrace{(B_1 - b).}_{\text{volcado en } B_1}}^{\text{volcado en } B_2}$$

Veamos la función recursiva:

$$f(i,b) = \begin{cases} 0 & \text{si } (r_i > b \text{ y } r_i > b_2) \text{ o } i = n+1 \\ f(i+1,\ b) + 1 & \text{si } r_i > b \\ f(i+1,\ b - r_i) + 1 & \text{si } r_i > b_2 \\ max\{f(i+1, b),\ f(i+1,\ b - r_i)\} + 1 & \text{si no} \end{cases}$$

Explicada en palabras,

- Si ($r_i > b$ y $r_i > b_2$), ninguno de los dos baldes tiene capacidad suficiente para albergar el contenido del recipiente i-ésimo. Como no se pueden saltear, terminamos.
- Si $i = n + 1$, ya consideramos todos los recipientes.
- Si no entra en B_1 continuamos por B_2, y viceversa.
- Si entra en ambos, tomamos ambos caminos y devolvemos el mejor.

b) Es fácil ver que tiene la propiedad de *superposición de subproblemas*, basta con que alguna cantidad de recipientes sumen lo mismo, y así volcarlos en cada balde sería indistinto y llevaría al mismo resultado. Por ejemplo, cuando más de un recipiente tiene la misma cantidad de agua. Sean $r_1 = r_2 = 5, r_3 = 10$ y $B_1 = B_2 = 20$.

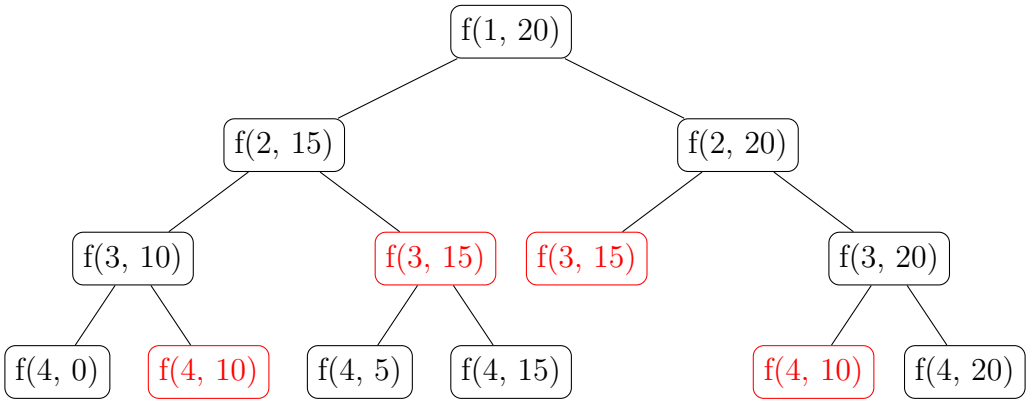


Figure 4: Ejemplo de superposoción de subproblemas. En rojo las llamadas que se superponen.

c) A continuación un algoritmo de programación dinámica *top-down* que implementa la función recursiva de forma bastante directa, agregando memoización mediante una matriz M de $n \times B_1$, cuyo valor por defecto es -1. Suponiendo que B_1, B_2, los r_i y M son globales,

Algorithm 2 Implementación con programación dinámica.

```
1:  function BALDES(i, b)
2:      b_2 ← B_2 − Σ_{j=1}^{i−1} r_j − (B_1 − b)                ▷ O(n)
3:      if i = n + 1 o (r_i > b y r_i > b_2) then
4:          return 0
5:      if M[i][b] = −1 then                                     ▷ No está memoizado
6:          if r_i > b then
7:              M[i][b] ← Baldes(i + 1, b) + 1
8:          else if r_i > b_2 then
9:              M[i][b] ← Baldes(i + 1, b − r_i) + 1
10:         else
11:             M[i][b] ← max{Baldes(i + 1, b), Baldes(i + 1, b − r_i)} + 1
12:     return M[i][b]
```

El problema se resuelve con el llamado $Baldes(0,\ B_1)$

d) La complejidad temporal del algoritmo *top-down* es igual a cualquier otro algoritmo de PD, (#subproblemas × costo de cada subproblema). En este caso, la cantidad de subproblemas es $n \times B_1$, y cada uno se resuelve en $O(n)$ (cálculo de b_2 y el resto operaciones en tiempo constante). Por lo tanto la complejidad es $O((n \times B_1) \times n) = O(n^2 \times B_1)$. Noto que esto podria mejorarse a $O(n \times B_1)$ evitando calcular b_2 en cada llamado, incrementandolo de a 1 en cada llamado.

Ejercicio 3

a) La imagen muestra un ejemplo de por qué las *social bubbles* no funcionan. Uno cree que está en una burbuja pequeña, con solo la gente con la que se ve, pero en realidad la burbuja se extiende con la burbuja de cada individuo. De esa forma, termina siendo mucho más grande de lo esperado.

Visto como grafos, cada persona sería un vértice, y dos personas son adyacentes entre sí si tuvieron contactos estrechos. Las burbujas sociales pasarían a ser componentes conexas, y uno cree que está en una componente conexa chica pero en realidad se extiende por los vecinos de cada nodo, y termina siendo un grafo conexo. Cada contagio que se produzca en cualquier lado termina llegando a su burbuja.

b) *Probar que un grafo de n vértices que tiene más de* $((n-1)(n-2))/2$ *aristas es conexo.*

qvq si $m > \frac{(n-1)(n-2)}{2}$ entonces G cualquiera es conexo. Lo voy a probar haciendo uso del siguiente lema.

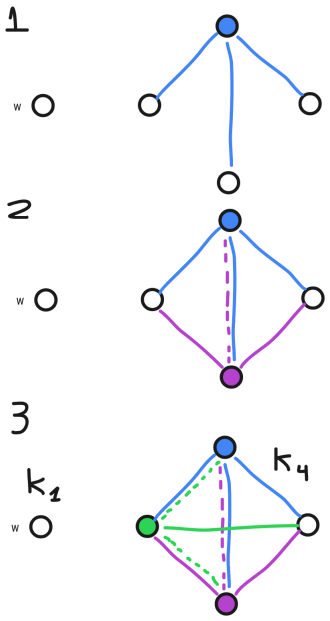
Lema 3. Sea G un grafo cualquiera,

$$m = \frac{(n-1)(n-2)}{2} \implies G \text{ es conexo o es } K_1 + K_{n-1}$$

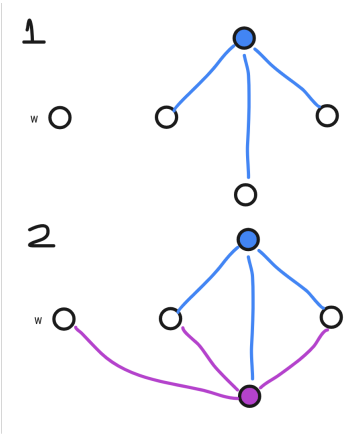
Dem (Lema 3). Veamos con más detalle cómo se puede interpretar la fórmula

$$m = \frac{\overbrace{(n-1)}^{\substack{\text{Todos los nodos} \\ \text{menos 1}}} \times \overbrace{(n-2)}^{\substack{\text{Conectan con todo} \\ \text{el resto menos 1}}}}{\underbrace{2}_{\substack{\text{Porque son pares} \\ \text{no ordenados}}}}.$$

Por lo tanto, puedo construir iterativamente a cualquier grafo que tenga esta cantidad de ejes conectandolo a $(n-1)$ nodos con todos menos 1. Por ejemplo,



(a) Ejemplo de construcción que lleva a $K_1 + K_{n-1}$



(b) Ejemplo de construcción que lleva a un grafo conexo

Figure 5: Algunos grafos posibles para $n = 5, m = 6$

Y para grafos generales, para cada nodo $v_i \in V \setminus \{w\}$ con $w \in V$ cualquiera,

- Para el primero, lo conecto con todos los nodos excepto uno, w. De esta manera quedan definidas dos componentes conexas, w y el resto de los nodos.
- Para el segundo, tengo dos opciones. Puedo conectarlo con w y todos menos 1 de $V \setminus \{w\}$, con lo cuál quedaría conexo, ya que ellos ya estaban conectados a través de v_1, o puedo conectarlo con todos los nodos de $V \setminus \{w\}$ y así seguir dejando a w aislado.
- Sigo así hasta el último de $V \setminus \{w\}$, si lo conecto con w quedaría G conexo, y sino queda partido en dos: w por un lado, y K_{n-1} por el otro, ya que cada nodo estaba conectado con todos menos w, formando así un subgrafo completo.

□

Ahora si, demuestro el ejercicio pedido

Dem (3b). Ya que $m > \frac{(n-1)(n-2)}{2}$, se agrega al menos una arista a G', el subgrafo de G que tiene exactamente $m = \frac{(n-1)(n-2)}{2}$. Por (3), G' es conexa o tiene dos componentes conexas, K_1 y K_{n-2}.

- Si G' es conexa, agregar solo aristas a un grafo conexo seguirá siendo conexo.

- Sino, $G' = K_1 + K_{n-2}$, y las aristas que se agregan no pueden ir dentro de cada componente conexa, ya que son grafos completos. Necesariamente tienen que tener un extremo en cada una. Al unirse las componentes conexas, G queda conexo.

Para ambos casos G es conexo. □

Ejercicio 4

*Un **puente** de un grafo es un eje del grafo tal que al removerlo se obtiene un grafo con más c.cs. Sea $G = (V, X)$ un grafo conexo y $e \in X$. Demostrar que e es un puente de G sii e pertenece a todo AG de G.*

Dem. Veamos la ida y la vuelta.

$\Rightarrow$) Sea e puente de G. qvq pertenece a todo AG de G.

Voy a probar el contrarecíproco, $e \notin T$ un AG de G $\implies$ e no es puente de G

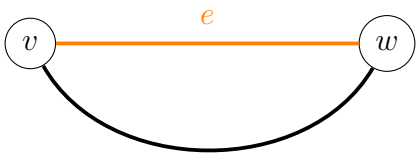
$$
\begin{aligned}
e \notin T &\implies T + e \text{ tiene un ciclo C que contiene a e} && \text{(Teo. 1.2, } T \text{ árbol)} \\
&\implies C \in G && (C \in T + e \subseteq G) \\
&\implies e \text{ pertenece a un ciclo de G} \\
&\implies G - e \text{ es conexo} && \text{(Lema 2)} \\
&\implies G - e \textbf{ no} \text{ tiene menos componentes conexas que G} \\
&\implies e \textbf{ no} \text{ es puente de G.} && \text{(por def de puente)}
\end{aligned}
$$

$\Leftarrow$) qvq e pertenece a todo AG de G $\implies$ e es puente de G.

Se que e es puente $\iff$ $G - e$ tiene más componentes conexas que G. Y si G es conexo, eso quiere decir que $G - e$ no lo es. Supongo que no es puente y que $e = (v, w)$.

$$
\begin{aligned}
e \text{ no es puente} &\implies G - e \text{ es conexo} && \text{(G conexo)} \\
&\implies e \in C \text{ un circuito simple de G,} && \text{(por Lema 2)}
\end{aligned}
$$

y lo descompongo de la siguiente manera,



P_{vw}

$$C : e + P_{vw}$$

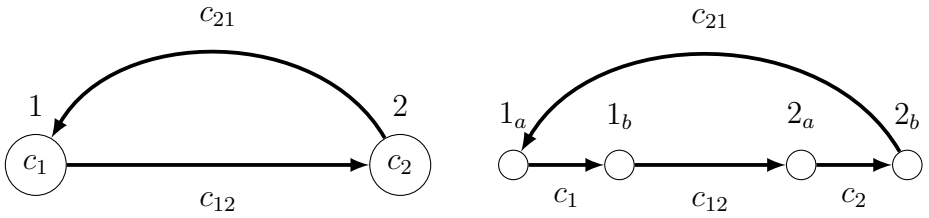
Como hay dos caminos para ir de v a w, voy a tener al menos dos AG distintos, uno para cada camino. Sea T el que contiene a P_{vw}. Como e pertenece a todo AG, en particular $e \in T$. Pero como $P_{vw} \in T \wedge e \in T \implies C : e + P_{vw} \in T$. Abs! T es un arbol, no puede tener circuitos. Como el absurdo provino de suponer que e no era puente de G, concluyo que sí lo es, que es lo que quería probar.

□

Ejercicio 5

Cabinas de peaje inverso, en las que se le paga un monto al conductor que pase. Cada cabina de peaje $i \in \{1 \dots C\}$ tiene un costo asociado c_i, el cual es negativo para las cabinas inversas, y el costo c_{ij} del viaje de la cabina i a la j, en caso de que se pueda de forma directa.

a) Para modelar el problema, me gustaría representar a las cabinas como nodos, donde una cabina es adyacente a otra si se puede viajar de forma directa. Pero tenemos un problema, las aristas y los vértices tendrían peso, cuando los grafos que modelamos solo tienen peso en las aristas. Vamos a tener que usar una representación muy similar a la de la clase de Mirko[1]. Veamos un ejemplo



(a) Digrafo original (b) Digrafo adaptado

Figure 6: Ejemplo de representación de cabinas con digrafos

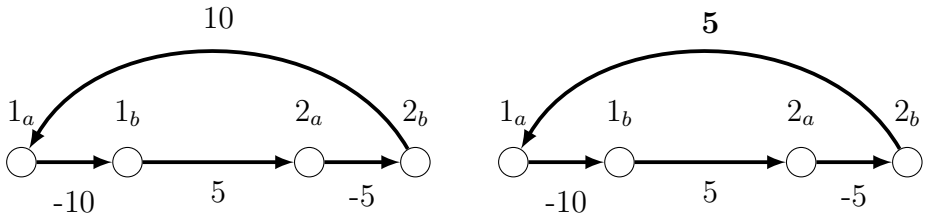
Separo cada nodo i con peso c_i en dos, i_a e i_b, con un arco de peso c_i de i_a a i_b. Todos los arcos salen de i_b, y llegan a i_a. De esta forma, me aseguro que cualquier camino que pase por el nodo pague el costo del peaje. Luego, habrá un arco de i_b a j_a con peso c_{ij} si es posible viajar de forma directa de la cabina i a la j.

Matemáticamente, $G = (V, X)$ donde

$$V = \{1_a, 1_b, \dots C_a, C_b\},$$
$$X = \{(i_b, j_a, c_{ij}) \mid \text{se puede viajar de forma directa de i a j}\} \cup \{(i_a, i_b, c_i) \mid i \in \{1, \dots, C\}\}.$$

Con este modelo, se podrá obtener una ganancia recorriendo *eternamente* las cabinas si el grafo tiene **ciclos negativos**, circuitos que luego de recorrerlos dan una ganancia.

[1] `https://campus.exactas.uba.ar/pluginfile.php/232068/mod_resource/content/1/CAMINOS%20MINIMOS.pdf`



(a) Cabinas sin ciclos negativos (b) Cabinas con ciclos negativos

Figure 7: Ejemplo de cabinas con y sin ciclos negativos

En la figura 7 se pueden ver dos ejemplos de cabinas y montos. En 7a no hay ningún ciclo negativo, a pesar de que el camino $P : 1_a\ 1_b\ 2_a\ 2_b$ tiene costo -10, al volver a 1_a se vuelve 0, con lo cual recorrer infinitamente nunca daría ganancias. En cambio, en 7b se puede ver como con un pequeño ajuste, el ciclo $C : 1_a\ 1_b\ 2_a\ 2_b\ 1_a$ tiene costo -5, con lo cual se puede recorrer eternamente, generando ganancias.

b) Para resolver el problema del inciso anterior, basta con ver si hay ciclos negativos en algún camino de una cabina a la otra. Para ello, se puede usar el algoritmo de **Floyd**, pensado para resolver camino mínimo *múltiples orígenes, múltiples destinos*, que puede ser adaptado para detectar ciclos negativos. Este tiene una complejidad temporal de $O(n^3)$ con n la cantidad de vertices. Y en nuestro caso, $n = 2 \times |C|$, el doble de la cantidad de cabinas, ya que cada una se separa en dos nodos.