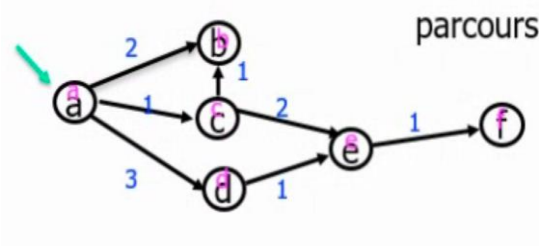
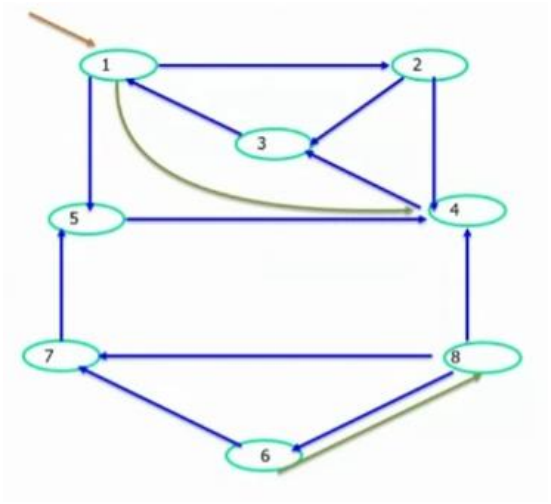
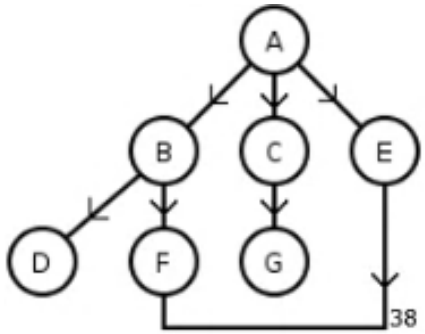
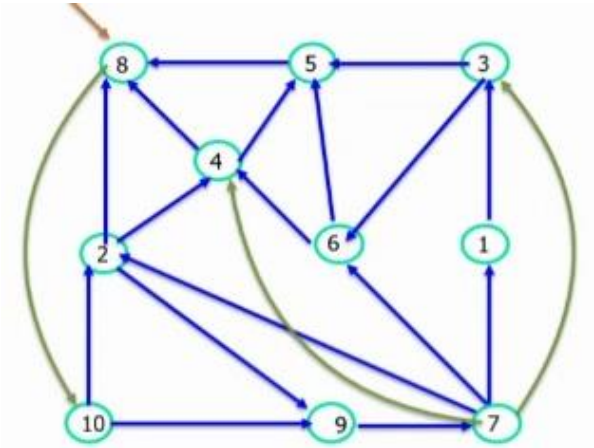
Parcours en profondeur DFS et en largeur BFS



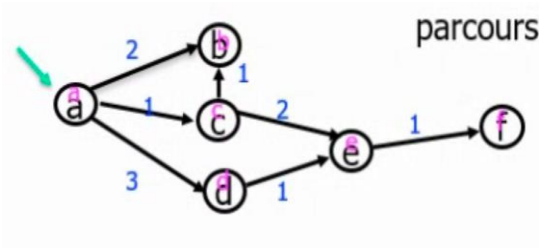




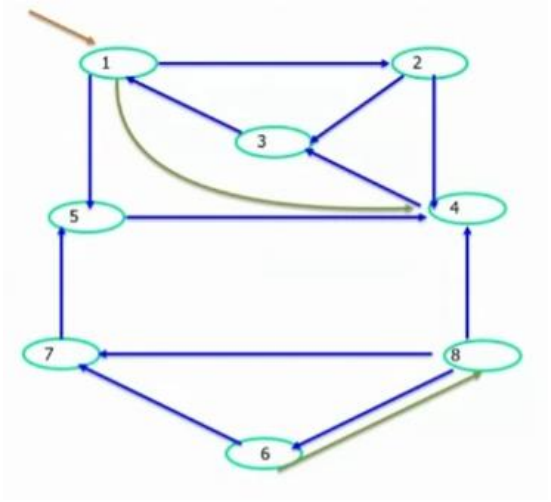


DFS Croissant : b-f-e-c-d-a	DFS Croissant : 3-4-2-5-1-7-8-6	DFS Croissant : D-F-B-G-C-E-A	DFS Croissant : 5-4-6-3-1-7-9-2-10-8
DFS Décroissant : f-e-d-b-c-a	DFS Décroissant : 3-4-5-2-1-7-6-8	DFS Décroissant : F-E-G-C-D-B-A	DFS Décroissant : 5-4-6-3-2-1-7-9-10-8
BFS Croissant : a-b-c-d-e-f	BFS Croissant : 1-2-4-5-3-6-7-8	BFS Croissant : A-B-C-E-D-F-G	BFS Croissant : 8-10-2-9-4-7-5-1-3-6
BFS Décroissant : a-d-c-b-e-f	BFS Décroissant : 1-5-4-2-3-8-7-6	BFS Décroissant : A-E-C-B-F-G-D	BFS Décroissant : 8-10-9-2-7-4-6-3-1-5

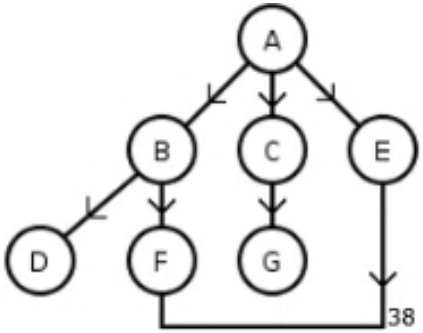
Déclaration des graphes en Python



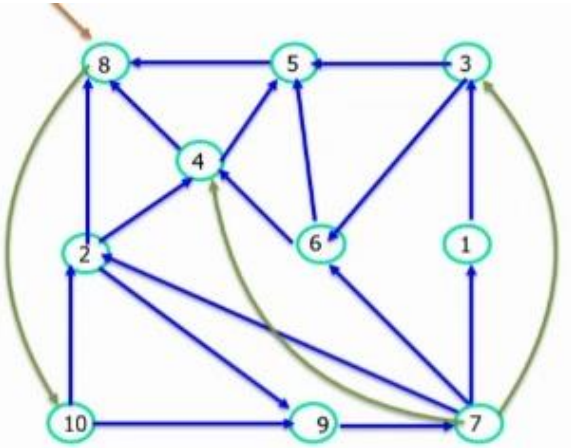

```
graphe = {
   'a': ['b', 'c', 'd'],
   'b': [],
   'c': ['b', 'e'],
   'd': ['e'],
   'e': ['f'],
   'f': []
}
```




```
graphe = {
   1: [5, 2, 4],
   2: [3, 4],
   3: [1],
   4: [3],
   5: [4],
   6: [7, 8],
   7: [5],
   8: [4, 6, 7]
}
```

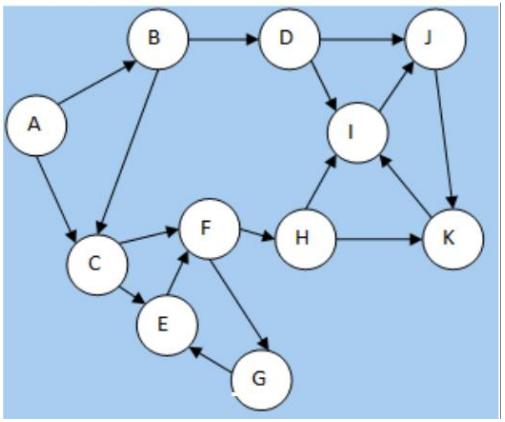


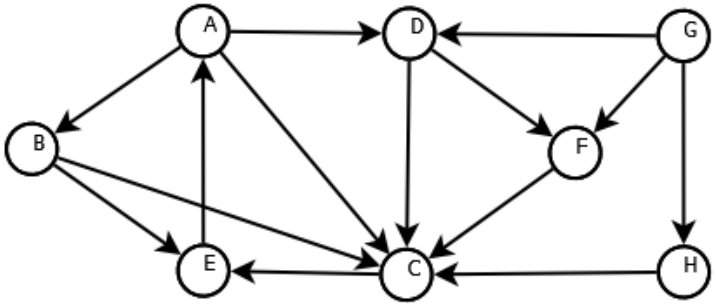

```
graphe = {
   'A': ['B', 'C', 'E'],
   'B': ['D', 'F'],
   'C': ['G'],
   'D': [],
   'E': ['F'],
   'F': [],
   'G': []
}
```

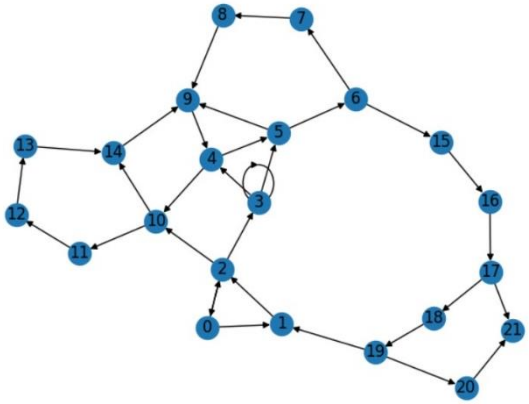


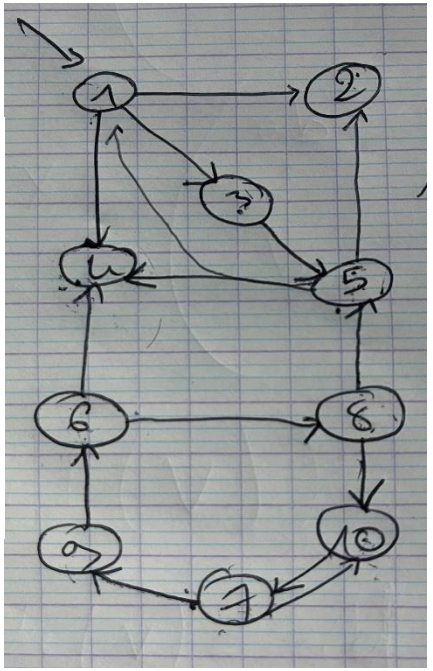

```
graphe = {
   1: [3],
   2: [4, 8, 9],
   3: [5, 6],
   4: [5, 8],
   5: [8],
   6: [4, 5],
   7: [1, 2, 3, 4, 6],
   8: [10],
   9: [7],
   10: [2, 9]
}
```

Parcours en profondeur DFS et en largeur BFS









DFS Croissant : G-K-J-I-H-F-E-C-D-B-A

DFS Décroissant : J-I-K-H-E-G-F-C-D-B-A

BFS Croissant : A-B-C-D-E-F-I-J-G-H-K

BFS Décroissant : A-C-B-F-E-D-H-G-J-I-K

DFS croissant : E-C-B-F-D-A-H-G

DFS décroissant : E-C-F-D-B-A-H-G

BFS croissant : A-B-C-D-E-F-G-H

BFS décroissant : A-D-C-B-F-E-H-G

DFS croissant :

0-9-8-7-21-20-19-18-17-16-15-6-5-14-13-12-11-10-4-3-2-1

DFS décroissant :

21-20-19-18-17-16-15-8-7-6-5-4-9-14-13-12-11-10-3-0-2-1

BFS croissant :

1-2-0-3-10-4-5-11-14-6-9-12-7-15-13-8-16-17-18-21-19-20

BFS décroissant :

1-2-10-3-0-14-11-5-4-9-12-6-13-15-7-16-8-17-21-18-19-20

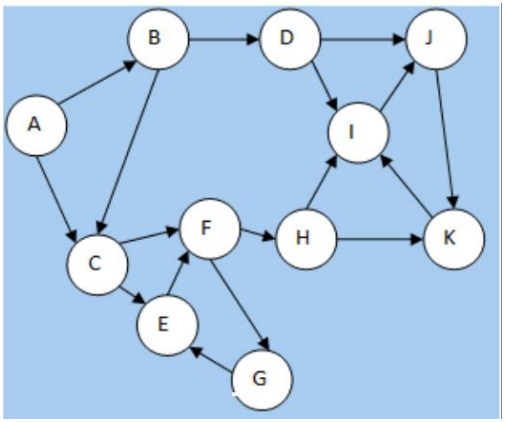
DFS Croissant : 2-4-5-3-1-9-7-10-8-6

DFS Décroissant : 4-2-5-3-1-8-6-9-7-10

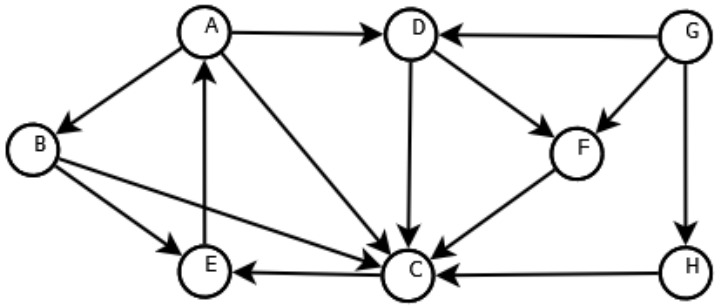
BFS Croissant : 1-2-3-4-5-6-8-10-7-9

BFS Décroissant : 1-4-3-2-5-10-7-9-6-8

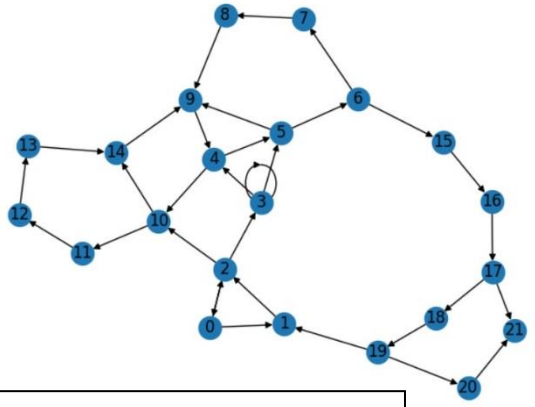
Déclaration des graphes en Python



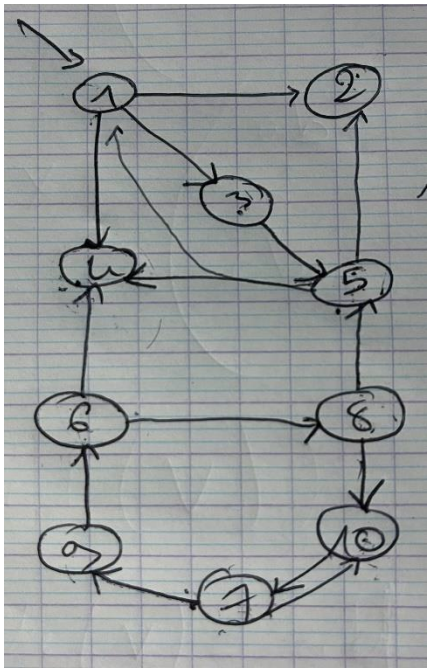

```
graphe = {
    'A': ['B', 'C'],
    'B': ['C', 'D'],
    'C': ['E', 'F'],
    'D': ['I', 'J'],
    'E': ['F'],
    'F': ['H', 'G'],
    'G': ['E'],
    'H': ['I', 'K'],
    'I': ['J'],
    'J': ['K'],
    'K': ['I']
}
```




```
graphe = {
    'A': ['B', 'C', 'D'],
    'B': ['E', 'C'],
    'C': ['E'],
    'D': ['F', 'C'],
    'E': ['A'],
    'F': ['C'],
    'G': ['D', 'F', 'H'],
    'H': ['C']
}
```




```
graphe = {
    1: [2],
    0: [1, 2],
    2: [0, 3, 10],
    3: [3, 4, 5],
    4: [5, 10],
    5: [6, 9],
    6: [7, 15],
    7: [8],
    8: [9],
    9: [4],
    10: [11, 14],
    11: [12],
    12: [13],
    13: [14],
    14: [9],
    15: [16]
```



```
graphe = {
    1: [2, 3, 4],
    2: [],
    3: [5],
    4: [],
    5: [1, 2, 4],
    6: [4, 8],
    7: [9, 10],
    8: [5, 10] ,
    9: [6] ,
    10: [7]
}
```

Mehdi RHOULAM

Parcours DFS & BFS Mme. ELOMRI

Yahia TOUHAMI EL ALAMI

Code des parcours DFS & BFS en Python

Parcours en profondeur DFS

```python
# Fonction pour gérer toutes les composantes connexes avec un nœud de départ
def dfs_global(graph, start_node=None, order='ascending'):
    visited = set()
    result = []
    all_nodes = sorted(graph.keys(), reverse=(order == 'descending'))  # Trier les nœuds tout en respectant l'ordre demandé
    sort_function = sorted if order == 'ascending' else lambda x: sorted(x, reverse=True)  # Choisir l'ordre de tri pour les voisins
    current_node = start_node if start_node is not None else all_nodes[0]  # Si un nœud de départ est spécifié, commencer par celui-ci
    while len(visited) < len(graph):  # Répéter jusqu'à ce que tous les nœuds soient visités
        if current_node not in visited:
            dfs_component(graph, current_node, visited, result, sort_function)
        # Trouver le prochain nœud non visité
        for node in all_nodes:
            if node not in visited:
                current_node = node
                break
    return result

# Fonction DFS pour une composante donnée
def dfs_component(graph, node, visited, result, sort_function):
    if node not in visited:
        visited.add(node)
        for neighbor in sort_function(graph[node]):
            if neighbor not in visited:
                dfs_component(graph, neighbor, visited, result, sort_function)
        result.append(node)
```

Parcours en largeur BFS

```python
def bfs_global(graph, start_node=None, order="asc"):
    visited = set()
    result = []
    all_nodes = sorted(graph.keys(), reverse=(order == "desc"))  # Trier les nœuds tout en respectant l'ordre demandé
    queue = deque([start_node] if start_node else [all_nodes[0]])  # Démarrer par le nœud donné ou le premier nœud déclaré
    while len(visited) < len(graph):  # Répéter jusqu'à ce que tous les nœuds soient visités
        while queue:
            node = queue.popleft()
            if node not in visited:
                visited.add(node)
                result.append(node)
                # Ajouter les voisins triés dans la file d'attente
                neighbors = sorted(graph.get(node, []), reverse=(order == "desc"))
                for neighbor in neighbors:
                    if neighbor not in visited:
                        queue.append(neighbor)
        # Trouver le prochain nœud non visité et le rajouter dans la file
        for node in all_nodes:
            if node not in visited:
                queue.append(node)
                break
    return result
```

Demande et conversion du nœud de départ en Python

```python
# Fonction pour valider et convertir le nœud de départ
def get_start_node(graph, input_node):
    try:
        # Essayer de convertir en entier si possible
        input_node = int(input_node) if input_node.isdigit() else input_node
        if input_node in graph:
            return input_node
    except ValueError:
        pass
    print("Le nœud entré n'est pas valide. Un nœud par défaut sera utilisé.")
    return list(graph.keys())[0]  # Utiliser le premier nœud par défaut

# Demander à l'utilisateur de choisir un nœud de départ
user_input = input("Entrez le nœud de départ: ").strip()
start_node = get_start_node(graphe, user_input)
```

Création de figure pour affichage de 2 axes en Python

```python
# Code pour afficher le graphe avec un agencement en cercle
G = nx.DiGraph()  # Utilisation d'un graphe dirigé pour les flèches
for node, neighbors in graphe.items():   # Ajouter les noeuds et les arêtes au graphe
    for neighbor in neighbors:
        G.add_edge(node, neighbor)
pos = nx.circular_layout(G) # Choisir un layout circulaire pour arranger les nœuds en cercle
node_colors = ['red' if node == start_node else 'skyblue' for node in G.nodes()] # Définir la couleur des nœuds
# Créer une figure à la fois le graphe et le tableau
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 8))  # Taille du graphe et du tableau
# Dessiner le graphe dans le premier axe (ax1)
nx.draw(G, pos, with_labels=True, node_color=node_colors, node_size=2000, font_size=16, font_weight='bold', edge_color='gray', arrows=True, ax=ax
#--------------------ax1.set_title("Présentation du Graphe")
# Affichage du tableau dans le deuxième axe (ax2)
table_data = [
    ["DFS en ordre Croissant", result_ascending],
    ["DFS en ordre Décroissant", result_descending],
    ["BFS en ordre Croissant", result_croissant],
    ["BFS en ordre Décroissant", result_decroissant]
]
ax2.axis('off') # Désactiver les axes pour le tableau
# Afficher le tableau dans ax2
table = ax2.table(cellText=table_data, colLabels=["Parcours", "Résultats"], cellLoc='center', loc='center', colColours=["lightblue", "lightgreen"
```

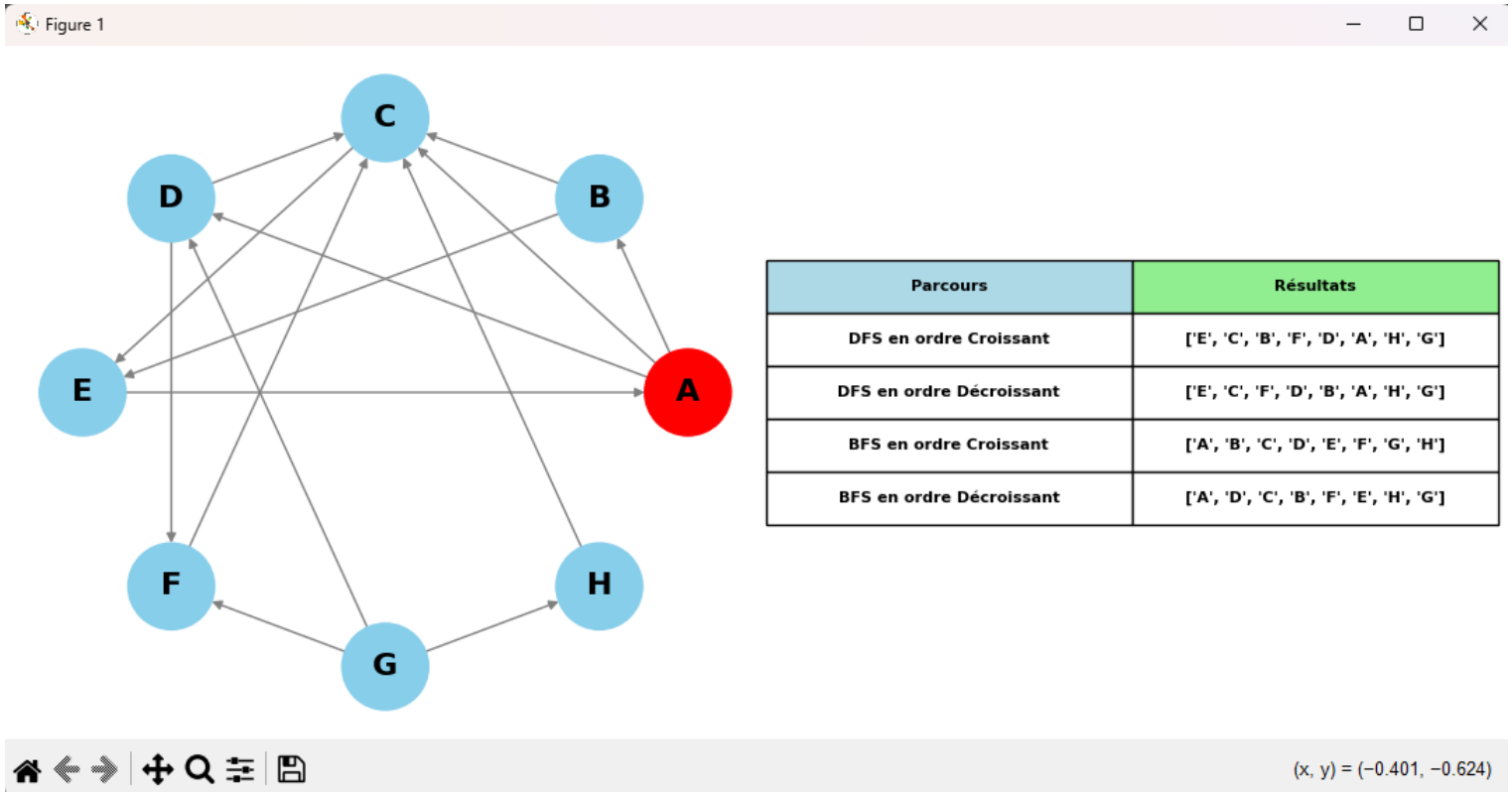
```python
# Modifier la hauteur des lignes (données)
for key, cell in table.get_celld().items():
    if key[0] != 0:  # Eviter la ligne d'entête
        cell.set_height(0.1)  # Ajuster la hauteur des lignes
        cell.set_fontsize(25)  # Modifier la taille de la police pour les cellules de données
        cell.set_text_props(fontweight='bold')  # Rendre la police de l'en-tête en gras
# Modifier la hauteur et le style des cellules d'entête (colLabels)
for cell in table.get_celld().values():
    if cell.get_text() is not None:  # S'assurer que la cellule contient du texte
        cell.set_height(0.08)  # Ajuster la hauteur de l'en-tête
        cell.set_fontsize(25)  # Modifier la taille de la police pour les cellules d'en-tête
        cell.set_text_props(fontweight='bold')  # Rendre la police de l'en-tête en gras

# Fixer la taille de la fenêtre d'affichage (en pouces)
fig.set_size_inches(16, 7)  # (Largeur,Hauteur)
# Afficher l'ensemble
plt.tight_layout()  # Ajuster pour un meilleur espacement
plt.show()
```

Affichage des résultats

Terminal

```
Entrez le nœud de départ: A
Parcours en profondeur DFS en ordre croissant   :  E - C - B - F - D - A - H - G
Parcours en profondeur DFS en ordre décroissant :  E - C - F - D - B - A - H - G
Parcours en largeur    BFS en ordre croissant   :  A - B - C - D - E - F - G - H
Parcours en largeur    BFS en ordre décroissant :  A - D - C - B - F - E - H - G
```

Figure en 2 axes



Parcours	Résultats
DFS en ordre Croissant	['E', 'C', 'B', 'F', 'D', 'A', 'H', 'G']
DFS en ordre Décroissant	['E', 'C', 'F', 'D', 'B', 'A', 'H', 'G']
BFS en ordre Croissant	['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
BFS en ordre Décroissant	['A', 'D', 'C', 'B', 'F', 'E', 'H', 'G']