P vs NP

Daniel Penazzi

7 de mayo de 2021

Tabla de Contenidos

- Este tema es uno de los mas importantes dentro de las ciencias de la computación.
- Al menos para los que hacen teoría .
- El tema requeriría todo un curso, pero como hay muchos temas en esta materia, sólo podremos dar un pantallazo de la superficie del mismo.
- En realidad para verlo bien formalmente deberiamos esperar a que vean bien maquinas de Turing en 4to año, pero el tema lo han puesto en esta materia ahora en tercer año, asi que hacemos lo mejor posible en el contexto dado.
- Un problema de decisión es un problema cuyas unicas respuestas posibles son “SI" o “NO".
- Algunos de estos problemas tienen “nombre” por ser usados frecuentemente como ejemplo.

Problemas de decisión

- Ejemplos:
  - PRIMO: Dado un numero natural n, ¿Es n primo?
  - INVERTIBILIDAD: Dada una matriz A cuadrada (es decir, $n \times n$ para algún n), ¿Es A invertible?
  - CONECTITUD: Dado un grafo G, ¿Es G conexo?
  - v-FLUJO: Dado un network N y vertices s, t ¿existe un flujo de s a t con valor v?
  - k-COLOR: Dado un grafo G ¿Existe un coloreo propio con k colores de G?
  - k-CLIQUE: Dado un grafo G ¿Existe un subgrafo completo de G con k vértices? (la “clique")
- En el caso de v-FLUJO, k-COLOR y k-CLIQUE lo que tenemos es una familia de problemas: un problema para cada k, o cada v.

Problemas de decisión

- En general serán de la forma:
- “Dado $x \in I$, ¿tiene x la propiedad Q?”
- Los elementos del conjunto I se llaman “instancias" del problema.
  - En el caso de PRIMO, una instancia es un número.
  - En el caso de INVERTIBILIDAD, una instancia es una matriz,
  - En el caso de k-COLOR, una instancia es un grafo.

Problemas de decisión

- Los problemas de decisión son todos casos particulares de
- “dados" conjuntos I, S con $S \subseteq I$, y dado $x \in I$, ¿está $x \in S$?
- El problema es que en general el S no está “dado" explícitamente sino en terminos de una propiedad, que puede ser difícil de verificar.
- En general se tendrá un algoritmo para, dado $x \in I$, poder verificar si $x \in S$ o no, de alguna forma.
- Y los problemas de decisión se clasifican, como los problemas de cálculo de algúna cantidad, de acuerdo con su complejidad.

Problemas de decisión

- La mas obvia es la clase P de “polinomial".

Definición

La clase P es la clase de problemas de decisión tal que existe un algoritmo que lo resuelve que es polinomial en el tamaño de sus instancias.

- Por ejemplo, deberían haber visto en Análisis Númerico que INVERSIBILIDAD está en P.
- Dado que BFS y DFS son polinomiales y cualquiera de ellos resuelve CONECTITUD, entonces CONECTITUD está en P

Ejemplos de problemas en P

- v-flujo está en P, por ejemplo usando Edmonds-Karp
- Dado que 2-COLOR es el problema de determinar si un grafo es bipartito o no, y como hemos dado un algoritmo polinomial que hace eso, vemos que 2-COLOR está en P.
- Pero no hemos visto ningún algoritmo polinomial para, pej, 3-COLOR, asi que por ahora no sabemos si 3-COLOR está en P o no.
- De hecho, nadie sabe si 3-COLOR está en P o no
- Y quien pueda responder esa pregunta, por la afirmativa o negativa, se gana un millón de dolares.

Sutilezas

- Estamos hablando de “polinomial en el tamaño de sus instancias" pero ¿qué es el “tamaño" de una instancia?
- Es la cantidad de bits necesarios para representar la instancia, o algo que este relacionado con eso.
- Por ejemplo, en un grafo el número de vértices y el número de lados (juntos) da una idea del tamaño de la instancia, asi que “polinomial" es polinomial en n, m.
- En una matriz, el número de filas y columnas es el “tamaño".
- Pero en las instancias de PRIMO, es decir, un número, ¿cual es el “tamaño"?

P

- Se puede estar tentado de decir que el tamaño de n es n, y en ese caso PRIMO seria claramente polinomial, pues el algoritmo de mirar todos los números menores a ver si alguno divide a n es $O(n)$.
- Pero eso estária mal salvo que usaramos una computadora que “guarda" el número n usando n bits.
- Pero sabemos que no: un número n se representa usando $\ell = lg(n)$ bits, donde lg es el logaritmo en base 2.
- Asi que un algoritmo $O(n)$ es un algoritmo $O(2^{\ell})$, exponencial en el tamaño ℓ con el cual estamos representando a n y el algoritmo naive de chequear que un número es primo mirando todos los números menores que el es exponencial, no polinomial.

P

- Por ejemplo, cuando pasamos de números de 30 bits a números de 90 bits, no tenemos que chequear el triple de números: Tenemos que chequear $2^{60} \simeq 10^{18}$= un trillón de veces mas números.
- (no un trillón de números: un trillón DE VECES mas números)
- Asi que el algoritmo “naive" para PRIMO no es polinomial.
- Luego hablaremos mas de este problema.
- Hay otra sutileza con el concepto de polinomial en el tamaño de la entrada.
- k-CLIQUE nos permite explorar esa sutileza.

k-CLIQUE

- Para k-CLIQUE podemos buscar todos los subonjuntos de k vértices de G.
- Y comprobar si hay lados entre todos ellos.
- Si alguno da que si hay lados entre todos los vértices, damos como respuesta SI, de lo contrario, damos como respuesta NO.
- La comprobación para cada uno es $O(k^2)$.
- Y la cantidad de subconjuntos de k vértices de G es $\binom{n}{k}$ que es un polinomio de grado $min\{k, n-k\}$ en n.
- Asi que k-CLIQUE es polinomial, para todo k.

CLIQUE

- Observar que el algoritmo anterior es $O(n^{min\{k,n-k\}})$ asi que si bien k-CLIQUE es polinomial para todo k, el algoritmo no es muy útil cuando k es grande.
- Pej 10000-CLIQUE es $O(n^{10000})$ si $n > 20000$, lo cual hace imposible resolverlo para esos n.
- Asi que no siempre “estar en P" significa que exista un algoritmo “bueno" que lo resuelva.
- Pero desde el punto de vista de teoría de complejidad, k-CLIQUE$\in P\ \forall k$.
- Esto muestra una vez mas que es un error asumir que para demostrar $\chi(G) < k$ basta con probar que no hay ningún subgrafo completo con k vértices.

CLIQUE

- Vieron varios ejemplos en los prácticos, pero ademas desde el punto de vista de complejidad computacional, k-CLIQUE$\in P$ $\forall k$ pero no se sabe ni siquiera si 3-COLOR$\in P$ o no.
- Hay un problema similar a k-CLIQUE, llamado CLIQUE. (sin la k).
- CLIQUE es:
  - "Dado un grafo G y un número k, ¿Existe un subgrafo completo de G con k vértices?
- En CLIQUE, k es PARTE DE LA INSTANCIA.
- Es decir, 2-CLIQUE es polinomial, 3-CLIQUE es polinomial, 4-CLIQUE es polinomial, etc, pero ahi 2,3,4 son números fijos, independientes de n.

CLIQUE

- En cambio en CLIQUE el k es parte de la instancia, y eso hace toda la diferencia pues podria depender de n.
- Podriamos tomar una instancia donde k fuese igual a $\frac{n}{2}$
- Y si quisieramos usar el anterior algoritmo deberiamos revisar todos los $\binom{n}{\frac{n}{2}}$ subconjuntos de $\frac{n}{2}$ vértices de G.
- Y $\binom{n}{\frac{n}{2}}$ no es polinomial en n, asi que ese algoritmo no es polinomial.
- Pero mas aún, no se conoce ningún algoritmo polinomial que resuelva CLIQUE, se sospecha que no existe ninguno, y determinar si CLIQUE está en P o no tiene un premio de un millón de dolares.

NP

- La siguiente clase de complejidad de algoritmos de decisión que estudiaremos es la clase *NP*.
- Las siglas son engañosas porque inducen a mucha gente a creer que *NP* significa “No Polinomial” y esto no es correcto.
- De hecho, resulta que lejos de ser clases disjuntas, lo que tenemos es que $P \subseteq NP$.
- La “N” es de “No determinístico”y *NP* significa “No determinístico polinomial.

NP

- Un algoritmo no determínistico es un algoritmo que en algunos lados toma decisiones no determísticas.
- Como dije, en realidad habria que esperar para definirlo técnicamente a ver maquinas de Turing y especificamente esperar para ver maquinas de Turing no determísticas.
- Pero la idea del concepto es entendible.
- Pej, un algoritmo no determístico para colorear un grafo seria el main de la parte 3 del proyecto que les dimos, pues hay varias partes donde se hacen elecciones no determísticas para ordenar ya sea los vértices o los colores.

NP

- Y ¿qué querria decir que un algoritmo no determinístico sea polinomial?
- Simplemente que para todo conjunto particular de decisiones no determinísticas que el algoritmo haga, el tiempo total para esa elección sea polinomial en el tamaño de las entradas.
- Obviamente si la cantidad de opciones no determinísticas es finita, podemos transformar el algoritmo no determinístico en uno determinístico simplemente recorriendo todas las opciones....pero si la cantidad de elecciones no es polinomial, el algoritmo determinístico asociado no será polinomial, mientras que el algoritmo no determinístico quizas si lo sea.

NP

- Pero aca hay una diferencia con el concepto de “resolver” para el caso determístico del caso no determinístico.
- En el caso determinístico, como justamente todas las opciones son determínisticas, para una instancia cualquiera x del problema, la respuesta será siempre la misma.
- Pero en el caso no determinístico, como puede haber distintas elecciones, la respuesta podria no ser siempre la misma.
- Pej si corremos el algoritmo del main de la parte 3 para intentar resolver 3COLOR y cuando nos detenemos decimos “SI” si coloreamos el grafo con 3 colores o menos, y “NO” si no, muy probablemente tendremos distintas respuestas para distintas elecciones.

NP

- En realidad, observemos que tendriamos respuestas distintas si el grafo es 3-coloreable, si no las respuestas siempre seran “NO”
- Pero hay una diferencia: una respuesta “NO” no garantiza que el grafo no sea 3-coloreable, mientras que una respuesta “SI” garantiza que se colorea con 3 colores.
- Asi que la respuesta “NO” en realidad deberia ser “NO SÉ”.
- Entonces ¿qué quiere decir que un algoritmo no determísitico “resuelva” un problema?
- Como el ejemplo lo indica, en realidad tambien hay que aclarar qué significa que lo resuelva para “SI”o que lo resuelva para “NO”

NP

- Decir que un algoritmo no determísitico “resuelve” un problema de decisión para el “SI” es decir que:
  - Sus únicas respuestas posibles son “SI” o “NO SÉ”.
  - Si la respuesta que da el algoritmo con una instancia x dada es “SI”, entonces la respuesta del problema de decisión con instancia x es “SI”.
  - Si la respuesta del algoritmo es “NO SÉ ” no podemos concluir nada.
  - Si la respuesta del problema a la instancia x es “SI”entonces existe alguna opción de las elecciones no determinísticas que hacen que la respuesta del algoritmo sea “SI”.
- Similar para resolver para el “NO” , intercambiando todos los “SI” por “NO” .

NP

- Supongamos que miramos la versión del main con parametro *a* fijado en 0. (es decir, sin hacer nunca ningún AleatorizarVertice).
- Este es un algoritmo que colorea un grafo no determinísticamente, pero no “resuelve” el problema pej, 3-COLOR
- Pues no tenemos ninguna garantia que simplemente cambiando el orden de los colores y ordenando los vértices según ese orden podamos calcular $\chi(G)$, aún si revisamos todos los ordenes posibles de colores.

NP

- En cambio con parametro $a > 0$, ahi sabemos que “resuelve” el problema (para el “SI”) porque vimos que algún orden de los vértices causa que Greedy de como respuesta $\chi(G)$.
- Pero no resuelve el problema para el “NO” porque no obtener un coloreo con 3 colores con el algoritmo no garantiza que la respuesta al problema sea “NO” .
- Ya podemos definir la clase *NP*.

NP

Definición

La clase *NP* es la clase de problemas de decisión ρ que pueden ser "resueltos" por algún algoritmo polinomial no determinístico para todas las instancia *x* de ρ cuya respuesta sea "SI".

- La clase co-NP tiene la misma definición reemplazando SI por NO en la definición.

Ejemplos de problemas NP

- Es obvio que $P \subseteq NP \cap co - NP$ pues todo algoritmo determínistico se puede mirar como un algoritmo no determinístico.
- Asi que cualquiera de los ejemplos de P está en NP, pej.
- Un problema que está en NP y no sabemos si está o no en P es 3-COLOR.
- Está en NP pues dimos un algoritmo no determinístico que lo resolvia para el caso “SI”
- De hecho, otro algoritmo no determínistico polinomial para el “SI” es simplemente elegir no determinísticamente algún coloreo con 3 colores de los vértices y verificar polinomialmente si ese coloreo es propio.

Ejemplos de problemas NP

- Lo que nadie sabe es si 3-COLOR$\in co-NP$, es decir, no se conoce ningún algoritmo polinomial no determinístico (y obviamente tampoco determinístico) que resuelva el problema para el "NO" .
- Vimos que k-CLIQUE$\in P$ pero mostramos la diferencia con CLIQUE y dijimos que nadie sabe si CLIQUE$\in P$.
- Pero es claro que CLIQUE está en NP:
  - Elegir no determísticamente un subconjunto de k vértices.
  - Verificar si es un conjunto completo.
  - Esto es $O(k^2)$, polinomial en k.
- Tampoco nadie sabe si CLIQUE$\in co-NP$ o no.

Ejemplos de problema $co - NP$

- Es fácil ver que PRIMO$\in co - NP$.
- Simplemente dado n elegimos no determinísticamente un par de números $1 < a, b < n$ y chequeamos a ver si $n = ab$.
- Si $n = ab$ respondemos “NO” .
- La multiplicación de dos números es polinomial en el número de bits, y si la respuesta es “NO” es porque efectivamente $n = ab$ asi que n definitivamente no es primo.
- Por otro lado, si n no es primo, entonces a, b existen, asi que alguna elección no determinística daria la respuesta “NO”.
- Pero si $n \neq ab$, eso no dice que n sea primo, solo que no se factoriza como ab.

PRIMO

- Asi que este algoritmo prueba que PRIMO$\in co - NP$ pero no dice nada sobre si PRIMO$\in NP$ o no.
- A diferencia de CLIQUE o 3COLOR, aca se sabe si PRIMO$\in NP$.
- El algoritmo no es obvio y recién se dió en 1975, pero se probó que efectivamente PRIMO$\in NP$, es decir, existe un algoritmo no determinístico polinomial que resuelve el caso “SI”
- Por lo tanto PRIMO está en $NP \cap co - NP$
- En 2002 un profesor indio, Maninda Agrawal, junto con dos estudiantes, Neeraj Kayal y Nitin Saxena, quisieron hacer un ensayo mostrando porqué, a pesar de estar PRIMO en $NP \cap co - NP$, no estaba en P....y terminaron probando que en realidad PRIMO está en P (!)

PRIME

- El algoritmo que desarrollaron se llama el test de primalidad de AKS por las iniciales de su nombre.
- Agrawal,Kayal y Saxena ganaron el premio Gödel por su algoritmo....y también el premio Fulkerson, que es un premio en honor al Fulkerson de Ford-Fulkerson por contribuciones significativas en el area de matemática discreta.
- El AKS original tenia una complejidad de $\tilde{O}(\ell^{12})$ donde $\ell = lg(n)$.
- (es decir, era $O(\ell^{12} lg(\ell)^k)$ para algún k).
- Y luego se ha reducido a $\tilde{O}(\ell^6)$.
- En la práctica no se usa pues es mas lento que otros métodos, que son probabilísticos.

PRIME

- Es decir, hay tests de primalidad que, dado n, si dicen NO, entonces n no es primo, y si dicen SI, entonces hay una probabilidad p de que n sea primo.
- Esa probabilidad p se puede hacer tan cerca de 1 como quiera, haciendo mas iteraciones del test.
- Y los algoritmos usados son mas rápidos que AKS
- Hay un algoritmo determinístico de los 70s que es polinomial $O(\ell^4)$....excepto que no se sabe si es correcto o no, pues la correctitud depende de una hipotesis de matemática que no ha sido probada. (la hipotesis generalizada de Riemann)

FACTORIZACIÓN

- Hay un problema similar a PRIME, que suele llamarse FACTORIZACIÓN
- El problema es: Dado n y $k \leq n$ ¿Existe a con $1 < a < k$ y $a|n$?
- Es decir: Dado n y $k \leq n$, ¿tiene n un factor menor que k?
- Obvio que si n es primo la respuesta es NO, pero aún si supieramos que n no es primo, podria no tener ningún factor menor a k.
- Claramente está en NP: simplemente buscamos no determinísticamente un número menor que k y verificamos polinomialmete si divide a n.

FACTORIZACIÓN

- Como PRIMO está en P, se puede probar que FACTORIZACIÓN está en co-NP también:
  1. Elegir no determinísticamente $a_1, ..., a_r$ todos mayores o iguales que k.
  2. Verificar que $n = a_1 ... a_r$ (multiplicar números es polinomial en el número de bits)
  3. Verificar que cada uno de los a_i es primo. (polinomial gracias a AKS)
  4. Si [2] y [3] son por la afirmativa, responder “NO” , si no , “NO SÉ”.

FACTORIZACIÓN

- Este algoritmo correctamente resuelve el caso “NO”pues:
  1. Si se verifica todo eso, entonces sabemos que $n = a_1 ... a_r$ es la factorización prima de n, y como esta es única, sabemos que no puede haber ningún factor menor a k.
  2. Por otro lado, si efectivamente n no tiene ningún factor menor a k, su factorización prima debe ser de esa forma y por lo tanto alguna elección de a_i’s va a dar la respuesta “NO”.
- Entonces FACTORIZACIÓN$\in$NP$\cap$co-NP.
- ¿Podria ser que, al igual que PRIME, en realidad FACTORIZACIÓN esté en P?

FACTORIZACIÓN

- Nadie lo sabe, y si la respuesta fuese SI, con un algoritmo razonable, una parte importante de la seguridad de internet se caeria.
- Esto es porque una buena parte de pej las transacciones con tarjetas de crédito o la transmisión de passwords de mail, etc dependen de que no exista ningún algoritmo eficiente para resolver FACTORIZACIÓN.
- Asi que si existiese uno, habria que cambiar casi toda la estructura que se está usando.

- En realidad ya se están tomando pasos para hacer esto si fuese necesario, pues HAY un algoritmo polinomial que resuelve FACTORIZACIÓN....sólo que no corre en computadores “normales".
- Es el algoritmo de Shor, que es polinomial en una computadora cuántica
- Las clases de complejidad de las que estamos hablando son clases de complejidad en maquinas comunes (o, como verán el año que viene, en máquinas de Turing)
- Al momento no existen computadoras cuánticas suficientemente grandes como para implementar Shor en números que tengan sentido (el número mas grande factorizado hasta hace poco tiempo era 21).

- Algunos sostienen que nunca podrán construirse computadoras cuánticas suficientemente grandes como para resolver este problema para números grandes.
- Pero debido a la posibilidad de que si se puedan construir, hay una competencia internacional para elegir algoritmos “inmunes" al algoritmo de Shor, que reemplacen a los algoritmos actuales que dependen de FACTORIZACIÓN u otros problemas similares que tambien son vulnerables a Shor.

- Volviendo a lo nuestro, muchos problemas en $NP \cap co - NP$ se han terminado probando que están en P.
- Pero hay otros que no se sabe, como FACTORIZACIÓN.
- Otro problema que se sabe que está en $NP \cap co - NP$ pero no se sabe si está en P es ISOMORFISMO DE GRAFOS: dados grafos G y H ¿son G y H isomorfos entre si?
- Una pregunta que ha sido hecha es: ¿Será cierto que $P = NP \cap co - NP$?
- Respuesta: nadie sabe.

P y NP

- Si lo resuelven tienen fama eterna y un puesto en cualquier universidad del mundo que quieran.
- Otra pregunta sobre estas clases que nadie sabe la respuesta es si $NP = co - NP$ o no.
- Tambien tendrán fama eterna si lo resuelven.
- Pero hay otra pregunta que no sólo les dará fama eterna y un puesto en cualquier universidad del mundo que quieran si lo resuelven si no que ademas tiene un premio de un millón de dolares y cuya respuesta positiva implicaria que $NP = co - NP$ y que $P = NP \cap co - NP$.

P y NP

- Esa pregunta es: ¿Es $P = NP$?
- De hecho, parece de locos plantearsela.
- Si $P = NP$ estariamos diciendo que todo problema que puede ser resuelto por un algoritmo polinomial no determinístico, puede ser resuelto por uno polinomial determínistico, y eso no parece razonable.
- Mas aún si pensamos que la clase NP es en realidad igual a la clase de algoritmos verificables polinomialmente.
- ¿Qué quiere decir esto?

Certificados

- En muchos problemas de decisión tenemos “certificados" para el problema: algo que, sin importar como lo conseguimos, nos “certifica" una de las respuestas posibles al problema.
- Por ejemplo, en INVERTIBILIDAD, un certificado es una matriz B tal que $AB = I$.
- Observemos que verificar que $AB = I$ se hace en tiempo polinomial.
- En este caso sabemos que podemos calcular B en tiempo polinomial, pero podria pasar (de hecho, pasa) que tuvieramos un problema donde calcular el certificado no sea obvio, pero que si nos lo dan, verificar que es un certificado si sea polinomial.

Certificados

- En el caso de INVERTIBILIDAD, una matriz B para la cual queremos chequear que $AB = I$ es un certificado para la respuesta “SI", pero no para la respuesta “NO" puesto que si $AB = I$ es verdadero, sabemos que A es inversible, pero si $AB \neq I$, no sabemos si A es o no inversible, sólo sabemos que B no es su inversa.
- Un certificado para el NO seria un vector columna X no nulo tal que $AX = 0$, como vieron en álgebra.

Certificados

- Que exista un certificado para el SI no necesariamente implica que exista uno para el NO, y viceversa. o al menos no es obvio que exista uno.
- En el caso de INVERTIBILIDAD lo de los certificados es medio irrelevante pues INVERTIBILIDAD$\in P$.
- Pero si un problema no está, o no se sabe que esté, en P, la existencia de certificados es mas relevante.
- Podriamos tener un problema que no sepamos resolver en tiempo polinomial, pero podamos verificar en tiempo polinomial.
- Es decir, si alguien “nos da” algo que podria ser un “certificado” , verificar si ese certificado es efectivamente correcto o no podria ser solamente polinomial.

Certificados

- Para que algo sea certificado para el “SI” debe pasar que si la verificación dice que es correcto, entonces la respuesta es “SI”.
- Y pedimos ademas que si la respuesta es SI, entonces exista un certificado para el SI.
- (similar para NO).
- Pero entonces queda claro que *NP* es la clase de problemas cuyas instancias de “SI” pueden verificarse polinomialmente.(y $co - NP$ lo mismo con el NO).

Certificados

- Pues si un problema esta en *NP* el certificado es simplemente la elección concreta de elecciones no determísticas que hicimos.
- Por otro lado si un problema es verificable polinomialmente, tomamos un algoritmo no determínistico que elija un certificado al azar y lo verifique, asi que esto demuestra que si es verificable polinomialmente está en *NP*.
- Entonces estariamos diciendo, si P=NP, que todo problema que puede verificarse en tiempo polinomial puede resolverse en tiempo polinomial.

Certificados

- Pej, como verificar una password es polinomial, estarian diciendo que hay un algoritmo polinomial que encuentra passwords.
- La mayoria de los computólogos (pero no todos) creen que esto no puede ser cierto.
- Cierto o no, nadie ha podido probar en el casi medio siglo desde que se hizo la pregunta por primera vez, si P es igual o no es igual a NP.
- Es el problema mas famoso de las ciencias de la computación
- Este es un tema para cursos enteros, aca sólo daremos un breve pantallazo de las ideas detras de toda esta área.

P y NP

- Probar $P = NP$ parece ademas imposible: habria que dar un algoritmo polinomial PARA CADA problema de NP!
- Bueno, en realidad no es tan asi, pero necesitamos otra definición la cual daremos la clase que viene.
- Para probar P$\neq$NP habria que dar un problema que esté en NP y probar que no hay ningún algoritmo polinomial capaz de resolverlo.
- Probar que no existe tal algoritmo es dificil, como lo prueba casi medio siglo de intentos.

P y NP

- Pero ademas, pareceria que hay que encontrar el problema “justo" para probarlo, porque como muestra la historia de PRIMO, que nadie conozca un algoritmo polinomial o parezca que no existe, no significa que en el futuro un grupo de estudiantes indios o chinos encuentre uno.
- Uno de esos problemas “candidatos” es 3-COLOR.
- Para terminar la clase de hoy veremos otro “candidato” a ser ese problema, muy famoso.

CNF

Definiciones

- Una variable booleana es una variable que solo toma los valores 1 (“True") o 0 (“False").
- Una expresion booleana es una función en variables booleanas.
- Dadas variables booleanas $x_1, ..., x_n$, un literal es una variable x_i o la negación de una variable: $\overline{x}_i$.
- Dadas expresiones booleanas $B_1, ..., B_m$:
  - La disjunción de ellas es $B_1 \vee ... \vee B_k$
  - La conjunción de ellas es $B_1 \wedge ... \wedge B_m$.
- Una expresion booleana esta en forma conjuntiva normal (CNF: conjuntive normal form) si es una conjunción de disjunciones de literales.

CNF

- Ejemplo de una booleana en CNF:

$$(x_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3 \vee x_4) \wedge (\overline{x}_2 \vee \overline{x}_3) \wedge (x_1 \vee \overline{x}_2 \vee \overline{x}_4)$$

- Tambien se puede definir la forma disjuntiva normal (DNF: disjuntive normal form) si es una disjunción de conjunciones de literales.
- O la forma algebráica normal (AFN) si es un xor de conjunciones de variables mas el número 1.
- Pero la que es útil en este contexto es la CNF.

SAT

Definición

El problema SAT es el problema: Dada una expresión booleana B, ¿existe un asignamiento de valores a las variables de B que la vuelven verdadera?

- Por ejemplo, con $B(x_1, x_2, x_3, x_4)$ la expresión booleana de la pagina anterior, podemos asignar los valores a las variables: $x_i = 1$ para todo $i \neq 2$, $x_2 = 0$ y $B(1, 0, 1, 1) = 1$ es decir SAT es SI para B.
- Si B está en DNF o ANF es muy fácil resolver SAT para ella.
- Pero si B está en CNF no. Nadie sabe como hacerlo en tiempo polinomial.

CNF-SAT

- El problema CNF-SAT es igual a SAT, pero cuando se especifica que la expresión booleana B debe estar en CNF.
- Como siempre usaremos esto, y ademas porque es común hacerlo asi, de ahora en mas SAT significa CNF-SAT.
- Este problema está claramente en NP pues simplemente elegimos no determinísticamente un asignamiento de valores a las variables y luego verificamos polinomialmente si B lo satisface.
- Como dije, nadie sabe si (CNF-)SAT está en P o no, y es uno de los fuertes candidatos a que se demuestre que $P \neq NP$ demostrando que no SAT no se puede resolver en tiempo polinomial.
- Pero ademas el problema tienen otras propiedades extras, que veremos la próxima clase.