Tecnológico de Monterrey

Actividad 3. Algoritmo de Dinic

Cruz Daniel Pérez Jiménez - A1736214

Erwin Porras Guerra - A01734881

Marlon Yahir Martínez Chacón - A01424875

Análisis y diseño de algoritmos avanzados

Luciano Gracía Bañuelos

Fecha de entrega:
30 de noviembre del 2023

Tecnológico de Monterrey

Algoritmo de Dinic

El algoritmo de Dinic es un algoritmo que nos sirve para encontrar el flujo máximo en un grafo dirigido. Este algoritmo es eficiente y se puede lograr una complejidad de O(V^2 E) donde V son los vértices del grafo, y E son los ejes o aristas de este. Para este algoritmo se deben realizar los siguientes pasos:

- Breadth First Search:Se implementa un algoritmo de breadth first search para construir capas de niveles en el grafo, esta comienza desde el origen y se van asignando niveles a los vértices en términos de la distancia más corta desde estos al origen.
- Depth First Search: Se hace uso de un algoritmo de depth first search para buscar el camino que únicamente toma incrementos de nivel.
- Aumento de flujo: Se va aumentando el flujo por cada iteración, se hace encontrando el valor mínimo de capacidad a lo largo del camino de aumento.

El algoritmo tiene una complejidad temporal de O(V^2 E) y una complejidad espacial de O(V^2).

Visualización

Para lograr que el grafo se visualizará adecuadamente, procedimos a utilizar la función “printGrafo”, la cual recibe al grafo que dibujará, sus niveles, el título del gráfico y el layout. Posteriormente definimos algunos parámetros del gráfico como el color del camino resultante, el color de los nodos de acuerdo al nivel o el grosor de las líneas.

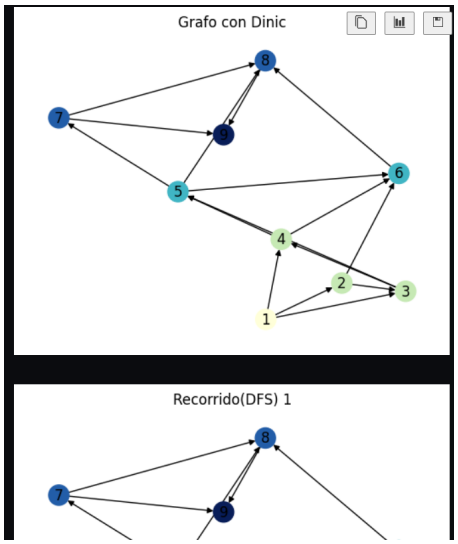
```python
#Mostrar grafo con nx.draw, modificando los colores de acuerdo al nivel
def printGrafo(g, levels, path=[], title='', pos=None):
    plt.title(title)
    nodeColor = list(levels.values())
    pathColor = 'green'

    if pos is None:
        pos = nx.spring_layout(g)

    nx.draw(g, pos, with_labels=True, node_color=nodeColor, cmap=plt.cm.YlGnBu)

    if path:
        edges = [(path[i], path[i + 1]) for i in range(len(path) - 1)]
        nx.draw_networkx_edges(g, pos, edgelist=edges, edge_color=pathColor, width=2)

    plt.show()
    return pos
```

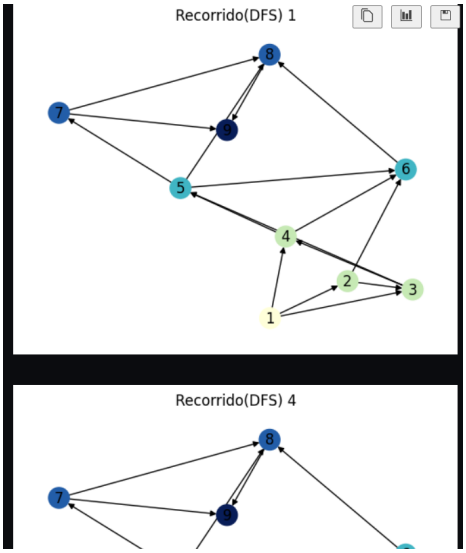
También hacemos uso de una función llamada “printDinic”, que se encarga de imprimir el recorrido que el algoritmo de Dinic hace por el grafo, mediante el dfs, especificando de igual manera las posiciones a las que va en el título

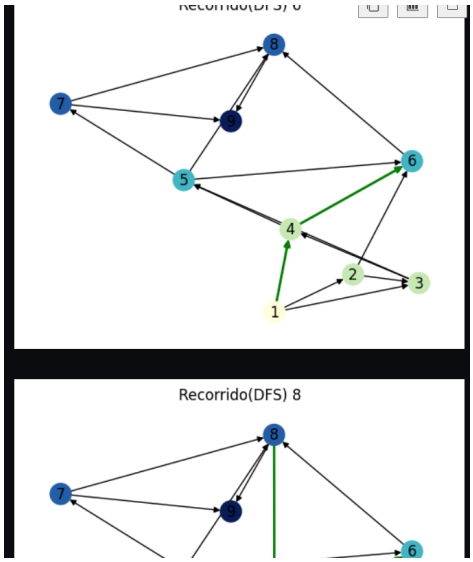
```python
#%%
def printDinic(g_networkx,g_dinic, start, pos):
    visited = set()
    stack = [start]
    while stack:
        current = stack.pop()
        if current not in visited:
            visited.add(current)
            pos = printGrafo(g_networkx, bfslevels(g_dinic, start), path=list(visited),
                             title=f'Recorrido(DFS) {current}', pos=pos)
            stack.extend(neighbor for neighbor in g_networkx.neighbors(current) if neighbor not in visited)
    return pos
```

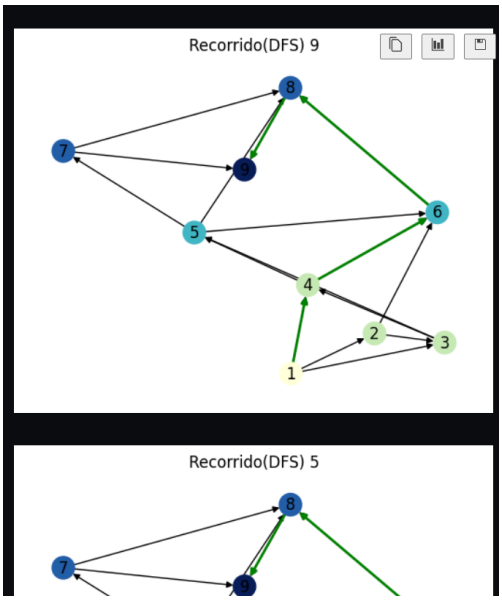
Obteniendo el siguiente resultado:

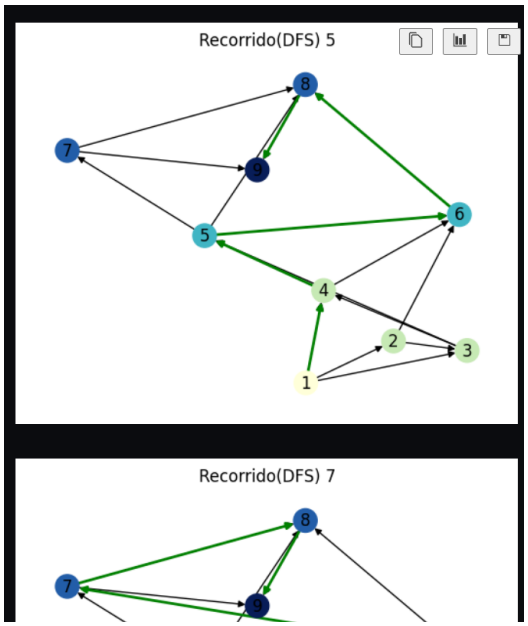
CASO 1



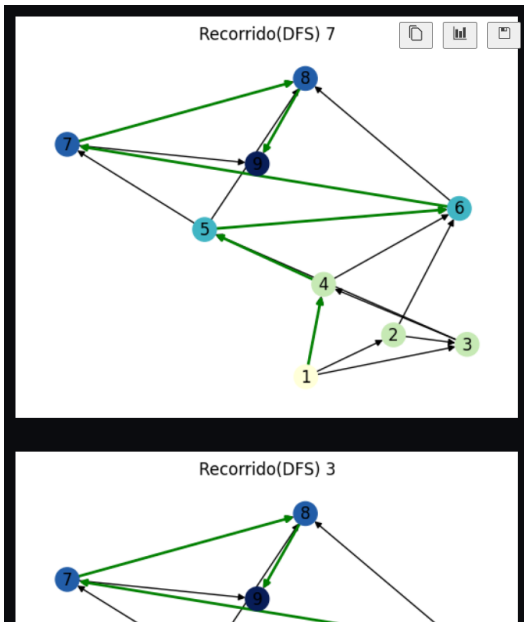




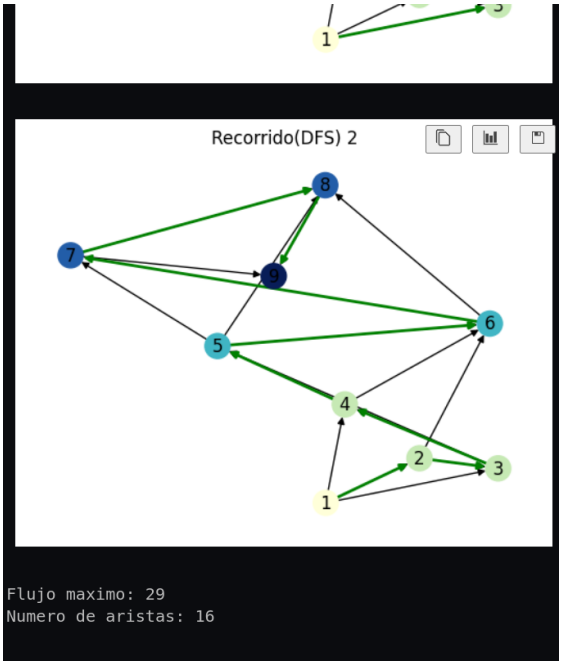



Recorrido(DFS) 5
8
7
9
6
5
4
2
3
1

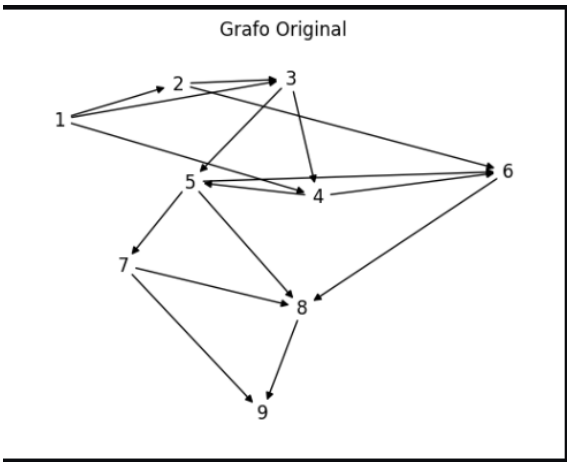
Recorrido(DFS) 7
8
7
9


Recorrido(DFS) 7
8
7
9
6
5
4
2
3
1

Recorrido(DFS) 3
8
7
9


1
Recorrido(DFS) 2
8
7
9
6
5
4
2
3
1
Flujo maximo: 29
Numero de aristas: 16

Siendo que nuestro nodo original fue este:



Reflexiones personales:

Cruz Daniel Pérez:

Mediante el algoritmo de Dinic se pudo vislumbrar una solución elegante, pero compleja, a la cuestión de encontrar un camino en un grafo dirigido con el mayor flujo posible, siendo así que con esto se optimiza la manipulación de información en grafos de gran escala, lo que resulta clave al momento de implementar este tipo de cuestiones en productos grandes. Siendo así que se obtiene un algoritmo eficiente para calcular en el menor tiempo posible el camino con el flujo máximo.

Erwin Porras Guerra:

El algoritmo de Dinic es bastante interesante ya que ha sido el algoritmo con complejidad más baja que hemos visto para recorrer grafos y encontrar el mejor camino. Lo que se me hace curioso es que a pesar de que sea el algoritmo más eficiente, este aún tiene una complejidad “alta”, no es como otros algoritmos para otras aplicaciones que llegan hasta complejidades

lineales, esto me hace cuestionar: ¿Será así por la naturaleza de los grafos? ¿O es posible obtener un algoritmo para esta misma aplicación con una menor complejidad?

Marlon Yahir Martínez Chacón:

El algoritmo que se nos presentó fue el algoritmo de Dinic, este algoritmo lo encontré bastante interesante debido a que dentro de su implementación, lo que hace es utilizar las dos técnicas para recorrer los grafos, pero con algunas modificaciones para poder verificar el flow de las aristas del grafo, y una vez que utilizamos una librería gráfica para poder visualizar los grafos que construimos se muestra como una solución muy elegante y a su vez compleja por todo los atributos que necesita de una arista como lo son la capacidad, el flow y el borde inverso, cosas que son manipuladas para aprovechar esa información y encontrar el flujo máximo de ese grafo, mediante la comprobación de cada uno de los caminos disponibles que son marcados por la visualización de los gráficos, una actividad muy interesante y retadora por el uso de un nuevo algoritmo y también de una biblioteca impresionante.

Link del repositorio: https://github.com/MarlonY123/AlgoritmosE3.git