Juan Cisneros Resendiz
Algoritmos en gráficas 2021
Tarea 1
Fecha de entrega: 30 de enero

El objetivo de esta tarea es que recuerdes y te sientas cómodo haciendo demostraciones, en particular la técnica de demostración por inducción. Además, que entiendas con un poco más de profundidad el algoritmo de PageRank. También que puedas entender un algoritmo genérico de recorrido en gráficas, como el que aprendimos en clase, pues vamos a revisitar ese algoritmo varias veces a lo largo del curso.

1. Un par de demostraciones sobre árboles.

a. Demuestra que un árbol es una gráfica conexa minimal, es decir, que al eliminar cualquier arista la gráfica se desconecta.

b. Demuestra usando inducción que todo árbol tiene al menos dos vértices de grado uno (es decir, dos hojas). Hint: usa 1.

Para demostrar el inciso a. es es necesario demostrar primero un lema.

Lema 1 *En un arbol para un $v \in V$ dado existe exáctamente un camino desde v a cualquier $w \in V$*

Sea V el conjunto de vértices de la gráfica, y sea n igual a el número de vértices de la gráfica. Supongamos por contradicción que existen dos o más caminos desde $v \in V$ a algún vértice w. Sean $v \to u_1 \to u_2 \to \ldots \to u_i \to w$ y $v \to t_1 \to t_2 \to \ldots \to t_k \to w$, $0 \leq i \leq n-2$, estos caminos. Esto contradice que la gráfica no contiene ciclos, ya que podemos formar con estos dos caminos el ciclo $v \to u_1 \to u_2 \to \ldots \to u_i \to w \to t_k \to t_{k-1} \to \ldots \to t_1 \to w$. Esto completa la demostración

a. Sea T un árbol y sean $u, v \in V(T)$. El lema 1 implica que únicamente existe un camino desde v hasta u, y como T es conexo sabemos que dicho camino existe. Sea (x, y) una arista en el camino de v a u. Si eliminamos (x, y) el vértice u deja de ser alcanzable desde v, y la gráfica se desconecta. Esto completa la demostración.

b. Haremos una demostración por inducción sobre el número de vértices.

- Caso base $n = 2$
  En este caso solo existen dos vértices y claramente los dos tienen grado uno.
- Caso inductivo $n = k + 1$.
  Supongamos que $n \geq 3$ y que el teorema se cumple para $0 \leq n \leq k$

  Sea G un árbol con $k+1$ vértices, y sea u, v dos vértices de G adyacentes. Al considerar los grados de u y v pueden ocurrir tres casos,

  1) v y u tienen grado 1.
  2) v tiene grado mayor o igual a 2, y u tiene grado 1.
  3) v y u tienen grado mayor o igual a 2

  Ahora veamos que se cumple el teorema para los tres casos.

  1) Este caso no puede existir, ya que para una gráfica conexa $n \geq 3$ y dos vértices adyacentes u y v, al menos un tercer vértice tiene que ser adyacente a alguno de estos dos vértices, digamos u, esto hace que u tenga grado 2.

2) Este es el caso en que $deg(u) = 1$ y $deg(v) \geq 2$. G es una gráfica conexa minimal con $k+1$ vértices. Al remover la arista (u, v) obtenemos las gráficas $H = G - \{u, v\}$ y $T = \{u\}$. H es un árbol de k vértices, así que por H.I H tiene al menos dos hojas. Al vovler a agregarle la arista a $G = H \cup T \cup \{u, v\}$ a lo mucho un vértice de H deja de ser hoja, así que H "contribuye" al menos una hoja a G, y v tiene grado 1 en G, así que G tiene al menos dos hojas.

3) Este es el caso en que $deg(u), deg(v) \geq 2$. Al remover la arista $\{u, v\}$ de G obtenemos dos componentes conexas H y T, estas componentes son árboles con a lo mucho k vértices cada una, así que por la hipótesis de inducción H y T tienen al menos dos hojas cada uno, ahora volvamos a considerar a $G = H \cup T \cup \{u, v\}$, al agregar la arista u, v tanto en H como en T a lo sumo un vértice deja de ser hoja, así que G tiene al menos dos hojas.

■

2. Un recolector de basura es un mecanismo a través del cual los espacios de memoria, de una computadora, que ya no se estén utilizando son liberados. Generalmente el mecanismo ocurren en dos etapas: uno, marcar los espacios que se estén usando aún, y dos, eliminar los no marcados. En 1978, Dijkstra, Lamport, Martin, Scholten y Steffens publicaron un artículo en donde presentaron el primer método recolector de basura que actúa de manera concurrente. Este artículo contiene, implícitamente, una de las primeras versiones de un algoritmo genérico de recorrido de gráficas. El algoritmo es el siguiente:

```
ThreeColorSearch(s):
    color all nodes white
    color s gray
    while at least one vertex is gray
      ThreeColorStep()
```

```
ThreeColorStep():
    v ← any gray vertex
    if v has no white neighbors
      color v black
    else
      w ← any white neighbor of v
    parent(w) ← v
    color w gray
```

1. Demuestra que durante toda la ejecución del algoritmo, y también una vez que este termina, nunca ocurre que un vértice negro sea vecino de uno blanco. Hint: es súper fácil.

2. Demuestra que una vez que el algoritmo ThreeColorSearch(s) termina:

   a) todos los vértices alcanzables desde s son negros

   b) todos los vértices no alcanzables desde s son blancos

   c) las "aristas padre" $v \to \text{parent}(v)$ definen un árbol generador enraizado del componente que contiene a s

Respuesta:

1. El algoritmo comienza coloreando todos los vértices de blanco, y en la única parte del algoritmo donde un vértice se colorea de negro es en el if de ThreeColorStep(), ese if solo colorea al vértice v de negro si no tiene vecinos blancos, así que ni durante la ejecución ni cuando termina el algoritmo ocurre que un vértice negro es vecino de uno blanco. ■

2. *c*) Para demostrar este inciso necesitamos primero demostrar un lema

   Lema 3 *Para todo vértice v coloreado de gris durante la ejecución del algoritmo, el camino $v \to parent(v) \to parent(parent(v)) \to \ldots$* en algún punto vuelve a s, y además no tiene ciclos.

   Demostraremos por inducción sobre el orden en que los vértices son marcados.

El primer vértice en ser marcado de gris es s, en este caso el camino consiste únicamente en s.

Supongamos que $v \neq s$, donde v es un vértice coloreado de gris, parent(v) debió necesariamente ser colorearlo antes de v (ya que asignamos como parent(v) a un vértice gris). Entonces nuestra hipótesis de inducción implica que el camino parent(v) $\rightarrow$ parent(parent(v)) $\rightarrow \ldots$ llega a s. Al agregar la arista $v \rightarrow$ parent(v) obtenemos que el camino $v \rightarrow$ parent(v) $\rightarrow$ parent(parent(v)) $\rightarrow \ldots$ vuelve a s

Además el camino $v \rightarrow$ parent(v) $\rightarrow \ldots$ no tiene ciclos (es idéntico al caso de las notas de clase).

■

Ahora demostraremos las "aristas padre" $v \rightarrow$ parent(v) definen un árbol generador enraizado del componente que contiene a s.

Por el inciso a) todos los vértices alcanzables desde s son negros, y por el b) todos los no alcanzables son blancos, así que el conjunto de los vértices alcanzables desde s consiste en los vértices negros. Por el lema 3 el conjunto de aristas padres de los vértices v que han sido grises forman un camino acíclico a s, esos vértices grises al finalizar el algoritmo se tienen que volver negros. Durante la ejecución del algoritmo un vértice coloreado de negro v es coloreado de gris una sola ves, así que cada vértice coloreado de negro tiene un único padre, excepto s, Como la gráfica es conexa, el número de aristas padre es el número de vértices menos 1, eso implica que es un árbol generador con la raíz s de todos los vértices alcanzables por s, es decir, de todos los vértices de la componente conexa que contiene a s.

– Para los incisos a) y b) primero necesitamos demostrar un lema.

Lema 2 *El vértice v es alcanzable desde s si y solo si v es coloreado de gris durante la ejecución del algoritmo.*

Aquí nos referimos a que un vértice v es coloreado de gris durante la ejecución del algoritmo si v es coloreado de gris en algún instante de la ejecución del algoritmo.

($\Longrightarrow$) *Si el vértice v es alcanzable desde, entonces es coloreado de gris durante la ejecución del algoritmo*

Está demostración procederá por inducción sobre la longitud del camino mas corto entre s y v.

Caso base. Como caso base tenemos un camino de longitud 0, s es coloreado de gris al inicio de ThreeColorSearch(s), así que se cumple este caso.

Caso inductivo. Sea $v \neq s$ un vértice alcanzable desde s, y sea $s \rightarrow \ldots \rightarrow u \rightarrow v$ cualquier camino de s a v con el menor número de aristas.

El prefijo $s \rightarrow \ldots \rightarrow u$ es mas corto que el camino mas corto de s a v, la hipótesis de inducción (el algoritmo colorea de gris a todos los vértices en el prefijo $s \rightarrow \ldots \rightarrow u$ implica que el algoritmo colorea de gris a u.

Después de que se colorea de gris a u, el algoritmo vuelve al ciclo while de ThreeColorSearch. v no ha sido coloreado de gris así que tiene que estar coloreado de blanco, ya que el otro color que puede tener v es negro, pero para que eso pase primero se tuvo que colorear de gris (eso es por primera condición de ThreeColorStep().

Entonces por la primera condición de ThreeColorStep() u se quedará de color gris mientras v no sea coloreado, entonces necesariamente en el ciclo while de ThreeColorSearch se ejecutará ThreeColorStep, en donde la variable v tomará el valor del vértice u o algún otro vértice vecino de v, y lo coloreará, así que se cumple el caso inductivo.

($\Longleftarrow$) *Si v es coloreado de gris durante la ejecución del algoritmo, entonces v es alcanzable desde s*

Por el lema 3 para todo vértice v coloreado de gris durante la ejecución del algoritmo existe un camino $v \rightarrow$ parent(v) $\rightarrow$ parent(parent(v)) $\rightarrow \ldots \rightarrow s$ de v a s, así que v es alcanzable desde s

■

a) Por el lema 2 todos los vértices alcanzables desde s (y solo esos) son coloreados de gris en algún instante del algoritmo, entonces todos los vértices de la componente conexa a la que pertenece s son coloreados de gris durante la ejecución del algoritmo, y una vez que son coloreados de gris ya no pueden volver a ser blancos. Consideremos el momento en el cual cada uno de los vértices alcanzables desde s ha sido coloreado de gris en algún instante durante la ejecución del algoritmo, ningún vértice que pertenece a la misma componente conexa que s tiene vecinos blancos, eso es, esa componente solo tiene vértices coloreados de gris o de negro. En el caso de que no halla vértices grises, entonces todos están coloreados de negro y se cumple el teorema. En el caso que queden vértices grises, cada uno de estos entrará a ThreeColorStep mediante el ciclo while de ThreeColorSearch, en el cual será coloreados cada uno de ellos de color negro ya que no tienen vecinos blancos.

b) De manera similar al inciso *a*), por el lema 2 solo los vértices alcanzables desde s son coloreados de gris en algún instante del algoritmo, y el color inicial de todos los vértices es el blanco, así que al terminar el algoritmo todos los vértices no alcanzables desde s se quedan coloreados de blanco.

■

3. En este problema vamos a estudiar PageRank. Considera la gráfica que se muestra en la Figura .

a. Intuitivamente ¿qué página crees que debe tener el menor PageRank, y qué página crees que debe tener el mayor? Explica tu respuesta.

b. Para responder esta pregunta vas a necesitar un dado (de seis lados). En la gráfica de la Figura repite el siguiente algoritmo por distintos periodos de tiempo, los que tú decidas, al menos tres periodos distintos. Para cada ronda anota en una tabla el PageRank de cada página. Discute ¿cuáles valores te parece que reflejan mejor el comportamiento que esperabas de la red? ¿Después de cuántas rondas crees que se estabilice el PageRank? Cada vértice tiene un contador que es su PageRank

   1. Inicializa los contadores de todos los vértices a cero
   2. Elige un vértice de la gráfica arbitrariamente, llamemos p a dicho vértice
   3. Aumenta el contador de p en uno
   4. Lanza el dado
   5. Mientras que el número que cae en el dado sea menor que 6:
      i. Elige aleatoriamente una arista que salga de p, sea dicha arista (p, q). Para elegir aleatoriamente reparte los seis números del dado equitativamente entre las aristas, lanza el dado, y sigue la arista a la que le corresponda el número que cayó en el dado.
      ii. Ahora hacemos q el nuevo p.
      iii. Lanza el dado. Si el número que cae es seis, ve al punto 2 del algoritmo.

c. Para este inciso puedes decidir o bien programar en Python el algoritmo iterativo de PageRank que vimos en clase o bien usar Sage para estudiar PageRank.

   Si decides programar el algoritmo deberás entregar tu código junto con un brevísimo reporte en donde muestres los resultados de hacer iteraciones de distintos tamaños (en el algoritmo que vimos en clase se hacían diez iteraciones) y los efectos que éstas tienen en la gráfica de este problema (las instrucciones de cómo hacerlo las colocaré en el NoteBook).

   Si decides usar Sage deberás hacer un programa en Python que use la función pagerank() (ver el muro de Teams para obtener la liga con la documentación de esta función) en la gráfica de este problema. La

función pagerank() cuenta con un parámetro que te permite elegir cuál algoritmo utilizar para calcular el PageRank, deberás usar cada uno y entregar un brevísimo reporte en donde cuentes y constrastes los resultados. Entrega también tu código.

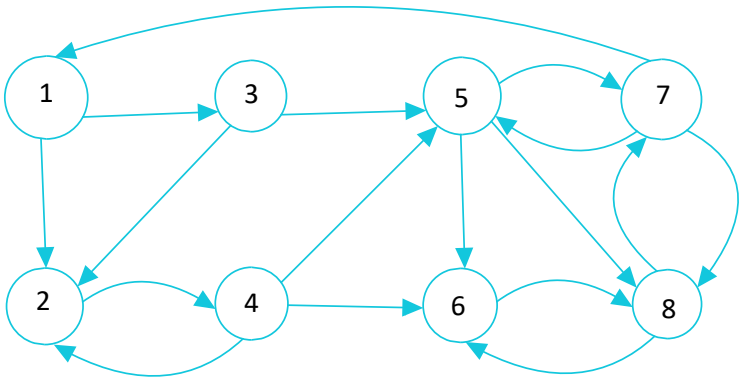

Figura 1: Gráfica para PageRank

Respuesta:

a. La página tres es la que creo que tiene el menor PageRank, ya solo tiene la página uno como vecino de entrada, y la página uno tiene un grado de entrada igual a 1.

   La página que debe de tener el mayor PageRank es la página ocho, las razones son las siguientes. Las cuatro páginas con el mayor in-grado son: la página 8 (3), la 5 (3), la 2 (3) y la 6 (3), pero de estas páginas, las páginas vecinas de 8 tienen en promedio grado de entrada 2.66, a diferencia de la página 5 (promedio 1,66), la página 2 (promedio 1) ,la página 6 (promedio 2.33) y la página 7 (promedio 3, pero solo tiene dos vértices de entrada).

b. En la tabla una ronda se refiere a los valores del PageRank que tienen los valores en el momento que algoritmo sale del ciclo while de la línea 5 Se ejecutó el algoritmo en tres periodos distintos.

   Los valores que parecen que reflejan mejor el valor del PageRank son los del experimento 2, en ese experimento se incrementó el valor del PageRank 26 veces, a diferencia del experimento 1 (24 veces) y del experimento 3 (19 veces).

   Creo que después de 8 rondas se estabilizará el PageRank, ya que con experimentos de muy pocas rondas como estos el algoritmo es propenso a quedarse ciclado en alguna ronda en particular.

c. Se comparó las tres implementaciones de PageRank que tiene Sage, con cinco valores del damping factor distintos. Entre esas implementaciones no se notan diferencias significativas. El experimento dos es el que se parece más a los valores del PageRank de Sage (usando damping factor 0,8333 = 1/6). Cuando el damping factor es igual o menor a 0,5 todas las páginas tienen valores del PageRank similares, soportando la intuición que con un valor del damping factor el caminante hará muchos saltos y se comportará más como una búsqueda aleatoria.

■

	PageRank experimento 1			
Página	Ronda 1	Ronda 2	PageRank total	Porción del PageRank
v_1	1	1	1	0.042
v_2	0	0	0	0
v_3	1	1	1	0.042
v_4	0	0	0	0
v_5	1	1	1	0.042
v_6	2	6	6	0.25
v_7	2	5	5	0.208
v_8	3	10	10	0.416

	PageRank experimento 2				
Página	Ronda 1	Ronda 2	Ronda 3	PageRank total	Porción del PageRank
v_1	0	0	1	1	0.038
v_2	0	0	2	2	0.077
v_3	1	1	2	2	0.077
v_4	0	0	1	1	0.038
v_5	1	3	5	5	0.192
v_6	0	0	4	4	0.153
v_7	0	2	4	4	0.153
v_8	0	1	7	7	0.270

	PageRank experimento 3					
Página	Ronda 1	Ronda 2	Ronda 3	Ronda 4	PageRank total	Porción del PageRank
v1	0	0	0	0	0	0
v2	0	0	3	3	3	0.157
v3	1	0	0	0	0	0
v4	0	1	3	3	3	0.157
v5	0	1	1	1	1	0.052
v6	0	4	4	4	4	0.210
v7	0	1	1	1	1	0.052
v8	1	6	6	7	7	0.368

Cuadro 1: Tablas de la ejecución de PageRank

```
PageRank con damping factor = 0.85
Networkx
1 0.06556443162164646
2 0.06642601670601414
3 0.04661465693114701
4 0.07521222484145068
5 0.11734092305660798
6 0.19719114940619267
7 0.1652260177460983
8 0.2664245796908426
Numpy
1 0.06556420527748096
2 0.0664260718211743
3 0.0466147872429294
4 0.07521216104799812
5 0.11734065830112494
6 0.1971917753070965
7 0.1652266068616974
8 0.26642373414049836
scipy
1 0.06556443162164646
2 0.06642601670601414
3 0.04661465693114701
4 0.07521222484145065
5 0.11734092305660797
6 0.1971911494061927
7 0.1652260177460983
8 0.2664245796908426
```

```
PageRank con damping factor = 0.833 (5/6)
Networkx
1 0.0661099739665120
2 0.0685722885604626
3 0.0484095675064640
4 0.0779958314654158
5 0.118757892076960
6 0.195394788988555
7 0.162909591932105
8 0.261850065503525
Numpy
1 0.0661097354896811
2 0.06857234689375191
3 0.048409704831452224
4 0.07799576496249538
5 0.11875761365886009
6 0.19539544795447014
7 0.16291021184759086
8 0.2618491743616983
scipy
1 0.066109973966512
2 0.0685722885604626
3 0.048409567506464024
4 0.0779958314654158
5 0.11875789207696004
6 0.19539478898855497
7 0.1629095919321055
8 0.2618500655035251
```

Figura 2: Ejecución algoritmo PageRank

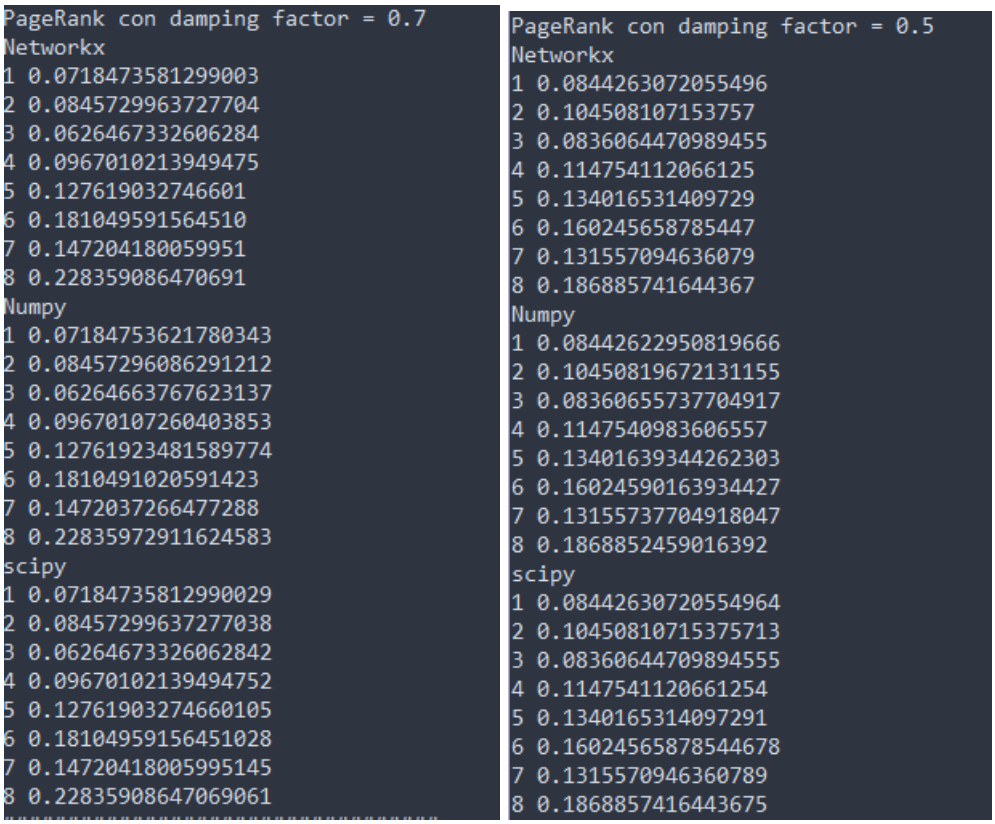

Figura 3: Ejecución algoritmo PageRank

```
PageRank con damping factor = 0.3
Networkx
1 0.0999292351562500
2 0.117862855859375
3 0.102489420312500
4 0.122858738281250
5 0.133731286328125
6 0.142719371875000
7 0.124290756250000
8 0.156118335937500
Numpy
1 0.09992908491445114
2 0.11786276714774294
3 0.1024893627371676
4 0.12285883014432292
5 0.13373131384667492
6 0.14271967366616092
7 0.12429084914451251
8 0.15611811839896703
scipy
1 0.09992923515624999
2 0.11786285585937499
3 0.1024894203125
4 0.12285873828124999
5 0.13373128632812498
6 0.142719371875
7 0.12429075625
8 0.15611833593749996
```

Figura 4: Ejecución algoritmo PageRank