



**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	4
1.1. Explicación formal del problema	4
1.2. Explicación de la solución	5
1.2.1. Nuevo grafo para el modelado	5
1.2.2. Correctitud y optimalidad	7
1.2.3. Explicación del código	8
1.3. Complejidad del algoritmo	11
1.4. Performance del algoritmo	12
1.4.1. Método de experimentación	14
2. El Imperio Contraataca	15
2.1. Explicación formal del problema	15
2.2. Explicación de la solución	15
2.2.1. Explicación del código	15
2.2.2. Pseudocódigo	15
2.2.3. Correctitud	15
2.2.4. Optimalidad	15
2.3. Complejidad del algoritmo	15
2.3.1. Complejidad en peor caso	15
2.3.2. Complejidad en mejor caso	15
2.4. Performance del algoritmo	15
2.4.1. Método de experimentación	17
3. El Retorno del que te Jedi	18
3.1. Explicación formal del problema	18
3.1.1. Ejemplos	18
3.2. Formulación Recursiva	18
3.2.1. Demostración de Correctitud	19
3.3. Pseudocódigo	20
3.3.1. Enfoque top-down vs. bottom-up	20
3.4. Complejidad del algoritmo	20
3.4.1. Complejidad en peor caso	20
3.4.2. Complejidad en mejor caso	20

3.5. Performance del algoritmo	20
3.5.1. Método de experimentación	21
4. Apéndice	22

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 v_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.

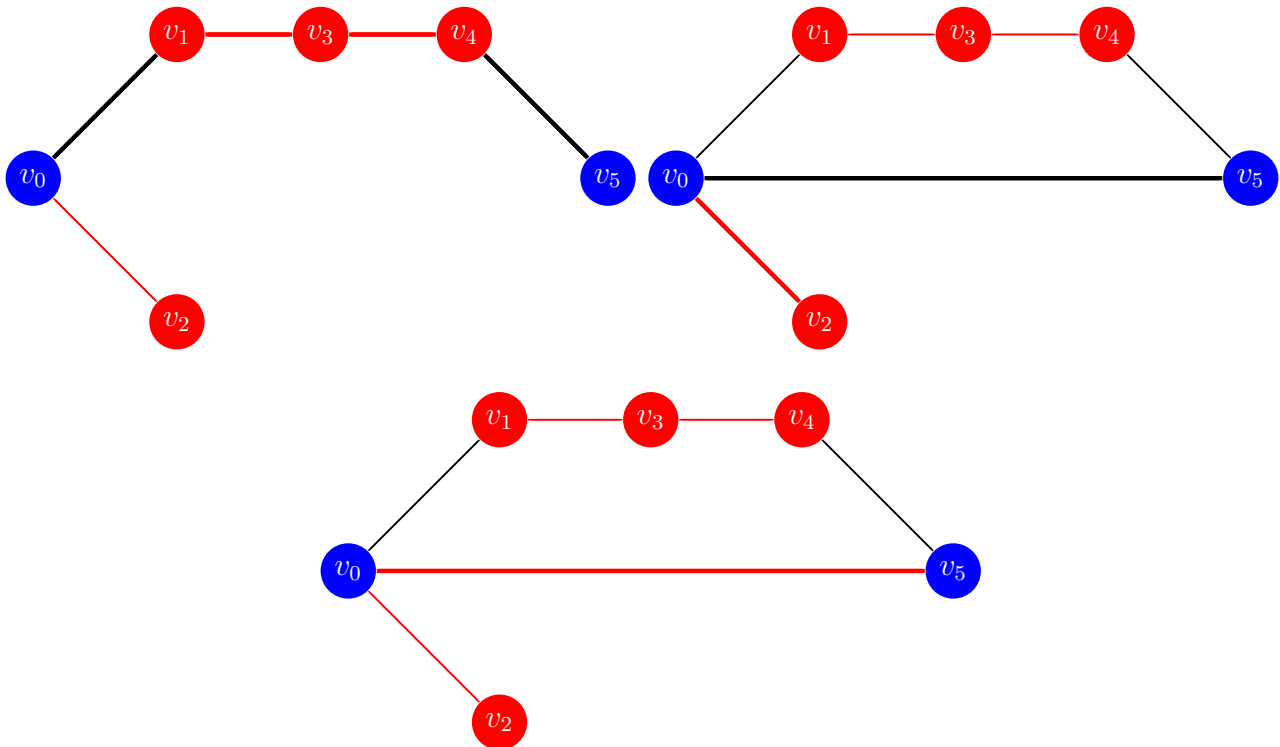


Figura 1: Ejemplos del problema. Las aristas especiales están pintadas de rojo y las comunes de negro. Las aristas que pertenecen a la solución están engrosadas. En azul distinguimos a los nodos inicial y final.

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 $f_k : V \rightarrow V_k / f_k(v_i) = v_{k \times n + i}$ 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.

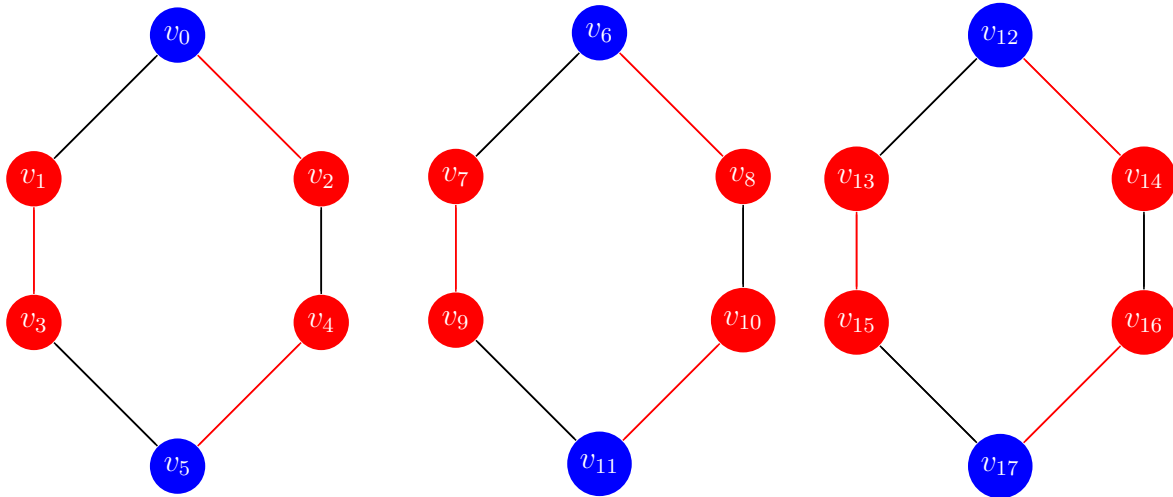


Figura 2: Tres isomorfismos del grafo original, contruidos de la forma indicada.

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

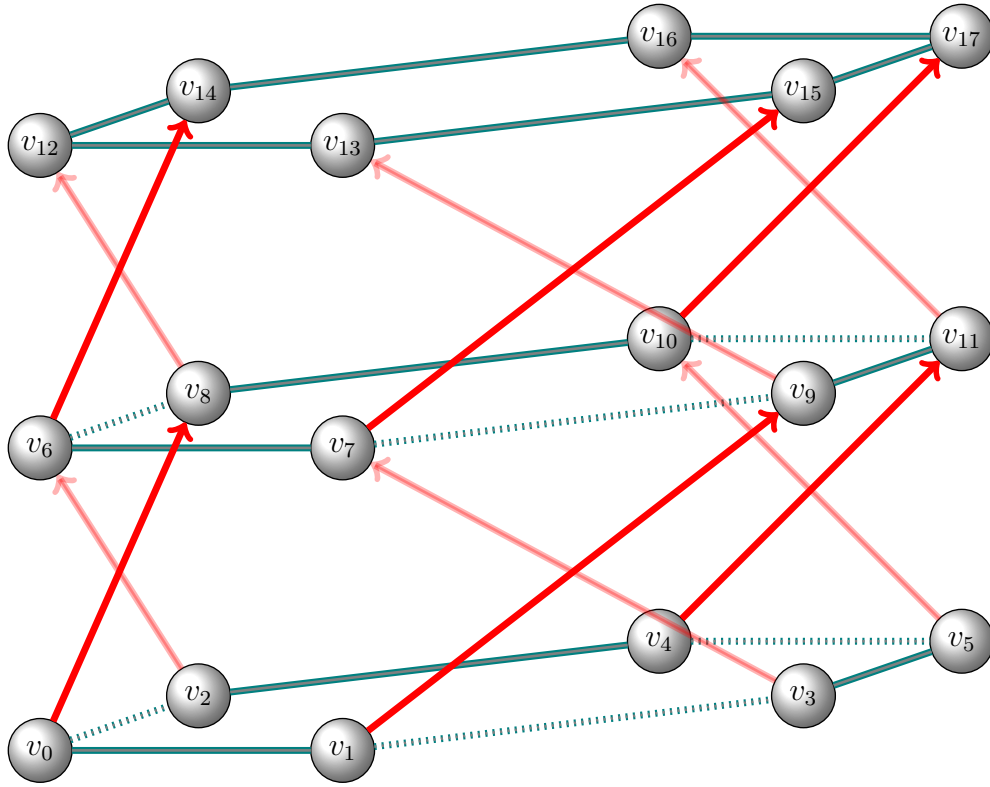


Figura 3: G' para el ejemplo dado en la figura (2). Las líneas punteadas señalan dónde solía haber una arista especial.

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

¹En 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.

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. \square

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 $P' = (v_0, v_{i_1}, \dots, v_{i_p}, v_{3n-1})$ un camino mínimo de v_0 a v_{3n-1} en G' , 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.

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 $f_k : V \rightarrow V_k / f_k(v_i) = v_{k \times n + i}$ para $k \in \{0, 1, 2\}$ son las biyecciones de los isomorfismos planteados en la sección anterior. Además una pequeña observación que usaremos fuertemente:

Observación 1.1: $G = (V, E)$. $(u, w) \in E$ pero $(u, w) \notin M$ si y solo si $(f_0(u), f_0(w)), (f_1(u), f_1(w))$ y $(f_2(u), f_2(w))$ son aristas de G' .

En cambio, $(u, w) \in M$ si y solo si los arcos $(f_0(u), f_1(w)), (f_0(w), f_1(u)), (f_1(u), f_2(w))$ y $(f_1(w), f_2(u))$ están en G' .

Ambas cosas valen por como construimos G' .

- 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 .

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

Si $v_i, v_j \in P$ son consecutivos entonces $v_{h+i} \in P'$ tiene que ser adyacente con $v_{h'+j} \in P'$ (pues son nodos consecutivos en un camino), 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 la observación 1.1, 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 .

El método propuesto a continuación para armar Q' hace uso fuertemente de la observación 1.1, pues esta nos garantiza que las aristas y arcos mencionados efectivamente existen en G' .

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 $(v_i, v_{n+j}) = (f_0(v_i), f_1(v_j))$ 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 $(v_{n+s}, v_{2n+t}) = (f_1(v_s), f_2(v_t))$ 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 $v_{2n+t} = f_2(v_t)$ hasta $v_{3n-1} = f_2(v_{n-1})$, 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 $|Q'| = |Q| < |P| = |P'|$, encontramos un camino más corto que P' en G' tal 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. \square

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 puesto que 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 ← Vacía()                                ▷  $O(1)$ 
3:   vector<int> distancia(n',  $\infty$ )                        ▷  $O(n')$ 
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  $\neg$ 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] =  $\infty$  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)                    ▷  $O(long\_sol) \subseteq O(m')$ 
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 precondition 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  $adj\_list(3 * n, VerticesAdyacentes())$  ▷  $O(3 \times n) = O(n)$ 
6:   for  $(v_1, v_2, e) \in input$  do ▷  $m$  veces
7:     if  $e = True$  then ▷  $O(1)$ 
8:        $adj\_list[v_1].push\_back(v_2 + n)$  ▷  $O(1)$ 
9:        $adj\_list[v_1 + n].push\_back(v_2 + 2 \times n)$  ▷  $O(1)$ 
10:       $adj\_list[v_2].push\_back(v_1 + n)$  ▷  $O(1)$ 
11:       $adj\_list[v_2 + n].push\_back(v_1 + 2 \times n)$  ▷  $O(1)$ 
12:     else
13:        $adj\_list[v_1].push\_back(v_2)$  ▷  $O(1)$ 
14:        $adj\_list[v_2].push\_back(v_1)$  ▷  $O(1)$ 
15:        $adj\_list[v_1 + n].push\_back(v_2 + n)$  ▷  $O(1)$ 
16:        $adj\_list[v_2 + n].push\_back(v_1 + n)$  ▷  $O(1)$ 
17:        $adj\_list[v_1 + 2 \times n].push\_back(v_2 + 2 \times n)$  ▷  $O(1)$ 
18:        $adj\_list[v_2 + 2 \times n].push\_back(v_1 + 2 \times n)$  ▷  $O(1)$ 
19:   vector<vertice>  $solucion \leftarrow bfs(adj\_list, 0, 3 \times n)$  ▷  $O(3n + 3m) = O(n + m)$ 
20:    $print(solucion.size() + 1)$ 
21:   for  $v \in solucion$  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).

En nuestro problema concreto $n' = 3 \times n$ y $m' = 3 \times m$ (por cada arista que saque estoy poniendo un arco que lo compensa).

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. En definitiva no tiene sentido hablar de mejor o peor caso pues ambos son iguales.

1.4. Performance del algoritmo

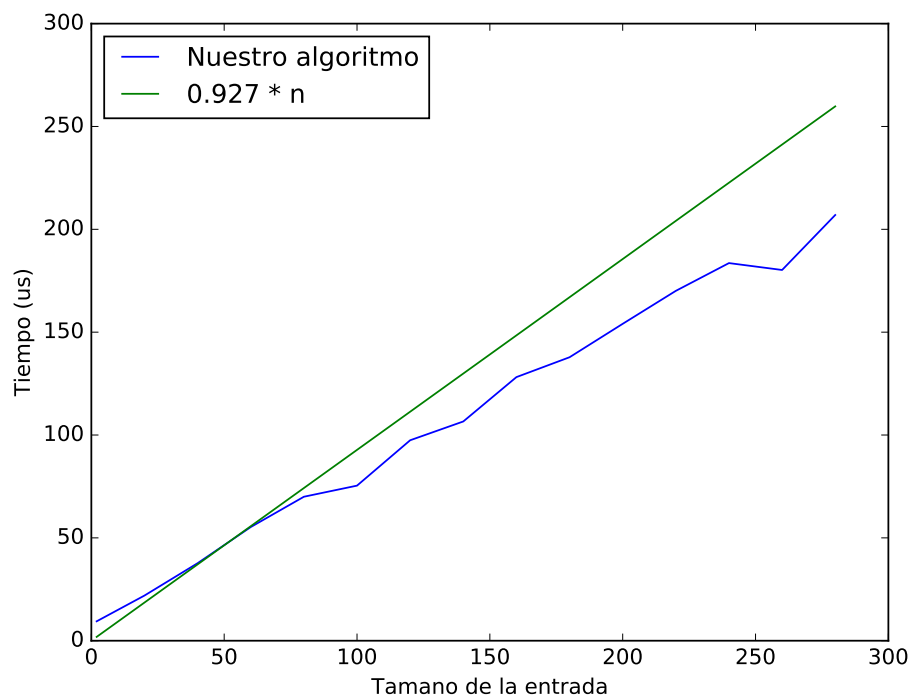


Figura 4: Tiempo que toma el algoritmo en μs para una entrada de tamaño n ($m \in O(n)$).

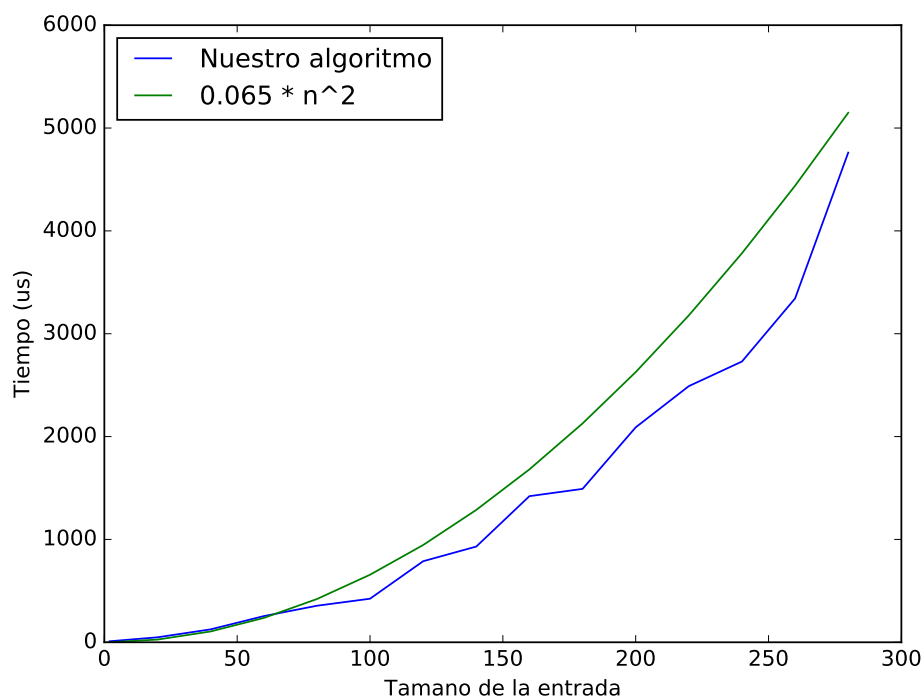


Figura 5: Tiempo que toma el algoritmo en μs para una entrada de tamaño n ($m \in O(n^2)$).

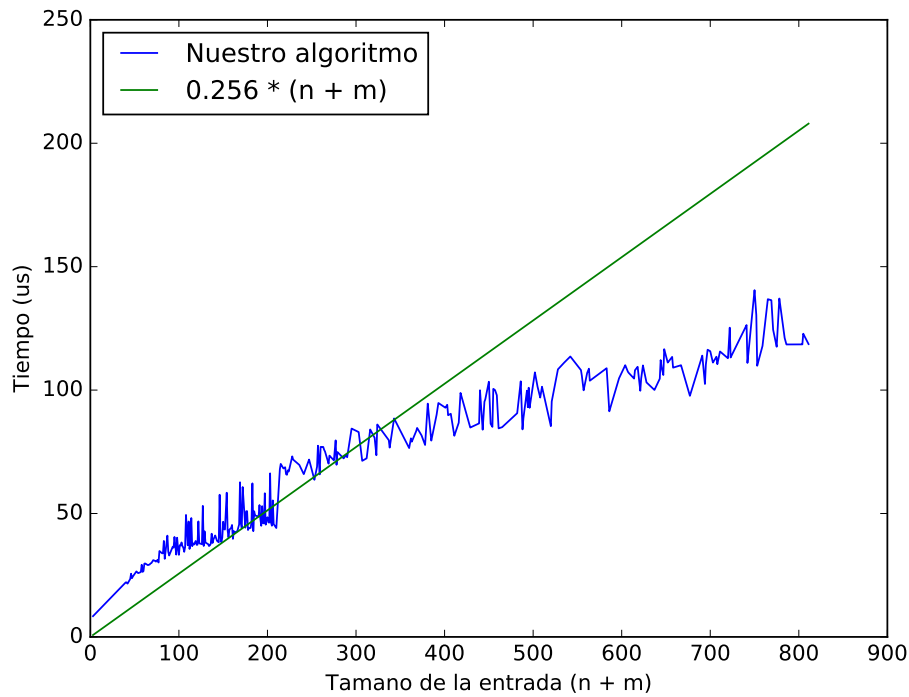


Figura 6: Tiempo que toma el algoritmo en μs para una entrada de tamaño $n + m$. m al azar entre $n - 1$ y $\frac{n(n-1)}{2}$.

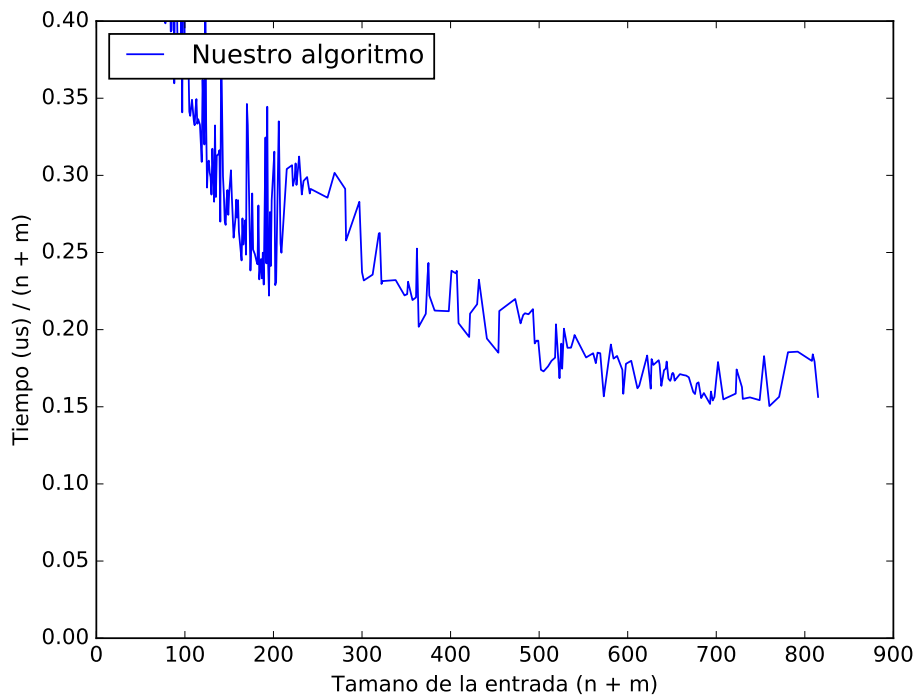


Figura 7: Tiempo que toma el algoritmo en μs dividido $n + m$ para una entrada de tamaño $n + m$. m al azar entre $n - 1$ y $\frac{n(n-1)}{2}$.

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

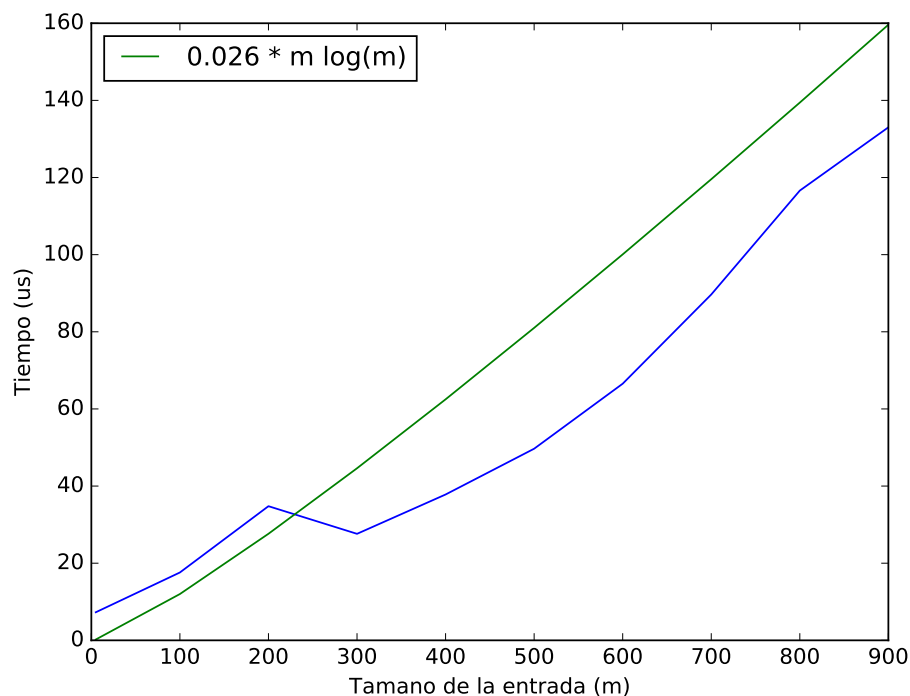


Figura 8: Tiempo que toma el algoritmo en μs para una entrada de tamaño m . n al azar.

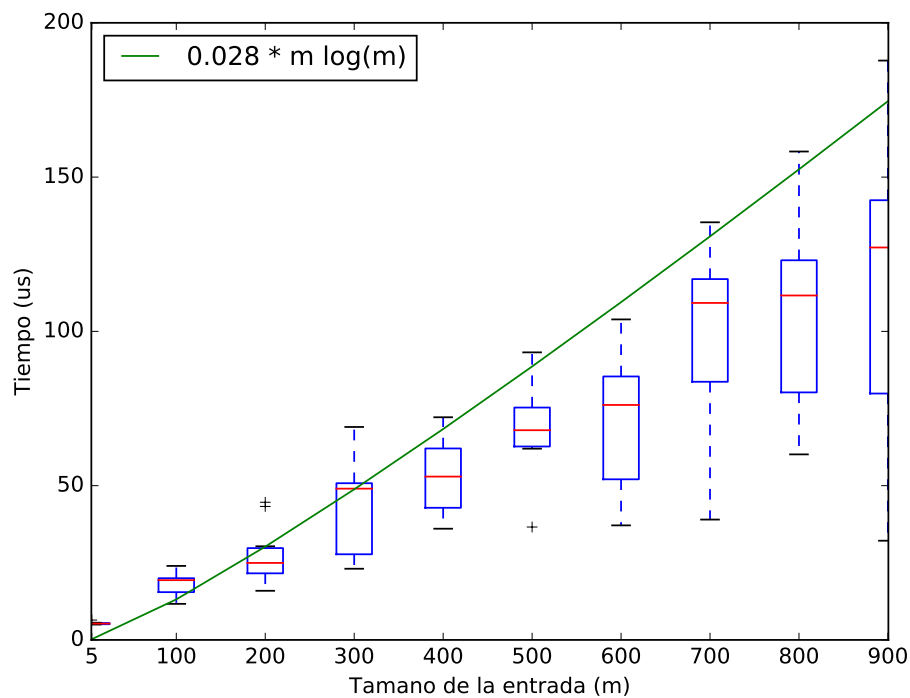


Figura 9: Tiempo que toma el algoritmo en μs para una entrada de tamaño m . n al azar. Se indican los valores del primer al tercer cuartil con un rectángulo azul y la mediana con una línea roja. El máximo y mínimo se indican con líneas negras arriba y abajo del rectángulo.

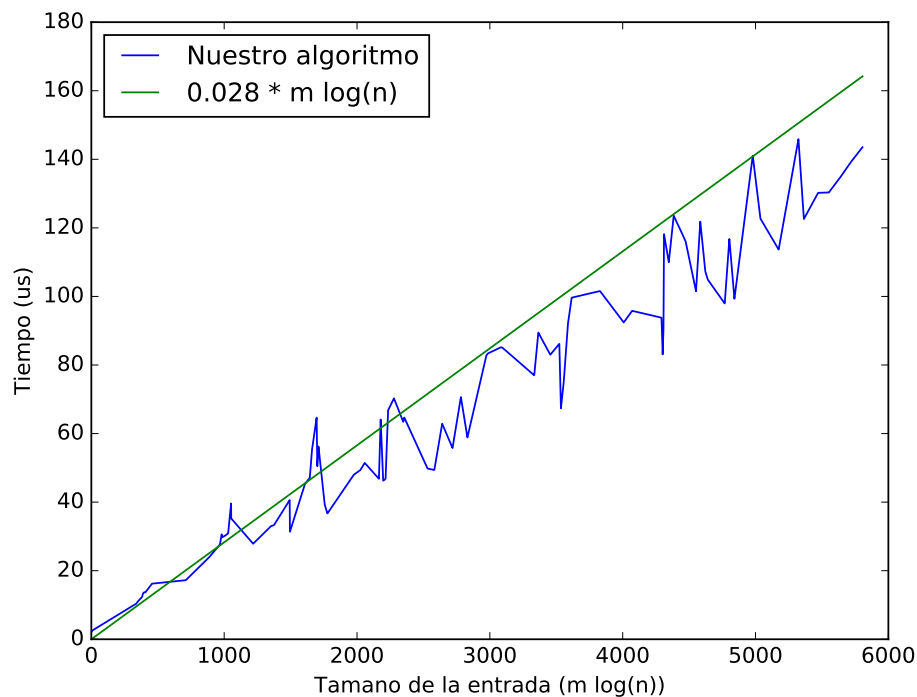


Figura 10: Tiempo que toma el algoritmo en μs para una entrada de tamaño $m \log n$.

2.4.1. Método de experimentación

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

3.1. Explicación formal del problema

Sea una matriz $M \in \mathbb{R}^{n \times m}$, donde cada posición de la matriz M_{ij} tiene un valor asociado h_{ij} . El problema consiste en calcular el camino mínimo de casilleros desde la posición (1,1) hasta la posición (N,M), donde los únicos dos movimientos posibles son:

Moverse hacia el casillero superior:

$$M_{i,j} \rightarrow M_{i+1,j}$$

ó *Moverse hacia el casillero de la derecha:*

$$M_{i,j} \rightarrow M_{i,j+1}$$

Realizar estos movimientos tiene un costo que depende de un parámetro de entrada H :

$$Costo(M_{i,j} \rightarrow M_{i+1,j}) = \begin{cases} 0 & \text{si } |h_{i,j} - h_{i+1,j}| \leq H \\ |h_{i,j} - h_{i+1,j}| - H & \text{caso contrario} \end{cases} \quad (2)$$

$$Costo(M_{i,j} \rightarrow M_{i,j+1}) = \begin{cases} 0 & \text{si } |h_{i,j} - h_{i,j+1}| \leq H \\ |h_{i,j} - h_{i,j+1}| - H & \text{caso contrario} \end{cases} \quad (3)$$

3.1.1. Ejemplos

3.2. Formulación Recursiva

Obtendremos la solución del problema usando un algoritmo de programación dinámica. Para eso, planteamos una formulación recursiva del problema. Sea la función f :

$f(i, j)$ = costo de un camino óptimo desde $M_{1,1}$ hasta $M_{i,j}$

Entonces, la solución del problema está dada por $f(n, m)$. Se propone la siguiente recursión para calcular f :

$$f(i, j) = \min \left(\begin{array}{l} f(i-1, j) + Costo(M_{i-1,j} \rightarrow M_{i,j}), \\ f(i, j-1) + Costo(M_{i,j-1} \rightarrow M_{i,j}) \end{array} \right) \quad (4)$$

con algunos casos particulares:

$$f(1, 1) = 0$$

$$f(1, j) = f(1, j-1) + Costo(M_{1,j-1} \rightarrow M_{1,j})$$

$$f(i, 1) = f(i-1, 1) + Costo(M_{i-1,1} \rightarrow M_{i,1})$$

3.2.1. Demostración de Correctitud

El caso base $f(1, 1)$ es trivial, ya que no realizo ningún movimiento.

Dada cualquier otra posición (i, j) de la matriz, el camino mínimo para llegar a ella tiene como última posición visitada o la casilla de su izquierda $(i, j - 1)$ (si existe), o la casilla de abajo $(i - 1, j)$ (si existe) por el enunciado del problema, ya que los únicos movimientos posibles son moverse hacia arriba o hacia la derecha.

Para demostrar que la función f propuesta calcula el camino mínimo hasta la casilla cualquiera $M_{i,j}$ debemos ver que se cumple el **Principio de Optimalidad**. Es decir, que dado un camino óptimo desde $M_{1,1}$ hasta $M_{i,j}$, $P_{i,j}$, entonces, el subcamino desde $M_{1,1}$ hasta el inmediato antecesor de $M_{i,j}$ ($P_{i-1,j}$ o $P_{i,j-1}$) debe ser óptimo:

Sea $P_{i,j}$ el camino óptimo desde $M_{1,1}$ hasta $M_{i,j}$. SPGE, puedo suponer que el inmediato antecesor de $M_{i,j}$ en P es $M_{i,j-1}$. Supongamos que el sub camino hasta el antecesor de $M_{i,j}$ ($P_{i,j-1}$) no es óptimo. Entonces, $\exists P_{i,j-1}$ tal que $Costo(P_{i,j-1}) < Costo(P_{i,j-1})$.

Pero entonces, puedo tomar el camino $P_{i,j-1}$ y de ahí moverme a la posición $M_{i,j}$. $Costo(P_{i,j-1}) + Costo(M_{i-1,j} \rightarrow M_{i,j}) < Costo(P_{i,j-1}) + Costo(M_{i-1,j} \rightarrow M_{i,j}) = Costo(P_{i,j})$. Pero esto es absurdo, ya que existiría un camino mejor que el óptimo hasta la posición (i, j) .

Como aplica el principio de optimalidad y sólo hay dos posibles antecesores para una determinada casilla $M_{i,j}$, para calcular el camino mínimo hasta una posición (i, j) basta con tomar el mínimo entre las dos posibilidades, tomando en cuenta el costo de realizar el último movimiento.

Para los casos borde de la matriz, en la primera columna y en la primera fila, donde sólo tengo un sólo posible antecesor (el inmediato de abajo y el inmediato de la izquierda respectivamente), el camino mínimo entonces es el camino mínimo hasta el único antecesor más el último movimiento. \square

3.3. Pseudocódigo

3.3.1. Enfoque top-down vs. bottom-up

3.4. Complejidad del algoritmo

3.4.1. Complejidad en peor caso

3.4.2. Complejidad en mejor caso

3.5. Performance del algoritmo

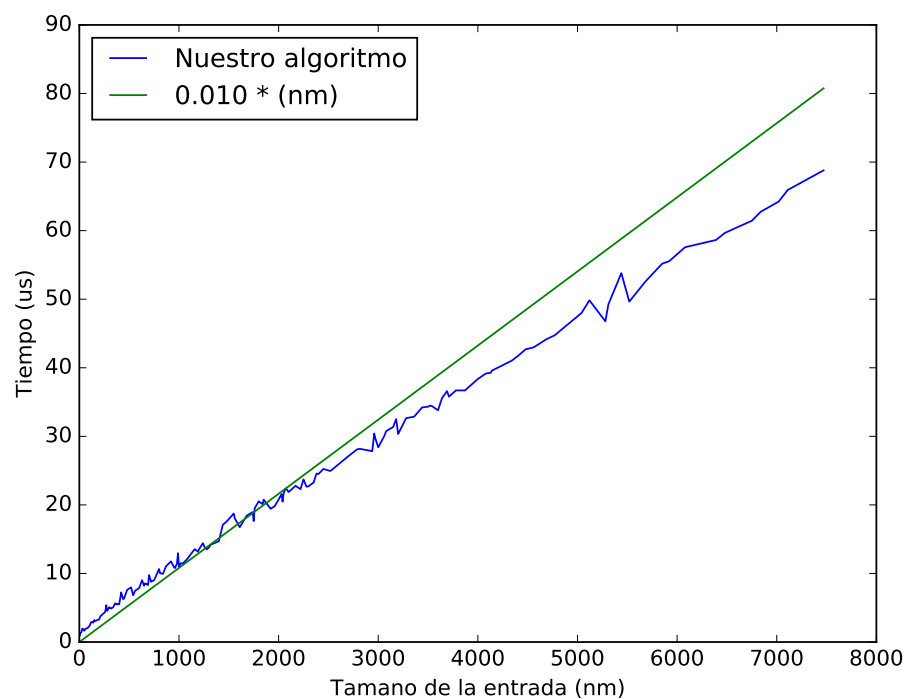


Figura 11: Tiempo que toma el algoritmo en μs para una entrada de tamaño mn .

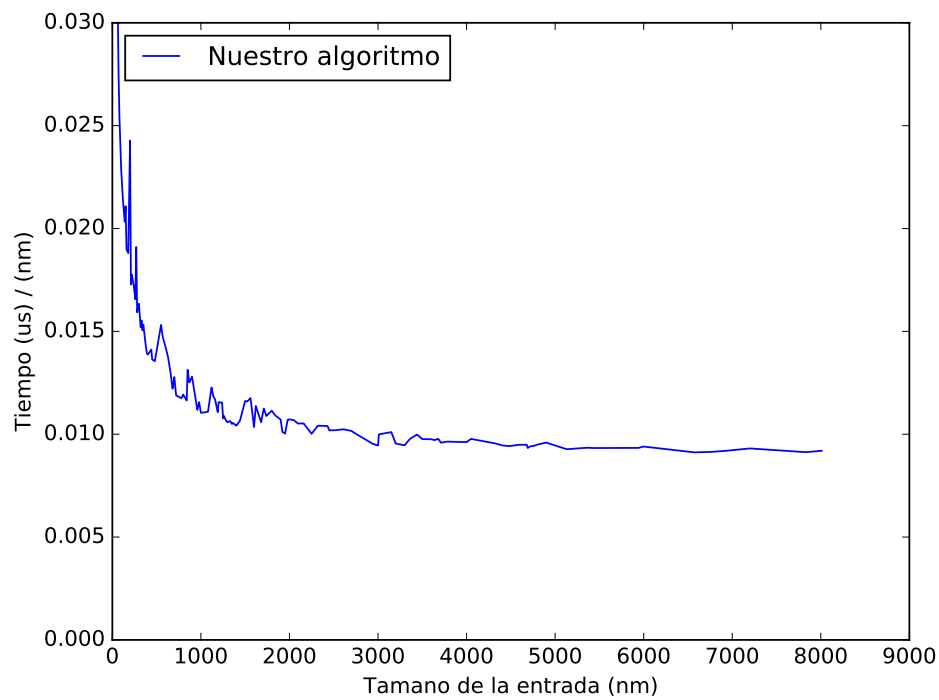


Figura 12: Tiempo que toma el algoritmo en μs dividido mn para una entrada de tamaño mn .

3.5.1. Método de experimentación

4. Apéndice