Universidad Nacional Autónoma de México

Facultad de Ciencias

Departamento de Matemáticas

México, Septiembre 2020.

Introducción al Análisis de Algoritmos

Notas de Clase

(Primera Parte, Segunda Versión)

Dra. María De Luz Gasca Soto

Licenciatura en Ciencias de la Computación,
Departamento de Matemáticas,
Facultad de Ciencias, UNAM.

Prefacio

Estas notas forman parte del material que se revisa en el curso de Análisis de Algoritmos I que se imparte en la Facultad de Ciencias de la UNAM, para la Licenciatura en Ciencias de la Computación.

He tenido la oportunidad de impartir este curso los últimos años y he estado preparando y corrigiendo las presentes notas de clase para el curso, de las cuales esta resulta ser la segunda versión.

Considero que las áreas de Análisis, Diseño y Justificación de algoritmos debe ser tomado más en serio por las personas que de una u otra manera diseñan programas o bien están involucradas con la computación.

El presente trabajo, pretende dar una panorámica general de lo que es el análisis de algoritmos. Enfatizando que el análisis, diseño y justificación de algoritmos se puede realizar de manera formal usando como herramienta a la Inducción Matemática.

La bibliografía básica, para el material presentado en estas notas, está basada en los libros:

- **Udi Manber** [13]
  Introduction to Algorithms. A Creative Approach.
- **J. Kingston** [12]
  Algorithms and Data Structures: Design, Correctness and Analysis.
- **R. Neapolitan & K. Naimipour** [16].
  Foundations of Algorithms.

Dra. María De Luz Gasca Soto
Profesor Asociado, T.C.
Departamento de Matemáticas
Facultad de Ciencias, UNAM.

Índice general

Capítulo 1

Conceptos Básicos

En este capítulo se definen algunos de los conceptos básicos más importantes relacionados con la noción de algoritmos, tales como: problema, parámetros, ejemplares, solución, entre otros.

1.1. Problemas y Algoritmos.

Un **problema** es una cuestión para la cual buscamos una respuesta.

Ejemplo 1.1 Dada una secuencia de números encontrar:
(a) el mínimo elemento; (b) el máximo elemento;
(c) el valor promedio de los elementos; (d) al elemento x en la secuencia.

Ejemplo 1.2 Dada una gráfica conexa, no dirigida con dos vértices distinguidos s y t, encontrar la ruta que una a s y a t que tenga el menor número posible de aristas.

Ejemplo 1.3 Cambiar la llanta averiada de un automovil.

Un problema puede contener *datos* (variables) que no posean valores específicos al enunciar el problema. Tales datos son llamados **Parámetros del Problema.**

Ejemplo 1.4 Para el Ejemplo 1.1 los parámetros son: la secuencia de datos y su tamaño, además del valor de x, para el inciso (d); para el Ejemplo 1.2 los parámetros son: la gráfica y los vértices s y t; para el Ejemplo 1.3 los parámetros son: el auto con la llanta averiada, una llanta en buen estado y las herramientas necesarias para cambiar el neumático.

Los parámetros determinan una clase de problemas para cada asignación de valores que puedan tomar. A cada asignación específica de valores para los parámetros se le llama **Ejemplar del Problema**. Los parámetros clasifican los problemas, podemos considerar en todos los ejemplares de tamaño n, para un número fijo n.

Una **Solución** para un ejemplar de un problema es una respuesta a la cuestión hecha por el problema.

De acuerdo con los conceptos planteados, redefiniremos los Ejemplos 1.1, 1.2, 1.3.

Ejemplo 1.1 Dada una secuencia S de n números encontrar: (a) el mínimo elemento; (b) el máximo elemento; (c) el valor promedio de los elementos; (d) al elemento x en la secuencia.

Parámetros: Sea $S = [10, 7, 11, 5, 13, 8]$, $n = 6$. (Ejemplar del problema)
Soluciones. (a) El mínimo elemento es: 5; (b) el máximo elemento es: 13; (c) el valor promedio es: 9; (d_1) Sea $x = 20$, x no está en la secuencia; (d_2) Sea $x = 11$, x sí está en S; (d_3) Sea $x = 9$, x no está en la secuencia.

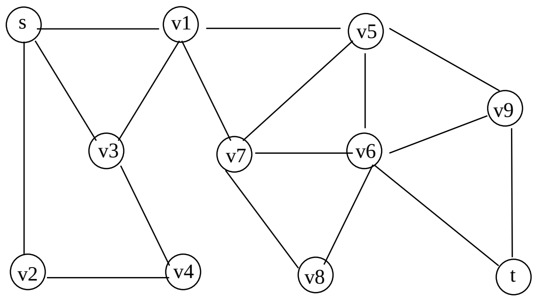


Figura 1.1: Gráfica del Ejemplo 1.2.

Ejemplo 1.2 Dada una gráfica $G = (V, A)$ conexa, no dirigida con dos vértices distinguidos s y t, encontrar la ruta que una al vértice s con t y que tenga el menor número posible de aristas. Se tiene que V es el conjunto de vértices con número de vértices $|V| = n$. A es el conjunto de aristas con número de aristas $|A| = m$.

Parámetros: Sea $G = (V, A)$ la gráfica de la Figura 1.1, tenemos que:
$V = \{s, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, t\}$, $n = 11$.
$A = \{(s, v_1), (s, v_3), (s, v_2), (v_1, v_7), (v_1, v_5), (v_2, v_4), (v_3, v_4), (v_7, v_5), (v_6, v_5), (v_5, v_9), (v_7, v_6), (v_8, v_7), (v_8, v_6), (v_6, v_9), (v_9, t), (v_6, t)\}$; $m = 16$.
Soluciones: $\mathcal{P}_1 : s \to v_1 \to v_7 \to v_6 \to t$;
$\mathcal{P}_2 : s \to v_1 \to v_5 \to v_6 \to t$; $\mathcal{P}_3 : s \to v_1 \to v_5 \to v_9 \to t$.

Podemos encontrar *fácilmente* la solución de un problema cuando el ejemplar es *pequeño* o es *manipulable*. Sin embargo, un ejemplar puede tener un valor muy grande para n y ya no resuta tan *natural* utilizar métodos visibles o intuitivos, se requiere entonces formalizar un método (proceso) para solucionar el problema independientemente de su tamaño. Se debe especificar paso a paso el método que produzca la solución al problema para cada uno de los ejemplares que éste tenga.

Un proceso que se describe paso a paso es llamado **Algoritmo**. Un **Algoritmo es Correcto** si garantiza la creación de una respuesta correcta para cada ejemplar del problema. Si diferentes respuestas son equivalentemente correctas, el algoritmo deberá ser capaz de producir cualquiera de ellas o generarlas todas.

Listado 1 Algoritmo Distancias

```
Distancias(G,s)
    for (cada  vertice v de G)
       do {
           Color[v] <-- Negro
           d[w] <-- MaxInt       }
    d(s) <-- 0
    Color[s] <-- Gris.
    while (exitan vertices con color Gris) do{
         Sea un vertice v de color Gris  con etiqueta minima
            Entonces   Color[v] <-- Blanco
         for  (todos los vertices w vecinos de v pintados de Negro)
            do{      d[w] <-- d(v) + 1
                  Color[v] <--  Gris.        }
   }
```

Ejemplo 1.5 Algoritmos para el Ejemplo *1.1.*

Algoritmo 1a.- Encontrar el mínimo elemento en la secuencia S.
Sea m el primer elemento de S. Comparamos a m con cada elemento s de S, cuando s sea menor que m entonces cambiamos el valor de m por el de s; hasta terminar con los elementos en la secuencia.

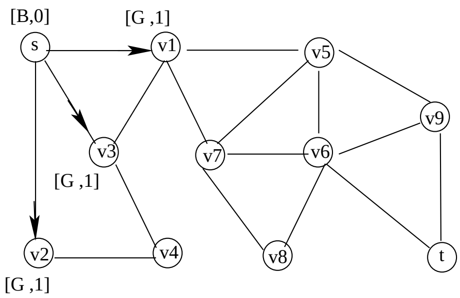
Algoritmo 1b.- Encontrar el máximo elemento en la secuencia S.
Sea M el primer elemento de S. Comparamos a M con cada elemento s de S, cuando s sea mayor que m entonces cambiamos el valor de M por el de s; hasta terminar con los elementos en la secuencia.

Algoritmo 1c.- Calcular el valor promedio.
Sea s el valor de la suma de los n elementos de la secuencia S. El valor promedio resulta ser $p = s/n$.

Algoritmo 1d.- Buscar al elemento x en S.
Iniciando con el primer elemento en la secuencia S, comparamos a x con cada elemento en la secuencia hasta que x sea igual a alguno de los elementos o hasta que terminemos de revisar toda la secuencia.

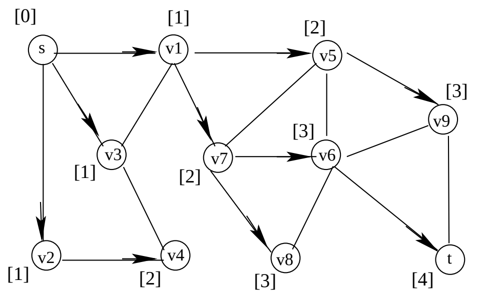
Ejemplo 1.6 Algoritmo para el Ejemplo *1.2.*

El Algoritmo Distancias recibe una gráfica $G = (V, A)$ conexa y un vértice inicial s, a partir del cual se inicia la búsqueda. El objetivo es encontrar la distancia mínima, en número de aristas, entre el vértice s y el resto de los vértices en G. La etiqueta $d(v)$ denotará la distancia mínima de s a v.
Usaremos tres colores: Negro para los vértices no revisados; Gris para los vértices en revisión; y Blanco para los que ya alcanzaron la mínima distancia.
Al inicio, a todos los vértices se les da color negro, al revisarlo se le asigna el color gris, una vez encontrada la distancia mínima al vértice se le asignará el color blanco.
El Listado 1 nos ilustra el algoritmo.



vi	s	1	2	3	4	5	6	7	8	9	t
color	B	G	G	G	N	N	N	N	N	N	N
d	0	1	1	1	*	*	*	*	*	*	*

(a)



vi	s	1	2	3	4	5	6	7	8	9	t
color	B	B	B	B	B	B	B	B	B	B	B
d	0	1	1	1	2	2	3	2	3	3	4

(b)

Figura 1.2: Resultado de aplicar el Algoritmo Distancias

Ejemplo 1.7 Aplicación del algoritmo Distancias.

Considere la gráfica de la Figura 1.1, aplicaremos el algoritmo a partir del vértice s. Al inicio, todos los vértices son coloreados con N, negro, y tienen distancia infinito. Actualizamos el color del vértice s con G, gris, y su distancia con 0. Como es el único vértice gris se colorea de B, blanco, y se revisan sus vecinos. Ahora tenemos que los vértices v_1, v_2, v_3 tienen distancia 1 y color G. Se toma un vértice gris, digamos v_2, su único vecino con color N es v_4, lo coloreamos de gris y su distancia será 2. Tomamos otro vértice gris, digamos v_1 y lo revisamos, sus vecinos negros son v_7 y v_5, se colorean con G y su distancia será 2. Así continuamos hasta terminar con los vértices grises. La Figura 1.2(a) muestra como queda la gráfica y sus atributos después de la primera iteración, y la Figura 1.2(b) nos ilustra el resultado final de aplicar el algoritmo Distancias.

Nótese que este algoritmo no calcula o construye ninguna ruta, sólo deja etiquetado a cada vértice de la gráfica, y a partir de ahí es posible construir la ruta.

Ejemplo 1.8 Algoritmos para Rutas.

A continuación describimos algunas ideas para realizar un algoritmo que construya la ruta, utilizando los resultados otorgados por el algoritmo Distancias.

Ejemplar: Sea $G = (V, A, d)$ la gráfica que regresa el algoritmo Distancias, donde $d : V \rightarrow \mathbb{N}$ es la función que asigna el valor de distancia a cada vértice de G. Sea s el vértice donde inició el algoritmo Distancias. Sea t uno de los vértices más alejado de s, es decir, que posea la mayor $d(v), \ \ \forall\, v \in V$.

Algoritmo Ruta_1. A partir del vértice t se construye una ruta tomando cada vez un vértice cuya distancia sea una unidad menor al último vértice tomado.

Algoritmo Ruta_2. A partir de t se construye la ruta tomando cada vez una arista que lleve a un vértice cuya distancia sea una unidad menor al último vértice tomado.

Algoritmo Ruta_3. A partir de t se construye la ruta. Se define $v = t$, cada vez la arista (w, v) tal que $d(w) = d(v) - 1$, entonces se agrega la arista a la ruta P y se repite el proceso hasta llegar al vértice s.

Listado 2 Algoritmo Ruta3

```
Ruta3(s,t)
   P <-- null
   while (v != s) do{
     v <-- t
     Sea a=(w,v) una arista tal que d(w) = d(v) - 1
     P <-- P + a
     v <-- w
   }
   Return P
```

Ejemplo 1.9 Aplicación del algoritmo Ruta_3.

Considere la gráfica de la Figura 1.2(b), iniciamos en el vértice t, así que sea $v = t$, tomamos la arista $a = (v_6, v)$, la anexamos a la ruta: $P = \{(v_6, t)\}$. Ahora $v = v_6$, tomamos la arista $a = (v_7, v_6)$, la agregamos a la ruta: $P = \{(v_7, v_6, v), (v_6, t)\}$. Entonces $v = v_7$, tomamos la arista $a = (v_1, v_7)$, la agregamos a la ruta, quedando $P = \{(v_1, v_7), (v_7, v_6), (v_6, t)\}$. Tenemos que $v = v_1$, tomamos la arista (s, v_1). Finalmente, $v = s$ y hemos terminado. La ruta construida es $P = \{(s, v_1), (v_1, v_7), (v_7, v_6), (v_6, t)\}$. La Figura 1.3 muestra la ruta tomada, sobre la gráfica dada por el Algoritmo Distancias.

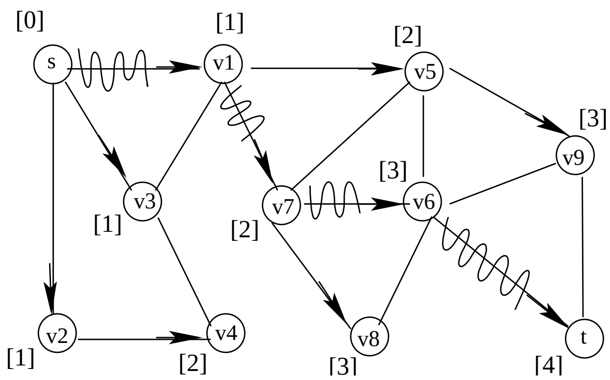


Figura 1.3: Resultado de aplicar el Algoritmo Ruta_3

1.2. Características de los Algoritmos.

Hemos visto que un algoritmo es una descripción de *cómo* un problema específico puede ser resuelto. Un algoritmo es usado para resolver un problema sencillo, o más comúnmente, para solucionar una clase general de problemas similares. Un algoritmo debe poseer las siguientes características:

1. **Especificación Precisa del Ejemplar.** El número y tipo de datos del ejemplar (datos de entrada) debe estar bien especificado, al igual que las condiciones iniciales que estos parámetros deben satisfacer para obtener una ejecución exitosa. A tales condiciones iniciales las llamaremos PreCondiciones.

2. **Especificación Precisa cada Instrucción.** Cada paso de un algoritmo debe estar definido con total precisión, no debe haber ambigüedad sobre las acciones a realizar en cada instrucción.

3. **Integridad.** El algoritmo debe ser correcto.
   Se espera que el algoritmo resuelva un problema, por lo cual se debe demostrar formalmente que, en efecto, soluciona el problema para el que fue creado.

4. **Terminación, en cuanto a tiempo de ejecución.** Se debe garantizar que para todo ejemplar, es decir para cualquier valor de una entrada, el algoritmo termina después de un número finito de pasos. Será conveniente proveer de una cota superior y argumentar que el algoritmo siempre termina en un número finito de pasos menor que la cota superior propuesta. Usualmente, tal cota es dada como una función de algunos valores de la entrada.
   Por ejemplo, si la entrada consiste de los valores enteros n y m se podría decir para un algoritmo particular que éste terminará en menos de $(n + m)$ pasos.

5. **Descripción del resultado.** El resultado o efecto del algoritmo debe estar completamente caracterizado. Es decir, el resultado puede ser expresado como una serie de condiciones a las cuales denominaremos PostCondiciones.

1.3. Tipos de Problemas.

Podemos clasificar los problemas computacionales en *Problemas de Requerimientos (Problem Requirements)* y *Problemas de Dificultad (Problem Difficulty).*

Los Problemas de Requerimientos se dividen en seis problemas computacionales:

Problemas de Búsqueda. Encontrar los valores X, en los datos de entrada, que satisfagan la propiedad P.

Problemas de Estructura. Transformar los datos de entrada para satisfacer la propiedad P.

Problemas de Construcción. Construir un dato X que satisfaga la propiedad P.

Problemas de Optimización. Encontrar, en los datos de entrada, la mejor X que satisfaga la propiedad P.

Problemas de Decisión. Decidir si una entrada satisface o no la propiedad P.

Problemas Adaptivos. Mantener la propiedad P todo el tiempo.

Se definen cuatro categorias para clasificar a los Problemas según su dificultad:

Problemas Conceptualmente Dificiles. No se tiene un algoritmo que resuelva el problema, ya que no es posible entender suficientemente el problema.

Problemas Analíticamente Dificiles. Se tiene el algoritmo que resuelve el problema, pero no se sabe cómo analizarlo ni cómo se resuelve cada ejemplar.

Problemas Computacionalemente Dificiles. Se tiene el algoritmo, el cual es posible analizar, pero el análisis indica que resolver un ejemplar se toma años. Esta categoria se divide en dos grupos:
(1) Problemas que se *sabe* son computacionalmente dificiles y
(2) Problemas que se *sospecha* son computacionalmente dificiles.

Problemas Computacionalmente sin solución. No se tiene un algoritmo que resuelva el problema, ya que no es factible construir tal algoritmo.