Equipo 4

# PROGRAMACIÓN DEL ALGORITMO PERT

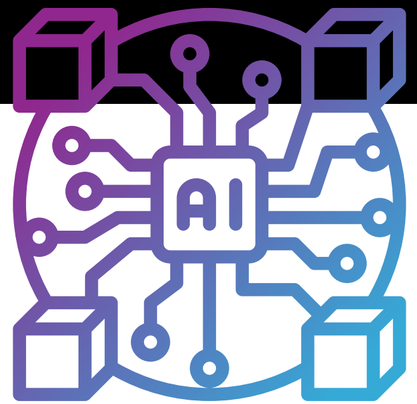
Hernández Soc Tomás David
Limón Cruz Andrés

# Objetivo

El objetivo de este proyecto es programar el algoritmo PERT, para ello, usaremos el lenguaje de programación python.

Por supuesto, ademas de programarlo queremos que funcione óptimamente, para esto vamos a probarlo en ejercicios resueltos en el pasado y asi cotejar los resultados.

# Algoritmo PERT




El algoritmo PERT, por sus siglas en ingles, es un algoritmo de gestión de proyectos utilizado para gestionar tareas involucradas en un proyecto utilizando una representación gráfica de las actividades de donde se obtiene la duración del proyecto y las actividades criticas. Los pasos a seguir son:

1. Creamos la red de actividades con la dependencia y duración de cada una.
2. Calculamos el calendario de fechas más cercanas y mas lejanas, que son las fechas en la que se espera que una actividad comience y termine lo más temprano posible, considerando las actividades predecesoras.
3. Cálculo de holguras de tiempo para cada actividad, que representan el margen disponible antes de que la actividad se convierta en crítica.
4. Identificación de la ruta crítica, la cual es la secuencia de actividades criticas (con holgura igual a cero) y determina la duración mínima del proyecto.

# Ejemplos

En el código probamos varios ejemplos, para esta presentación nos enfocaremos en uno, el cual extrajimos de la tarea 9.

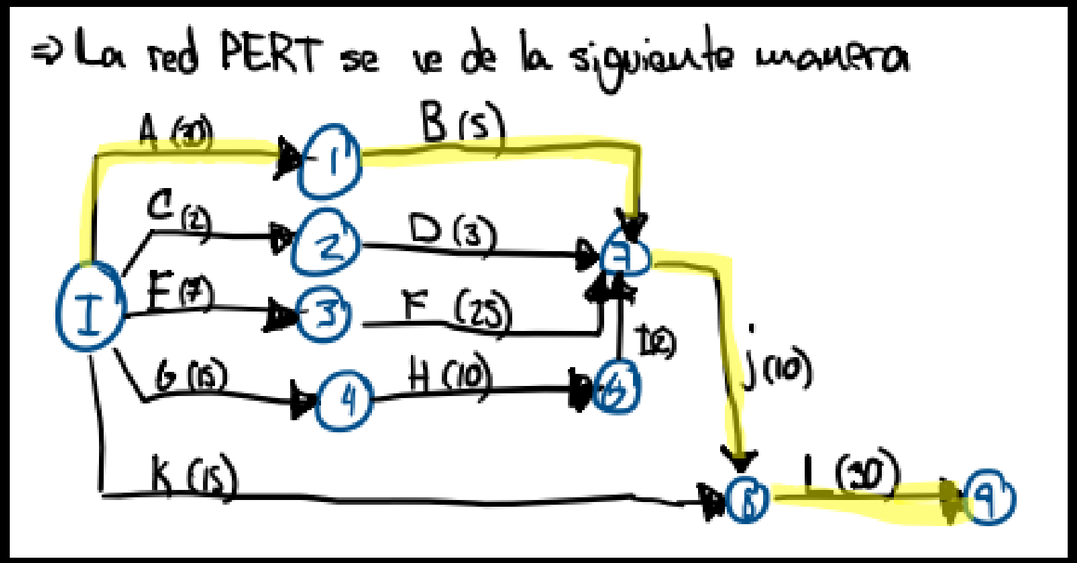
**2. Ustedes y varios amigos van a preparar una lasagna para cenar. Las tareas a realizar, sus predecesores inmediatos y sus duraciones estimadas son como sigue:**

| Actividad | Descripción | Precedente | Duración |
|---|---|---|---|
| A | Comprar los ingredientes | — | 30 |
| B | Rayas el queso mozzarella | A | 5 |
| C | Batir dos huevos | — | 2 |
| D | Mezclar los huevos con el queso ricotta | C | 3 |
| E | Picar cebolla y champinones | — | 7 |
| F | Cocer la salsa de tomate | E | 25 |
| G | Hervir agua | — | 15 |
| H | Hervir la pasta de lasagna | G | 10 |
| I | Escurrir la pasta de lasagna | H | 2 |
| J | Juntar todos los ingredientes | I,F,D,B | 10 |
| K | Precalentar el horno | — | 15 |
| L | Meter la lasagna al horno | J,K | 30 |

# Resultados

**De la tarea 9 sabemos los resultados del problema, vamos a plasmarlos para tenerlos presentes y cotejarlos con los resultados del código.**

## ¿Cómo es la red PERT?




## ¿Cuales son las fechas mas próximas?

[a(1): 0, a(2): 30, a(3): 2, a(4): 7, a(5): 15, a(6): 25, a(7): 35, a(8): 45, a(9): 75]

## ¿Cuales son las fechas mas lejanas?

[b(1): 0, b(2): 30, b(3): 32, b(4): 10, b(5): 23, b(6): 33, b(7): 35, b(8): 45, b(9): 75]

## ¿Cuales son las holguras?

[h(A): 0, h(C): 30, h(F): 3, h(G): 8, h(K): 30, h(B): 0, h(D): 30, h(F): 3, h(H): 8, h(L): 0, h(J): 0, h(I): 8]

## ¿Cuál es la ruta critica?

[('I', 1), (1, 7), (8, 'F'), (7, 8)]

# Codigo del ejemplo

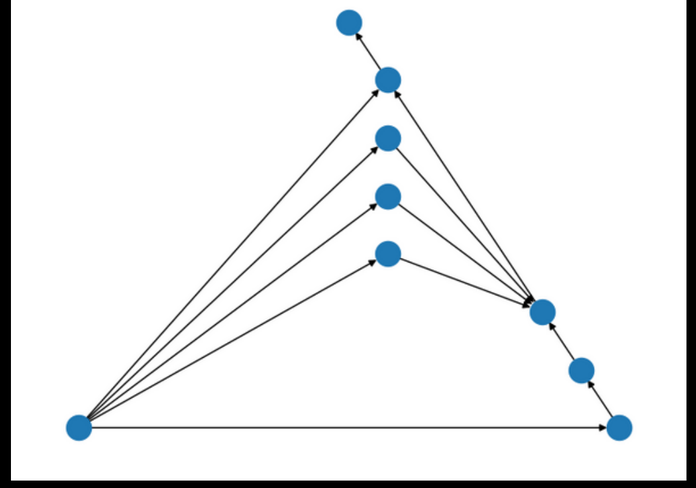
**Formamos la red PERT con los datos de la tabla apoyándonos con la librería networkx, y la usamos para correr la función PERT.**

```
Pert2 = nx.DiGraph()
edges = [("I",1),('I',2),('I',3),('I',4),('I',8),(1,7),(2,7),(3,7),(4,6),(6,7),(7,8),(8,"F")]
Pert2.add_edges_from(edges)
weight = [("I",1,30),('I',2,2),('I',3,7),('I',4,15),('I',8,15),(1,7,5),(2,7,3),(3,7,25),(4,6,10),(6,7,2),(7,8,10),(8,"F",30)]
Pert2.add_weighted_edges_from(weight)

PERT(Pert2,'I','F')
```

**Los resultados del programa, junto a la red PERT, nos arroja lo siguiente:**

```
Calendario de fechas mas próximas: {'I': 0, 1: 3, 2: 18, 3: 28, 4: 30, 5: 38, 6: 40, 7:
40, 8: 38, 10: 50, 9: 60, 12: 80, 13: 83, 11: 83, 15: 98, 14: 108, 16: 116, 'F': 124}
Calendario de fechas mas lejanas: {'F': 124, 16: 116, 14: 108, 15: 98, 11: 83, 13: 83,
7: 75, 8: 77, 12: 80, 9: 60, 5: 40, 10: 50, 6: 40, 4: 30, 3: 28, 2: 18, 1: 3, 'I': 0}
Holguras: {('I', 1): 0, (1, 2): 0, (2, 3): 0, (3, 4): 0, (4, 5): 2, (4, 6): 0, (4, 7):
35, (4, 8): 39, (4, 15): 63, (5, 9): 2, (6, 10): 0, (7, 11): 35, (8, 11): 39, (15, 14):
0, (9, 12): 0, (10, 9): 0, (11, 15): 0, (14, 16): 0, (12, 13): 0, (13, 11): 0, (13, 14)
: 17, (16, 'F'): 0}
Ruta critica: [('I', 1), (1, 2), (2, 3), (3, 4), (4, 6), (6, 10), (15, 14), (9, 12), (1
0, 9), (11, 15), (14, 16), (12, 13), (13, 11), (16, 'F')]
```



**Si observamos y comparamos a los resultados obtenidos haciendo el algoritmo a mano vemos que todo coincide, asi que el programa funciona correctamente**

# Conclusion

## ¿Qué aprendimos?

El algoritmo de PERT nos ayuda a organizar un calendario optimo para una lista de actividades, con este proyecto logramos automatizar el algoritmo en python, por otro lado aprendimos a usar la librería networkx para hacer redes.
Además lo aplicamos a diversos ejemplos que nos ayudaron a comprobar que el código funciona.
Este proyecto nos ayudo mucho a aplicar los conocimientos obtenidos en el curso en programación, lo que nos dio una vision mas amplia de el algoritmo y el alcance de la investigación de operaciones.

# Referencias


- Hernandez Ayuso, M. (2017).Introducción a la programación lineal. Facultad de ciencias UNAM
- Lopez Claudia, "Investigación de operaciones". UNAM, CDMX.
- https://colab.research.google.com/drive/18zTRFGQ0BaaqEH8zpvfoa86bB-NmtH6w#scrollTo=g2skTKJlhXjC

# Código

Para el código del algoritmo PERT usamos un total de 6 funciones:

- **sucesores_lista_s():**

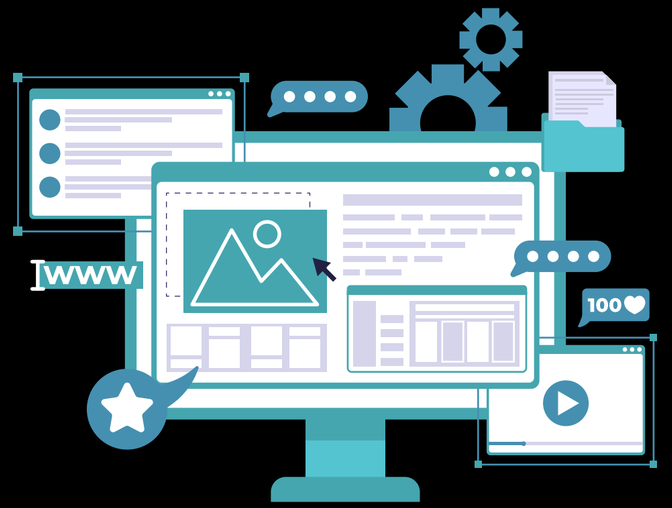
Esta función recibe la red PERT que debe de ser de la forma networkx.DiGraph() y un valor S y devuelve una lista con los sucesores.

```python
def sucesores_lista_S(Pert,S):
    sucesores = []
    for i in range(len(S)):
        for j in Pert.successors(S[i]):
            if j not in sucesores:
                sucesores.append(j)
    return sucesores
```

- **predecesores_lista_sucesores():**

Esta función recibe la red PERT que debe de ser de la forma networkx.DiGraph() y una lista con los sucesores y devuelve un diccionario con todos los predecesores.

```python
def predecesores_lista_sucesores(Pert,sucesores):
    diccionario_de_predecesores = {}
    for i in range(len(sucesores)):
        predecesores_de_cada_sucesor = []
        for j in Pert.predecessors(sucesores[i]):
            predecesores_de_cada_sucesor.append(j)
        diccionario_de_predecesores[sucesores[i]] = predecesores_de_cada_sucesor
    return diccionario_de_predecesores
```



# Código

- **valor_a()**

Esta función recibe la red PERT que debe de ser de la forma networkx.DiGraph(), un diccionario con los predecesores, un valor numérico que indica el indice en una lista, un diccionario con las actividades, y devuelve el valor de la fecha mas próxima.

```
def valor_a(Pert,predecesores_dic,indice,diccionario_a):
    maximo = []
    for i in range(len(predecesores_dic[indice])):
        maximo.append(diccionario_a[predecesores_dic[indice][i]] +
        Pert[predecesores_dic[indice][i]][indice]['weight'])
    return max(maximo)
```

- **valor_b()**

Esta función recibe la red PERT que debe de ser de la forma networkx.DiGraph(), un diccionario con las actividades, un valor numérico que indica el indice en una lista,, la lista con los sucesores y devuelve el valor de la fecha mas lejana.

```
def valor_b(Pert,diccionario_b,indice,sucesores):
    minimo = []
    for i in range(len(sucesores)):
        minimo.append(diccionario_b[sucesores[i]] -
        Pert[indice][sucesores[i]]['weight'])
    return min(minimo)
```

- **holguras()**

Esta función recibe la red PERT que debe de ser de la forma networkx.DiGraph(), un diccionario con las fechas mas cercanas, un diccionario con las fechas mas lejanas, y devuelve un diccionario con las holguras.

```
def holguras(Pert,diccionario_a,diccionario_b):
    arcos = list(Pert.edges)
    holguras = {}
    for i in range(len(arcos)):
            holguras[(arcos[i][0],arcos[i][1])] =
            diccionario_b[arcos[i][1]] - diccionario_a[arcos[i][0]] -
            Pert[arcos[i][0]][arcos[i][1]]['weight']
    return holguras
```

- **PERT()**

En esta función se usan todos los anteriores para hacer el algoritmo, recibe la red PERT que debe de ser de la forma networkx.DiGraph(), un nodo inicial y un nodo final y devuelve con la función print los returns en conjunto de varios de los algoritmos anteriores, es decir, devuelve el calendario de las fechas mas próximas, mas lejanas, las holguras y a su vez una ruta critica junto a la gráfica formada con ayuda de la librería networkx.

```
def PERT(Pert,I,F):
    S = [I]
    a = {I : 0}
    while len(S) != len(Pert.nodes):
        sucesores = sucesores_lista_S(Pert,S)
        predecesores = predecesores_lista_sucesores(Pert,sucesores)
        for i in range(len(sucesores)):
            if sucesores[i] not in S:
                if set(predecesores[sucesores[i]]) <= set(S):
                    a[sucesores[i]] = valor_a(Pert,predecesores,sucesores[i],a)
                    S.append(sucesores[i])


    S = [F]
    b = {F : a[F]}
    while len(S) != len(Pert.nodes):
        predecesores = predecesores_lista_sucesores(Pert,S)
        for i in predecesores:
            sucesores = predecesores[i]
            for j in sucesores:
                if j not in S:
                    if set(sucesores_lista_S(Pert,[j])) <= set(S):
                        b[j] = valor_b(Pert,b,j,sucesores_lista_S(Pert,[j]))
                        S.append(j)

    Holguras = holguras(Pert,a,b)
    Ruta_critica = []
    for i in Holguras:
        if Holguras[i] == 0:
            Ruta_critica.append(i)


    print(f'Calendario de fechas mas próximas: {a}\nCalendario de fechas mas lejanas:\
    {b}\nHolguras:{Holguras}\nRuta critica: {Ruta_critica}') #Mostramos el resultado
    nx.draw_planar(Pert)
```

# GRACIAS POR SU ATENCION

"Si ser expertos en investigación de operaciones es un delito, nos declaramos culpables"