Análisis de Algoritmos y Estructura de Datos

Algoritmos sobre grafos

Prof. Violeta Chang C

Semestre 2 – 2023

TDA grafo

- **Contenidos**:
  - Algoritmo de recorrido DFS
  - Algoritmo de camino mínimo Dijkstra
  - Algoritmo de flujo máximo Ford-Fulkerson
- **Objetivos**:
  - Comprender funcionamiento algoritmo de recorrido DFS en grafos y aplicarlo para resolver problemas específicos
  - Comprender funcionamiento algoritmo Dijkstra de camino mínimo en grafos y aplicarlo para resolver problemas específicos
  - Comprender funcionamiento algoritmo Ford-Fulkerson de flujo máximo en grafos y aplicarlo para resolver problemas específicos

Ruta de la sesión

Recorrido en profundidad DFS

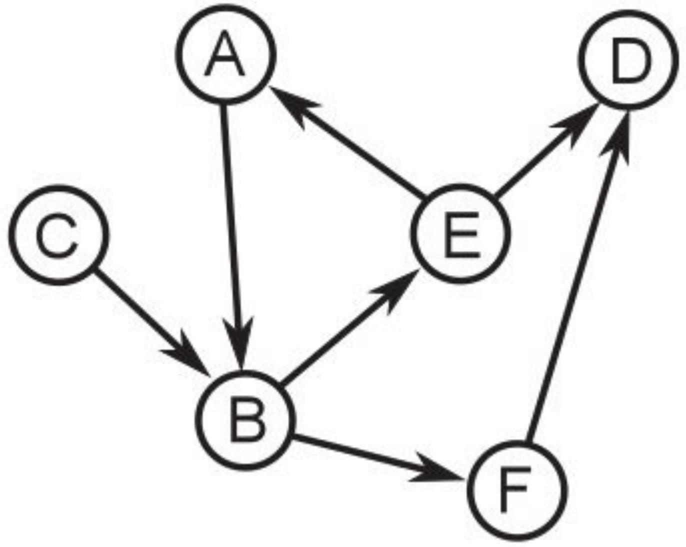
Camino mínimo DIjkstra

Flujo máximo Ford-Fulkerson

Próximas fechas

Terminología de grafos

- Un grafo G se define con:
  - **Conjunto de vértices** V={A,B,C,D,E,F}
  - **Conjunto de aristas** A={(A,B),(B,E),(B,F),(C,B),(E,D),(F,D)}

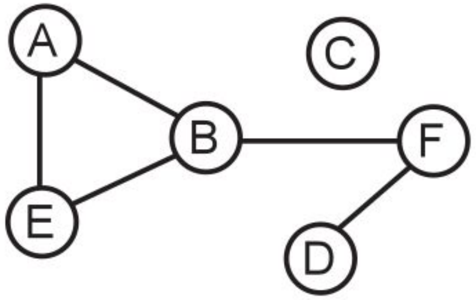


DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Representación de grafos

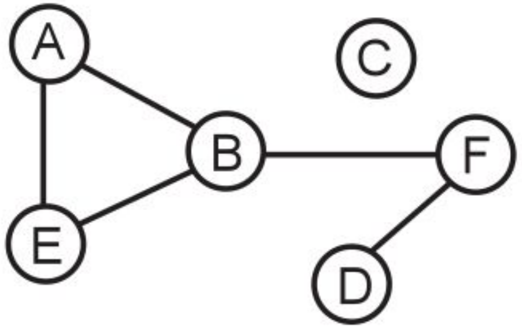
Matriz de adyacencia

Lista de adyacencia



	A	B	C	D	E	F
A	0	1	0	0	1	0
B	1	0	0	0	1	1
C	0	0	0	0	0	0
D	0	0	0	0	0	1
E	1	1	0	0	0	0
F	0	1	0	1	0	0

Matriz de adyacencia de un grafo no dirigido



A → B → E
B → A → E → F
C
D → F
E → A → B
F → B → D

Lista de adyacencia de un grafo no dirigido

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA UNIVERSIDAD DE SANTIAGO DE CHILE

TDA grafo

Definición de TDA grafo

- **Estructura de datos**
  - Un grafo es una colección no lineal de elementos homogéneos que se entiende como un conjunto de vértices y un conjunto de aristas que conectan dichos vértices. Las aristas pueden tener un peso asociado.
  - La estructura de datos que representa un grafo consiste de una de las siguientes alternativas:
    - Número de vértices + listas de adyacencia
    - Número de vértices + matriz de adyacencia

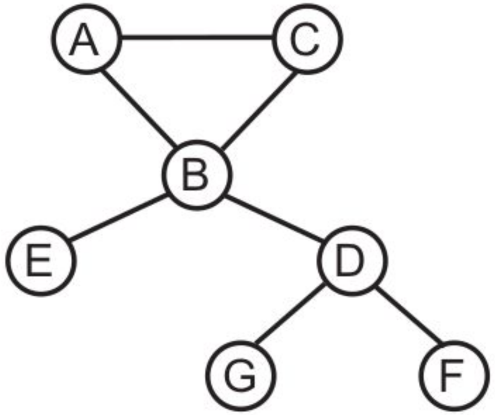
Definición de TDA grafo

- **Operaciones**
  - ***crearGrafo(V,A)***
  - ***agregarVertice(v)***
  - ***agregarArista(v1,v2)***
  - ***eliminarVertice(v)***
  - ***eliminarArista(v1,v2)***
  - **obtenerAdyacentes(v)**
  - **recorrerGrafo(g)**
  - **obtenerCaminoMinimo(g,v1,v2)**
  - **obtenerArbolCoberturaMinimo(g)**
  - **calcularMaximoFlujo(g)**

Recorrido en profundidad (DFS)

- Recorrido en profundidad DFS (Depth First Search)
  - El objetivo es recorrer siempre el camino más largo posible a partir de un vértice inicial.
  - Se comienza visitando un vértice vértice, luego se visita algún vecino y cuando se encuentra a un vecino sin visitar, se sigue un recorrido desde allí. Así hasta que todos los vecinos sean visitados.
  - Una vez que todos los vecinos están visitados, se da por finalizado el recorrido desde el vértice.
  - Se va expandiendo ordenadamente en una dirección todo lo que se puede.

Recorrido en profundidad (DFS)



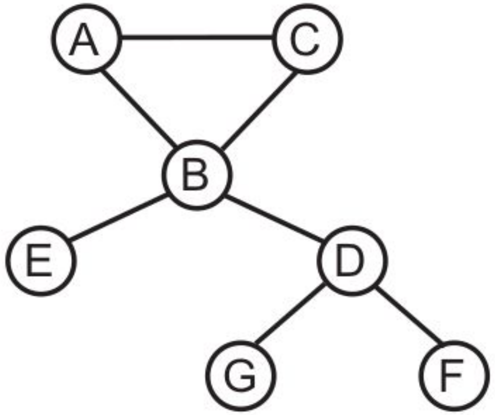
	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	1	1	1	0	0
C	1	1	0	0	0	0	0
D	0	1	0	0	0	1	1
E	0	1	0	0	0	0	0
F	0	0	0	1	0	0	0
G	0	0	0	1	0	0	0

A → B → C
B → A → C → D → E
C → A → B
D → B → F → G
E → B
F → D
G → D

Fondo de la pila →

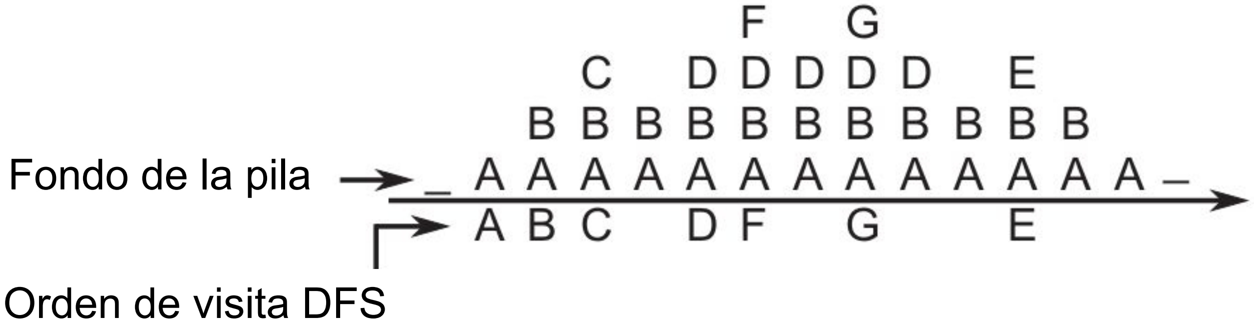
Orden de visita DFS

Recorrido en profundidad (DFS)



	A	B	C	D	E	F	G
A	0	1	1	0	0	0	0
B	1	0	1	1	1	0	0
C	1	1	0	0	0	0	0
D	0	1	0	0	0	1	1
E	0	1	0	0	0	0	0
F	0	0	0	1	0	0	0
G	0	0	0	1	0	0	0

A → B → C
B → A → C → D → E
C → A → B
D → B → F → G
E → B
F → D
G → D



Recorrido en profundidad (DFS)

```
DFS(grafo g, vertice inicio)
   pila←crearPilaVacia(calcularLargo(G→V))
   apilar(pila,inicio)
   marcarVisitado(inicio)
   mientras no(esPilaVacia(pila)) hacer
      topePila←tope(pila)→dato
      adyacentes←obtenerAdyacentes(G,topePila)
      w←adyacenteNoVisitado(adyacentes)
      si existe(w) entonces
         apilar(pila,w)
         marcarVisitado(w)
      sino
         desapilar(pilar)
```

Recorrido en profundidad (DFS)

```
DFS(grafo g, vertice inicio)
   pila←crearPilaVacia(calcularLargo(G→V))
   apilar(pila,inicio)
   marcarVisitado(inicio)
   mientras no(esPilaVacia(pila)) hacer
      topePila←tope(pila)→dato
      adyacentes←obtenerAdyacentes(G,topePila)
      w←adyacenteNoVisitado(adyacentes)
      si existeNoVisitado(w) entonces
         apilar(pila,w)
         marcarVisitado(w)
      sino
         desapilar(pilar)
```

Recorrido en profundidad (DFS)

```
DFS(grafo g, vertice inicio)
   pila←crearPilaVacia(calcularLargo(G→V))
   apilar(pila,inicio)
   marcarVisitado(inicio)
   mientras no(esPilaVacia(pila)) hacer
      topePila←tope(pila)→dato
      adyacentes←obtenerAdyacentes(G,topePila)
      w←adyacenteNoVisitado(adyacentes)
      si existe(w) entonces
         apilar(pila,w)
         marcarVisitado(w)
      sino
         desapilar(pilar)
```

Complejidad: $O(n^2)$

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

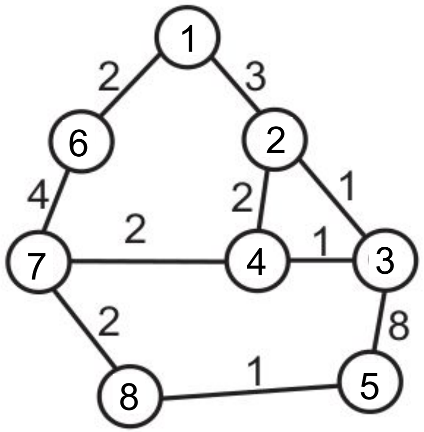
Recorridos: aplicaciones

- Recorrer un grafo
- Determinar árboles de cobertura
- Obtener información estructural del grafo:
  - Determinar las distancias de un vértice a los demás vértices de un grafo
  - Determinar si un grafo es o no es conexo
  - Identificar las componentes conexas de un grafo
  - Determinar si un grafo tiene o no tiene ciclos
  - Identificar los puntos de articulación de un grafo

Camino mínimo: Dijkstra

- El algoritmo fue descrito por Edsger Dijkstra en 1959.
- Determina el **camino más corto** entre un **vértice origen** y los demás vértices en un **grafo ponderado**.

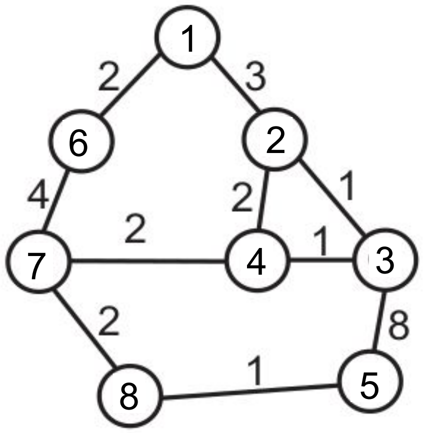
Camino mínimo: Dijkstra



	1	2	3	4	5	6	7	8
anterior	0	0	0	0	0	0	0	0
visitado	0	0	0	0	0	0	0	0
distancia	∞	∞	∞	∞	∞	∞	∞	∞

inicio= 1
n=8

Camino mínimo: Dijkstra



	1	2	3	4	5	6	7	8
anterior	0	1	2	2	8	1	6	7
visitado	1	1	1	1	1	1	1	1
distancia	0	3	4	5	9	2	6	8

inicio= 1
n=8

Camino mínimo: Dijkstra

```
Dijkstra(grafo G, vertice inicio): arreglo, arreglo
    n←calcularLargoArreglo(G→V)
    para i←1 hasta n
        anterior(i)←0
        si G→A(inicio,i)>0 entonces
            distancia(i)←G→W(inicio,i)
            anterior(i)←inicio
        sino
            distancia(i)←∞
    distancia(inicio)←0
    marcarVisitado(inicio)
    mientras existenVerticesSinVisitar hacer
        u←noVisitadoDistanciaMinima(distancia,g)
        marcarVisitado(u)
        adyacentes←obtenerAdyacentes(G,u)
        mientras adyacentes<>NULO hacer
            v←adyacentes→dato
            si distancia(v)>distancia(u)+G→W(u,v) entonces
                distancia(v)←distancia(u)+G→W(u,v)
                anterior(v)←u
            adyacentes←adyacentes→puntero
    devolver(distancia,anterior)
```

Camino mínimo: Dijkstra

```
Dijkstra(grafo G, vertice inicio): arreglo, arreglo
    n←calcularLargoArreglo(G→V)
    para i←1 hasta n
        anterior(i)←0
        si G→A(inicio,i)>0 entonces
            distancia(i)←G→W(inicio,i)
            anterior(i)←inicio
        sino
            distancia(i)←∞
    distancia(inicio)←0
    marcarVisitado(inicio)
    mientras existenVerticesSinVisitar hacer
        u←noVisitadoDistanciaMinima(distancia,g)
        marcarVisitado(u)
        adyacentes←obtenerAdyacentes(G,u)
        mientras adyacentes<>NULO hacer
            v←adyacentes→dato
            si distancia(v)>distancia(u)+G→W(u,v) entonces
                distancia(v)←distancia(u)+G→W(u,v)
                anterior(v)←u
            adyacentes←adyacentes→puntero
devolver(distancia,anterior)
```

Camino mínimo: Dijkstra

```
Dijkstra(grafo G, vertice inicio): arreglo, arreglo
    n←calcularLargoArreglo(G→V)
    para i←1 hasta n
        anterior(i)←0
        si G→A(inicio,i)>0 entonces
            distancia(i)←G→W(inicio,i)
            anterior(i)←inicio
        sino
            distancia(i)←∞
    distancia(inicio)←0
    marcarVisitado(inicio)
    mientras existenVerticesSinVisitar hacer
        u←noVisitadoDistanciaMinima(distancia,g)
        marcarVisitado(u)
        adyacentes←obtenerAdyacentes(G,u)
        mientras adyacentes<>NULO hacer
            v←adyacentes→dato
            si distancia(v)>distancia(u)+G→W(u,v) entonces
                distancia(v)←distancia(u)+G→W(u,v)
                anterior(v)←u
            adyacentes←adyacentes→puntero
devolver(distancia,anterior)
```

Complejidad: $O(n^2)$

Camino mínimo: aplicaciones

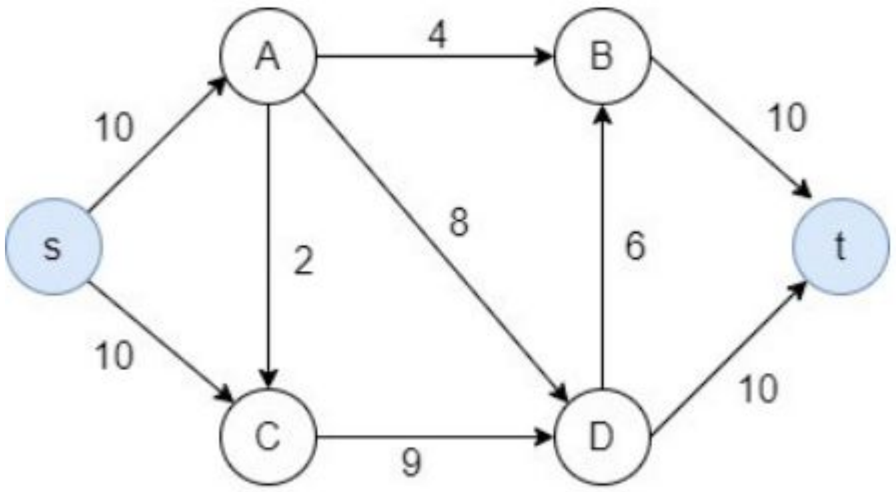
- Los algoritmos de los caminos más cortos se aplican para encontrar direcciones de forma automática entre lugares físicos, como las rutas de conducción en sitios de mapas web.
- Si un algoritmo se aplica en un grafo, donde los vértices describen estados, y las aristas posibles transiciones, el algoritmo del camino más cortos puede ser usado para encontrar una secuencia óptima de decisiones para llegar a un cierto estado final. Ej: Akinator, Cubo Rubik.

Flujo en redes: Ford-Fulkerson

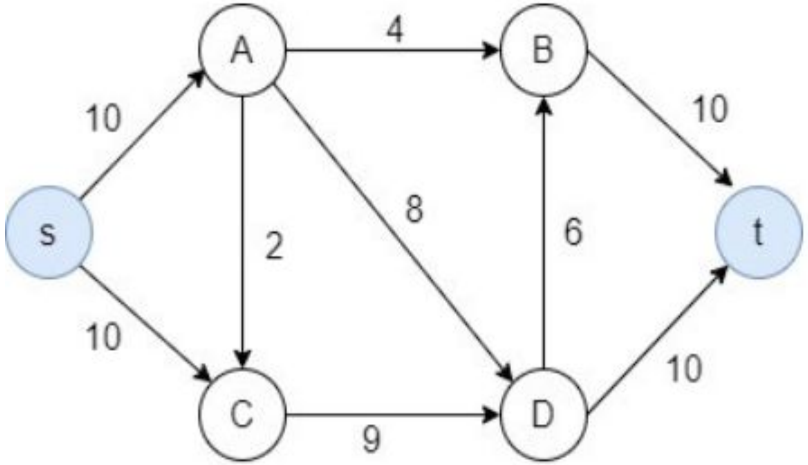
- Una red (grafo ponderado dirigido) tiene dos vértices particulares únicos: **fuente** (sale flujo) y **sumidero** (llega flujo). El peso de cada arista corresponde a su **capacidad**.
- Un **flujo** es una función que asigna a cada arista un valor entre 0 y su capacidad, señalando cuánto ha sido ocupado de la capacidad de dicha arista → **0 ≤ flujo ≤ capacidad**
- Los flujos deben respetar la **ley de conservación**, la cual establece que, para cada vértice excepto la fuente y el sumidero, el flujo que entra es igual que el que sale. El valor del flujo de una red corresponde a lo que entra al sumidero, y debe ser igual a lo que sale de la fuente.
- Se busca determinar el flujo máximo que puede pasar por la red.
- Uno de los algoritmos clásicos para resolver el problema de flujo en redes es el de Ford-Fulkerson.

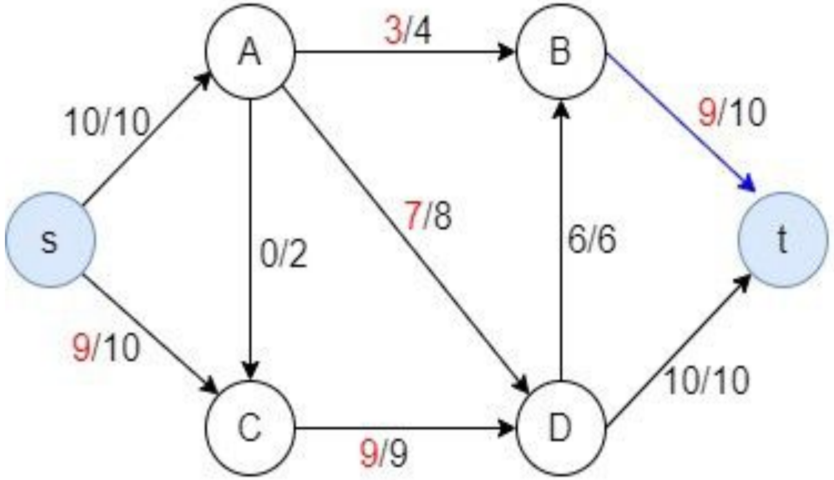
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Flujo en redes: Ford-Fulkerson



Flujo en redes: Ford-Fulkerson





Flujo máximo = 10 + 9 = 19

Flujo en redes: Ford-Fulkerson

```
FordFulkerson(grafo G, vertice s, vertice t):num
   GR→V ← G→V
   n←calcularLargo(GR→V)
   para u←1 hasta n
      para v←1 hasta n
         GR→A(u,v)→capacidad ← G→A(u,v)
         GR→A(u,v)→flujo ← 0
   flujoMaximo ← 0
   camino ← encontrarCamino(GR,s,t)
   mientras no(esListaVacia(camino)) hacer
      flujoCamino ← obtenerCapacidadMinima(GR,camino)
      mientras camino<>NULO y camino→puntero<>NULO hacer
         u ← camino→dato
         v ← camino→puntero→dato
         GR→A(u,v)→flujo ← GR→A(u,v)→flujo + flujoCamino
         GR→A(v,u)→flujo ← GR→A(v,u)→flujo - flujoCamino
         camino ← camino→puntero
      flujoMaximo ← flujoMaximo + flujoCamino
      camino ← encontrarCamino(GR,s,t)
   devolver(flujoMaximo)
```

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Flujo en redes: Ford-Fulkerson

```
FordFulkerson(grafo G, vertice s, vertice t):num
    GR→V ← G→V
    n←calcularLargo(GR→V)
    para u←1 hasta n
        para v←1 hasta n
            GR→A(u,v)→capacidad ← G→A(u,v)
            GR→A(u,v)→flujo ← 0
    flujoMaximo ← 0
    camino ← encontrarCamino(GR,s,t)
    mientras no(esListaVacia(camino)) hacer
        flujoCamino ← obtenerCapacidadMinima(GR,camino)
        mientras camino<>NULO y camino→puntero<>NULO hacer
            u ← camino→dato
            v ← camino→puntero→dato
            GR→A(u,v)→flujo ← GR→A(u,v)→flujo + flujoCamino
            GR→A(v,u)→flujo ← GR→A(v,u)→flujo - flujoCamino
            camino ← camino→puntero
        flujoMaximo ← flujoMaximo + flujoCamino
        camino ← encontrarCamino(GR,s,t)
    devolver(flujoMaximo)
```

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA UNIVERSIDAD DE SANTIAGO DE CHILE

Flujo Máximo en redes: Aplicaciones

- El estudio de algoritmos para resolver este problema teórico es de suma importancia, ya que sirve para modelar una gran variedad de problemas de la vida real, entre ellos:
  - Transporte de mercadería (logística).
  - Flujo de gases y líquidos por tuberías.
  - Flujo de componentes o piezas en líneas de montaje.
  - Flujo de corriente en redes eléctricas.
  - Flujo de paquetes de información en redes de comunicaciones.

DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Actividad de cierre

- Ir a menti.com e ingresar código 6122 4706

Próximas fechas...

- Resumen de la semana:
  - Algoritmo de recorrido DFS
  - Algoritmo de camino mínimo Dijkstra
  - Algoritmo de flujo máximo Ford-Fulkerson

  ~~cátedra~~ – refuerzo – laboratorio

- Próxima semana:
  - **PEP2**
  - **Entrega de Tarea 2**
  - Algoritmo de árboles de cobertura mínima (Prim y Kruskal)