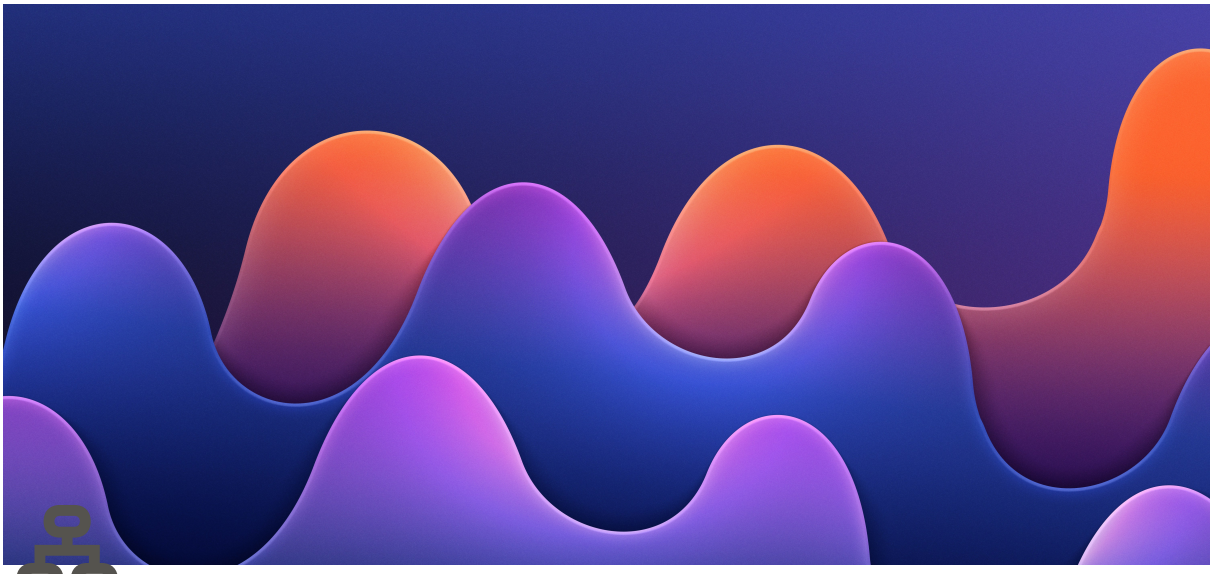


Apuntes EXAMEN - EDD IIC2133

CLASE 26

11/09/2024

BFS me permite descubrir el grafo por distancia

Encontrar todos los nodos que estan a cierta distancia

si la arista es la distancia, encontrar a todos los nodos a distancia 1, 2, 3 ...
Hasta tener el grafo completo

Va recorriendo por capas, de que forma nos damos cuenta que cual es el que tenemos que recorrer

Necesito ir guardando aquellos que ya visite, asi se que me falta recorrer aun los vecinos de cierto

Dijkstra a partir de DFS, enconuentra el **CAMINO** mas corto, que nos lleva a costo minimo de un lado a otro

Nota aparte: Dijkstra personaje problematico dentro del Campo de Computacion

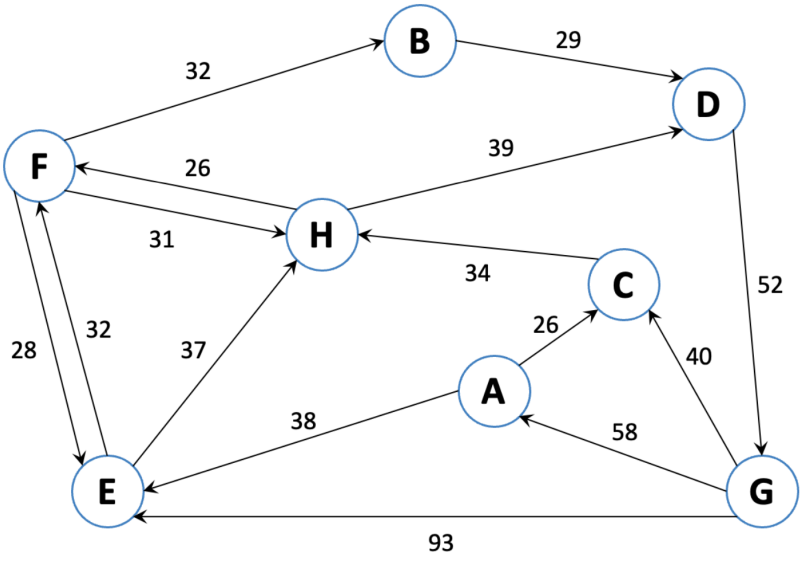
- PROBLEMA TIPO: Rutas en Viajes

Planificar un viaje en auto entre dos ciudades, podemos asociar costos a las aristas como funciones, entre varios parametros

No necesariamente con varios parametros

Esto es un **GRAFO DIRIGIDO CON COSTOS**

La forma mas barata de llegar de A a C



Notemos que en este problema, los costos son **no negativos**

Si todos los costos fueran iguales, puedo solo contar el numero de aristas. el camino con menos aristas gana!

Primero, haremos eso, lograremos buscar el camino mas corto **(SIN COSTO) → BFS**

BFS

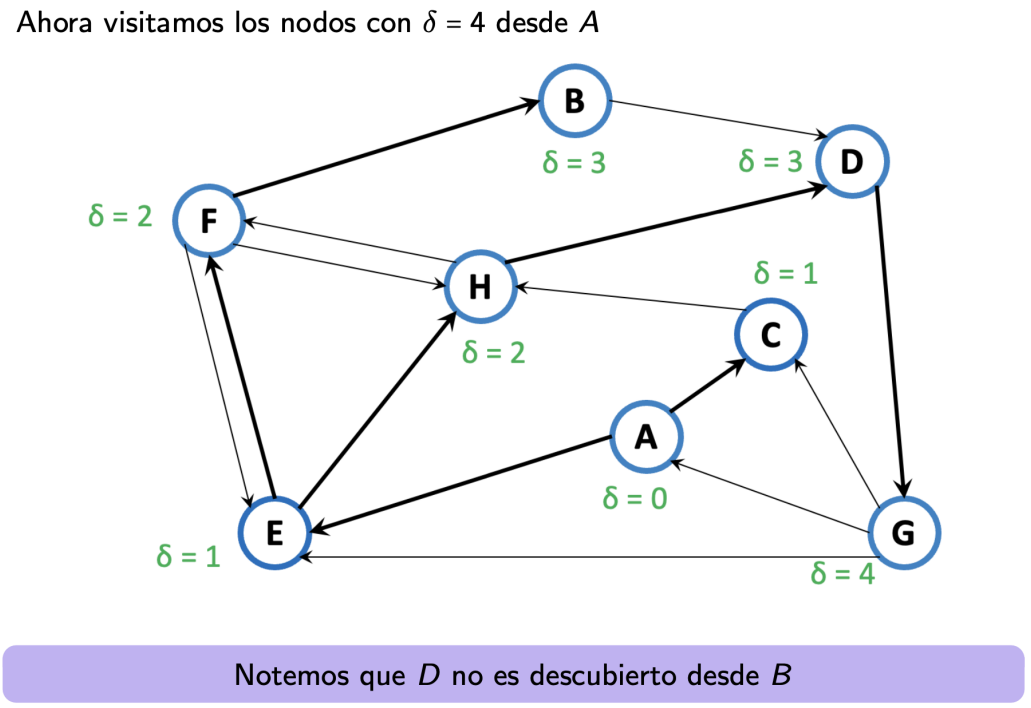
Busqueda en Amplitud

→ Similar a DFS, busqueda en el grafo → Busca vecinos inmediatos, recorre en base a **DISTANCIA DESDE UN NODO INICIAL**

Para nuestro problema Delta sera la fistancia desde el punto de inicio hasta cada nodo

Por def, delta es la **menor distancia de A a cualquier nodo**

Usaremos ademas una cola **FIFO**



IMPLEMENTACION BFS

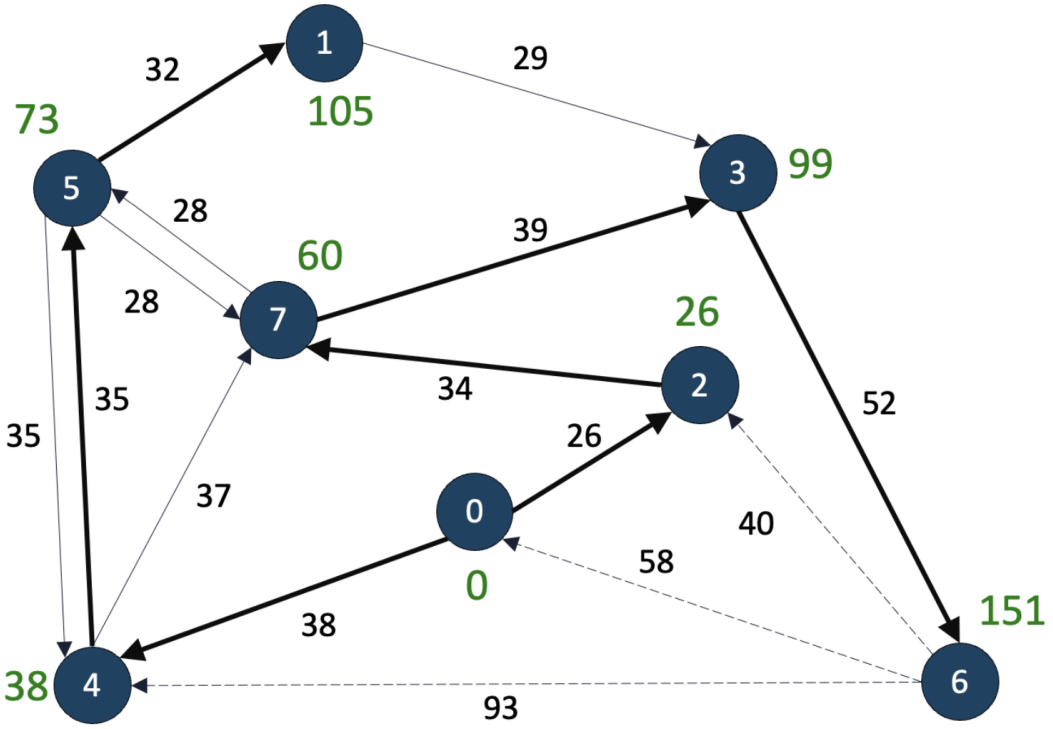
```
input  : vértice de inicio s

BFS(s):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅
    Q ← cola FIFO vacía
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco :
                v.color ← gris;  v.δ ← u.δ + 1
                π[v] ← u
                Insert(Q, v)
        u.color ← negro
```

- En estricto rigor, el negro no lo necesitamos, dado que si lo sacamos de la cola, ya no lo volveremos a ver, sin embargo el gris si es necesario, en caso de costos, podremos encontrar distancias mas cortas
- BFS deja en π la representacion de un arbol de rutas mas cortas desde S

Problema de los caminos mas cortos (con pesos)

- [ ] Estudiar Ejemplo




```
Dijkstra(s):
    for u ∈ V − {s} :
        u.color ← blanco;  d[u] ← ∞;  π[u] ← ∅
    s.color ← gris;  d[s] ← 0;  π[s] ← ∅
    Q ← cola de prioridades vacía (Min Heap)
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco ∨ v.color =gris :
                if d[v] > d[u] + cost(u, v) :
                    d[v] ← d[u] + cost(u, v);  π[v] ← u
                    DecreaseKey(Q, v, d[v])
                if v.color =blanco :
                    v.color ← gris;  Insert(Q, v)
        u.color ← negro
```

Solo cambia costo+ruta si hay arista que baja el costo

Hay un problema, si el nodo es blanco, lo inserto en la cola

Decrease Key deberia tener un if antes de que el nodo sea gris

Porque si es blanco, aun no lo he descubierto

Cuando es gris, solo lo actualizo si me conviene,

El nodo blanco que acabo de encontrar aun no esta en el **HEAP**

Por eso deberia tener un if gris → Decrease

Si es blanco, lo actualizo y lo inserto en la cola

DIJKSTRA es GREEDY

→ tiene subestrcutura optima

Programacion Dinamica utiliza subestructura optima

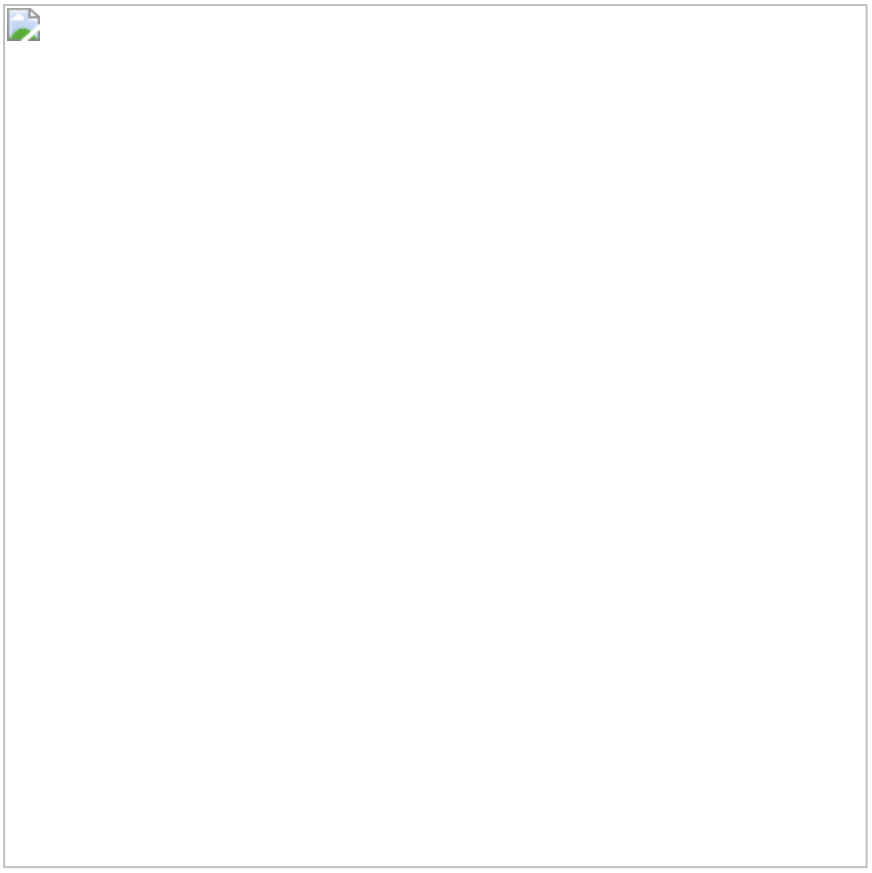
Para probar que dijkstra funciona, usaremos esa idea

Si esta es una ruta de costo minimo, entonces una ruta intermedia entre la ruta grande, tambien es la mas corta →

Lo que vamos a hacer, es que si dentro de la ruta mas corta que encontre, yo puedo encontrar una ruta intermedia que me lleve a una ruta mas corta, encunetro una forma mas barata, llegare a una contradiccion .> Debieron haber costos negativos

No puedo llegar una intermedia, eso implicaria que hay una forma de **Decrementar costos** en vez de incrementar → Lo veremos la proxima clase

Problema de poner Dijkstra con Costos negativos



Dijkstra no funciona para esto, porque pinto el camino mas corto, el B no tiene vecinos, no hay forma mas corta de llegar de A a B, luego en C no puedo llegar a B, sin embargo es mas barata

Dijkstra no se da cuenta de esto!

La proxima clase, veremos **BELLMAN FORD- > Restriccion menos duras**

Permite que hayan aristas negativas → Pero sin **CICLOS DE SUMA NEGATIVA**

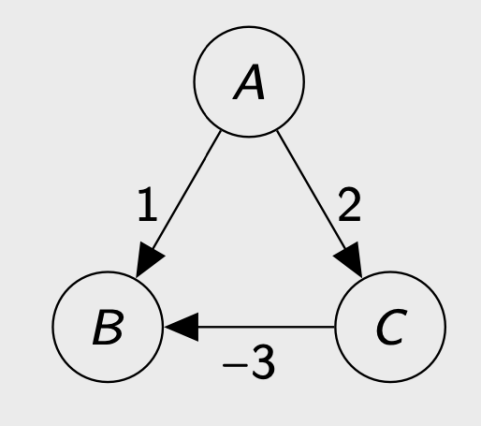
Ciclos que infinitamente se quedan dando vueltas reduciendo costos

BELLMAN FORD, ademas te **DICE CUANDO HAY CICLO NEGATIVO** dentro de un grafo!!

CLASE 27

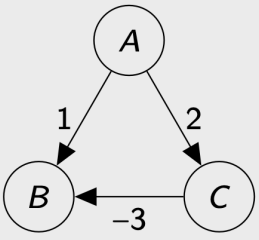
27/11/24

- Con dijkstra no puedo determinar caminos con aristar negativas
- Porque cuando termina de revisar un nodo → Lo saca → Lo pinta de negro
- Tener aristas negativas pueden darnos caminos mas largos y que sean mas “optimos”



- El problema de fondo → Tiene un Supuesto → Greedy asume que al agregar aristas el costo solo puede crecer o mantenerse, dijkstra asume que agregar una arista mas, lo mejor que puede pasar, es que cueste lo mismo, por ello la estrategia de **DIJKSTRA FALLA CON COSTOS NEGATIVAS**

El estado final de los parámetros nos entrega la ruta encontrada por Dijkstra



t	$(d[X], \pi[X])$ A	B	C	Q
4	$(0, \varnothing)$	$(1, A)$	$(2, A)$	0 1

La ruta de A hasta B entregada cumple

$$cost((A, B)) = 1 > -1 = cost((A, C, B))$$

Cuando los costos son negativos → Vamos a tener que seguir revisando **ARISTAs**

Lo que propone Bellman Ford

Que pasa si agrego mas aristas, y me encuentro con otra forma que tiene mas aristas, pero menor costo

Debo añadir las primeras aristas , y determino costos, luego las segundas, luego las terceras

Por lo tanto → No te detengas → Sigue buscando

Viendolo como **PROGRA DINAMICA**



Al final es como en progra dinamica, el optimo tomandolo, o no tomandolo

Al final, llegare al optimo final

Pareciera que podemos descomponer el problema, llegando a la solución final

El peor caso, va a ser el numero de nodos - 1, tendriamos que recorrer

Tenemos una forma de llegar

Si hay un camino mas largo → **IMPLICA CICLOS**

Si hay un camino mas corto → Pueden haber **Negativos**

Al descomponer, tenemos un numero finito de casos posibles

Lineal en el numero de nodos

En terminos de que el camino tendra un numero maximo de nodos

¿Que ocurre con los ciclos? → Que pasa si la ruta optima tiene un ciclo

- No puede tener ciclos la ruta mas corta, porque añadiria mas
- La forma de que ocurra, es que un ciclo negativo que nos reste
- El tema es que siempre estare bajando el costo
- **POR LO TANTO, no tiene ruta mas corta**

- Que grafo no tiene ruta mas corta → Un grafo que tiene **CICLOS NEGATIVOS**

Estrategia Bellman Ford

Añadir aristas, hasta que complete caminos de largo K, y que al ir actualizando costos, darse cuenta que hay un ciclo negativo, cuando inserte la arista K+1, y se movio la solucion, significa que en alguna parte hay un **COSTO NEGATIVO**

Usaremos entre medio varias cosas conceptuales

- Ej: MST tiene n-1 aristas → Lo usaremos para saer que terminamos
- Sabremos que hay un ciclo, cuando nos movamos por el camino, y no encontremos una solucion

Vamos a plantear la estrategia de:

1. Buscar rutas mas cortas desde s con a lo mas 1 arista
2. Luego con 2 aristas
3. Luego con 3 ....

Vamos aprovechar Progra DINAMICA

Usaremos la rutas mas cortas con a lo mas **K-1 Aristas**

Iteraremos hasta el numero de nodos - 1

EJEMPLO

La Gran diferencia con dijkstra, es que la ruta optima, puede ir cambiando a lo largo del recorrido

Encontrando un camino mas corto por otro lado

PSEUDOCODIGO

```
BellmanFord(s):
1     for u ∈ V :
2         d[u] ← ∞;   π[u] ← ∅
3     d[s] ← 0
4     for k = 1 . . . |V| − 1 :
5         for (u, v) ∈ E :
6             if d[v] > d[u] + cost(u, v) :
7                 d[v] ← d[u] + cost(u, v)
8                 π[v] ← u
```

Al revisar $|V| - 1$ veces **todas** las aristas, nos aseguramos de actualizar rutas en caso de encontrar atajos

El algoritmo en si, esta muy condensado

Aparentemente dice que esta recorriendo todas las aristas, no necesariamente desde nuestro punto de partida

Sin embargo, como hay infinitos, d[v] > d[u] + cost → d[u] es infinito, por lo tanto no entro a eso, no sigo estrictamente la forma del camino que me dice el grafo

Simpelemente recorro, y agrego aristas → Las que **FUERON DESCUBIERTAS**

- [ ] Estudiar esta parte

- El paso de darse cuenta que estaba en 0, y con eso parte los que estaban desde el origen, esta implicito en el codigo, no pareciera estar, pareciera que esta malo, es **DIFICIL DE VERLO, estudiar esta parte**
- Se puede hacer eficiente ? → Vale la pena?

A medida que avanza el problema, se **VUELVE MAS EFICIENTE ,** Por algo es DINAMICA, vas pasando solo por aquellos que se actualizan!!!

Hasta el momento, no nos estamos dando cuenta que hay ciclos negativos

La respuesta ya la dijimos, si hicera un ciclo mas aca, y me doy cuenta que hya una actualizacion en alguna de las rutas

Como el camino optimo, si el costo cambia, hay un ciclo negativo

Hay un camino de **MENOR COSTO, y mas LARGO !! → CICLO NEGATIVO**

for k = 1 —- |V| - 1 → Eso lo uncio que hace, es un contador, restringe al numero maximo de aristas

El de abajo, va iterando por las aristas a añadir, y actualizar!

PARA DETECTAR CICLOS

```
ValidBellmanFord(s):
1     for u ∈ V :
2         d[u] ← ∞;   π[u] ← ∅
3     d[s] ← 0
4     for k = 1 . . . |V| − 1 :
5         for (u, v) ∈ E :
6             if d[v] > d[u] + cost(u, v) :
7                 d[v] ← d[u] + cost(u, v)
8                 π[v] ← u
9     for (u, v) ∈ E :
10        if d[v] > d[u] + cost(u, v) :
11            return false
12    return true
```

El resultado de este algoritmo indica si los costos en d son válidos como costos de rutas más cortas

COMPLEJIDAD

Actualización de costos

- $\mathcal{O}(V)$ iteraciones
- Cada una hace $\mathcal{O}(E)$ comparaciones
- Total: $\mathcal{O}(VE)$

Chequeo de ciclos

- $\mathcal{O}(E)$ comparaciones

COMPLEJIDAD: O(V * E)