TEMA 3.

Introducción a la complejidad computacional.

1- Complejidad computacional:

Vamos a estudiar los problemas decidibles desde el punto de vista de los recursos que se requieren para resolverlos. Escogemos las Máquinas de Turing como modelo de cómputo.

Nos referimos al tiempo como: cantidad de pasos efectuados por las MT.

Nos referimos al espacio como: cantidad de celdas utilizadas por las MT.

A las medidas de complejidad de este tipo las conocemos como dinámicas. Otra medida dinámica sería la cantidad de veces que en la ejecución de una MT con una sola cinta, su cabezal cambia de dirección. De las medidas dinámicas es de lo que trata la **complejidad computacional**.

La complejidad también se puede basar en medidas estáticas, en cuyo caso hacemos referencia a la **complejidad estructural** de los problemas. Un ejemplo de estas medidas de complejidad es el producto entre la cantidad de estados y el tamaño del alfabeto de una MT.

2- Complejidad temporal.

Nota: Como estamos trabajando con problemas de decisión, emplearemos de forma indistinta los términos de problema y lenguaje.

Complejidad (computacional) temporal:

Idea: Una MT que resuelve un problema, tarda más cuando mayor sea el tamaño de la instancia del problema.

Tratamos de medir el tiempo de ejecución de una MT a partir de una entrada w con |w|=n, mediante una función T(n) y analizar la razón de crecimiento de la función temporal. Categorizamos la complejidad del problema según el tiempo en el que lo resuelva la MT (lineal, polinómico..)

Como los factores constantes no influyen en la medición final, vamos a trabajar con funciones O(T)={funciones f / f(n)≤cT(n), c=cte, y n natural}.

Sea una función T:N→N. Se dice que una **MT M trabaja en un tiempo T(n)** ↔ para toda entrada w con |w|=n, M hace a lo sumo T(n) pasos, en su única computación si es determinista, o en cualquiera de sus computaciones si es no determinista.

Sea una función O(T):N→N. Se dice que una **MT M trabaja en un tiempo O(T(n))** ↔ para toda entrada w con |w|=n, M hace a lo sumo O(T(n)) pasos, en su única computación si es determinista, o en cualquiera de sus computaciones si es no determinista.

Se asume que una MT hace al menos n+1 pasos.

Los problemas que pueden ser resueltos por MT que trabajan en tiempo O(T(n)) se agrupan en una misma clase: Un problema (lenguaje) pertenece a la **clase DTIME(T(n))** (resp **NTIME(T(n)**)) ↔ existe una MTD(resp MTND), con una o varias cintas, que lo resuelve (reconoce) en tiempo O(T(n)).

El criterio de medición que estamos usando es el del peor caso (cota superior). Podríamos usar el criterio del caso promedio, para ello necesito conocer cómo se distribuyen las entradas. Si un problema pertenece a una clase temporal, no implica que no tenga un tiempo de resolución menor.

Lo ideal es buscar la cota temporal mínima, establecer una **complejidad intrínseca** del problema, independientemente de los algoritmos que conocemos. Lo normal es calcular la complejidad temporal de un problema usando el algoritmo más eficiente que conozcamos.

Ejemplo: Una MTD M1, con una cinta, que reconoce el lenguaje L de los palíndromos, trabaja en tiempo T(n)=O(n^2), es decir, LєDTIME(n^2). Pero una MT M2, con dos cintas reconoce el mismo lenguaje en tiempo lineal.

Relación entre los tiempos de trabajo de MT:

1) $T(n) \geq n \rightarrow$ cualquier MT multicinta que trabaje en tiempo T(n) tiene una equivalente de una sola cinta en tiempo $O(T(n)^2)$. Se puede demostrar también que al pasar de una a varias cintas, el retardo disminuye a $O(n*\log_2 n)$.
2) Un problema que se resuelve en tiempo polinomial con una MTD con K1 cintas, también se revuelve en tiempo polinomial con una MTD con K2 cintas.
3) $T(n) \geq n \rightarrow$ cualquier MTND que trabaje en tiempo T(n) tiene una equivalente de una sola cinta en tiempo $2^{O(2\text{^}T(n))}$.

3- Representación de problemas.

Se debe elegir un modelo razonable para representar los datos tratados por las MT para evitar inconsistencias. Si un problema es tratable (intratable) también lo debe de ser con otra. Esta es la noción de representación razonable.

Una **representación razonable** consiste en "cadenas estructuradas" de un determinado alfabeto a partir de las cuales codificar objetos de diferentes categorías. Han de tener las siguientes características:

- Enteros: los representamos usando el alfabeto {0,1,-}. Usar representaciones unarias sería inviable.
- Conjuntos, listas, etc…: secuencia de sus elementos, separados por símbolos apropiados. Cada elemento se representa como un entero y cada entero con su representación binaria.
- Grafo G=(V,E): usando una cadena estructurada {x1,x2,…,xn, {(xi,xj),i,j=1,2,..,n}}
- Racional q: como un par de enteros (x,y) donde x/y = q.
- Función finita F:{U1,U1,…Un}→ W: mediante una cadena estructurada {(xi,yi) i=1,2,…,n} donde xi, yi son la representación de Ui y f(Ui).

Ejemplo: Sea L={(G,v1,v2,K)} el lenguaje que representa el problema de establecer si un grafo tiene un camino de longitud al menos K entre los vértices v1 y v2.

El grafo G= ({1, 2, 3, 4}, {(2, 4), (4, 1)}) tiene representación 1#10#11#100##10#100#100#1. Asumimos que G es no orientado si no se dice lo contrario. Denoto el tamaño de V por m. Ni con ésta ni con otras representaciones razonables se conoce una resolución eficiente para L.

Si represento el grafo enumerando todos sus caminos si tengo una resolución eficiente pero esta representación no es razonable. El tiempo de transformación de cualquiera de las representaciones razonables conocidas a ésta es exponencial.
La complejidad temporal del problema está oculta en su representación.

4- Las clases P y NP.

El conjunto R de los lenguajes recursivos o problemas de decisión decidibles se distinguen en dos clases temporales:

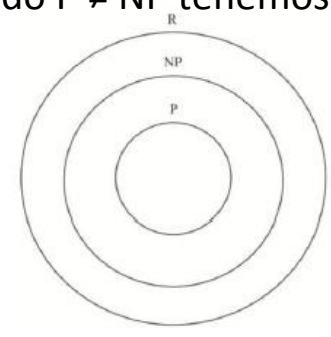
$P = \bigcup i \geq 0\ DTIME(n^i)$
$NP = \bigcup i \geq 0\ NTIME(n^i)$

P es la clase que agrupa los problemas que se resuelven en tiempo determinístico polinómico (para todo problema existe una MTD M que lo resuelve en tiempo $O(n^i)$)

NP es la clase que agrupa los problemas que se resuelven en tiempo no determinístico polinómico.

$P \subseteq NP$. Ambas están incluidas estrictamente en R.

Se sospecha que $P \neq NP$. Constituye el problema más importante de la complejidad computacional. En P y NP se identifican numerosos e importantes problemas de la computación. Asumiendo $P \neq NP$ tenemos la siguiente primera versión de la **jerarquía temporal**:



P es la clase de los problemas tratables (aceptables, eficientes), en el sentido de que si bien todos los problemas de R son decidibles, las resoluciones que consumen más que tiempo determinístico polinómico no se consideran aceptables.

La **Tesis de Cobham-Edmonds** identifica la clase P como la de los problemas que se resuelven eficientemente.

{R\P} queda como la clase de los problemas intratables, siendo entonces la frontera de la clase P la que separa los problemas con resolución temporal aceptable de los de resolución temporal ineficiente. Asumiremos que lo que no es polinomial es exponencial.

Definición alternativa de la clase NP:

En problemas fuera de la clase P los tiempo de cálculo pueden ser prohibitivos, de modo que optamos por la verificación de soluciones y no por calcularlas.

Una MTND construye un árbol de computación a partir de una entrada del lenguaje y eventualmente para sobre una rama, que puede interpretarse como una solución que se verifica.

Def: Un **verificador** para un lenguaje L es una MTD tal que
L = {w/ M(w,c) = 1, para una cadena c}.
Un **verificador en tiempo polinómico** es aquel que verifica L en tiempo polinómico en la longitud de w.
Un lenguaje **L se dice polinomicamente verificable** si tiene un verificador en tiempo polinómico.

La cadena c se llama certificado o demostración de la pertenencia de w a L. Si M es polinómico, entonces c tiene longitud polinómica.

Def: La **clase NP** está formada por todos los lenguajes que tienen verificadores en tiempo polinómico.

Teorema: Las dos definiciones de NP son equivalentes.

5- **Relación entre clases** (en base a cotas temporales).

Hablaremos de tiempo determinístico pero todo se aplica igual para tiempo no determinístico.

$T1(n)=O(T2(n)) \rightarrow DTIME(T1(n)) \subseteq DTIME(T2(n))$

El **Teorema de Aceleración Lineal** establece que si una MT M1 con K cintas traba en tiempo T(n) → existe una MT M2 con K+1 cintas que trabaja en tiempo $cT(n) + (n+1)$, $c>0$.

En otras palabras, una clase DTIME(T2(n)) no incluye más problemas que una clase DTIME(T1(n)), si es que T2(n) difiere de T1(n) en un factor constante.

Las funciones temporales de buen comportamiento se llaman **tiempo-construibles**.
Def: Una función $T:N \rightarrow N$ es tiempo-construible si $T(n) \geq n$ y para toda entrada w, $|w|=n$, existe una MT que trabaja en tiempo determinístico exactamente T(n) que computa T(n). (T(n) se ejecuta en T(n) pasos (se utilizan como relojes) → a veces se define la jerarquía temporal solo en términos de estas funciones)

Todas las funciones con las que se trabaja habitualmente en la complejidad temporal, n^k, 2^n, $n!$,… con tiempo-construibles.

Si T1 y T2 son tiempo-construibles → su suma, producto y 2^{t1} lo son.

El uso de funciones no tiempo-construibles puede producir efectos no deseados en la jerarquía temporal, por ejemplo:

Teorema de Aceleración: que existan MT con cotas de tiempo siempre mejorables.

Teorema del Hueco: posibilidad de la existencia en la jerarquía de amplias franjas vacías de problemas, a pesar de estar delimitadas por funciones temporales muy distintas entre sí.

Resultado 1:
Si T2(n) es un función tiempo-construible y T2(n)>T1(n)*logT1(n) a partir de un cierto n, entonces DTIME(T1(n)) ≠ DTIME(T2(n)).

Resultado 2:
Si T(n) es una función total computable → existe un lenguaje recursivo L tal que L ∉ DTIME(T(n)). Se trata del lenguaje:
L = {wi | wi no es aceptado por la MT Mi en T(|wi|)pasos}

6- Algunos problemas P y NP.

P:

PATH: {Dado grafo G determinar si hay un camino entre 2 vértices dados}.
RELPRIME: {dados x,y enteros determinar si son relativamente primos}
MCD: {dados x,y enteros calcular su mcd} (Clase FP).
2-SAT: {determinar si una formula booleana en forma normal conjuntiva con 2 literales por clausula es satisfacible}

NP:

SAT: {determinar si una formula booleana es satisfacible}
CSAT, k-SAT
k-CLIQUE: {Grafo G, ver si tiene un clique con exactamente k nodos}
HC: {Si G es un grafo, determinar si tiene Circuito Hamiltoniano}
TSP: {Si G es un grafo, determinar si tiene Circuito Hamiltoniano de mínimo costo}