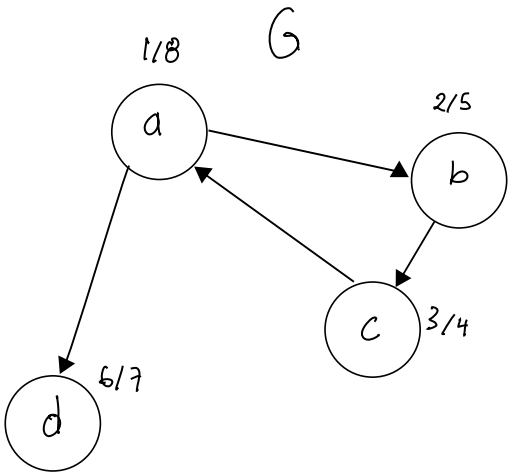
Juan Cisneros Resendiz
Algoritmos en gráficas 2021
Tarea 4
Fecha de entrega: 20 de febrero

El objetivo de esta tarea es que
r mejor el algoritmo de Kosaraju-Sharir y las ideas que sustentan que sea correcto.

**1.** El profesor John Smith asegura que el algoritmo de Kosaraju-Sharir puede simplificarse usando la siguiente estrategia: usar la gráfica original, en lugar de la invertida, en el segundo DFS y escanear los vértices en orden creciente respecto a su *.post*. Argumenta por qué esta estrategia simplifica el algoritmo o da un contraejemplo.

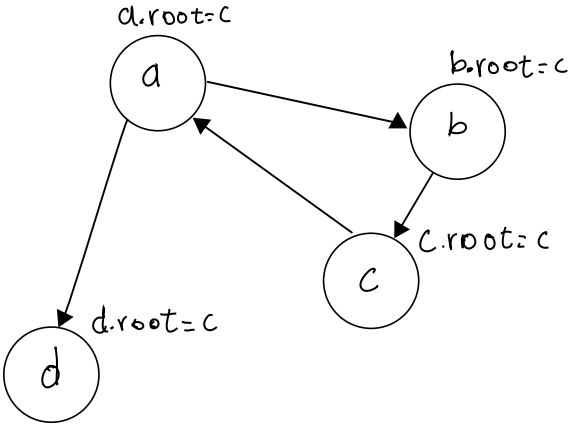
Esa estrategia en general no funciona, consideremos el siguiente contraejemplo.




Postorden

1 c
2 b
3 d
4 a

( a ) ( b ) ( d ) ( c )




Al hacer este recorrido postorden particular tenemos al vértice $c$ como el primero en el orden postorden, y en la segunda fase a todos los vértices marcamos como su raíz a $c$, ya que $c$ alcanza a todos los demas, es decir, que solo hay una componente conexa, pero $d$ solo se alcanza a si mismo. Las componentes conexas de está gráfica son $\{a, b, c\}$ y $\{d\}$. ∎

**2.** Considera las siguientes definiciones.

Un *árbol enraizado* $T(x)$ es un árbol $T$ con un vértice específico $x$ al que llamamos la *raíz* de $T$. Una orientación de un árbol enraizado en la cual cada vértice, salvo la raíz, tiene in-grado igual a uno es llamada *branching*. Denotaremos un branching enraizado en un vértice $x$ como $x$*-branching*.

En este problema deberás diseñar un algoritmo que use Sage para comprobar computacionalmente el siguiente teorema.

**Theorem 1.** *Una digráfica $D = (V, E)$ es fuertemente conexa si y sólo si tiene un $v$-branching generador para cada vértice $v \in V$.*

Programa tu algoritmo usando Sage. Ejecuta el programa para distintas gráficas y haz un reporte discutiendo tus resultados.

Tu reporte debe contener: tu algoritmo, análisis de la complejidad de tu algoritmo como función de la complejidad de las funciones que uses de Sage, la lista de gráficas que utilizaste para tus experimentos, una discusión de los resultados de tus experimentos. En particular deberás argumentar si tu algoritmo comprueba computacionalmente el teorema o lo refuta.

**Respuesta:**

Para encontrar si un vértice $v$ tiene un $v-$branching generador en la gráfica dirigida $G$ necesitamos encontrar un árbol generador tal que tenga como única raíz a $v$ y todos los demás vértices tienen un único padre (in-grado igual a uno). Usando la clasificación de aristas, si existe tal árbol podemos encontrarlo haciendo un recorrido DFS comenzando en $v$ y viendo el árbol generado por los tree edges, eso es porque los tree edges forman un árbol dirigido tal que, salvo el vértice raíz, todos los demás vértices del árbol tienen un solo padre (véase la clasificación de aristas), y un recorrido *WhateverFirstSearch* comenzando sobre el vértice $v$ marca a todos los vértices alcanzables por $v$, en particular un recorrido DFS marca a todos los vértices alcanzables por $v$ (suponiendo que no se haya ejecutado anteriormente con otro vértice raíz).

Así que solo tenemos que ver cuantos vértices podemos agregar al árbol formado por los tree edges (no necesitamos construirlo explícitamente), y si ese árbol tiene tamaño $n$, donde $n$ es el número de vértices de $G$, entonces existe un $v-$branching generador. Contamos el número de vértices del árbol enraizado en $v$ formado por los tree edges con el algoritmo ArbolGenerador(v)

```
ArbolGenerador(v)
    for all vertices w
      unmark w
    Tree = 0 (tamaño del árbol generado por v)
    DFSArbolGenerador(v)
    return Tree
```

```
DFSArbolGenerador(v):
    mark v
    for each edge v → w
      if w is unmarked
        // parent(w) ← v
        Tree = Tree + 1
        DFSArbolGenerador(w)
```

Y el código en Sage.

```
def arbol_generador(G, v):
    marcado = [False for i in G.vertices()]
    num_marcados = 0

    def DFS(v):
        nonlocal marcado
        nonlocal num_marcados
        marcado[v] = True
        num_marcados += 1
        for w in G.neighbor_out_iterator(v):
            if (not marcado[w]):
                DFS(w)

    DFS(v)
    return num_marcados
```

Entonces para comprobar computacionalmente la primera parte del teorema, si la digráfica $G$ es fuertemente conexa entonces existe un $v$-branching generador para cada $v \in G.V$, necesitamos repetir el siguiente experimento un número suficiente de veces.

- Generar una gráfica aleatoria fuertemente conexa $G$
- Verificar para cada $v \in G.V$ si existe un $v$-branching generador
- Si para algún $v$ falla el paso anterior, entonces el teorema es falso.

Sage no tiene una función específica para generar una gráfica conexa aleatoria, para generarla tenemos que generar una gráfica dirigida aleatoria y verificar si es fuertemente conexa, si no lo es hay que generar otra gráfica aleatoria y repetir ese procedimiento hasta obtener una gráfica fuertemente conexa. Eso lo hacemos de la siguiente manera en Sage.

```
#probabilidad de que una arista esté presente
p = 0.3
#genera una gráfica conexa aleatoria
def randomGraficaConexa(num_vertices):
    #genera una digráfica aleatoria
    G = digraphs.RandomDirectedGNP(num_vertices, p)
    #mientras la gráfica aleatoria generada no sea conexa,
    #volver a generar otra
    while (not G.is_strongly_connected()):
        G = digraphs.RandomDirectedGNP(num_vertices, p)
    return G
```

■

Para verificar si una gráfica $G$ conexa tiene un $v$-branching generador para cada vértice $v \in G.V$ se comprueba si existe un branching generador por cada vértice, eso tiene complejidad cuadrática. El algoritmo en Sage es el siguiente

```
def testGraficaBranchings(G):
    n = len(G.vertices())
    for v in G.vertex_iterator():
        if arbol_generador(G,v) != n:
            return False
    return True
```

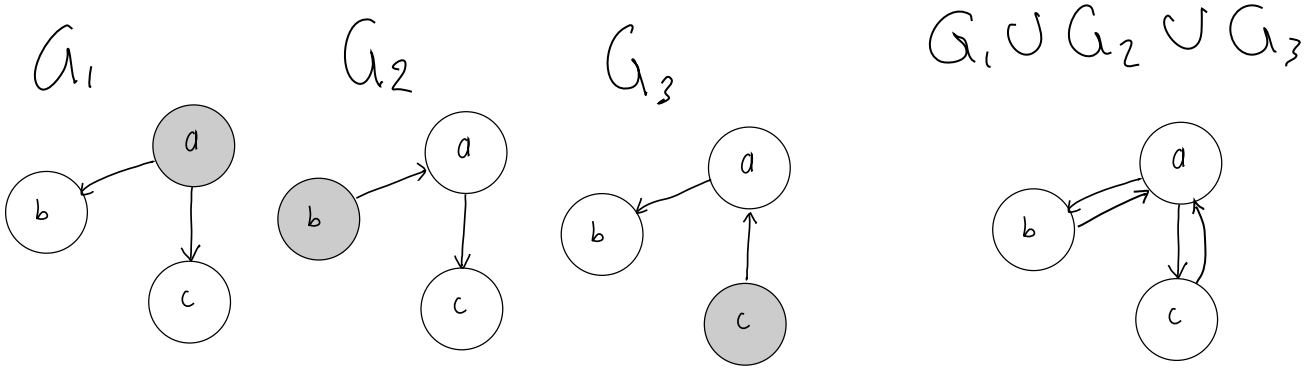
El algortimo en Sage para verificar computacionalmente si se cumple la primera parte del teorema (si la digráfica $G$ es fuertemente conexa entonces existe un $v$-branching generador para cada $v \in G.V$) es el siguiente.

```python
import random

#comprueba computacionalmente que dada
#una gráfica conexa existe un v-branching para
#cada vértice de la gráfica
def ConexaTestBranching(num_experimentos = 10**5, tam_maximo = 30):
    for exp in range(num_experimentos):
        #da un valor aleatorio en el rango [1, tam_maximo]
        #para determinar el tamaño de la gráfica aleatoria
        tam_random = random.randint(1, tam_maximo)
        G = randomGraficaConexa(tam_random)
        for v in G.vertex_iterator():
            #si no podemos generar un arbol generador con
            #los tree edges que tenga todos los vértices de la
            #gráfica, entonces no hay un v-branching generador
            if (arbol_generador(G,v) != tam_random):
                return False

    return True
```

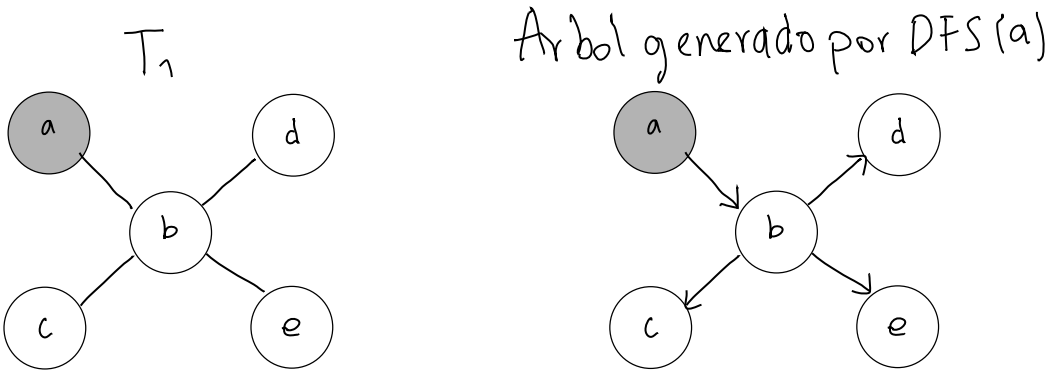
Si este algoritmo retorna el valor verdadero, entonces tenemos evidencia de que esta primera parte del teorema es verdadero, pero si retorna falso encontramos un contraejemplo de que el teorema es falso.

El análisis de la complejidad es el siguiente. El generar un número aleatorio toma tiempo constante. El generar la gráfica conexa aleatoria toma tiempo $O(1/P(n) \cdot (V+E))$, donde $P(n)$ es la probabilidad de que una digráfica de $n$ vértices aleatoria sea conexa. Verificar si existe un $v$-branching para cada vértice $v \in G.V$ toma tiempo $O(V+E)$ por vértice, en total toma $O(V(V+E))$, y haciendo $N$ experimentos la complejidad total del algoritmo es $O(N(1/P(n) \cdot (V+E) + V(V+E)))$. Según los experimentos de la tarea pasada, la probabilidad de que una gráfica aleatoria sea conexa se acerca a uno mientras mas grande sea $n$, así que situaciones prácticas la complejidad del algoritmo es $O(V(V+E))$ por cada iteración del experimento.

Para comprobar la segunda parte del teorema, si dada una gráfica $G = (V, E)$ tal que existe un $v$-branching generador por cada $v \in V$, entonces es conexa, necesitamos una manera de generar aleatoriamente gráficas que cada uno de sus vértices tenga un $v$-branching generador. Para eso generaremos un árbol que tenga $n$ vértices y que sea un $v$-branching, eso lo hacemos por cada vértice y unimos todos esos arboles. para generar la gráfica aleatoria.




Para generar aleatoriamente un $v$-branching primero generamos un árbol aleatorio no dirigido, y después ejecutamos un WhateverFirstSearch($v$) (en particular un DFS) tomando como vértice raíz $v$, por las notas de la clase 4, ese algoritmo genera un árbol con raíz $v$, y todos los vértices alcanzables desde $v$ (que no son $v$, todos los demás vértices de la gráfica en este caso) tienen un único padre, es decir, tienen in-grado uno.




El algoritmo en Sage para eso es el siguiente.

```
#genera una gráfica que es un v-branching aleatorio
def random_v_branching(v, num_vertices):
    lista_adj = dict()
    for i in range(0, num_vertices):
        lista_adj[i] = list()
    marcado = [False for i in range(num_vertices)]
    #genera un arbol no dirigido aleatorio
    G = graphs.RandomTree(num_vertices)
    #recorrido DFS para generar un v-branching del
    #arbol no dirigido
    arbolDFS(G,v, lista_adj, marcado)
    #convierte de lista de adjacencia a digráfica
    D = DiGraph(lista_adj, format='dict_of_lists')
    return D

#realiza un recorrido DFS sobre el arbol no dirigido de n vértices G
#y guarda el arbol dirigido generado por DFS en la lista de adjacencia
#lista_adj
def arbolDFS(G,v, lista_adj, marcado):
    marcado[v] = True
    for w in G.neighbor_iterator(v):
        if (not marcado[w]):
            #parent(w) = v
            lista_adj[v].append(w)
            arbolDFS(G, w, lista_adj, marcado)

#genera una gráfica aleatoria de num_vertices vértices
#tal que para cada vértice existe un v-branching generador
def randomGraficaBranchings(num_vertices):
    if num_vertices >= 1:
        #un v-branching aleatorio para el primer vértice
        G = random_v_branching(0, num_vertices)
        for i in range(1, num_vertices):
            #v-branching aleatorio para cada vértice
            rnd_v_branching = random_v_branching(i, num_vertices)
            #union de todos los v-branching
            G = G.union(rnd_v_branching)
        return G
```

Ahora lo único que nos falta es hacer los experimentos para verificar esta segunda parte del teorema (si dada una gráfica $G = (V, E)$ tal que existe un $v$-branching generador por cada $v \in V$, entonces es conexa). El algoritmo en Sage es el siguiente.

```
def BranchingTestConexa(num_experimentos = 10**5, tam_maximo = 30):
    for exp in range(num_experimentos):
        tam_random = random.randint(2 ,tam_maximo)
        #genera una gráfica aleatoria tal que cada por cada vértice haya
        #un v-branching generador
        G = randomGraficaBranchings(tam_random)
        for v in G.vertex_iterator():
            if (not G.is_strongly_connected()):
                return False

    return True
```

La complejidad de este algoritmo es la siguiente: para generar una gráfica aleatoria tal que para todo vértice exista un $v$-branching es necesario generar $n$ árboles que son un $v$-branching, donde $n$ es el número de vértices, cada árbol tiene $E = V - 1$ aristas, así que toma tiempo $O(V)$ generar cada $v$-branching, en total generar $v$-branchings toma tiempo $O(V^2)$, además toma tiempo $f(V + E)$ unir un nuevo $v$-branching a la gráfica que vamos generando de $v$-branchings, así que en total el costo de generar una gráfica aleatoria con estas propiedades es $O(f(V + E) \cdot V^2)$, y si se quiere hacer $N$ experimentos el costo del algoritmo es de $O(N(f(V + E) \cdot V^2))$

**Resultados**

Se realizaron los dos experimentos para verificar las dos partes del teorema, a partir de la gráfica conexa ver si existen $v$-branchings para cada vértice, y a partir de una gráfica con $v$-branchings para cada vértice ver si la gráfica es conexa, con $10^5$ gráficas aleatorias cada una, de tamaño 1 a 30, en los dos experimentos no se encontró ningún contraejemplo, eso da algo de evidencia de que el teorema es cierto.

**3.** Sea $G$ una gráfica con pesos en sus aristas, y supón que todos los pesos son distintos. Sea $e$ la arista de $G$ con menor peso. Demuestra que $e$ está en el árbol generador de peso mínimo de $G$. Hint: utiliza cut & paste.

Por contradicción sea $T$ el árbol generador de peso mínimo de $G$ tal que $T$ no tenga a la arista de peso mínimo $e$.

La subgráfica $T\{e\}$ tiene un único ciclo $C$ que contiene a la gráfica $e$. Sea $e'$ cualquier arista de $C$ que no sea $e$. Tenemos que $w(e') > w(e)$ ya que todas las aristas tienen peso diferente.

Consideremos el árbol generador $T' = T + e - e'$

$$\begin{aligned} w(T') &= w(T) + w(e) - w(e') \\ &< w(T) \qquad\qquad \text{ya que } w(e') > w(e) \end{aligned}$$

Pero eso contradice que $T$ es el árbol generador de peso mínimo, así que $T$ tiene que tener a la arista de peso mínimo $e$. ∎