

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0217 – Estructuras de Datos Abstractas y Algoritmos para ingeniería
II ciclo 2020

Proyecto Final
Algoritmo de Bellman-Ford

Syndell Poveda Cubero Carné: B25225

Profesor: Juan Carlos Coto Ulate

14 de Diciembre

Índice

1. Resumen	1
2. Historia y Contexto	2
3. Funcionamiento	2
4. Usos	4
5. Complejidad	5
6. Implementación en Python	6

Índice de figuras

1.	Grafo inicial	3
2.	Soluciones parciales obtenidas aplicando algoritmo de Bellman Ford a Grafo de Figura 1	4
3.	Camino mínimo final de todos los nodos al primero	4
4.	Código para imprimir laberinto	5
5.	Salida de código de algoritmo de Bellman-Ford	7
6.	Salida de código de algoritmo de Bellman-Ford igual a figura 2	8

Índice de tablas

1. Relaciones de distancia de grafo inicial 3

1. Resumen

Este proyecto busca exponer un resumen del algoritmo de Bellman-Ford cómo se creó y su contexto histórico. Además una explicación de cómo funciona, junto con un ejemplo que se utilizará a lo largo del proyecto. Seguidamente se mencionan algunos usos actuales y propuestas de usos. También se realiza un análisis breve de su complejidad y finalmente se realiza una implementación en python siguiendo con el ejemplo antes mencionado. Todo esto con el fin de profundizar un poco más el tema de grafos y algoritmos.

2. Historia y Contexto

Los métodos de matrices para encontrar el camino más corto en distancia eran desarrollados para estudiar las relaciones en redes, como encontrar el cierre transitivo de una relación; esto significa, identificar en un grafo direccionado los pares de puntos t, s de manera que se puede llegar a t desde s . Estos métodos eran estudiados para usarlos en la aplicación de redes de comunicación (incluyendo las redes neuronales) y la sociología animal [1].

Estos métodos de matriz consisten en la representación de gráficos dirigidos usando una matriz, y entonces tomando los productos iterativos de la matriz para calcular el cierre transitivo. Esto fue estudiado por Landahl y Runge (1946), Landahl (1947), Luce y Perry (1949), Luce (1950), Lunts (1950, 1952) y A. Shimbel.

El interés de Shimbel en los métodos de matrices fue motivada por sus aplicaciones en redes neuronales. Él analizó con matrices cuáles sitios en una red pueden comunicarse con cada una de las otras y cuánto tiempo tomaba. En 1951 Shimbel observó que las entradas positivas en una matriz S^t corresponden a pares entre los cuales existe comunicación en t pasos.

Un sistema con adecuada comunicación es aquel cuya matriz S^t es positiva para algún at . Otra de las observaciones de Shimbel (1951) es que en un sistema de comunicación adecuado, el tiempo que toma que todos los sitios tengan toda la información es igual al mínimo de valores de t para los cuales S^t es positivo.

Shimbel (1953) mencionó que la distancia de i a j es igual al número de ceros en las posiciones de las matrices $S^0, S^1, S^2, \dots, S^t$. Esencialmente dio un algoritmo $O(n^4)$ para encontrar todas las distancias en un digrafo con distancias unitarias. Si un digrafo $D=(V,A)$ y una función de longitud $l: A \rightarrow \mathbb{R}$ son dados, se pueden pedir las distancias y el camino más corto dado un vértice s . Para esto existen dos métodos bien conocidos: el método de Bellman-Ford y el de Dijkstra. El último es más rápido pero está restringido a funciones con distancias no negativas.

El marco general de ambos métodos es el siguiente esquema, descrito de forma general por Ford (1956). Se mantiene una función d de distancia provisional. Inicialmente se configura $d(s):=0$ y $d(v):=\infty$ para cada $v \neq s$. Ahora, de manera iterativa se selecciona un arco (u,v) con $d(v) > d(u) + l(u,v)$ y se reinicia $d(v):=d(u) + l(u,v)$. Si el arco no existe, entonces d es la función de distancia.

La diferencia en los métodos es la regla en la cual el arco (u,v) con $d(v) > d(u) + l(u,v)$ es elegida. El método de Bellman-Ford consiste en considerar todos los arcos consecutivamente y aplicando donde es posible, y repitiendo esto. Este es el método descrito por Shimbel (1955), Bellman (1958) y Moore (1959) [1].

3. Funcionamiento

El algoritmo de Bellman-Ford como se vio anteriormente determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos. El objetivo es el mismo que el del algoritmo de Dijkstra, la diferencia de estos algoritmos es que en de Bellman-Ford los pesos pueden tener valores negativos ya que este permite detectar la existencia de un ciclo negativo.

El algoritmo parte de un vértice origen que será ingresado, a diferencia de Dijkstra que utiliza una técnica rápida para seleccionar vértices de menor peso y actualizar sus distancias mediante el paso de relajación, Bellman-Ford simplemente relaja todas las aristas y lo hace $|V| - 1$ veces, siendo $|V|$ el número de vértices del grafo.

Para la detección de ciclos negativos se realiza el paso de relajación una vez más y si se obtuvieron mejores resultados es porque existe un ciclo negativo, para verificar porque tenemos un ciclo se puede seguir relajando las veces que se quiera y se sigue obteniendo mejores

resultados.

A continuación se verá su funcionamiento con un ejemplo práctico. Se tiene el grafo de la figura 1, con sus respectivas relaciones de distancia inicial en la tabla 1. Se ponen infinito entre los vértices que no son vecino, y en lo que sí se pone su respectivo peso. En caso de un grafo sin peso se asigna valor de peso unitario.

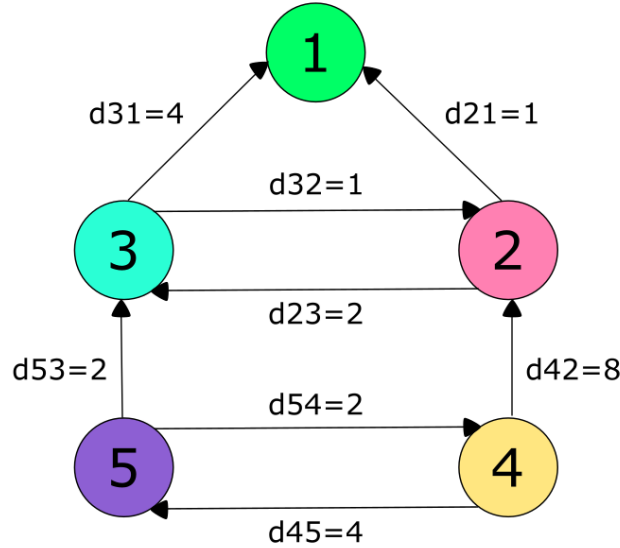


Figura 1: Grafo inicial

destino/ origen	1	2	3	4	5
1	0	∞	∞	∞	∞
2	1	0	2	∞	∞
3	4	1	0	∞	∞
4	∞	8	∞	0	4
5	∞	∞	2	2	0

Tabla 1: Relaciones de distancia de grafo inicial

Seguidamente se visita cada arista y se relaja las distancias de los caminos si estos no son los mínimos. Se realiza esto de manera iterativa V veces, donde V corresponde a la cantidad de vértices del grafo, esto es porque en el peor de los casos, el camino debe ser reajustado V veces, pero podría salir en menos pasos. Tomando en cuenta el nodo 1 como destino, se realizan los siguientes pasos:

- Paso 1: Se agrega la distancia de los nodos que tienen acceso directo al vértice 1 y se suma a la distancia mínima acumulada que hay hasta el vértice oportuno. En este primer paso la distancia acumulada sería 0 para 1, debido a que se trata de la distancia a él mismo, e infinito para el resto porque todavía no se han analizado.
- Paso 2: Al tener ya una distancia mínima acumulada desde los nodos 2 y 3 hasta 1, se pueden actualizar las distancias mínimas de los nodos 4 y 5.
- Paso 3: Se van actualizando las distancias mínimas acumuladas (D) de los distintos vértices hasta 1, se van utilizando en los pasos siguientes para optimizar el camino mínimo. El final del algoritmo se da cuando no hay ningún cambio de un paso a otro, es decir, cuando ya no se puede encontrar un camino más corto o al llegar a la iteración $= V$.

Podemos ver estos pasos de manera más gráfica en la figura 2, donde se marca en azul siempre la distancia más corta.

n	D ₂		D ₃		D ₄		D ₅	
	$d_{21}+D_1^n$	$d_{23}+D_3^n$	$d_{31}+D_1^n$	$d_{32}+D_2^n$	$d_{45}+D_5^n$	$d_{42}+D_2^n$	$d_{53}+D_3^n$	$d_{54}+D_4^n$
1	1+0 1	2+∞ ∞	4+0 4	1+∞ ∞	4+∞ ∞	8+∞ ∞	2+∞ ∞	2+∞ ∞
2	1+0 1	2+4 6	4+0 4	1+1 2	4+∞ ∞	8+1 9	2+4 6	2+∞ ∞
3	1+0 1	2+2 4	4+0 4	1+1 2	4+6 10	8+1 9	2+2 4	2+9 11
4	1+0 1	2+2 4	4+0 4	1+1 2	4+4 8	8+1 9	2+2 4	2+9 11
5	1+0 1	2+2 4	4+0 4	1+1 2	4+4 8	8+1 9	2+2 4	2+8 10

Figura 2: Soluciones parciales obtenidas aplicando algoritmo de Bellman Ford a Grafo de Figura 1

En la figura 3 se observa como queda el camino mínimo final de todos los nodos al primero.

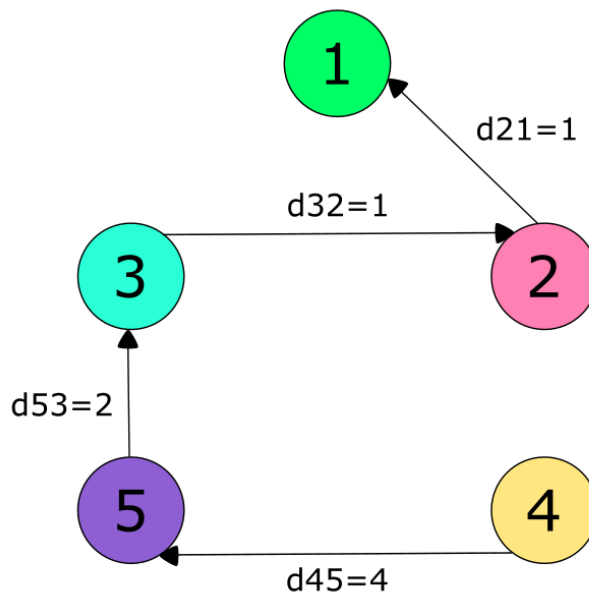


Figura 3: Camino mínimo final de todos los nodos al primero

4. Usos

- Enrutamiento: se ve en el método de enrutamiento de Vector de distancias. Se trata de uno de los más importantes junto con el de estado de enlace. Utiliza el algoritmo de Bellman-Ford para calcular las rutas. Lo usa el protocolo RIP (Routing Information Protocol), que hasta 1988 era el único utilizado en internet. El enrutamiento de un protocolo basado en vector de distancias requiere que un router informe a sus vecinos de los cambios en la topología periódicamente y en algunos casos cuando se detecta un cambio en la topología de la red [2].

- En Boddu (2019) se propone el algoritmo cooperativo modificado de Bellamn-Ford (CMB-FA) utilizando el modo de ahorro de batería (PSM) en teléfonos móviles para mejorar el desempeño de la red [3].
- Detección de Arbitraje: Es cuando se utiliza la función de detección de ciclos negativos del algoritmo para encontrar un camino en el cuál se pueda encontrar una ganancia al realizar una serie de cambios de moneda, de manera que el producto al final es más grande que el inicial.
- Resolución de juego escalera de palabra (Word ladder): la teoría de grafos se puede ver aplicada a este tipo de problema donde se toma en cuenta las palabras como vértices del grafo y las aristas como las transformaciones posibles para pasar de una palabra a otra.
- Un uso puede ser en un mapa para encontrar el camino más corto entre localidades. Donde se considera el mapa como un grafo, las localidades en el mapa como vértices y la distancia entre un vértice y otro como aristas y pesos.

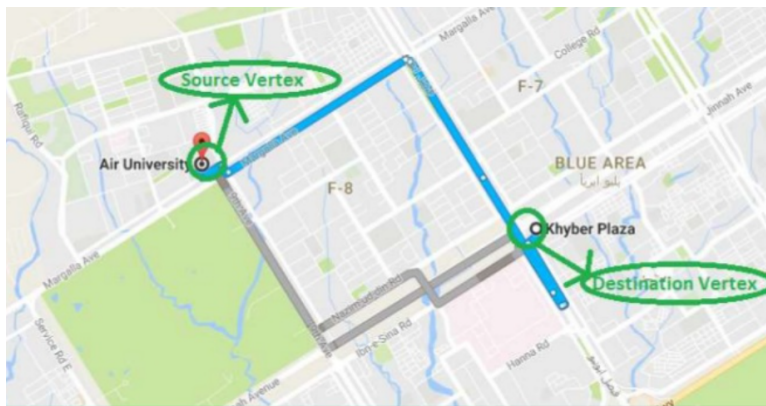


Figura 4: Código para imprimir laberinto

- En una red telefónica las líneas tienen un determinado ancho de banda (BW). Se quiere enrutar las llamadas telefónicas a través de los anchos de banda más altos, de manera que se consideran las líneas de transmisión con el mayor valor de ancho de banda los caminos más largos. En este caso se puede considerar la red telefónica como un grafo, las estaciones de conmutación de la red como vértices, las líneas de transmisión como aristas y los anchos de banda de la red como pesos.

5. Complejidad

La complejidad del algoritmo de Bellman-Ford es de $O(|V| \cdot |E|)$, donde $|V|$ y $|E|$ son el número de vértices y aristas respectivamente. A continuación se va a ver un análisis de dónde es que proviene esta complejidad tomando en cuenta un pseudocódigo del algoritmo.

BELLMAN-FORD(G, l, s)

1. INITIALIZE-SINGLE-SOURCE(G, s)
2. for $i = 1$ to $|G.V| - 1$
3. for each edge $(u, v) \in G.E$
4. RELAX(u, v, l)

5. for each edge(u,v) \in G.E
6. if $v.d > u.d + l(u,v)$
7. return FALSE
8. return TRUE

En la línea 1 del algoritmo: El loop inicial de for corre por cada vértice una vez. De esta forma este loop corre en tiempo $O(V)$. En las líneas 2-4: Consisten de 2 loops, uno ejecutando V veces y el otro E veces. De manera que la complejidad de tiempo va a ser $O(VE)$. En las líneas 5-7 el loop corre por cada arista, corriendo así en tiempo $O(E)$.

Entonces, la complejidad total es $O(V.E)$.

6. Implementación en Python

Para realizar esta sección se modificó un script de python encontrado en la página <https://www.geeksforgeeks.org/shortest-path-bellman-ford-algorithm/> [4] y se modificó para poder adaptarlo a este proyecto. A continuación se muestra el código utilizado para implementar el algoritmo junto con una breve explicación de cada paso.

```

1 class Grafo:
2
3     def __init__(self, vertices):
4         self.V = vertices    # Numero total de vertices en el grafo
5         self.arista = []     # Arreglo de aristas
6
7     # Agregar Aristas
8     def add_arista(self, s, d, w): #s=source, d=destiny, w=weight
9         self.arista.append([s, d, w])
10
11    # Imprimir soluci n
12    def mas_corto(self, dist):
13        print("Camino mas corto desde vertice")
14        for i in range(self.V):
15            print("{0}\t\t{1}".format(i, dist[i]))
16
17    def alg_bellman_ford(self, src):
18
19        # Paso 1: llenar el array de distancias y el array predecesor
20        dist = [float("Inf")] * self.V
21        # Se alar v rtice fuente
22        dist[src] = 0
23
24        # Paso 2: Relajar las aristas |V| - 1 veces
25        for _ in range(self.V - 1):
26            for s, d, w in self.arista:
27                if dist[s] != float("Inf") and dist[s] + w < dist[d]:
28                    dist[d] = dist[s] + w
29
30        # Paso 3: detectar ciclos negativos
31        # si el valor cambia entonces tenemos un ciclo negativo en el grafo
32        # y no se pueden encontrar las distancias mas cortas
33        for s, d, w in self.arista:
34            if dist[s] != float("Inf") and dist[s] + w < dist[d]:
35                print("Grafo contiene un ciclo negativo")
36                return
37
38        # Si no tiene ciclos negativos

```

```

39         # Imprimir la distancia y array predecesor
40         self.mas_corto(dist)

```

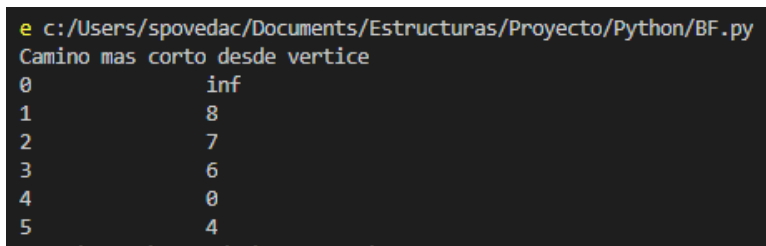
Para correr este código primero se crea el grafo con la cantidad de vértices que tenemos en el grafo. Siguiendo con el ejemplo de la figura 1 tenemos 5 vértices, y se agrega el vértice 0 para un total de 6 vértices. Seguidamente se va agregando cada una de las aristas del grafo, por ejemplo en la línea 2 tenemos la arista que va del vértice 2 al vértice 1 con un peso de 1. Finalmente, se especifica el vértice de origen desde donde se quiere encontrar el camino más corto, en este caso seleccionamos el 4.

```

1 g = Grafo(6)
2 g.add_arista(2, 1, 1)
3 g.add_arista(2, 3, 2)
4 g.add_arista(3, 1, 4)
5 g.add_arista(3, 2, 1)
6 g.add_arista(5, 3, 2)
7 g.add_arista(5, 4, 2)
8 g.add_arista(4, 5, 4)
9 g.add_arista(4, 2, 8)
10
11 g.alg_bellman_ford(4)

```

La salida de esta implementación la podemos observar en la figura 6. Como podemos observar los resultados son consistentes con los encontrados en la figura 3, donde se observa que del vértice 4 al vértice 5 hay un peso de 4, y al vértice 3 hay 2 pesos más para un total de 6, al vértice 2 hay un peso total de 7 y al vértice 1 un total de 8. Mientras que para él mismo la distancia es de 0.



```

e c:/Users/spovedac/Documents/Estructuras/Proyecto/Python/BF.py
Camino mas corto desde vertice
0          inf
1           8
2           7
3           6
4           0
5           4

```

Figura 5: Salida de código de algoritmo de Bellman-Ford

Si se quieren ver los mismos valores de la figura 4, donde se observa la distancia de cada punto hacia el punto 1, se logra cambiando el 1 a vértice origen como sigue:

```

1 g = Grafo(6)
2 g.add_arista(1, 2, 1)
3 g.add_arista(3, 2, 2)
4 g.add_arista(1, 3, 4)
5 g.add_arista(2, 3, 1)
6 g.add_arista(3, 5, 2)
7 g.add_arista(4, 5, 2)
8 g.add_arista(5, 4, 4)
9 g.add_arista(2, 4, 8)
10
11 g.alg_bellman_ford(1)

```

```
e c:/Users/spovedac/Documents/Estructuras/Proyecto/Python/BF.py
Camino mas corto desde vertice
0          inf
1          0
2          1
3          2
4          8
5          4
```

Figura 6: Salida de código de algoritmo de Bellman-Ford igual a figura 2

Referencias

- [1] Alexander Schrijver. On the history of the shortest path problem. *Documenta Math*, pages 155–157, 210.
- [2] Charles E Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. *ACM SIGCOMM computer communication review*, 24(4):234–244, 1994.
- [3] Boddu Devi, Kishan Kalitkar, and M. Rani. Application of modified bellman-ford algorithm for cooperative communication. *Wireless Personal Communications*, 109, 12 2019.
- [4] unknown. "bellman–ford algorithm — dp-23", 2020.