3 MÉTODOS GENERALES DE SOLUCIÓN DE PROBLEMAS

En el Capítulo 1 se introdujo como notación para especificación de problemas algo como

```
[Ctx {Pre} ? {Pos}]
```

En [Car91] se esboza una metodología de programación, guiada por objetivos, cuyas técnicas corresponden a decidir qué clase de instrucción de un lenguaje de programación como GCL (i.e., imperativa) sirve para solucionar un problema. En principio, esta metodología está orientada a satisfacer la especificación, de modo que para tener en cuenta restricciones de eficiencia o consumo de recursos, debería empezarse por complementar la notación de especificación y describir una manera sistemática de involucrar estas restricciones dentro de la búsqueda de soluciones.

Ya en [Car91] se criticaba el método de empezar "desde cero" a buscar soluciones para un problema de programación. La experiencia del programador puede capitalizarse con el uso de paradigmas, que no son otra cosa que programas genéricos que resuelven familias determinadas de problemas. Otra posibilidad es utilizar técnicas de solución de problemas, algo menos esquemáticas que los paradigmas, pero, en medio de su generalidad, útiles para atacar clases específicas de problemas.

En este capítulo se estudiarán superficialmente las siguientes técnicas de solución de problemas:

- Dividir y conquistar
- Programación dinámica
- Cambio de representación
- Heurísticas
- Algoritmos de aproximación

Las clases de problemas que se pueden resolver usando estas técnicas no son disyuntas ni, mucho menos, cubren todo posible problema soluble. De hecho, cualquier intento de organización de lo que podría ser la disciplina de solución de problemas está condenado a tratar el conocimiento de

manera incompleta, por el solo hecho de que el mismo problema de "solucionar problemas" no está bien definido.

Algunas de las técnicas esbozadas volverán a tratarse, con más profundidad, en capítulos posteriores. El estudio de algoritmos de aproximación no se profundiza, debido a que un buen tratamiento quedaría por fuera de los objetivos de este libro.

3.1 DIVIDIR Y CONQUISTAR

Dividir y conquistar es una técnica muy utilizada en informática, si se tiene en cuenta el que muchos de los objetos informáticos (estructuras de datos, programas) son construidos de manera inductiva o recurrente. La técnica consiste en partir un problema en subproblemas, solucionarlos y unir las soluciones para construir la solución del problema original. La partición en subproblemas está, en muchos casos, relacionada con la forma de construcción de los objetos.

Los subproblemas pueden ser o no del mismo tipo que el original. Si no lo son, la complejidad temporal es la suma de las complejidades de las partes más la de la unión de las soluciones. En otro caso, la complejidad temporal -peor caso o promedio- satisface usualmente una recurrencia de la forma

$$\begin{aligned} T(n) &= O(1) && , n \le n_0 \\ &= k\, T(\tfrac{n}{k}) + g(n) && , n > n_0 \end{aligned}$$

donde k es el número de subproblemas y $g(n)$ el costo de la reunión de las soluciones. Este costo puede pagarse al principio de un algoritmo solución, en lo que sería un pre proceso.

Ejemplo A: Quicksort

Se puede argumentar que el comportamiento promedio de un algoritmo de *quicksort*, contando como operación básica el número de comparaciones, satisface una recurrencia de la forma:

$$\begin{aligned} \overline{T}(n) &= 0 && , n \le 1 \\ &= 2\,\overline{T}(\tfrac{n}{2}) + n-1 && , n > n_0 \end{aligned}$$

de donde se obtiene que

$$\overline{T}(n) = O(n \log n).$$

Sin embargo, el peor caso satisface una recurrencia de la forma:

$$\begin{aligned} T(n) &= 0 && , n \le 1 \\ &= T(n-1) + n-1 && , n > 1 \end{aligned}$$

de modo que:

$$T(n) = O(n^2)$$

‖

Ejemplo B: Búsqueda binaria

En una búsqueda binaria se resuelve el problema de encontrar la dirección de un objeto, en una estructura de datos ordenada, donde los datos pueden ser accedidos directamente (i.e., con costo constante). El método consiste en comparar el objeto buscado con el más cercano a la mitad del intervalo de incertidumbre. De pendiendo de que la comparación dé un resultado de "menor" o "mayor", se elige el subintervalo que debería contener el objeto buscado.

El tamaño del problema es n, la longitud del intervalo inicial de incertidumbre. La operación básica es la comparación de dos objetos, con costo $O(1)$. De esta manera, resulta fácil estimar $T(n)$ el peor caso:

$$T(n) = 1 \quad , n=1$$
$$= T(\tfrac{n}{2}) + 1 \quad , n>1$$

En este caso, se llega a que $T(n) = O(\log n)$.

‖

3.2 PROGRAMACIÓN DINAMICA

La *programación dinámica* es una técnica de la Investigación de Operaciones que busca reducir el tiempo de cálculo de una solución recurrente, usualmente inspirada en "dividir y conquistar", evitando, en lo posible, la repetición de cálculos costosos y la realización de otros innecesarios.

En general, se trata de transar espacio por tiempo: el espacio adicional deberá guardar estructuras de datos que almacenen los cálculos que no se quieren repetir. Un refraseo de esta idea es una forma de "poner a trabajar al invariante", esto es, usar variables (de tipo tan complejo como se requiera) cuyo significado para el programa se define en aserciones intermedias y, más exactamente, dentro de invariantes que describan la ejecución de ciclos del algoritmo.

Ejemplo A: Fibonacci

Los algoritmos del Capítulo 2 tienen las siguientes complejidades:

$T(n) = O(\phi^n)$

$S(n) = O(1)$

La alta complejidad temporal se debe al hecho de no recordar los cálculos ya efectuados. Entonces, se puede pensar en una mejora que consista en guardar los resultados de cálculos ya efectuados en una estructura de datos:

Para calcular F_n, calcular $F_0, \ldots, F_{n-1}$, guardarlos y recordar que, para $n>1$: $F_n = F_{n-2} + F_{n-1}$. Para este algoritmo (operación básica: suma):

$$T_3(n) = 0 \quad , n\leq 1$$
$$= 1 + T_3(n-1) \quad , n>1$$

es decir:

$T_3(n) = O(n)$

y además:

$S_3(n) = n$

Ejemplo B: Más Fibonacci!

Todavía se puede ir algo más lejos:

Para calcular F_n, calcular $F_0, \ldots, F_{n-1}$, guardar sólo los dos últimos valores y recordar que, para $n>1$:

$$F_n = F_{n-2} + F_{n-1}$$

Para este algoritmo:

$$T_4(n) = 0 \quad , n \leq 1$$
$$= 1 + T_4(n-1) \quad , n>1$$

es decir:

$T_4(n) = O(n)$

pero, además:

$S_4(n) = O(1)$

∎

3.3 CAMBIO DE REPRESENTACIÓN

Un método fundamental en la solución de problemas consiste en refrasear el planteo y el dominio del discurso del problema inicial en términos de estructuras de información que sean fácilmente manipulables. La facilidad de manipulación depende de los intereses que se tengan: el replanteo del problema puede hacer que la solución sea barata en algún sentido. Por ejemplo, las nuevas estructuras de datos pueden ser recorridas eficientemente, puede haber teoría que permita hacer cálculos más rápidos, etc.

Al hacer un cambio de representación se podrá decir que el prblema en cuestión se *reduce* a otro, que se espera sea más simple. Con esta idea, la posibilidad de una reducción resulta interesante en la medida que la transformación en sí sea barata, pues, de hecho, el costo de la solución sigue siendo el de la reducción sumado al del problema transformado.

La noción de reducción da lugar a pensar en jerarquías de problemas, ordenados por dificultad. Si las reducciones son poco costosas, se puede argumentar que un problema es "por lo menos, tan fácil" como el problema al que se transformó. Al lado de la noción de "más fácil" surge, en forma natural, la noción de "equivalencia (de dificultad)" entre problemas: dos problemas son equivalentes cuando uno es más fácil que el otro y viceversa.

Ejemplo A: Uso de grafos

Una técnica muy utilizada en solución de problemas consiste en plantear las cosas dentro de la teoría de grafos. El nuevo problema puede, por ejemplo, corresponder a la búsqueda de un camino, de un ciclo, etc. en un grafo.

Normalmente se utilizan grafos dirigidos de la forma `G(V,E)` o `G(V,E,c)` donde:

- `V` es un conjunto de *vértices* o *nodos*, usualmente finito.
- `E` $\subseteq$ `V x V` es un conjunto de *arcos*. Cuando $\langle$`i,j`$\rangle \in$ `E` se dice que hay un arco del vértice `i` al vértice `j`.
- `c: E` $\rightarrow$ `L` es una función que *etiqueta* los arcos con elementos de un conjunto `L`. Las etiquetas pueden denotar costos de transitar, capacidad de transmitir información, etc., por el arco correspondiente.

La solución que se da al siguiente problema ilustra el uso de la representación con grafos:

Problema: Se tienen dos vasijas, de capacidad `7` y `5` lts., respectivamente. Se quiere dejar exactamente `4` lts. en una de las vasijas.

En primer término se define un grafo `G(V,E)`. La idea es representar con los vértices los posibles estados de las vasijas, teniendo en cuenta su contenido de agua; por su parte, los arcos representan transiciones posibles entre estados. De este modo:

`V := {<u,v> | 0≤u≤7, 0≤v≤5}`

Para definir los arcos se utilizará la notación

`<u,v>` $\rightarrow_{et}$ `<u',v'>` "acción representada por et"

para significar que se puede pasar del estado `<u,v>` al estado `<u',v'>` mediante la acción representada por la etiqueta `et`. Más exactamente, se definen los siguientes arcos, para `<u,v>`$\in$`E`:

`<u,v>` $\rightarrow_{L7}$ `<7,v>` "llenar la vasija de 7"

`<u,v>` $\rightarrow_{L5}$ `<u,5>` "llenar la vasija de 5"

`<u,v>` $\rightarrow_{V7}$ `<0,v>` "vaciar la vasija de 7"

`<u,v>` $\rightarrow_{V5}$ `<u,0>` "vaciar la vasija de 5"

`<u,v>` $\rightarrow_{C75}$ `<7,u+v-7>` "completar la vasija de 7 con la de 5"
si `0≤u+v-7≤5`

`<u,v>` $\rightarrow_{C57}$ `<u+v-5,5>` "completar la vasija de 5 con la de 7"
si `0≤u+v-5≤7`

`<u,v>` $\rightarrow_{T75}$ `<0,u+v>` "transvasar la vasija de 7 en la de 5"
si `0≤u+v≤5`

`<u,v>` $\rightarrow_{T75}$ `<u+v,0>` "transvasar la vasija de 5 en la de 7"
si `0≤u+v≤7`

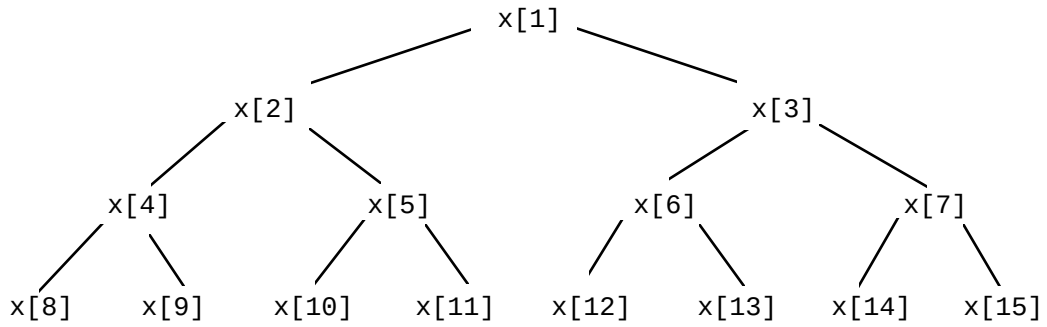
Puesto que se puede pensar que se comienza conl las vasijas vacías, el problema refraseado en términos de grafosconsiste en encontrar un camino de <0,0> a <4,x>, para algún x tal que <4,x>∈ V. Por ejemplo:

$$\langle 0,0\rangle \rightarrow_{L7} \langle 7,0\rangle \rightarrow_{C57} \langle 2,5\rangle \rightarrow_{V5} \langle 2,0\rangle \rightarrow_{T75} \langle 0,2\rangle \rightarrow_{L7} \langle 7,2\rangle \rightarrow_{C57} \langle 4,5\rangle$$

‖

Ejemplo B: Árboles binarios como arreglos

Un árbol binario completo de n nodos puede representarse con un arreglo x[1..n], de acuerdo a la siguiente figura (n=15)



La representación es muy eficiente para recorrer la estructura arborescente sin otra ayuda que el acceso directo que ofrecen los índices del arreglo. De hecho, se tienen las siguientes correspondencias:

nodo i	:	x[i]
raíz	:	x[1]
hijo izquierdo de i	:	x[2i]
hijo derecho de i	:	x[2i+1]
padre de i	:	$x[\lfloor \frac{i}{2} \rfloor]$
i es hoja	:	2i>n

Si el árbol no fuera completo, se puede continuar con la misma correspondencia, asignando valores nulos en los elementos del arreglo que no deban corresponder a nodos del árbol. En este caso, el precio que se paga por el rápido cálculo de padres e hijos de los nodos es, precisamente, el posible desperdicio de espacio que no se ocupa en el arreglo, que, de todas maneras, debe reservarse.

3.4 ALGORITMOS DE APROXIMACIÓN

En la práctica se presentan, con más frecuencia de la que se podría desear, problemas cuya solución no es viable, i.e., no se conocen métodos rápidos de solución. Como, de todas maneras, estos problemas siguen existiendo, se opta por soluciones aproximadas cuyo cálculo se pueda llevar a acabo dentro de los gastos de recursos admisibles.

Debe esperarse un compromiso eficiencia : precisión. Es decir, si se quiere más precisión se deben sacrificar más recursos. Las teorías que miden este compromiso son complejas y escapan del

alcance de estas notas. Como ya se anunció, se mencionarán, superficialmente, ejemplos e ideas que pueden utilizarse en esta clase de enfoques.

Los algoritmos probabilísticos podrían, a su vez, considerarse como algoritmos de aproximación, en el sentido de que dan respuestas -quizás exactas- con algún grado de incertidumbre. Más aun, pueden considerarse algoritmos probabilísticos que aproximen las soluciones exactas.

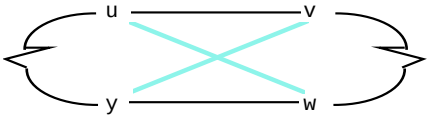
Un algoritmo puede usar *heurísticas*, es decir, reglas de decisión que justifican una intuición. Es decir, la razón para usar una regla puede basarse en consideraciones algo informales o incompletas. Este el caso de un programa que hace "optimización local" buscando un óptimo global; una tal estrategia no siempre es buena, pero muchas veces es la mejor[1].

Otra técnica de aproximación, conocida como *acondicionamiento*, consiste en transformar una solución aproximada en otra, mejorando una medida de optimalidad, y repetir este proceso hasta alcanzar algún grado de aproximación satisfactorio. Por ejemplo, puede ser deseable llegar a un punto fijo, i.e., una solución no mejorable "localmente".

Ejemplo: El agente viajero (acondicionamiento)

El problema del agente viajero (AV), consiste en encontrar un circuito de costo mínimo que pase por todos los nodos de un grafo completo (arcos entre cada par de nodos), con valores en los arcos que representan los costos de transiciones entre nodos. Los arcos se consideran no dirigidos, i.e., se pueden transitar en cualquier sentido.

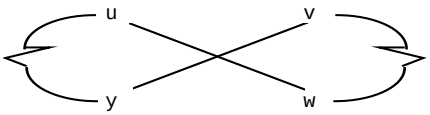
Un algoritmo puede comenzar proponiendo un circuito arbitrario, o bien, un circuito en el que se prefieren arcos de menor valor que no generen ciclos en el recorrido, excepto en el último arco, escogido para completar el circuito total. Se puede obtener un circuito de la forma:



donde los arcos `<u,w>` y `<y,v>` no forman parte del circuito. Si se tiene que

```
costo(u,v) + costo(w,y) > costo(u,w) + costo(v,y)
```

entonces debe ser preferible un circuito.



La optimización puede continuar, para cada cuarteta de nodos, mientras sea posible rebajar el costo.

∎

3.5 REDUCCIÓN Y EQUIVALENCIA DE PROBLEMAS

[1] Por ejemplo, un programa que juegue ajedrez que tenga como mejor jugada la que le reporta más ganancia de material. Es una estrategia casi siempre buena, aunque puede conducir a errores graves.

En esta sección se profundiza en el estudio de las nociones de reducción y equivalencia de problemas esbozadas en 3.3. Se mostrará un ejemplo importante de problemas equivalentes, que por su nivel de abstracción resultan útiles en el momento de resolver otros problemas menos generales.

El problema de determinar la ruta más corta entre un par de nodos de un grafo es un tema clásico dentro de la teoría de análisis de algoritmos. La importancia práctica que, de por sí, ya tiene este problema, se ve realzada por resultados teóricos que prueban cómo el problema de encontrar la distancia mínima entre cada para de nodos de un grafo de n nodos es equivalente (i.e. algorítmicamente igual de costoso de resolver) al problema de multiplicar matrices de $n \times n$.

Aunque no se demostrará este resultado, sí se estudiará en forma abstracta lo que significa multiplicar matrices y se mostrarán ejemplos interesantes en los que se ilustrará el hecho de que un algoritmo eficiente de multiplicación de matrices puede resolver, mediante un cambio de representación adecuado a cada caso, muchos problemas aparentemente no relacionados.

Las mejoras que se consigan en algoritmos que resuelvan el problema de la ruta más corta entre cada par de nodos de un grafo redundarán, así mismo, en mejoras en algoritmos que multipliquen matrices cuadradas. Este tipo de observaciones hace que el poder considerar clases de problemas equivalentes parece aumentar las posibilidades de encontrar más y mejores soluciones.

3.5.1 MULTIPLICACIÓN GENERAL DE MATRICES

Se quiere extender la multiplicación de matrices a una en que sus elementos puedan ser sumados y multiplicados en forma análoga a como lo son los números reales.

Sea $M_{mn}(K)$ el conjunto de las matrices de dimensiones $m \times n$ cuyos elementos se toman de un conjunto K. $M_n(K)$ denota el conjunto $M_{nn}(K)$, las matrices cuadradas de elementos de K.

Para $A, B \in M_n(\mathbf{R})$ se tiene que:

$$(AB)_{ij} = \langle +k: 1 \leq k \leq n : A_{ik} * B_{kj} \rangle$$

En el caso general, se considerarán matrices cuyos elementos se toman de un semianillo $(E, \oplus, \otimes, 0, 1)$.

Definición

Un *semianillo* $(E, \oplus, \otimes, 0, 1)$ consta de un conjunto E, dotado con dos operaciones binarias $.\oplus.$ y $.\otimes.$, cerradas en E y dos constantes $0, 1 \in E$.

Se usan las siguientes denominaciones:
$.\oplus.$: es la *suma* de E
$.\otimes.$: es la *multiplicación* de E
0 : es el *cero* de E
1 : es el *uno* de E.

En un semianillo se cumplen los siguientes axiomas:

[1] $(E,\oplus,0)$ es un *monoide conmutativo*, i.e.,

$a \oplus 0 = a$

$(a \oplus b) \oplus c = a \oplus (b \oplus c)$

$a \oplus b = b \oplus a$

[2] $(E,\otimes,1)$ es un *monoide* , i.e.,

$a \otimes 1 = a$

$1 \otimes a = a$

$(a \otimes b) \otimes c = a \otimes (b \otimes c)$

[3] $\otimes$ distribuye sobre $\oplus$ (a la izquierda y a la derecha):

$(a \oplus b) \otimes c = (a \otimes c) \oplus (a \otimes c)$

$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$

[4] El módulo de la suma es cero de la multiplicación:

$a \otimes 0 = 0$

$0 \otimes a = 0$

Para $a \in E$, se definen las potencias de a como:

$a^0 := 1$

$a^{n+1} := a^n \otimes a \ , \ n \geq 0$

Un semianillo se llama *cerrado*, si en él están bien definidas las sumas de un número enumerable de sumandos (para las que las propiedades de conmutatividad y distributividad se extienden). Cuando éste es el caso, se define además la *clausura* de un $a \in E$, como

$a^* := \langle \oplus k: 0 \leq k : a^k \rangle$.

‖

Ahora pueden introducirse operaciones algebraicas sobre matrices:

- La matriz $0 \in M_{pn}(E)$, se define de modo que

  $0_{ij} := 0$.

- La matriz $I \in M_n(E)$, se define de modo que

  $I_{ij} := 0$, si $i=j$

  $:= 1$, si $i \neq j$.

- Para matrices $A,B \in M_{pn}(E)$, se define la matriz $(A \oplus B) \in M_{pn}(E)$, de modo que:

  $(A \oplus B)_{ij} := A_{ij} \oplus B_{ij}$.

- Para matrices $A \in M_{pn}(E), B \in M_{nq}(E)$, se define la matriz $(A \overset{\oplus}{\otimes} B) \in \in M_{pq}(E)$, de modo que:

  $(A \overset{\oplus}{\otimes} B)_{ij} := \langle \oplus k: 1 \leq k \leq n : A_{ik} \otimes B_{kj} \rangle$.

Siempre que no haya lugar a confusión, se escribirá AB, en lugar de $A \overset{\oplus}{\underset{\otimes}{}} B$.

Ejemplos de semianillos

1 $(\mathbf{R},+,*,0,1)$

2 $(M_n(E),\oplus,\overset{\oplus}{\underset{\otimes}{}},0,I)$

Es decir: las matrices cuadradas cuyos elementos forman parte de un semianillo son, a su vez, un semianillo. La multiplicación no es, en general, conmutativa.

3 $(\mathbf{B},\vee,\wedge,0,1)$
En este caso, nótese que:

$$(AB)_{ij} = \langle \vee k: 1 \le k \le n : A_{ik} \wedge B_{kj} \rangle$$

Sea A matriz booleana que representa las conectividad de un grafo cuyos vértices son los números $1..n$, i.e.,

$$A_{ik} = 1 \Leftrightarrow \quad \text{"hay un arco de i a k"}$$

En el grafo representado por A^2, hay un arco de i a j si y sólo si, existe un vértice k tal que, hay un arco A_{ik} y hay un arco A_{kj}. Es decir:

$$A^2_{ij} = 1 \quad \Leftrightarrow \quad \text{"En A hay un camino de 2 arcos, de i a j"}$$

Más generalmente, para $m \ge 0$:

$$A^m_{ij} = 1 \Leftrightarrow \quad \text{"En A hay un camino de m arcos, de i a j"}$$

Si se denomina A* la matriz de conectividades entre nodos, i.e.,

$$A^*_{ij} = 1 \Leftrightarrow \quad \text{"En A hay un camino de i a j"}$$

se concluye que:

$$A^* = \langle \vee k: 1 \le k \le n : A^k \rangle$$

4 $(\mathbf{R}^*,\min,+,\infty,0)$
Se define $\mathbf{R}^* := \{r \in \mathbf{R} \mid r \ge 0\} \cup \{\infty\}$, con las extensiones "naturales" de las operaciones $.+.$ y $.\min.$ para que cubran los caso en que algún argumento sea ∞, es decir:

$x + \infty = \infty + x = \infty$

$x \min \infty = \infty \min x = x.$

Entonces se verifica que, también:

$(a \min b) \min c = (a \min b) \min c$

$(a + b) + c = a + (b + c)$

$a + 0 = 0 + a = a$

$(a \min b) + c = (a + c) \min (b + c)$

Este semianillo es útil para analizar rutas más cortas en grafos donde cada arco se etiqueta con un número que representa la distancia entre los nodos que conecta. En el caso de nodos no conectados por ningún arco, se puede agregar un arco artificial, etiquetado con ∞.

Así, si se tiene un grafo $G(V,E,c)$, donde
$V = 1..n$
$E \subseteq VxV$
$c: E \to \{r \in \mathbf{R} \mid r \geq 0\}$

se puede definir una *matriz de distancias* (*asociada a* G), como la matriz $D \in M_n(\mathbf{R}^*)$ definida así:

$$\begin{aligned} D_{ij} &:= 0 && , \text{ si } i=j \\ &:= c(i,j) && , \text{ si } (i,j) \in E \\ &:= \infty && , \text{ si } (i,j) \notin E \end{aligned}$$

Entonces, se tiene que:

$$D^2_{ij} = \langle \min k: 1 \leq k \leq n : D_{ik} + D_{kj} \rangle$$

i.e., la distancia mínima de i a j, usando a lo sumo dos arcos[2]. Más aun:

D^m_{ij} : distancia mínima de i a j, usando a lo sumo m arcos

D^{n-1}_{ij} : distancia mínima de i a j

De esta manera:

$$D^* = D^{n-1}.$$

3.5.2 EQUIVALENCIA

Intuitivamente, dos problemas se quieren llamar equivalentes cuando el solucionar uno de los dos conlleva una solución para el otro. Desde el punto de vista de la complejidad, se espera que las dos soluciones sean, al menos del mismo orden.

Cuando se tienen dos problemas equivalentes, se cuenta con una ventaja adicional en la búsqueda de algoritmos eficientes que los resuelvan: las mejoras a las soluciones de uno de los dos redundan en mejoras en las soluciones del otro.

El concepto de equivalencia se ilustrará señalando cómo el problema de multiplicar matrices es equivalente al de calcular distancias mínimas entre los nodos de un grafo etiquetado. Así, las mejoras que se puedan lograr en algoritmos que multipliquen matrices se pueden trasladar a los algoritmos que calculan distancias mínimas. Esto es cierto, en teoría; sin embargo, el hecho de que se estén comparando órdenes de complejidades y no complejidades exactas puede parecer engañoso: las constantes de proporcionalidad, pueden ser demasiado grandes para que los resultados teóricos puedan ser aprovechados en muchos casos prácticos.

---

[2] Nótese que el caso "un arco" está considerado, cuando se exige que D sea 0 en la diagonal.

Sea P_0 un problema soluble con complejidad $T_0(n)$. Para un problema P, se dice que:

- P es *reducible a* P_0 $:\Leftrightarrow$ $P \leq P_0$
  $:\Leftrightarrow$ Existe un algoritmo para solucionar P, mediante P_0, con complejidad $T(n) = O(T_0(n))$.
- P es *equivalente a* P_0 $:\Leftrightarrow$ $P \cong P_0$
  $:\Leftrightarrow$ $P \leq P_0$ y $P_0 \leq P$

Dados los problemas:

P^* : calcular D^*, dado $G(V,E,c)$, con $V=\{1,\ldots,n\}$

P_m : calcular $(A \overset{\min}{+} B)$, con $A,B \in M_n(\mathbf{R}^*)$.

se tiene el siguiente resultado:

Teorema

$$P^* \cong P_m$$

Demostración:
Se omite.

□

3.5.3 MULTIPLICACIÓN EFICIENTE DE MATRICES

El siguiente teorema es debido a Strassen y Winograd:

Teorema

Dado un semianillo $(E,\oplus,\otimes,0,1)$, si existe un número k, para el cual hay un algoritmo que permite miltiplicar matrices de $M_k(E)$ que utiliza $f(k)$ multiplicaciones y $g(k)$ sumas, entonces, para $n \geq k$, la multiplicación de matrices en $M_n(E)$ puede llevarse a cabo en

$$O(n^{\log_k f(k)})$$

operaciones (sumas y multiplicaciones).

Demostración:
Se omite.

□

Un corolario del anterior teorema asegura la existencia de algoritmos de multiplicación de matrices con orden menor que n^3:

El algoritmo de Strassen

Sean A,B,C matrices cuadradas de lado 2 tales que

$$AB = C.$$

Los coeficientes de C pueden calcularse de la siguiente forma:

$$m_1 = (a_{12} - a_{22}) * (b_{21} + b_{22})$$
$$m_2 = (a_{11} + a_{22}) * (b_{11} + b_{22})$$
$$m_3 = (a_{11} - a_{21}) * (b_{11} + b_{12})$$
$$m_4 = (a_{11} + a_{12}) * b_{22}$$
$$m_5 = a_{11} * (b_{12} - b_{22})$$
$$m_6 = a_{22} * (b_{21} - b_{11})$$
$$m_7 = (a_{21} + a_{22}) * b_{11}$$
$$c_{11} = m_1 + m_2 - m_4 + m_6$$
$$c_{12} = m_4 + m_5$$
$$c_{21} = m_6 + m_7$$
$$c_{22} = m_2 - m_3 + m_5 - m_7$$

Nótese que, en este caso, los números de sumas y multiplicaciones son, en los términos del teorema anterior ($k = 2$):

$$f(2) = 7$$
$$g(2) = 18$$

El teorema afirma que la multiplicación se puede lograr en $O(n^{\log 7})$.

Una manera de probar el resultado, en este caso particular, consiste en plantear la recurrencia:

$$T(n) = 1 \qquad , \text{ si } n = 1$$
$$= 7T(n/2) + 18(n/2)^2 \qquad , \text{ si } n \geq 2$$

que resulta de observar que la multiplicación matricial puede ser llevada a cabo por bloques. Esta es, de hecho, la idea que apoya la demostración del caso general que plantea el teorema.

3.6 EJERCICIOS

1 Considere el problema de las torres de Hanoi: dadas tres pilas (A,B,C), inicialmente de la forma (<1,2,…,n>,<>,<>), se busca transformarlas hasta el estado (<>,<>,<1,2,…,n>), de acuerdo a las siguientes reglas:
i. Un *movimiento* consiste en llevar el tope de una pila a ser tope de otra.
ii. Despues de cada movimiento, las tres pilas están ordenadas ascendentemente.
Se quiere encontrar una solución que demande el mínimo número de movimientos.

a Construya un algoritmo que solucione el problema utilizando "dividir y conquistar". Encuentre la complejidad exacta de su solución (número de movimientos).

b Exprese el problema como un problema de grafos. Haga una estimación del peor caso de un algoritmo que resuelva el problema, con base en una estimación del número de vértices y arcos que pueda tener el grafo en cuestión.

2 El *problema de la planicie* consiste en encontrar, para un arreglo de enteros ordenado ascendentemente `b[0..n-1]`, la longitud de la planicie más larga. Una *planicie* es un subarreglo `b[i..j]`, con $0 \le i \le j < n$, tal que $b[i]=b[i+1]=...=b[j]$. Encuentre un algoritmo que resuelva el problema en tiempo $O(n)$, y comente cómo puede considerarse esa solución como una aplicación de programación dinámica.

3 **a** Cuántos *tours* posibles hay, en el problema del agente viajero?

b Dado un *tour*, considere la operación descrita para explicar el algoritmo de acondicionamiento de 3.4. Muestre que si se puede cambiar un *tour* por cualquier otro que resulte de efectuar una tal operación, sin que sea necesario disminuir el costo, toda solución óptima es alcanzable desde cualquier tour inicial. Utilice esta observación para estimar el costo del peor caso del algoritmo de acondicionamiento, si se le exige alcanzar el óptimo.

4 Pruebe que las siguientes estructuras son semianillos:

a $(\mathbf{R}^*, \max, \min, 0, \infty)$

b $(\mathbf{R} \cup \{-\infty, +\infty\}, \max, +, -\infty, 0)$, donde $(-\infty)+(+\infty)=(-\infty)$.

5 Muestre que los semianillos de **4** son cerrados. Encuentre cómo se definen las clausuras en cada caso.

6 Sea A matriz booleana que representa la conectividad de un grafo sobre el conjunto de vértices $V = 1..n$. Supóngase, por convención, que $A_{ii}=1$. Demuéstrese que $A^* = A^{n-1}$.

7 Sea $G(V,E,c)$ un grafo etiquetado con distancias. Sea D la matriz de distancias correspondiente, pero con $D_{ij}=-\infty$, si $\langle i,j \rangle \notin E$. Pruebe que la clausura D^* en el semianillo de **4b** puede interpretarse como la matriz de distancias máximas entre cada para de nodos.