EDMONDS-KARP ALGORITHM

ISAAC STALLEY
B97756

UNIVERSIDAD DE COSTA RICA
ESTRUCTURAS ABSTRACTAS DE DATOS Y ALGORITMOS PARA INGENIERÍA
PROYECTO DE INVESTIGACIÓN
II PERIODO 2020
GRUPO 01

Introducción

El algoritmo de Edmonds Karp se basa completamente en el algoritmo de Max Flow Ford Fulkerson, la única diferencia entre los algoritmos es que a diferencia del algoritmo de Ford Fulkerson que no especifica cómo encontrar los augmenting paths, el algoritmo de Edmonds Karp especifica que para encontrar los augmenting paths debemos usar un Breadth-First Search. El algoritmo de Edmonds Karp tiene una complejidad de $O(V^2N)$, donde V es el número de vértices y N es el número de nodos.

El algoritmo fue publicado por primera vez por Yefim Dinitz en 1970 **(1)** y posteriormente publicado de forma independiente por Jack Edmonds y Richard Karp en 1972. **(2)**

Discusión

Máximo Flujo

Para encontrar el flujo máximo de una red, comenzamos con un diagrama de flujo. El flujo máximo a través del gráfico de flujo que comienza en la fuente y termina en el target está limitado por la capacidad de los vértices, el flujo máximo es un valor de "cuello de botella" para la cantidad de flujo que la fuente puede entregar.

Augmenting Paths

Es una ruta de vértices en el gráfico de flujo con capacidad no utilizada mayor que cero desde la fuente "s" hasta el target "t". El cuello de botella del augmenting path se encuentra como el valor de capacidad más pequeño de todos los vértices del augmenting path.

Algoritmo de Ford Fulkerson

Para encontrar el flujo máximo de una red, el algoritmo Ford Fulkerson encuentra repetidamente augmenting paths a través del gráfico de flujo y aumenta el flujo en ellas hasta que no se pueden encontrar más augmenting paths, una vez que no hay más augmenting paths que se puedan encontrar, encontramos el valor de cuello de botella de la red mejor conocido como valor de flujo máximo, con la suma del flujo en todos los vértices de la fuente. La complejidad del algoritmo de Ford Fulkerson depende de la complejidad del algoritmo utilizado para encontrar los augmenting paths.

El algoritmo de Ford Fulkerson se publicó en 1956 por L. R. Ford Jr. yD. R. Fulkerson. **(3)**

Breadth-First Search

Es un algoritmo de búsqueda que se utiliza para encontrar la ruta más corta en un gráfico no ponderado. Utiliza una cola para visitar cada nodo en un gráfico nivel por nivel. Tiene una complejidad de $O(V+N)$, donde V es el número de vértices y N es el número de nodos.

Algoritmo de Edmonds Karp

El algoritmo de Edmonds Karp es una implementación del algoritmo de Ford Fulkerson que utiliza un algoritmo de Breadth-First Search para encontrar los augmenting paths. Tiene una complejidad de $O(V^2N)$, donde V es el número de vértices y N es el número de nodos.

Implementación del Algoritmo en Python

Para la implementación del algoritmo, crei dos funciones, uno llamado Edmonds_Karp y el otro llamado bfs. Para la función de Edmonds_Karp, lo que decidí hacer fue crear una matriz llamado “rm” (residual matrix) encargado de mantener los valor residuos de los vértices, es decir los valores de (capacidad - flujo) de los vértices, de esa manera cada vez que se obtiene el valor de capacidad mínima de un augmenting path, puedo “empujar ese flujo” por el augmenting path restando el valor de la capacidad mínima a los vértices en la matriz. Así, cuando un vértice se satura, en su índice en la matriz queda un 0, de manera que cuando vuelve a pasar la matriz por bfs, el vértice saturado no se tomará en cuenta.

```python
import copy
# Edmonds Karp Algorithm, inputs are M = Adjacency matrix, s = source and t =
sink.
def Edmonds_Karp(M, s, t):
    rm = copy.deepcopy(M) # rm = residual matrix (capacity - flow)
    max_flow = 0
    ap = bfs(rm, s, t) # ap = augmenting path
    while ap != None:
        min_flow = min(rm[ap[i]][ap[i + 1]] for i in range(len(ap) -1))
        for i in range(len(ap) -1):
            rm[ap[i]][ap[i + 1]] -= min_flow
        ap = bfs(rm, s, t)
    for i in range(len(M)):
        max_flow += M[s][i] - rm[s][i]
    return max_flow
```

Codigo 1. Edmonds-Karp Function

Para la función de bfs, es una implementacion normal del Algoritmo, con la única excepción de que retorna una lista con el camino más corto. La forma que se obtiene el camino es por medio de la lista llamada visited, la lista guarda en los índices de cada nodo, el índice del nodo anterior en el camino. El último while loop es utilizado luego para obtener el camino de la lista visitada, empezando en el target y terminando en el source, últimamente se utiliza el método reverse() para hacer flip al orden del camino.

```
# BFS algorithm, inputs are M = Adjacency matrix, s = source and t = sink.
# Returns the shortest path or None.
def bfs(M, s, t):
    queue = [s]
    visited = [-1] * len(M)
    visited[s] = -2
    path = [t]
    while queue:
        visit = queue[0]
        queue.pop(0)
        for i in range(len(M)):
            if (M[visit][i] > 0 and visited[i] == -1):
                queue.append(i)
                visited[i] = visit
    if visited[t] == -1:
        return None
    else:
        i = t
        while visited[i] != s:
            path.append(visited[i])
            i = visited[i]
        path.append(s)
        path.reverse()
    print(path)
    return path
```

Codigo 2. Breadth-First Search Function

Ejemplos

Ejemplo 1

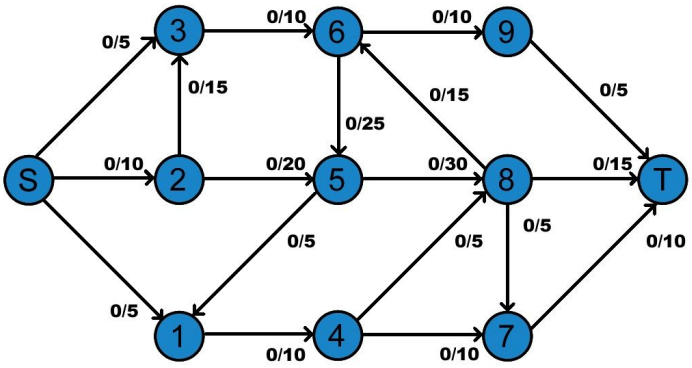

Imagen 1. Flow graph

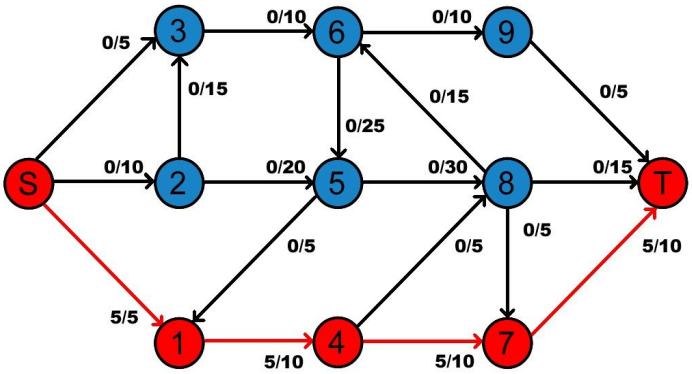

Imagen 2. Primer augmenting path.

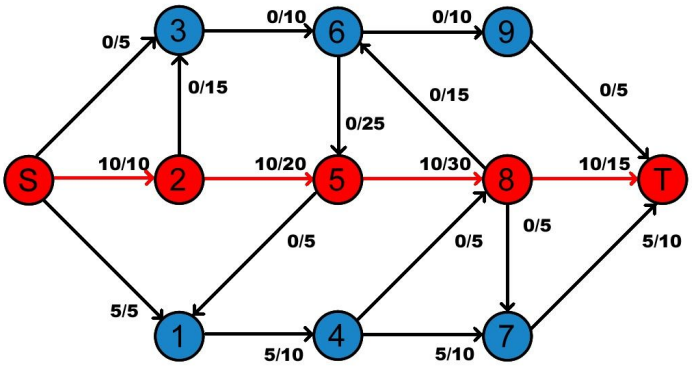

Imagen 3. Segundo augmenting path.

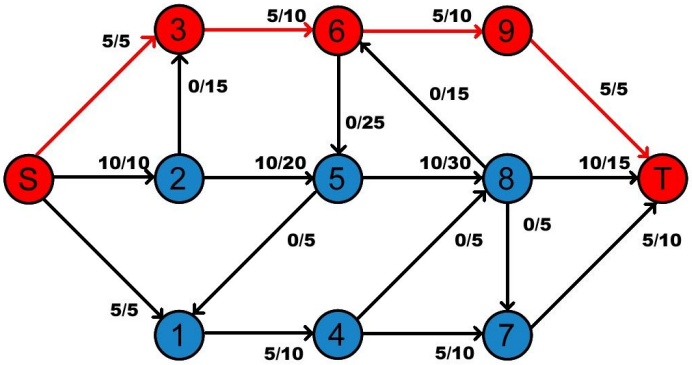

Imagen 4. Tercer augmenting path.

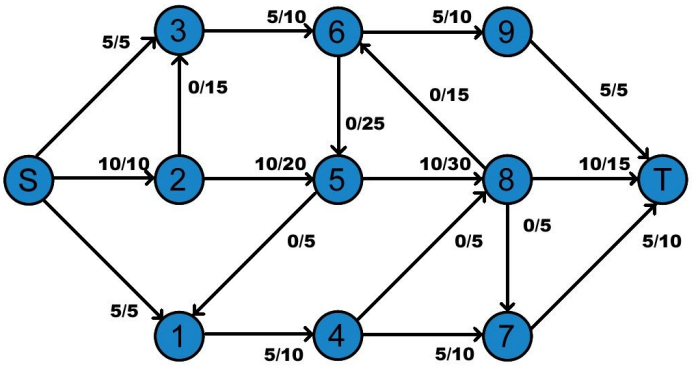

Imagen 5. Network con flujo máximo.

```
# Example 1
#     s, 1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,t
M = [[0 ,5 ,10,5 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ], # s
     [0 ,0 ,0 ,0 ,10,0 ,0 ,0 ,0 ,0 ,0 ], # 1
     [0 ,0 ,0 ,15,0 ,20,0 ,0 ,0 ,0 ,0 ], # 2
     [0 ,0 ,0 ,0 ,0 ,0 ,10,0 ,0 ,0 ,0 ], # 3
     [0 ,0 ,0 ,0 ,0 ,0 ,0 ,10,5 ,0 ,0 ], # 4
     [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,30,0 ,0 ], # 5
     [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,10,0 ], # 6
     [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,10], # 7
     [0 ,0 ,0 ,0 ,0 ,0 ,15,5 ,0 ,0 ,15], # 8
     [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,5 ], # 9
     [0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ]] # t
```

```
s = 0
t = 10
print("\n" + "Example 1:")
print("The augmenting paths are:")
max_flow = Edmonds_Karp(M, s, t)
print("\n" + "The maximum flow value is: " + str(max_flow))
```

Codigo 3. Ejemplo 1

Al correr se obtiene

Example 1:
The augmenting paths are:
[0, 1, 4, 7, 10]
[0, 2, 5, 8, 10]
[0, 3, 6, 9, 10]

The maximum flow value is: 20

Ejemplo 2

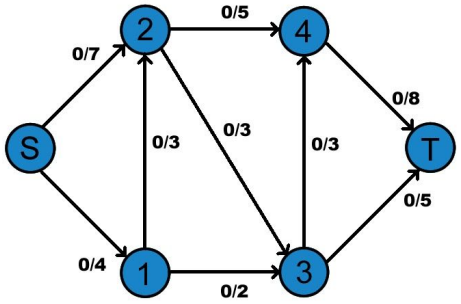

Imagen 6. Flow graph del ejemplo 2.

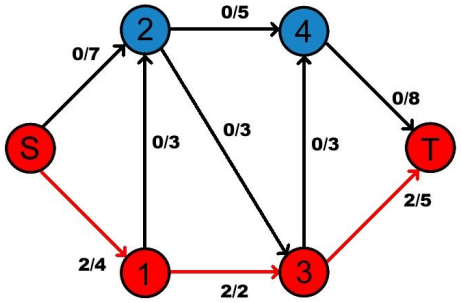

Imagen 7. Primer augmenting path.

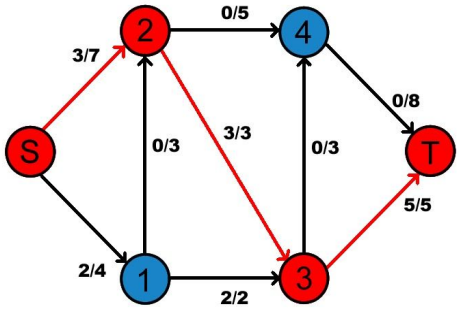

Imagen 8. Segundo augmenting path.

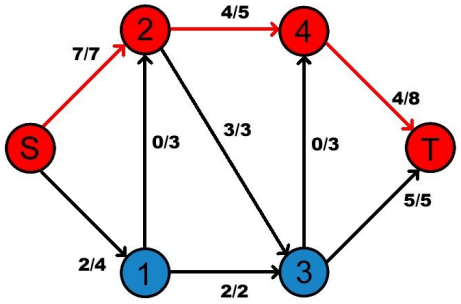

Imagen 9. Tercer augmenting path.

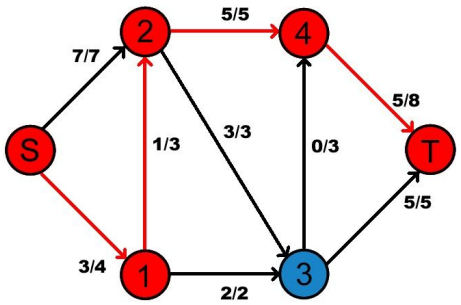

Imagen 10. Cuarto augmenting path.

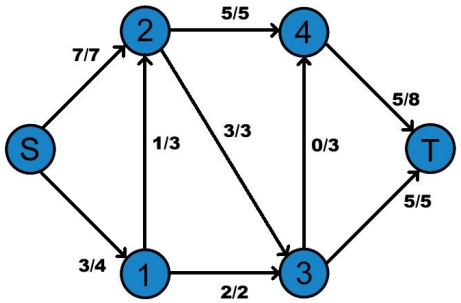

Imagen 11. Network con flujo máximo.

```
# Example 2
#      s, 1, 2, 3, 4, t
M = [[ 0, 4, 6, 0, 0, 0], # s
     [ 0, 0, 2, 3, 0, 0], # 1
     [ 0, 0, 0, 3, 5, 0], # 2
     [ 0, 0, 0, 0, 3, 5], # 3
     [ 0, 0, 0, 0, 0, 8], # 4
     [ 0, 0, 0, 0, 0, 0]] # t

s = 0
t = 5
print("\n" + "Example 2:")
print("The augmenting paths are:")
max_flow = Edmonds_Karp(M, s, t)
print("\n" + "The maximum flow value is: " + str(max_flow) + "\n")
```

Codigo 4. Ejemplo 2

Al correr se obtiene

Example 2:
The augmenting paths are:
[0, 1, 3, 5]
[0, 2, 3, 5]
[0, 2, 4, 5]
[0, 1, 2, 4, 5]

The maximum flow value is: 10

Conclusión

Como conclusión entonces el algoritmo de Edmonds Carp es una implementación del algoritmo de Ford Fulkerson donde los augmenting paths se obtienen utilizando un Breadth-First Search. El algoritmo tiene muchas aplicaciones ya que es una herramienta para el análisis de grafos de flujos y estos grafos son utilizados para modelar sistemas de red, por ejemplo un circuito, una red de conecciones en internet, una red de tubos, una red de caminos o cualquier otro tipo de red que se puede representar utilizando un grafo de flujos. El algoritmo se puede utilizar para conseguir el valor del flujo máximo de estas redes de manera eficiente.

Este algoritmo es más eficiente que el de Ford Fulkerson ya que aunque no se especifique cuál método utilizar para encontrar los augmenting paths en el algoritmo de Ford Fulkerson, si no se está utilizando el algoritmo de Edmonds Karp, la manera que se encontrarán es por medio de un Depth-First Search lo cual causaría que el algoritmo de Ford Fulkerson tuviera complejidad de $O(VN^2)$. Esto es menos eficiente que el algoritmo de Edmonds Karp ya que en un grafo de flujo siempre existe una mayor cantidad de nodos que vertices y el algoritmo de Edmonds Karp que utiliza un Breadth-First Search tiene complejidad de $O(NV^2)$ donde en ambos casos V es el número de vértices y N es el número de nodos.

Referencias

Dinic, E. A. (1970). "Algorithm for solution of a problem of maximum flow in a network with power estimation". Soviet Mathematics - Doklady. Doklady. 11: 1277–1280. **(1)**

Yefim Dinitz (2006). "Dinitz' Algorithm: The Original Version and Even's Version" (PDF). In Oded Goldreich; Arnold L. Rosenberg; Alan L. Selman (eds.). Theoretical Computer Science: Essays in Memory of Shimon Even. Springer. pp. 218–240. **(1)**

Edmonds, Jack; Karp, Richard M. (1972). "Theoretical improvements in algorithmic efficiency for network flow problems" (PDF). 248 - 264 **(2)**

Algorithms and Complexity. 63–69, Retrieved from: https://web.archive.org/web/20061005083406/http://www.cis.upenn.edu/~wilf/AlgComp3.html