Curso:
Teoría de la Computación.
Unidad 2, Sesión 7: Complejidad computacional

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Montevideo, Uruguay

dictado semestre 2 - 2009

Contenido:

1. Introducción
2. Formalismos
3. Etapas de abstracción
4. Clases de complejidad
5. Clases P y $\mathcal{N}P$.

Introducción

Nota: **Complejidad computacional.** La teoría de la complejidad computacional es el estudio de cuántos recursos computacionales (tiempo, memoria, etc.) son necesarios para resolver diversos problemas de cómputo. En la primer parte del curso discutimos que problemas eran computables y cuales no lo eran. Sin embargo, algunos problemas que son computables en principio, en la práctica precisan un uso de recursos demasiado grande para permitir su cálculo en la práctica. Es entonces de particular interés el poder distinguir problemas que son tratables de otros que no lo son. En este curso nos restringiremos a los modelos más clásicos de cálculo, pero la teoría de la complejidad computacional también incluye resultados cuando se agrega el uso de aleatoriedad, cuando se aceptan soluciones aproximadas en lugar de exactas, y cuando se permite trabajar con programas que dan el resultado correcto para la mayor parte de las entradas, pero no necesariamente para todas.

El material de esta sesión está fuertemente basado en el material del curso Computational Complexity dictado por Luca Trevisan (y en particular,

en la primer lectura de dicho curso). Dicho material fue obtenido en la dirección http://www.cs.berkeley.edu/∼luca/cs278-08/ (último acceso 2009-11-08).

En particular, la teoría de la complejidad computacional ha sido clásicamente desarrollada sobre el modelo de máquina de Turing (si bien es equivalente a hacerlo con otros formalismos como los programa **while**).

Formalismo

Nota: **Complejidad.** Supongamos que tenemos una función $f : N \to N$ computable. Podemos decir que construir un algoritmo que compute f es un problema computacional, en el sentido que queremos encontrar un código que, dada una entrada cualquiera (que vamos a suponer, sin pérdida de generalidad, que está codificada en el alfabeto $\{0, 1\}$), queremos calcular como salida una solución que satisface las propiedades que definen a f. Vamos a definir cuatro tipos de problemas computacionales usuales: problemas de decisión, problemas de búsqueda, problemas de optimización, problemas de conteo (esta clasificación, si bien es útil, es arbitraria, ya que todo problema puede ser visto como un problema de búsqueda). ♣

Definición de **Problema de decisión.**

Dados: Una función $f : \{0, 1\}^* \to N$

Escribimos: f corresponde a un problema de decisión

Condición: f corresponde a un problema de decisión si $f(x) \in \{0, 1\}\ \forall x$. Es decir, es un problema de decisión si la "respuesta" es de tipo SI/NO. Esto corresponde a clasificar las entradas en dos conjuntos, las que satisfacen una cierta propiedad y las que no lo hacen.

Ejemplo: **Problema de decisión.** Un ejemplo de problema de decisión es el de 3-coloreo. Dado un grafo no dirigido, determinar si hay una forma de asignar un color elegido entre tres posibles a cada vértice de manera que dos vértices adyacentes tengan colores distintos. ♣

Nota: **Problema de decisión.** Una forma conveniente de especificar un problema de decisión es dar el conjunto $L \in \{0,1\}^*$ de las posibles entradas para los cuales la respuesta es "SI". Si llamamos lenguaje a un subconjunto de $\{0,1\}^*$, cualquier problema de decisión puede ser especificado por un lenguaje equivalente (y viceversa). Utilizaremos ambos términos de manera intercambiable de aquí en más. ♣

Definición de **Problema de búsqueda.**

Dados: Una relación $R \subseteq \{0,1\}^* \times \{0,1\}^*$

Condición: Un problema de búsqueda corresponde a, dado una entrada $x \in \{0,1\}^*$, encontrar un valor $y \in \{0,1\}^*$ tal que $(x,y) \in R$.

♣

Ejemplo: **Problema de búsqueda.** Un ejemplo de problema de búsqueda ligado al problema de 3-coloreo es el, dado un grafo no dirigido, encontrar (si existe) un coloreo concreto de tres colores de sus vértices. El problema es distinto (y más exigente) que el problema de decisión asociado, que sólo nos pedía contestar si un coloreo existe o no, pero no suministrar un coloreo concreto. ♣

Definición de **Problema de optimización.**

Dados: Una función objetivo $f : \{0,1\}^* \rightarrow R$

Dados: Una función $g : \{0,1\}^* \rightarrow \{0,1\}$ que define el conjunto facbile

Escribimos: El espacio de soluciones factibles del problema es el conjunto $\{x/g(x) = 1\}$.

Condición: Este corresponde a un problema de optimización si el objetivo consiste en encontrar $x^* = argmax_{x|g(x)=1} f(x)$ (o argmin).

♣

Ejemplo: **Problema de optimización.** Un ejemplo de problema de optimización ligado al problema de 3-coloreo es el, dado un grafo no dirigido y un conjunto de pesos en las aristas , encontrar (si existe) un coloreo concreto de tres colores de sus vértices, que minimice (o maximice) un parámetro de mérito para el problema. ♣

Definición de **Problema de conteo.**

Dados: Una relación $R \subseteq \{0,1\}^* \times \{0,1\}^*$

Condición: El problema de conteo corresponde a, dado una entrada $x \in \{0,1\}^*$, encontrar el cardinal del conjunto $\{y \in \{0,1\}^*$ tal que $(x,y) \in R$.

♣

Ejemplo: **Problema de conteo.** Un ejemplo de problema de conteo ligado al problema de 3-coloreo es el, dado un grafo no dirigido y un conjunto de pesos en las aristas, dar la cantidad de coloreos de tres colores de sus vértices distintos. ♣

Etapas de abstracción

Nota: **.** La complejidad temporal de un procedimiento de cálculo efectivo para una función f y una entrada concreta x es el número de pasos elementales que se debe realizar en la máquina que lo computa desde el inicio hasta la parada.

Esta información, si bien puede ser muy útil (es en definitiva el "tiempo de cálculo" que el procedimiento lleva para calcular $f(x)$), tiene un bajo nivel de abstracción. ♣

Definición de **Tamaño de la entrada x.**

Condición: $|x|$ es el número de bits necesarios para representar x

♣

Nota: . Un nivel superior de abstracción corresponde a definir la complejidad en función del tamaño de la entrada x. De esta manera, la complejidad de un procedimiento de cálculo efectivo para entradas de tamaño n puede verse como el número de pasos que es necesario realizar en la máquina utilizada, en el peor caso calculado sobre todas las entradas x de tamaño menor o igual a n, para calcular $f(x)$. En particular, es interesante expresar dicha complejidad como una función analítica de n (si posible), o al menos estudiar el orden de dicha función.

Si el orden de dicha función es por ejemplo lineal o polínómico de bajo orden en n, el tiempo de cálculo requerido crecerá de manera relativamente benigna en función del tamaño de la entrada. En cambio, si fuera exponencial, pequeños incrementos en el tamaño de la entrada darán lugar a grandes incrementos en tiempo computacional, al punto de poder hacer inviable desde un punto de vista práctico el realizar el cómputo. ♣

Nota: **.** Observamos que aún en este nivel de abstracción, la complejidad está definida en base al procedimiento de cómputo efectivo utilizado para calcular la función f. Dado que para cualquier función f computable existen infinitos programas equivalentes del punto de vista de su salida, pero que pueden tener tiempos de cómputo muy diversos, es interesante definir la complejidad computacional de calcular la función f (o equivalentemente de resolver un problema P) como la complejidad del procedimiento de cálculo de f más eficiente posible (aquel entre todos los equivalentes que tendrá menor orden), en una determinada máquina o modelo de computación. ♣

Ejemplo: **Problemas de ordenamiento.** Un ejemplo del punto anterior es por ejemplo el problema de ordenar una lista de datos; si bien existen métodos que realizan esta tarea en orden cuadrático en el tamaño de la entrada, el problema en sí mismo es de orden $nlog(n)$. ♣

Clases de complejidad

Nota: . Las clases de complejidad es una manera de clasificar los problemas. En este curso miraremos únicamente los problemas de decisión, si bien existen clases de complejidad también para los problemas de búsqueda, optimización y conteo.

Es usual tomar como referencia la máquina de Turing como modelo de computación. Desde el punto de vista de la complejidad temporal de los problemas, este modelo es equivalente (módulo una transformación a lo sumo de tipo polinómico) a los modelos de computación de programas-while, y otros similares. Esto significa que si tengo un problema cuya complejidad en el modelo de máquina de Turing es de $O(f(n))$, en el modelo de programas while su complejidad será de $O(g(f(n))$, donde $g(n)$ es un polinomio de n de orden k independiente de la entrada (y viceversa si se realiza el pasaje de un modelo de programas-while a una máquina de Turing).

Dada la máquina de referencia, las clases se organizan naturalmente

de acuerdo al orden de complejidad de los problemas, si bien hay algunas grandes clases que son las más significativas.

Por ejemplo, la clase de complejidad lineal L corresponde a todos los problemas que pueden ser resueltos por una máquina de Turing en tiempo $O(n)$ en función del tamaño de la entrada.

Algunas de las clases más significativas son la clase P de los problemas que pueden ser resueltos en tiempo polinómico en función del tamaño de la entrada, y la clase EXP de los problemas que pueden ser resueltos en tiempo exponencial.

Nota: **.** Transformaciones polinomales: ya se mencionó la importancia de las transformaciones en tiempo polinómico.

En la Teoría de la Complejidad, se utiliza como herramienta central la noción de Reducción de problemas, que corresponde a la posibilidad de transformar en tiempo polinómico problemas.

Definición de **Reducción.**

Dados: Dos problemas de decisión A y B.

Escribimos: El problema A es reducible al problema B.

Condición: A es reducible a B (notación $A \leq B$) si existe una función f computable en tiempo polinómico tal que $x \in A$ ssi $f(x) \in B$.

Nota: . Si A es reducible a B, intuitivamente B es al menos tan difícil de calcular (o más) que A. Si A y B son mutuamente reducibles, entonces intuitivamente su dificultad es la misma. ♣

Nota: . Dada una clase cualquiera C, se define la clase $C - completo$ como el conjunto de los problemas de la clase C tales que cualquier otro problema de la clase puede ser reducido a ellos. Formalizando, $B \in C - completo$ si $B \in C$ y para todo A en la clase C, $A \leq B$. Intuitivamente, la clase $C - completo$ contiene a los problemas más difíciles de la clase C.

Nota: . Dada una clase cualquiera C, se define la clase C-difícil como el conjunto de los problemas que son al menos tan difíciles como los más difíciles de la clase C. Formalizando, $B \in C$-difícil si para todo A en la clase C, $A \leq B$. La clase C-difícil contiene a todos los problemas de la clase $C - completo$ y los problemas que no pertenecen a C y son más difíciles.

Clases de complejidad P y $\mathcal{N}P$

Nota: . La clase de complejidad P incluye como dijimos a todos los problemas que pueden resolverse en tiempo polinómico. En general, se considera que los problemas de esta clase admiten una solución eficiente (si bien un problema que para su solución requiera de n^20 pasos claramente solo podrá ser resuelto para instancias muy pequeñas; sin embargo, para la inmensa mayoría de los problemas de interés estudiados en esta clase, se ha podido identificar algoritmos de órdenes relativamente pequeños para resolverlos).

Por otro lado, los problemas en la clase EXP-difícil,que llevan tiempo de cálculo exponencial o peor, se consideran como fuera de la capacidad de solución práctica para instancias de tamaño mediano o grande (y en general se atacan con métodos no exactos de solución).

Existe una clase intermedia, de gran interés práctico y teórico, que incluye una cantidad importante de problemas para los que no conoce ningún método de cálculo de complejidad polinómico, pero tampoco se ha

podido probar teóricamente que no exista. Se trata de la clase $\mathcal{NP}$, sigla que corresponde a No determinista- polinómico (y no a No polinómico, como a veces erróneamente se piensa)..

Esta clase se define a partir de la Máquina de Turing no-determinista, e incluye a los problemas para los que existe un procedimiento efectivo que puede ser calculado en tiempo polinómico por una máquina de esas características.

Dada la importancia de esta clase, a continuación se dan más detalles sobre la misma.

Nota: . Una definición alternativa de la clase $\mathcal{NP}$ se puede hacer a través de los problemas de búsqueda.

Se dice que un problema de búsqueda definido por una relación R pertenece a la clase de problemas de búsqueda $\mathcal{NP}$ si existe un algoritmo de tiempo polinómico tal que, dados x e y, decide si (x, y) pertenece a R, y si existe una función polinómico p tal que el tamaño de y es menor o igual que $p(|x|)$.

Se dice que un problema de decisión L es un problema $\mathcal{NP}$ si existe una relación R perteneciente a $\mathcal{NP}$ tal que $x \in L$ sí y sólo sí existe un y tal que $(x, y) \in R$. Esto equivale a pedir que para todo x en el lenguaje L exista un y de tamaño polinómico en el tamaño de x, y que haya un algoritmo que acepte en tiempo polinómico el par (x, y) como perteneciente a R.

Dicho de otra forma, debe existir por un lado un certificado y para cada x del lenguaje L, y por otro un algoritmo dependiente sólo de L y de la forma de los certificados, tal que dado un valor x y un posible certificado y, en tiempo polinómico verifique que efectivamente este último es el certificado y que entonces x pertenece al lenguaje.

Nota: . Intuitivamente, se puede ver que para un problema $\mathcal{N}P$, si yo tengo un oráculo que, dado un $x \in L$, es capaz de producirme un certificado y de que x pertenece a L, es posible en tiempo polinómico con una máquina clásica verificarlo. Alternativamente, si tengo una máquina no-determinista capaz de explorar "al mismo tiempo" todos los certificados posibles (de largos polinómicos), o si tengo un número polinómico de máquinas de Turing clásicas que permitan explorar en paralelo estos certificados, también obtendré la misma respuesta (verificando que $x \in L$) en tiempo polinómico.

Es interesante ver que si $x \notin L$, no tengo la propiedad simétrica (es decir, puedo no saber en tiempo polinómico que no pertenece al lenguaje).

Nota: . Se conoce por clase $\mathcal{NP}$ al conjunto de los problemas de decisión $\mathcal{NP}$.

Siguiendo la notación que usamos anteriormente, la clase $\mathcal{NP}$-completa corresponde a los problemas más difíciles de $\mathcal{NP}$, aquellos tales que cualquier otro problema de $\mathcal{NP}$ puede ser resuelto via la reducción a un problema de estos. La clase $\mathcal{NP}$-difícil incluye a los problemas $\mathcal{NP}$-completos y a aquellos que son aún más difíciles y no pertenecen a $\mathcal{NP}$.

El interés de los problemas $\mathcal{NP}$-completos es que si es posible resolver de forma eficiente uno de ellos, a través de reducciones polinómicas será posible resolver de forma eficiente cualquier problema de $\mathcal{NP}$.
