**Fundamentos de Teoría de la Computación 2021**

**Examen Parte 1 Fecha 1 - Lunes 3 de mayo**

**Instrucciones**

1. El examen es INDIVIDUAL, no nos defraudes.
2. Ni bien recibas el examen enviame un email con tu nombre, DNI y legajo, confirmándome que lo recibiste.
3. Tenés 2 horas para responder el máximo de preguntas que puedas, no te detengas mucho en ninguna, y respondé claro y breve. Hay que responder preguntas de las 2 partes, Computabilidad y Complejidad Computacional. Respondé sobre el email o el Word, debajo de cada pregunta.
4. Cuando termines, enviame el examen resuelto por email, repitiendo en el asunto tu nombre, DNI y legajo. Cuando yo lo reciba te enviaré un ok.

¡Mucha suerte!

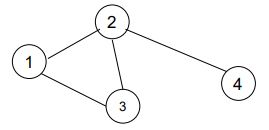
**Parte 1. Computabilidad.**

* Vimos en clase que el lenguaje SAT, que representa el problema que plantea si una fórmula booleana sin cuantificadores es satisfactible, es recursivo. ¿Qué significa que el lenguaje SAT es recursivo?
  + Que se puede construir una MTD M que lo decida y siempre pare
* ¿Por qué el complemento de un lenguaje de (RE – R) no puede pertenecer a RE?
  + Porque RE no es cerrada en cuanto al complemento, ya que si un L que pertenece a RE, si su complemento tambien pertenece a RE, L pertenece a R.
* Describir la función de transición de una Máquina de Turing que, recibiendo sólo cadenas de 1 y 0, las transforma cambiando 1 por 0 y 0 por 1.
  + δ(q0, 0) = (q0, 1, R)
  + δ(q0, 1) = (q0, 0, R)
  + δ(q0, B) = (qA, B, S)
* Describir la función de transición de un Autómata Finito con estado inicial q0 y conjunto de estados finales {qF} que, recibiendo sólo cadenas de 1 y 0, acepta las que tengan al menos una vez dos 0 consecutivos
  + δ(q0, 0) = (q1 , R)
  + δ(q0, 1) = (q0 , R)
  + δ(q1, 0) = (qF , R)
  + δ(q1, 1) = (q0 , R)
* Comentar la idea general de cómo se puede transformar una Máquina de Turing en otra equivalente que no modifique el input.
  + Dadas dos maquinas M1 y M2, si se quisiera transformar en M1 en M2, M1 podria ejecutar M2 (ya que esta segunda se va a comportar como M1 por ser equivalente).
* Vimos en clase el lenguaje QBF = {φ | φ es una fórmula booleana con cuantificadores (todos al comienzo), cerrada (es decir que no tiene variables libres) y verdadera}. Por ejemplo, φ1 = ∀x∀y∃z (x ∨ y ∨ z) ∈ QBF, y en cambio φ2 =∀x∀y (x ∧ y) ∉ QBF. Explicar por qué la siguiente Máquina de Turing no determinística no acepta QBF: dado un input φ, si no es correcto sintácticamente rechaza, y si lo es genera no determinísticamente una asignación A de valores de verdad para todas las variables de φ y luego en cada computación evalúa φ con A sin considerar los cuantificadores, aceptando sii la evaluación resulta verdadera. Ayuda: ver por ejemplo qué sucede con la fórmula ∀x∀y (x ∧ y).
  + El problema de la maquina esta en que no tiene en cuenta los cuantificadores, ya que para el caso∀x∀y (x ∧ y) una computacion encontrara la combinacion x = verdadero e y = verdadero y determinaria que como con esa combinacion la formula da verdadero, esta estaria en QBF, pero al tener en cuenta los cuantificadores sabemos que no, ya que nos dicen que para todos los valores de x e y, la formula tiene que dar verdadero.
* Sean L1 un lenguaje recursivamente numerable y L2 un lenguaje recursivo. Describir la idea general de cómo trabajaría una Máquina de Turing que acepte L1 ⋃ L2.
  + Siendo M1 una maquina que reconozca L1 y M2 una maquina que reconozca L2
  + Se puede construir una MT M’ que ejecute en paralelo M1 y M2 y si alguna acepta entonces M’ acepta, si ambas rechaza, M’ rechaza y si M2 rechaza y M1 loopea M’ loopea
* Explicar por qué la siguiente Máquina de Turing no acepta el lenguaje L = {<M> | L(M) ≠ ∅}: la máquina genera una cadena w, ejecuta M sobre w, y si acepta entonces acepta; si no, genera otra cadena w, ejecuta M sobre w, y si acepta entonces acepta; y así sucesivamente.
  + No lo acepta ya que se puede dar que le llegue una <M> que siempre rechace entonces MT probaria infinitamente (loopearia)
* Dados k lenguajes recursivamente numerables L1, L2, …, Lk, disjuntos dos a dos y cuya unión es ∑\*, describir la idea general de cómo trabajaría una Máquina de Turing que acepte L1 y pare siempre. Ayuda: M podría ejecutar “en paralelo” las Máquinas de Turing que aceptan L1, L2, …, Lk.
  + La MT M podria ejecutar de forma paralela las Mk maquinas que aceptan los lenguajes respectivos, haciendo un paso en cada una, de esta forma ya que la union de todos los lenguajes es ∑\* aunque sea una de las Mk va a aceptar.
* Explicar cómo se puede detectar si una Máquina de Turing entra en loop cuando se mueve en un espacio limitado de celdas.
  + Porque se puede calcular la cantidad maxima de pasos que puede hacer antes de entrar en loop, siendo C la cantidad de pasos y n. C = C = n .|Q|. |Γ|n
* Supongamos que una Máquina de Turing M genera un lenguaje L, es decir, imprime en una cinta, una a una, todas las cadenas de L. Explicar la idea general de cómo trabajaría una Máquina de Turing M´ que acepte el lenguaje L, es decir que dada una cadena w la acepte sii w ∈ L. Ayuda: M´ debería invocar a M.
  + M’ deberia ejecutar M sobre una cinta n, asi en esta cinta n quedaria el lenguaje L, entonces M’ recorreria el input y buscaria que alguna de las palabras generadas por M coincida con w. De hacerlo M’ acepta y de lo contrario rechaza
* ¿Por qué no puede existir una reducción del lenguaje HPC al lenguaje HP?
  + Si HPC se reduce a HP como HP esta en RE, por las propiedades de las reducciones HPC tambien estaria en RE y si HP y su complemento estan en RE, significaria que HP esta en R. Lo cual es absurdo.
* Vimos en clase que se puede reducir el lenguaje HP al lenguaje Lu por medio de una función que primero valida la sintaxis del input y luego, si es correcto, es decir si tiene la forma (<M>,w), lo transforma cambiando los estados qR de <M> por qA, y manteniendo w. Explicar por qué esta función es total computable.
  + Es total computable porque siempre pararia, ya que primero valida el input y luego lo volveria a recorrer cambiando los qR por qA entonces siempre recorreria dos veces el input tomando tiempo polinomial, O(2n) = O(n)
* Turing demostró que la lógica de predicados de primer orden no es recursiva, encontrando una reducción del halting problem (lenguaje HP) a dicha lógica. Si la misma fuese recursiva, también lo sería HP. ¿Por qué?
  + Por el lema b de las reducciones, dice que si L1 α L2 y L2 ϵ R entonces L1 ϵ R
* Sea L = {wi | Mi acepta wi}. Plantear la idea general de una reducción de L a LU. Ayuda: Usar la definición del lenguaje LU.
  + La funcion podria simplemente generar Mi y generar el output <Mi>, wi

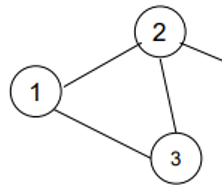
Si wi pertenece a L, f(wi) pertenece a Lu ya que, dada una Mu que reconozca Lu, si Mi acepta wi entonces aceptaria y si wi no pertenece a L entonces f(wi) no pertence a Lu porque Mi no aceptaria wi.

**Parte 2. Complejidad Computacional.**

* Probar que todo lenguaje finito pertenece a la clase P.
  + Se puede construir una MTD que tenga tantos estados como caracteres tengan todas las palabras. Entonces primero trata de reconocer la primera palabra (o conjunto de simbolos) si no puede pasa a la siguiente y asi sucesivamente.
* Sean f una función computable en tiempo determinístico polinomial y A un lenguaje de la clase P. Probar que el lenguaje B = {w | f(w) ∈ A} también pertenece a P.
  + Si A pertenece a P significa que hay una MTD Ma que lo reconoce en tiempo poly(n) y tambien sabemos que hay una MT Mf que computa la funcion de transicion. Entonces se puede construir una MTD Mb que dado un w ∈ B, este pasa por la Mf luego el output se pone como input de la Ma, reconociendo el lenguaje en tiempo poly(n). Entonces B ∈ P
* Probar que L = {(<M>, w, 1k) | M es una Máquina de Turing determinística con una sola cinta que acepta w en a lo sumo k pasos} ∈ P.
  + Se puede construir una MTD M’ que dado un input si no cumple con la forma <M>, w, 1k rechaza y de cumplir
    - Copia w en la cinta 2
    - Copia el contador unareo en la cinta 3
    - Ejecuta M sobre w en la cinta 2
      * Por cada paso de <M> se borra un 1
        + Si se queda sin 1 y no termino <M> M’ rechaza
      * Si termina <M> y acepta M’ acepta
      * Si termina <M> y rechaza M’ rechaza
* Analizamos en clase el problema que plantea si un número N tiene un divisor que termina en 3. El algoritmo propuesto hacía O(N) iteraciones. ¿Esto significa que el problema está en P? Justificar la respuesta.
  + No ya que lo que importa para ver si pertenece o no a P es el tiempo en funcion del tamaño binario del numero N, siendo n = |W|, seria O(2n)
* Probar que P ⊆ NP ⋂ CO-NP.
  + Dado un L ϵ P -> L ϵ NP ya que P ⊆ NP -> como P es cerrada para el complemento LC ϵ P y LC ϵ NP (por definicion) -> como LC ϵ NP -> L ϵ CO-NP -> Entonces L ϵ CO-NP
* Sea el lenguaje SI-NO-SAT = {φ | φ es una fórmula booleana sin cuantificadores, satisfactible al menos por una asignación e insatisfactible al menos por una asignación}. Probar que SI-NO-SAT ∈ NP. Ayuda: construir una Máquina de Turing no determinística que trabaje en tiempo poly(n), y recordar que la evaluación de una fórmula booleana con una asignación de valores de verdad consume tiempo determinístico polinomial.
* ¿Qué significa que todo elemento de un lenguaje de la clase NP cuenta con un certificado suscinto? Dar un ejemplo.
  + Significa que para cualquier lenguaje de NP tiene una solucion que se puede validar en tiempo poly(n). Por ejemplo el problema en el caso del problema CLIQUE, se puede plantear un certificado suscinto, este seria un par G, G1 tal que G es un grafo y G1 es un subgrafo clique de G. Entonces verificar que G1 es un clique de G se puede resolver en tiempo polinomial, siendo asi un certificado suscinto



Por ejemplo para este grafo G, el subgrafo dado por 1, 2 y 3



es un clique de G

* El problema VAL plantea si una formula booleana sin cuantificadores es válida, es decir, si cualquier asignación de valores de verdad la satisfice. Comentar por qué VAL no estaría en la clase NP.
  + No estaria en NP porque su complemente VALC dada una formula q, ver si alguna asignacion de verdad “a” no la satisface se puede verificar en tiempo polinomial, siendo a el certificado suscinto.

Entonces como VALC pertenece a NP, si complementeo (VAL) pertenece a CO-NP. El certificado de VAL seria ver que todas las asignaciones de verdad satisfacen la formula lo cual toma tiempo O(exp(n)) por ende no estaria en NP

* Se define que un lenguaje L es P-completo si está en P y además todos los lenguajes de P se reducen polinomialmente a L. Probar que el lenguaje {0} es P-completo. Ayuda: (a) La prueba de que {0} está en P es trivial. (b) También es sencillo construir una reducción polinomial de cualquier lenguaje L de P al lenguaje {0}: la función de reducción debería chequear si el input está o no en L y en base a ello generar un output adecuado.
  + L pertenece a P ya que se puede construir una MTD M que dado un input si es 0 acepte y de lo contrario rechace lo cual tomaria tiempo lineal O(n) (ya que solo seria recorrer el input), entonces para ver si es P-completo todos los lenguajes de P se deben reducir a el. Entonces la funcion de reduccion deberia dado un input w, siendo w = {<M>, w}. f(w) si <M> acepta w f(w) generaria un 0 y si <M> rechaza w f(w) generaria un 1.
* Probar, dados dos lenguajes cualesquiera NP-completos, que cada uno se reduce polinomialmente al otro.
  + Dados dos lenguajes L1 y L2 que pertenecen a NPC. Por la definicion de NPC ambos pertenecen a NP y todos los lenguajes de NP se reducen a ellos. Entonces como L1 pertenece a a NP y L2 a NPC L1 se reduce a L2 Y por otro lado como L2 pertenece a NP y L1 a NPC entonces L2 se reduce a L1.
* Vimos en clase el lenguaje TSP que representa el problema del viajante de comercio. Se prueba que TSP es NP-completo. Explicar por qué, si existe una reducción polinomial de TSP a un lenguaje L de la clase NP, entonces L también es NP-completo.
  + Esto se da porque, para ser NPC o NP-Completo se necesita que el lenguaje L este en NP (se da por definicion) y que todos los lenguajes de NP se reduzcan a el.
  + Como ya sabemos que TSP se reduce polinomialmente a L y que TSP pertenece a NPC, gracias a la propiedad de transitividad de las reducciones, todos los lenguajes de NP se reduciran a L.
  + Entonces L pertenece a NP y todos los lenguajes de NP se reducen a el, por ende L pertenece a NPC.
* El problema FSAT plantea encontrar una asignación de valores de verdad que satisfaga una fórmula booleana sin cuantificadores (si no existe ninguna el output es “no”). Suponiendo que una Máquina de Turing M resuelve FSAT en tiempo poly(n), se prueba fácilmente que SAT ∈ P. Comentar la idea general de la prueba. Ayuda: la Máquina de Turing que decida SAT en tiempo poly(n) debería invocar a M.
  + SAT estaria en P ya que se puede construir una MT M’ que reconozca SAT en tiempo poly(n). La M’ dado un input w si no es una formula booleana rechaza, de lo contrario ejecuta M sobre w y si el output de M es distinto de “no” M’ acepta y de lo contrario rechaza, como la ejecucion de M es poly(n) y chequear el output es O(n) la ejecucion final de M’ es O(n) + O(poly(n)) = O(poly(n)) demostrando que SAT esta en P.
* Volviendo al lenguaje QBF: se indicó en clase que pertenece a la clase PSPACE, ¿Qué significa que QBF pertenece a PSPACE?
  + Que se puede construir una MT M que reconozca QBF en espacio O(poly(n)) en funcion del tamaño del input.
* ¿Cuál es la clase espacial que se considera como clase de problemas de resolución eficiente en el marco de la complejidad espacial? Justificar la respuesta.
  + La clase espacial es LOGSPACE ya que si una MT M trabaja en espacio log n entonces trabaja en tiempo clog n con c constante, entonces M trabaja en tiempo poly(n)
* Explicar por qué la clase temporal P está incluida en la clase espacial PSPACE.
  + Porque, dado una MT M que trabaje en tiempo poly(n) como no puede escribir mas celdas que las que recorre, trabajara tambien en espacio poly(n).