DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Número 2

8 de Abril de 2016 — Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Ciruelos Rodríguez, Gonzalo	063/14	gonzalo.ciruelos@gmail.com
Costa, Manuel José Joaquín	035/14	manucos94@gmail.com
Gatti, Mathias Nicolás	477/14	mathigatti@gmail.com
Maddonni, Axel Ezequiel	200/14	axel.maddonni@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Una Nueva Esperanza

1.1. Explicación formal del problema

Sea $G = (V, E)$ un grafo simple conexo con $n \geq 2$ vértices y m aristas. Además sea $M \subseteq E$ tal que $M \neq \emptyset$. Se desea hallar un camino (no necesariamente simple) de v_0 a v_{n-1} que pase al menos dos veces por un eje de M (potencialmente el mismo) y que tenga longitud mínima.

En la figura (1) pueden verse tres ejemplos. Los tres son muy parecidos pero permiten ilustrar distintas situaciones. En el primero (de izquierda a derecha y de arriba a abajo) encontramos el camino simple $P = (v_0, v_1, v_3, v_4, v_5)$ de longitud 4 que cumple con lo pedido pues tiene dos aristas especiales y es de longitud mínima.

En el segundo se agregó una arista corriente entre los nodos v_0 y v_5. Puede notarse que en este caso el mejor camino es $P = (v_0, v_2, v_0, v_5)$ de longitud 3, el cual claramente no es simple pues pasa dos veces por el vértice 0.

En el último caso, la arista recientemente agregada pasa a ser especial. Al hacer esto tenemos dos caminos de longitud mínima entre los que pasan por dos aristas especiales: el P que encontramos antes, y $P' = (v_0, v_5, v_0, v_5)$. Notar que P' no solo no es simple, sino que también usa a v_5 como nodo intermedio. Cualquiera de los dos es igualmente aceptable.

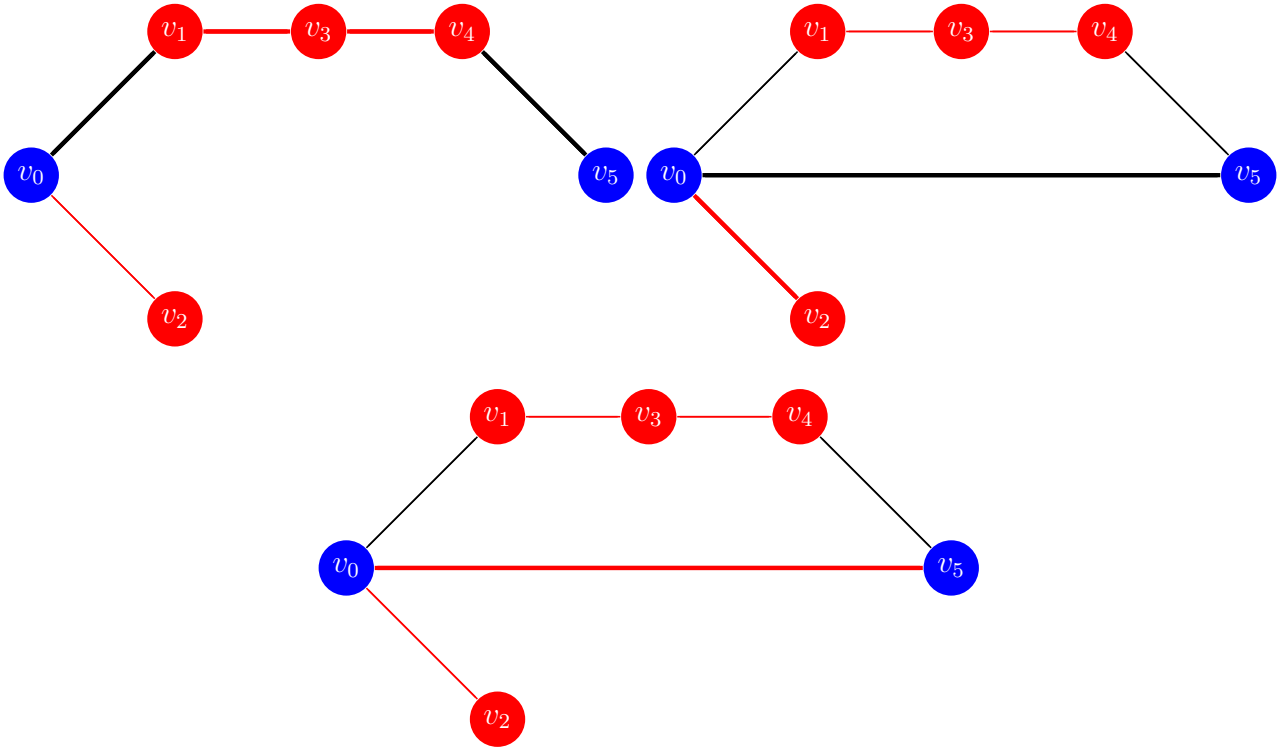

Figura 1: Ejemplos del problema. Las aristas especiales están pintadas de rojo y las comunes de negro. Las aristas que pertenecen a la solución están engrosadas. En azul distinguimos a los nodos inicial y final.

1.2. Explicación de la solución

Si bien lo que se pide es, en definitiva, encontrar un camino mínimo en G, la dificultad adicional que implica hacer que el camino contenga al menos dos aristas de M hace que no podamos aplicar de forma directa los algoritmos clásicos para este propósito. Para solventar esto consideraremos un grafo alternativo a G, G', con el cual resolver el problema planteado originalmente será equivalente a encontrar un camino mínimo en G' de forma tradicional (en este caso utilizando BFS).

1.2.1. Nuevo grafo para el modelado

Lo primero que haremos es armarnos un grafo nuevo G' a partir de G.

Consideramos tres grafos isomorfos a G: $G_0 = (V_0, E_0)$, $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$, tales que $f_k : V \to V_k / f_k(v_i) = v_{k \times n+i}$ para $k \in \{0, 1, 2\}$ son las biyecciones correspondientes. En particular, $G_1 = G$. La figura (2) ilustra la situación para un ejemplo puntual.

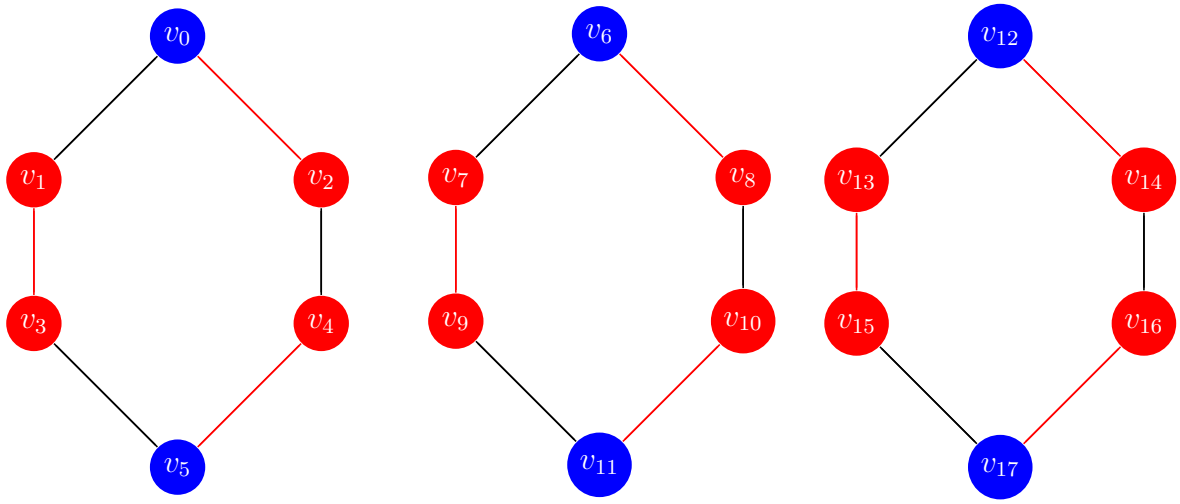

Figura 2: Tres isomorfismos del grafo original, construidos de la forma indicada.

Además, definimos M', un conjunto de arcos (aristas orientadas) de peso 1, como

$$M' = \{(f_0(v), f_1(w)) \text{ y } (f_0(w), f_1(v))/(v, w) \in M\} \cup \{(f_1(v), f_2(w)) \text{ y } (f_1(w), f_2(v))/(v, w) \in M\}$$

Por ejemplo, en la figura (2) la arista $(v_0, v_2) \in M$. Entonces queremos que M' tenga los arcos (v_0, v_8), (v_2, v_6), (v_6, v_{14}) y (v_8, v_{12}). Observar que en M' solo hay arcos que van de nodos de G_0 a nodos de G_1, y de G_1 a G_2. En ningún caso hay arcos de G_t a G_h, con $h < t$; ni arcos que vayan directamente de G_0 a G_2.

Entonces hasta acá tenemos tres grafos conexos isomorfos. Podemos pensarlos como las tres componentes conexas de un grafo con $3n$ vértices y $3m$ aristas. A continuación uniremos estas tres componentes mediante los arcos de M': sea $G^* = (V_1 \cup V_2 \cup V_3, E_1 \cup E_2 \cup E_3 \cup M')$.

Podemos pensar a G^* con la siguiente analogía: cada una de las tres componentes es un nivel, y los arcos "escaleras mecánicas" que permiten subir de un nivel al siguiente en forma unidireccional y que no se saltea niveles. Notar que entonces G' no es fuertemente conexo pues desde un nodo del nivel 2 o 3 no puedo alcanzar a un nodo en el nivel 1. Sin embargo, para

cualquier vértice en el nivel 1 todos los vértices del grafo son alcanzables. En particular esto vale para v_0.

Finalmente, definimos a G' como el resultado de quitarle a los niveles 0 y 1 de G^* todas las "aristas isomorfas" a las aristas de M[1]. En este caso ya no es cierto que cualquier vértice de G' sea alcanzable desde cualquier vértice del primer nivel: en efecto, si vemos la figura (2), al quitarle las aristas especiales al primer nivel el nodo v_5 no es alcanzable desde el v_1. No obstante, sigue valiendo el siguiente lema:

Lema 1.1: Todo vértice del tercer nivel (nivel 2) de G' es alcanzable desde todo vértice del primer nivel (nivel 0).

Demostración: Sean $u, w \in V_1$ (es decir, ambos del nivel 0). Supongamos que w no es alcanzable desde u en G'. Como G_1 inicialmente era conexo, esto significa que existía camino de u a w, Q, y seguro incluía alguna arista especial (sino seguiría existiendo en G' y w sería alcanzable desde u). Como todos los nodos que eran incidentes a una arista especial en G_1 son incidentes a un arco en G', seguro existe $z \in Q$, tal que z es incidente a un arco que permite pasar al segundo nivel. Luego, es posible llegar desde u a algún vértice del segundo nivel. Si no existe tal nodo w, entonces cualquier nodo del primer nivel es alcanzable desde u, y como M no era vacío, entonces seguro hay un camino desde u hasta el segundo nivel.

Es fácil ver que exactamente el mismo razonamiento se puede realizar para probar que es posible llegar desde cualquier nodo del segundo nivel a algún nodo del tercer nivel. Luego, por concatenación de ambas cosas, es posible llegar desde cualquier nodo del primer nivel a algún nodo del tercero. Pero como el tercer nivel sigue siendo conexo (pues nunca quitamos las aristas especiales) entonces es claro que esto es equivalente a poder llegar a cualquier nodo del tercer nivel. □

Debido a este lema, y como todas las aristas de G' tienen peso 1, es posible aplicar el algoritmo BFS para hallar el camino mínimo entre un nodo del nivel 0 y otro del 2. Particularmente, entre los nodos 0 y $3n - 1$.

En la sección siguiente probaremos que esto es equivalente a resolver el problema planteado originalmente.

1.2.2. Correctitud y optimalidad

La siguiente proposición garantiza la correctitud de nuestra solución.

Proposición 1.1: Sea $P' = (v_0, u_1, \ldots, u_k, v_{3n-1})$ un camino mínimo de v_0 a v_{3n-1} en G', de longitud $k + 1$, entonces $P = (P' \bmod n)$ [2]es camino de v_0 a v_{n-1} en G, mínimo entre los que pasan por al menos dos aristas especiales.

[1]En rigor, tanto G' como G^* sirven a nuestro propósito, pero G' tiene la ventaja de que permitirá reducir el uso de memoria (potencialmente mucho si hay muchas aristas especiales) y constantes en la complejidad de la implementación.

[2]$P = P' \bmod n \Leftrightarrow P_i = v_{j \bmod n}$ donde $P'_i = v_j$, $i = 0, \ldots, k-1$

Demostración: Hay que ver tres cosas respecto de P: que es camino de v_0 a v_{n-1}, que pasa por al menos dos aristas especiales, y que es mínimo respecto a los caminos que cumplen ambas cosas.

Va a ser útil recordar que $f_k : V \to V_k / f_k(v_i) = v_{k \times n+i}$ para $k \in \{0, 1, 2\}$ son las biyecciones de los isomorfismos planteados en la sección anterior.

- Es camino: Ante todo, por definición del operador módulo vale que $(\forall v_i \in P)\ 0 \leq i \bmod n \leq n-1$. Es decir que todos los vértices de P pertenecen a G, y además empieza en v_0 y termina en $v_{n-1} = v_{(3n-1) \bmod n} = v_{(2n+n-1) \bmod n}$. Queda ver que efectivamente nodos consecutivos en P son adyacentes en G.
  Si $v_i, v_j \in P$ son consecutivos entonces $v_{h+i} \in P'$ tiene que ser adyacente con $v_{h'+j} \in P'$, donde h y h' son un par de constantes múltiplos de n. Por el contexto del problema h y h' solo pueden ser $0, n$ o $2n$. Luego, si $h = h' = k \times n$, por definición de isomorfismo y por cómo está definida f_k, si v_{h+i} es adyacente a v_{h+j} en G_k (cosa que pasa, sino no podría pasar en G') entonces v_i es adyacente a v_j en G. Si $h \neq h'$, seguro que cada nodo es el extremo de un arco (pues están en diferentes niveles). Pero por construcción de G', solo puede haber un arco entre v_{h+i} y $v_{h'+j}$ si había una arista especial entre v_i y v_j en G, lo que significa que eran adyacentes. Luego, queda probado que P es un camino válido de v_0 a v_{n-1}.

- Pasa por al menos dos aristas especiales: el nodo v_{3n-1} pertenece al nivel 2 de G'. Esto significa que paso por dos arcos. Un arco entre $v_{k \times n+i}$ y $v_{(k+1) \times n+j}$ en G' solo existe si $(v_i, v_j) \in M$. Por definición de P, si tal arco pertenece a P' entonces tal arista especial pertenece a P. Luego, P tiene al menos dos aristas de M (podría tener más, pues en el tercer nivel las aristas especiales siguen existiendo y no son arcos).

- Es mínimo: Supongamos que P no es óptimo para el problema. Entonces existe Q tal que $|Q| < |P|$ y cumple con pasar por dos aristas de M. Construyamos Q', un camino de v_0 a v_{3n-1} en G', basado en Q.
  La idea es la siguiente: recorremos las aristas de Q en orden y las vamos poniendo en Q' hasta encontrar la primer arista especial, (v_i, v_j). En su lugar agregamos el arco (v_i, v_{n+j}) a Q'. Seguimos completando Q' con aristas $(f_1(u), f_1(w))$ por cada arista común (u, w) que encontramos en Q. Eventualmente llegamos a una segunda arista especial (por hipótesis existe), (v_s, v_t), y en su lugar agregamos el arco (v_{n+s}, v_{2n+t}) a Q'. Ahora bien, en este punto si Q es mínimo lo mejor que puede hacer es tomar el camino de distancia mínima desde v_t hasta v_{n-1}. Pero tal camino es isomorfo a un camino desde v_{2n+t} hasta v_{3n-1}, pues el tercer nivel es isomorfo a G. Por lo tanto agregando este camino a Q', llegamos a que Q' es un camino de v_0 a v_{3n-1} en G'.
  Pero como $|Q'| = |Q| < |P| = |P'|$, llegamos a que hay un camino más corto que P' en G' que conecta los mismos vértices. Esto es absurdo, pues por hipótesis P' era camino mínimo. Luego, el absurdo provino de suponer que P no era óptimo para el problema.

Finalmente, queda demostrada la proposición. □

De hecho, la recíproca de esta proposición también vale. No lo probamos sin embargo porque no es necesario para la correctitud de la solución. La forma de probarlo sería similar igualmente.

1.2.3. Explicación del código

Notar que para las dimensiones del grafo que toma BFS no usamos n y m sino n' y m', para no confundir con las dimensiones del problema original porque pueden ser diferentes, y de hecho por cómo lo vamos a usar, así va a ser.

Algorithm 1 Pseudocódigo del procedimiento BFS

```
1:  procedure BFS(ListaAdyacencia vs, vertice root, vertice target, int n')→ Vector<vertice>
2:      cola<vertice> c ← Vacia()                                   ▷ O(1)
3:      vector<int> distancia(n', ∞)                                ▷ O(n')
4:      vector<vertice> acm(n', −1)                                 ▷ O(n')
5:      distancia[root] ← 0                                         ▷ O(1)
6:      acm[root] ← root                                            ▷ O(1)
7:      c.push(root)                                                ▷ O(1)
8:      while ¬c.vacia?() do                                        ▷ O(n) veces
9:          actual ← c.pop()                                        ▷ O(1)
10:         VerticesAdyacentes vecinos ← vs[actual]                 ▷ O(1)
11:         for v ∈ vecinos do                                      ▷ O(d(v)) veces
12:             if distancia[v] = ∞ then                            ▷ O(1)
13:                 distancia[v] ← distancia[actual] + 1            ▷ O(1)
14:                 acm[v] = actual                                 ▷ O(1)
15:                 if v = target then                              ▷ O(1)
16:                     break                                       ▷ O(1)
17:                 c.push(v)                                       ▷ O(1)
18:     int long_sol ← distancia[target] − 1                        ▷ O(1)
19:     vector<vertice> solucion(long_sol, 0)                       ▷ O(long_sol) ⊆ O(m')
20:     vertice v ← acm[target]                                     ▷ O(1)
21:     for int i desde long_sol − 1 hasta 0 do                     ▷ O(m') veces
22:         solucion[i] ← v                                         ▷ O(1)
23:         v ← acm[v]                                              ▷ O(1)
24:     return solucion
```

La implementación de BFS que realizamos está compuesta por dos partes: la primera hasta la línea 17 inclusive, es la implementación clásica del algoritmo de búsqueda en anchura para determinar caminos mínimos, en particular modificada un poco para que termine a penas compute un camino hasta el nodo objetivo pues es el único que nos importa realmente; la segunda consiste en reconstruir el camino mínimo entre *root* y *target* a partir del árbol de caminos mínimos. Notar que la función tiene como precondición que efectivamente exista algún camino desde *root* hasta *target*.

Para la primer parte tenemos esencialmente tres estructuras importantes:

- una cola FIFO de vértices donde iremos encolando los vecinos del nodo en el que estamos actualmente y que todavía no hayamos visitado; la misma está implementada sobre una lista doblemente enlazada, lo que permite que las operaciones de encolar, desencolar y ver el siguiente elemento sean todas $O(1)$.

- un vector de distancias tal que la posición i-ésima del mismo guarda la distancia desde el *root* hasta el nodo i (o bien ∞ si todavía no pasamos por i, o simplemente i no es alcanzable desde *root*).
- un vector de vértices que representará nuestro árbol de caminos mínimos desde el *root* hasta cualquier nodo, de forma que en la posición i-ésima del vector tendremos al padre del nodo i en el árbol (o bien -1, si todavía no pasamos por i, o simplemente no es alcanzable desde *root*).

Que la búsqueda es correcta es resultado inmediato de que el algoritmo es el BFS tradicional cuya correctitud ya está probada.

Una vez hallado un camino mínimo hasta el nodo *target*, queremos ahora armar un vector de vértices que contenga a todos los vértices de dicho camino. Como la distancia es la cantidad de vértices en el camino menos uno, entonces el vector tendrá que tener tamaño igual a la distancia más uno. Pero como no nos interesa que el primer y último nodos estén en el vector entonces nos queda que el largo del mismo será $l = d(root, target) - 1$. Luego, es cuestión de llenar las posiciones de este vector de atrás para adelante, pues *a priori* para cada nodo solo sabemos cual es su padre en el árbol de caminos mínimos. La última posición tendrá al padre del nodo *target*, la anteúltima al padre del padre y así. Iterando l veces llegamos a que en la posición inicial del vector hay un nodo que es hijo de *root* y ancestro de *target*.

Finalmente devolvemos este vector.

Algorithm 2 Pseudocódigo del main

```
1:  procedure MAIN
2:      int n                                                          ▷ Cantidad de nodos
3:      vector(vector(int)) input      ▷ input[i] almacena los datos de la i-ésima arista pasada
4:      inicializar(input, n)          ▷ Leemos los datos pasados como parámetros e inicializamos
5:      ListaAdyacencia adj_list(3 * n, VerticesAdyacentes())          ▷ O(3 × n) = O(n)
6:      for (v1, v2, e) ∈ input do                                     ▷ m veces
7:          if e = True then                                           ▷ O(1)
8:              adj_list[v1].push_back(v2 + n)                         ▷ O(1)
9:              adj_list[v1 + n].push_back(b + 2 × n)                  ▷ O(1)
10:             adj_list[v2].push_back(v1 + n)                         ▷ O(1)
11:             adj_list[v2 + n].push_back(v1 + 2 × n)                 ▷ O(1)
12:         else
13:             adj_list[v1].push_back(v2)                             ▷ O(1)
14:             adj_list[v2].push_back(v1)                             ▷ O(1)
15:             adj_list[v1 + n].push_back(v2 + n)                     ▷ O(1)
16:             adj_list[v2 + n].push_back(v1 + n)                     ▷ O(1)
17:         adj_list[v1 + 2 × n].push_back(v2 + 2 × n)                 ▷ O(1)
18:         adj_list[v2 + 2 × n].push_back(v1 + 2 × n)                 ▷ O(1)
19:     vector<vertice> solucion ← bfs(adj_list, 0, 3 × n)             ▷ O(3n + 3m) = O(n + m)
20:     print(solucion.size() + 1)
21:     for v ∈ solucion do
22:         print(v %n)
```

Nuestra función main tiene tres partes importantes:

- El armado de la lista de adyacencias de G'.
- El llamado a BFS pasando como parámetros la lista de adyacencias anterior, tomando como *root* el nodo 0 y como target el $3n-1$. Dichos parámetros cumplen las precondiciones de BFS por el Lema 1.1 y por ser todas aristas de peso 1.
- La impresión del resultado. Acá es importantísimo notar que imprimimos los vértices módulo n pues lo que nos devuelve BFS son nodos del grafo G' y no de G. Por la Proposición 1.1 esto efectivamente constituye una solución al problema original.

1.3. Complejidad del algoritmo

La complejidad en peor caso de la solución es la complejidad de la función *main*. Omitiendo las partes de lectura y escritura de datos, tenemos que el costo de dicha función es el costo de armar el nuevo grafo más el costo de realizar BFS sobre él.

Viendo el algoritmo 2, el costo de armar el grafo es $O(n + 6m) = O(n + m)$. Vale destacar que esta complejidad es además claramente una cota inferior, pues el costo de armar el grafo nuevo depende únicamente de la cantidad de vértices y aristas, y no de las características topológicas particulares. Por lo tanto el algoritmo en general debe ser al menos $\Omega(n + m)$.

Por otra parte, observando el algoritmo 1, BFS tiene una complejidad en peor caso de

$$\begin{aligned} O(1 + 2n + 3 + 2n + (\sum_{i=0}^{n-1} d(v_i)) \times 5 + m + 1 + 2m) &= O(5 + 4n + 2m \times 5 + 3m) \\ &= O(4n + 13m) \\ &= O(n + m) \end{aligned} \tag{1}$$

Notar que en el primer término podemos escribir la sumatoria de los grados de todos los nodos debido a que en peor caso hará falta pasar por todos ellos, y por otra parte sabemos que pasamos por cada uno exactamente una vez. El segundo término resulta de agrupar y reemplazar la sumatoria por $2m$ (cosa que podemos hacer pues es una identidad válida para todos los grafos).

Luego, la complejidad asintótica del algoritmo en peor caso es $O(n+m+3n+3m) = O(4(n+m)) = O(n+m)$. Por otra parte como dijimos que también era $\Omega(n+m)$, tenemos que es $\Theta(n+m)$.

De hecho, asintóticamente también lo es en mejor caso (cuando existe un camino de longitud 3): si bien BFS puede ser $\Theta(1)$ debido a que nuestra implementación termina de buscar una vez que encuentra al nodo deseado, armar el grafo sigue siendo $\Theta(n+m)$ en cualquier caso.

1.4. Performance del algoritmo

1.4.1. Método de experimentación

2. El Imperio Contraataca

2.1. Explicación formal del problema

2.2. Explicación de la solución

2.2.1. Explicación del código

2.2.2. Pseudocódigo

2.2.3. Correctitud

2.2.4. Optimalidad

2.3. Complejidad del algoritmo

2.3.1. Complejidad en peor caso

2.3.2. Complejidad en mejor caso

2.4. Performance del algoritmo

2.4.1. Método de experimentación

3. El Retorno del ~~que te~~ Jedi

3.1. Explicación formal del problema

3.2. Explicación de la solución

3.2.1. Explicación del código

3.2.2. Pseudocódigo

3.2.3. Correctitud

3.2.4. Optimalidad

3.3. Complejidad del algoritmo

3.3.1. Complejidad en peor caso

3.3.2. Complejidad en mejor caso

3.4. Performance del algoritmo

3.4.1. Método de experimentación

4. Apéndice