Grafos, Representaciones y Algoritmos

EST-1132 / Estructuras Discretas

Juan Zamora O.

Otoño 2024

INSTITUTO DE ESTADÍSTICA
IES PUCV
PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

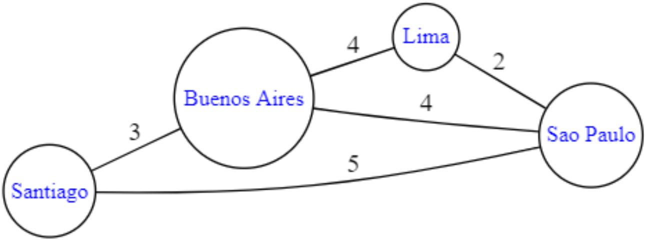
Introducción

Estudiaremos

- Qué es un Grafo y qué tipo de información permite representar
- Algoritmos para recorrer grafos
- Cómo poder representar estas estructuras computacionalmente
- Uso de grafos en problemáticas reales

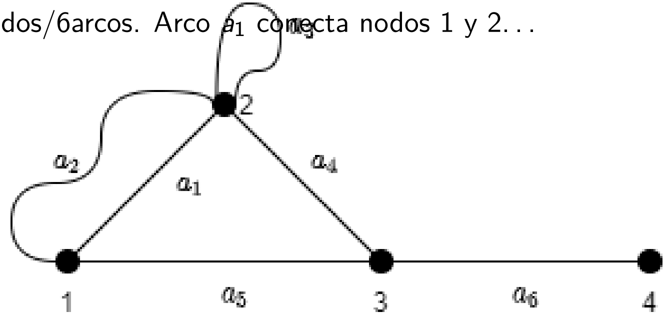
Ejemplo de la Aerolínea

- Imagine una aerolínea y sus rutas de viaje
- Un párrafo explicando esta información con distancias y quizás tiempos de viaje resulta algo dificil de asimilar en un primer acercamiento



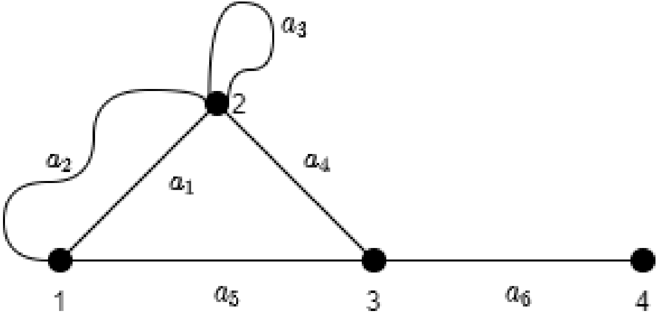
Una aproximación informal

- Un grafo es un conjunto no vacío de nodos o vértices, y arcos, donde cada uno conecta dos nodos
- 5 nodos/6arcos. Arco a_1 conecta nodos 1 y 2...



Definición formal

- Un grafo es una tupla (N, A, g) donde
  - N un conjunto no vacío de nodos
  - A un conjunto de arcos
  - g una función que asocia cada arco a con un par no ordenado de nodos, denominados puntos terminales de a



- función g asocia arcos con terminales
- $g(a_1) = 1 - 2; g(a_2) = 1 - 2; g(a_3) = 2 - 2; g(a_4) = 2 - 3; g(a_5) = 1 - 3; g(a_6) = 3 - 4;$

Ejercicio

Dibuje un grafo con nodos $\{1, 2, 3, 4, 5\}$, arcos $\{a_1, a_2, a_3, a_4, a_5, a_6\}$, y función $g(a_1) = 1 - 2; g(a_2) = 1 - 3; g(a_3) = 3 - 4; g(a_4) = 3 - 4; g(a_5) = 4 - 5; g(a_6) = 5 - 5;$

Grafo Dirigido

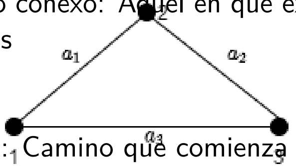
- Hasta ahora no hemos considerado orden en cada par de nodos
- Un grafo es una tupla (N, A, g) donde
  - N un conjunto no vacío de nodos
  - A un conjunto de arcos
  - g una función que asocia cada arco a con un par ordenado de nodos, donde el primero es el punto inicial y el segundo es el final de a

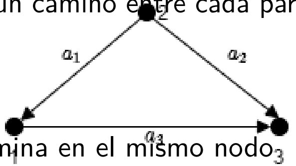
- 5 nodos/5 arcos. La función g hace asociaciones como $g(a_1) = (1, 2)$.
  - Esto significa que el arco a_1 comienza en el nodo 1 y termina en el 2.

Algo de terminología

- Nodos adyacentes: Nodos terminales asociados con un mismo arco
- Retorno: Arco con mismo nodo como inicial y terminal.
- Grafo libre de retornos: Arco sin arcos ciclicos
- Arcos paralelos: Arcos con mismos nodos terminales
- Grafo simple: Grafo libre de ciclos y sin arcos paralelos
- Grado de un nodo: Cantidad de arcos que terminan en el Nodo
- Grafo completo: Grafo en que cualquier par de nodos es adyacente

- Camino entre dos nodos: Secuencia de nodos y arcos donde cada par sucesivo de nodos y arcos sea adyacente al par siguiente.
- Largo de un camino: Cantidad de arcos en un camino
- Grafo conexo: Aquel en que existe un camino entre cada par de nodos





- Ciclo: Camino que comienza y termina en el mismo nodo donde ningun arco aparece más de una vez en la secuencia.

Representaciones computacionales

- Si bien una característica importante de un grafo es la posibilidad de visualizarlo, esta no es su virtud más relevante
- Grafos representan información relacional
- Esta información puede ser manipulada (filtrada, buscada, comparada . . .) independientemente de la visualización

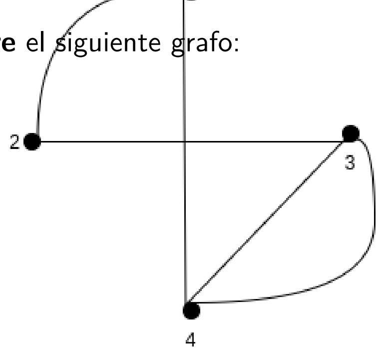
¿Qué debe ser almacenado para reconstruir la información y la visualización de un grafo?

Las dos representaciones más comunes de grafos en un computador son: 1. Matriz de adyacencia 2. Lista de adyacencia

Matriz de Adyacencia

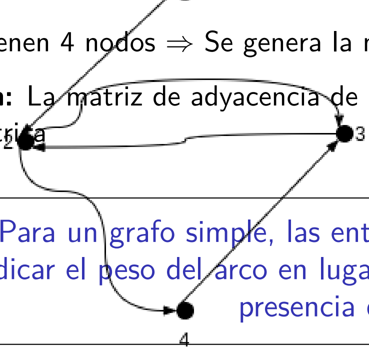
- Supongamos un grafo con n nodos numerados $n_1, n_2, \ldots, n_n$
- Esta numeración es arbitraria. Sin embargo, no tiene impacto en la interpretación del grafo.
- Gracias a este ordenamiento, podemos conformar una matriz de tamaño $n \times n$
- La entrada i, j representa la cantidad de arcos entre los nodos n_i y n_j.

Considere el siguiente grafo:



$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \\ 1 & 0 & 2 & 0 \end{bmatrix}$$

- Se tienen 4 nodos $\Rightarrow$ Se genera la matriz de 4×4
- **Nota:** La matriz de adyacencia de un grafo no dirigido es simétrica
-



$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Para un grafo simple, las entradas de la matriz pueden indicar el peso del arco en lugar de solo indicar con un 1 la presencia del arco

Para el caso de un grafo dirigido

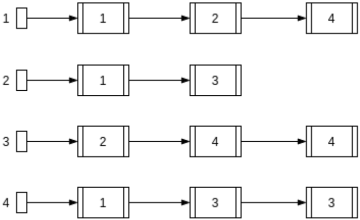
- Notar que no es simétrica

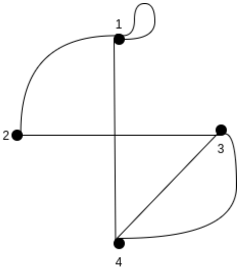
Lista de Adyacencia

- Al usar la matriz de adyacencia, para n nodos, independientemente de la cantidad de arcos, necesitamos almacenar n^2 items
- Adicionalmente, para encontrar todos los nodos adyacentes a un nodo n_i dado se necesita revisar la fila i de la matriz de adyacencia en su totalidad (n items)
- Considere el caso en que el grafo tenga muy pocos arcos en comparación con la cantidad de nodos

¿Será conveniente este esquema en este escenario?

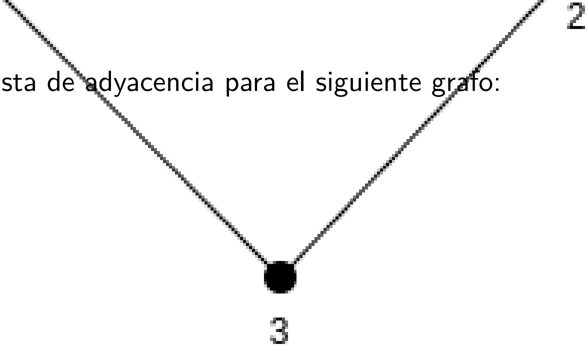
- Alternativa considerablemente más eficiente consiste en almacenar únicamente las entradas no nulas
- Se crea una lista para cada nodo con todos los nodos adyacentes a él
- Ahora para encontrar el vecindario de nodos adyacentes a uno dado, solo se debe recorrer esta lista que tiene menos de n items


1 1 2 4
2 1 3
3 2 4 4
4 1 3 3


1
2
3
4

Ejercicio

- Anote la lista de adyacencia para el siguiente grafo:



Recorriendo un grafo

Objetivo: Encontrar un camino que visite cada nodo al menos una vez

- Revisaremos dos métodos:
  a. DFS: Búsqueda en profundidad
  b. BFS: Búsqueda en anchura

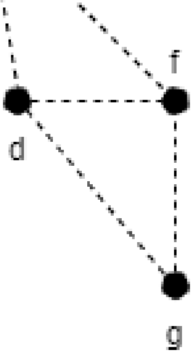
Depth-First Search (DFS)

- Comienza con un nodo arbitrario
- se marca como visitado
- se expande el listado de nodos a partir de sus vecinos adyacentes
- se repite recursivamente con el vecindario de cada nodo hasta que no quedan más nodos sin visitar

```
BuscarEnProfundidad: grafo G simple y conexo; nodo A
    marcar A como visitado
    registrar A
    Para cada nodo B adyacente a A...
        si B no ha sido visitado, entonces
            ejecutar BuscarEnProfundidad(G, B)
    fin
```

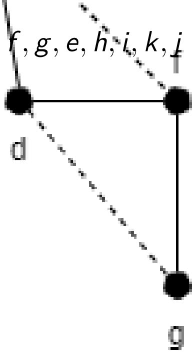
La secuencia de nodos registrados conformaran el oden por p

Ejemplo



- Se visita a, se marca como visitado y se anota.
- nodos adyacentes a a no visitados: $\{b, e, h, i\}$... se elije b
- comienza ejecución nuevamente sobre b, se marca como visitado y se anota
- nodos adyacentes $\{a, c\}$, solo c no visitado... se elije c
- comienza ejecución nuevamente sobre c, se marca como visitado y se anota ...
- al alcanzar g nos encontraremos sin nodos adyacentes no visitados. Finaliza la llamada `BuscarEnProfundida(G, g)` y continua ejecución de `BuscarEnProfundida(G, f)`

- $a, b, c, d, f, g, e, h, i, k, j$

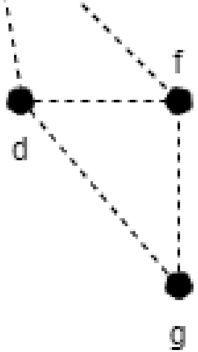


Breadth-First Search (BFS)

- Se visita *a*, se marca como visitado y se anota.
- se visita cada nodo adyacente, luego sus adyacentes . . .

```
BuscarEnAmplitud: grafo G simple y conexo; nodo A
    usar cola Q vacía
    marcar A como visitado y mostrar A
    encolar A en Q
    Mientras |Q| > 0
        Para cada nodo B adyacente al primer nodo en Q
            si B no ha sido visitado, entonces
                marcar B como visitado y mostrar B
                encolar B en Q
        remover el primer nodo en Q
```

Ejemplo



- comienza con a, se marca como visitado, muestra y agrega a Q (inicialmente vacía)
- nodos adyacentes a a no visitados: $\{b, e, h, i\}$... se marcan como visitados, muestran y agregan en ese mismo orden a la cola Q
- Se remueve el primer nodo en Q (esta vez es a)
- se revisan nodos adyacentes al tope de la cola que es b
- el único no visitado es c. Se marca como visitado, se muestra y encola en Q.
- Se remueve el primer nodo de la cola que es b... ...

- finalmente el orden BFS a partir de a es $a, b, e, h, i, c, j, k, d, f, g$