EXERCICI 1

(a) Demostreu que "3SAT - Twice at most" és NP-complet.

Per a demostrar que "3SAT - Twice at most" és NP-complet, hem de verificar dues propietats:

- És **NP**: Donada una interpretació I (amb espai polinòmic respecte la mida de la fórmula) podem recórrer linialment (polinòmic) i evaluar cada clàusula amb la interpretació I. Si totes les clàusules evaluen a cert, I és un model.
- És **NP-Hard**: Farem una reducció polinòmica 3-SAT <= "3SAT - Twice at most" i, donat que 3-SAT és **NP-complet**, haurem demostrat que "3SAT - Twice at most" és **NP-Hard**.

L'algorisme que implementa la reducció iterarà per tots els símbols de predicat, i farà un seguit de comprovacions:

Si la variable apareix com a màxim dues vegades positiva i dues negativa, mantenim la fórmula tal i com està i passem a analitzar la següent variable.

Si el símbol apareix més de dues vegades en positiu o més de dues en negatiu, i almenys una en positiu i una en negatiu, llavors procedirem a realitzar la següent transformació. Substituirem les *k* aparicions del símbol en positiu per *k* noves variables *x_1,… , x_k*, i les *t* aparicions en negatiu per les variables *x_(k+1), x_(k+2), …, x_(k+t)* **en positiu**. Afegirem una variable *x'* extra.

Afegim les implicacions [1] $x\_i \rightarrow x\_j$ per a $1 <= i < j <= k$, la implicació [2] $x\_k \rightarrow -x\_\{k+1\}$ i implicacions [3] $-x\_i \rightarrow -x\_j$ per a $k < i < j <= k+t$. A més afegirem les implicacions [4] $-x\_\{k+t\} \rightarrow x'$ i $x' \rightarrow x\_1$ [5].

Veiem que totes les variables *x_1, … ,x_{k+t}* ja apareixen una vegada en positiu en les clàusules originals. Les implicacions [1] les podem convertir en *-x_i v x_j v -x_i*, consumint així totes les possibles aparicions en negatiu de cada *x_i* a excepció de la *x_k*, i totes les aparicions en positiu de cada *x_i* a excepció de *x_1*. L'implicació $x\_k \rightarrow -x\_\{k+1\}$ es transforma en la clàusula *-x_k* v *-x_{k+1}* v *-x_k. Les implicacions* [3] $-x\_i \rightarrow -x\_j$ en *x_i v -x_j v -x_j*. La [4] en *x_{k+t} v x' v x'* i la [5] en *-x' v -x' v x_1*. D'aquesta manera fem que tots els *x_{1..k} i* x' hagin de prendre el mateix valor, i els *x_{k+1 .. k+t}* hagin de prendre el valor oposat.

Si el símbol només apareix en positiu o negatiu i *k* vegades, *k > 2*, llavors només afegim les variables *x_1, …, x_k* afegim les implicacions $x\_i \rightarrow x\_j$ per a $1 <= i < j <= k$ amb la clàusula *-x_i v x_j v -x_i*.

El número de clàusules noves afegides és polinòmic respecte el número d'aparicions dels literals amb >2 aparicions en positiu o negatiu. Per tant, l'algorisme que computa la reducció és polinòmic respecte l'entrada 3-SAT. A més, si existeix un model I per a la CNF de 3-SAT existeix un model I' per a la nova CNF on, si els *x_i* substitueix el literal *y*, llavors *I'(x_i) = I(y).* Si existei*x* un model per a la CNF 3SAT-Twice at most, llavors donat que per cada variable les seves *x's han de prendre el mateix valor (per la cadena d'implicacions),* el mateix model funciona per a 3-SAT original.

"3-SAT Twice at most" és NP-complet.

(b) Demostreu que “3SAT – Once at most” és decidible en temps polinòmic.

L’algorisme que presentarem iterarà per tots els símbols de predicat i farà una sèrie de comprovacions.

Si un símbol apareix només una vegada en positiu o negatiu a la clàusula *C*, podem assignar-li el valor per a que el literal sigui cert i podem eliminar la clàusula *C* de la fórmula. Aquesta acció no modifica pas la satisfactibilitat de la CNF.

Si un símbol apareix una vegada en positiu i una en negatiu en la mateixa clàusula *C*, la clàusula *C* és una tautologia i per tant podem eliminar-la de la fórmula. Aquesta acció tampoc modifica pas la satisfactibilitat.

Si un símbol *p* apareix una vegada en positiu a la clàusula (*p v C)* i una vegada en negatiu en la clàusula (*-p v D),* podem aplicar la regla de resolució i substituir les dues clàusules per una sola clàusula (*C v D).*

Si, després d’aplicar un pas de resolució, ens quedem amb una clàusula buida (és a dir, tenim la clàusula *p* i la clàusula *-p* per a una variable *p*), llavors la fórmula és insatisfactible. Si, en canvi, som capaços d’eliminar totes les clàusules de la CNF, llavors som capaços de satisfàcer totes les clàusules amb el mateix model i, per tant, la fórmula és satisfactible.

L’algorisme és polinòmic doncs cada iteració s’elimina una variable de la fórmula de l’entrada, i és correcte, doncs la resolució és una deducció lògica correcta i refutacionalment completa (si la fórmula és insatisfactible llavors generarem la clàusula buida en la clausura sota resolució).

EXERCICI 2 – CONJUNT DOMINANT

Demostreu que “Conjunt-Dominant” és NP-complet.

Per a veure que “Conjunt-Dominant” o DOMINATING-SET (DS) és NP-complet, hem de verificar dues propietats:

- És **NP**: Donat un subconjunt de vèrtexs D i una *b,* podem verificar que |D| <= b i podem recórrer V i verificar si per a tot vèrtex *v* es compleix que *v* és a D o un vèrtex adjacent a *v* pertany a D. Això es pot fer en temps polinòmic respecte el graf.
- És **NP-Hard**: Farem una reducció polinòmica VERTEX-COVER <= DOMINATING-SET i, donat que VERTEX-COVER és **NP-complet**, haurem demostrat que DOMINATING-SET és **NP-Hard**.

Recordem la versió decisional del problema del VERTEX COVER:

> Donats $G = (V, E)$ i un enter $k > 0$, decidir si existeix un subconjunt W de V, amb |W| <= k, t.q. tota aresta sigui adjacent a almenys un vèrtex de W.

Volem transformar instàncies (*G, k*) del VC a instàncies del DS en temps polinòmic. Primerament, creem el graf G’ de la següent manera. Inicialment G = G’. Per a cada aresta *{u,v}* de G, creem un nou vèrtex *w_uv* a G’ i afegim les arestes {*u, w_uv}* i {*v, w_uv*} a G’. Sigui A el subconjunt de vèrtexs aïllats a G (vèrtexs sense cap aresta adjacent), $b = k + \#A$. Retornar (G’, *b*).

L’algorisme que fa la reducció és polinòmic respecte la mida del graf i de *b*, doncs estem creant |E(G)| nous vèrtexs i afegint dos arestes per a cadascun d’aquests vèrtexs. A més, la suma és linial respecte la mida de *k*.

CORRECTESA: SI VC TÉ RECOBRIMENT LLAVORS DS TÉ CONJUNT DOMINANT

Sigui *S* un recobriment de vèrtexs a VC amb *#S* <= *k*, tenim que D = *S* U *A* és un DS per a G’. Primer, podem veure que es compleix la restricció de cardinalitat. És a dir, #D = | *S* U *A* | <= *k* + *#A* = *b.*

Per a veure que D és un DS a G’, hem de veure que tot vèrtex és de D o té un vèrtex adjacent a D. Tots els vèrtexs aïllats són dominats doncs A és un subconjunt de D. En segon lloc, tots els vèrtexs *w_uv* es corresponen a les arestes *{u, v}* de G. Donat que *S* és un VC de G, *u* membre de *S* o *v* membre de *S.* Per tant, donat que *w_uv* és adjacent a *u* i *v*, el *w_uv* queda dominat. Finalment, cada vèrtex no aïllat original *v* és incident a almenys una aresta a G, i aquesta aresta és adjacent a un vèrtex de *S.* Per tant, o bé *v és* de *S* (i per tant de *D)* o bé *v és adjacent* a almenys un vèrtex de *S* (i pertant de *D)*.

CORRECTESA: SI DS TÉ CONJUNT DOMINANT LLAVORS VC TÉ RECOBRIMENT

Volem demostrar que si G' té un cjt dominant *D* amb *D* <= *k* + *#A*, llavors G té un recobriment de mida màxima *k*. Veiem primer que necessàriament tots els vèrtexs aïllats són del conjunt dominant. Dels *k* vèrtexs restants, hem de demostrar que existeix un recobriment de mida màxima *k*.

Si algun vèrtex *w_uv* és membre de *D*, llavors perfectament podem canviar *w_uv* per *u* a *D*. La cardinalitat no augmenta, i si *D* era un DS després de la modificació ho seguirà essent, doncs *w_uv, v* i *u* seran dominats per *u*, i *w_uv* dominava aquests tres exactament.

Després d'aplicar totes aquestes modificacions possibles, anomenarem *T* el subconjunt resultant (després d'eliminar els vèrtexs aïllats i modificar els *w_uv* per *u*). Veiem que #T <= k. T és un VC a G, doncs si suposem que *{u,v}* no està cobert, *w_uv* no seria adjacent a cap vèrtex de *D*, però *D* era un cjt dominant de G'.

Q.E.D.

EXERCICI 3 - PEDRETES

El problema és pseudo-polinòmic en temps i espai respecte la mida de l'entrada.

Si intentem decidir el problema mitjançant el *game tree* i fent servir un algorisme de profunditat, veiem que donat que la profunditat és com a màxim el valor de *n*, llavors l'algorisme és pseudo-polinòmic en espai.

Si ara intentem optimitzar en temps, podem fer servir memorització (PD) per a estalviar-nos expandre una configuració ja visitada, on la configuració és un parell $< t, n >$ on t és el torn i n és el número de pedretes restants. Podem veure fàcilment que el nombre de configuracions màxim és $2*n$ (poden restar de 1 a n pedretes per a cada jugador).

Per tant, en temps serà també pseudo-polinòmic (polinòmic respecte la magnitud de n, exponencial respecte la mida de n en binari).

EXERCICI 4 – RANDOM 3SAT

Demostreu que hi ha un algorisme aleatori amb un temps esperat polinòmic tal que, donada una fórmula en Forma Normal Conjuntiva amb 3 literals per clàusula calcula una assignació que satisfà com a mínim 7/8 del nombre total de les clàusules

Primerament, considerem l'algorisme que, per a cada variable, li assignem 0 o 1 independentment i amb probabilitat ½. Per a calcular el nombre esperat de clàusules que seran satisfetes, definim la variable aleatòria Z_j = { 1 : si clausula C_j està satisfeta, 0 : altrament }. En aquest cas, el nombre esperat de clàusules satisfetes serà Z = Z_1 + Z_2 + … + Z_k.

Seguidament, E[Z_i] és igual a la probabilitat que sigui satisfeta C_i. Per tal que no ho sigui, cada literal haurà de tenir el valor 0; com que són independents tenim que 1 – E[Z_i] = (½) ³. → E[Z_i] = 7/8. Per linearitat de l'esperança, E[Z] = E[Z_1] + E[Z_2] + … + E[Z_k] = (7/8)*k. Això implica que, com a mínim, hi ha una interpretació que satisfà almenys (7/8) de les clàusules (sempre hi ha d'haver un valor >= a l'esperança). Hem demostrat que aquesta interpretació sempre existirà.

Tot i així, aquest algorisme no sempre retornarà una interpretació amb (7/8) de les clàusules satisfetes. L'algorisme que presentarem i que serà la solució al problema plantejat generarà interpretacions completament a l'atzar (com l'algorisme anterior), mentre no haguem satisfet almenys (7/8)*k clàusules.

> *Waiting for success*
> Si tenim una moneda que, al llançar-la, cau "HEADS" amb probabilitat *p*, llavors el número esperat de llançaments a efectuar serà E[X] = SUMinf j * Pr[X = j] = 1 / p.

Sigui *p_j* la probabilitat que exactament *j* clàusules siguin satisfetes. Per definició, el nombre esperat de clàusules satisfetes és E[Z] = SUM {j=0→ k} [j * p_j] = (7/8)*k. Quina és la probabilitat que se satisfacin >= (7/8)*k clàusules ? Això és *p* = SUM { j >=(7k/8)} *p_j*.

$$\frac{7}{8}k = \sum_{j=0}^{k} jp_j = \sum_{j<7k/8} jp_j + \sum_{j\geq 7k/8} jp_j.$$

Sigui *k'* = *floor* [(7/8)*k – 1]. La RHS de l'eqüació només incrementa si el modifiquem emprant *k* i *k'* en comptes de *j*. A més, SUM { j < (7k/8) } *p_j* = 1 – *p*.

$$\frac{7}{8}k \leq \sum_{j<7k/8} k'p_j + \sum_{j\geq 7k/8} kp_j = k'(1-p) + kp \leq k' + kp,$$

Tenim que (7/8)k – k' <= *kp*. A més, (7/8)k – k' >= (1/8) donat que *k'* és un natural estrictament menor que (7/8)*k. Ergo *kp* >= 1/8 => *p* >= *1 / (8k)*. Per la propietat de "Waiting for success", podem veure que el nombre esperat de generacions per a trobar l'assignació amb almenys 7/8 de clàusules satisfetes serà 8k.

Existeix un algorisme amb temps esperat polinòmic que garanteix produir una assignació que satisfà com a mínim 7/8 de totes les clàusules.

EXERCICI 5 - CONTRACTION ALGORITHM

Presenteu *the Contraction Algorithm* conegut també per *Karger's Algorithm* i analitzeu-ne el temps de computació i la probabilitat d'error.

L'algorisme treballa amb *multigrafs* (possibilitat de repetició d'arestes). Comença escollint una aresta *(u, v)* qualsevol i la "contrau" unificant els vèrtexs *u, v* en un de sol *w.* Totes les arestes que tenien *u* i *v* com a vèrtexs adjacents són destruides. Totes les altres arestes es mantenen, actualitzant els vèrtexs *u,v* per el nou vèrtex *w*. Fàcilment podem veure que encara que el graf inicial no sigui un *multigraf*, podem arribar-ne a un mitjançant una contracció.

L'algorisme de *Karger* recursivament es cridarà amb el nou graf obtingut. A mesura que l'algorisme avança, podem interpretar els vèrtexs com a *Hyper-vèrtexs,* constituits per la contracció d'un conjunt de vèrtexs originals. L'algorisme termina quan s'arriba a un multigraf amb només dos Hyper-vèrtexs. Els dos Hyper-vèrtexs (conjunts de nodes no buits i disjunts) són el *cut* que retornem.

Cada contracció es pot efectuar en un temps polinòmic respecte el graf (només unificar dos nodes i modificar les arestes afectades), en concret *O(m)*. En total, s'efectuen un total de *n – 2* contraccions. Ergo, l'algorisme és polinòmic respecte l'entrada, amb *O(nm)* on *n = #V, m = #E.*

The Contraction Algorithm retutns a global min-cut of G with probability at least 1 / (n choose 2).

Sigui (A, B) un *min-cut global* de G de cardinalitat *k,* i sigui C el cut-set de (A, B). Si l'algorisme d'en *Karger* escull per desgràcia una aresta de *C,* llavors dos nodes de *A* i *B* seran unificats i no podrem obtenir el *min-cut global òptim.* Si no es contrau una aresta de C, llavors encara existeix la probabilitat d'èxit.

Veiem que tot node de G ha de tenir grau >k (si un no el tingés el *cut-set* (V, V – {u}) seria més òptim). Ergo *#E >= 0.5*k*n*. La probabilitat que una aresta de *C* sigui contraguda és, com a màxim, *k / (0.5*k*n) = 2 / n.* Si considerem l'algorisme després de *j* iteracions on no s'ha contragut cap aresta de *C* encara. El # de Hyper-vèrtexs és *n – j*, el nombre d'arestes és com a mínim *0.5*k*(n – j).* La probabilitat d'escollir una aresta de *C* en la iteració *j+1* és 2 / (n – j).

La probabilitat d'error final serà que falli en alguna iteració. Més fàcilment, la probabilitat d'encert és que no falli en cap iteració, per tant SUM { j = 0 → (n-3) } [1 – 2/ (n – j)].

$$
\begin{aligned}
&\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{n-j}\right)\cdots\left(1-\frac{2}{3}\right)\\
&= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\cdots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)\\
&= \frac{2}{n(n-1)} = \binom{n}{2}^{-1}. \quad \blacksquare
\end{aligned}
$$

EXERCICI 6 – RSA

(a) Demostreu que si P = NP, aleshores el sistema RSA és fàcilment vulnerable.

Si P = NP, això implica que FACTORIZE (Donat un enter *x* trobar la seva factorització en nombres primers) és computable en temps polinòmic, doncs FACTOR (Donats x i y, té x un factor no trivial < y?) és a NP; pertant a P; i podem factoritzar en temps polinòmic qualsevol enter emprant un oracle de FACTOR (cerca binària per a trobar un factor).

Aleshores, donat un missatge encriptat *c* amb RSA i donada la clau pública (*N*, *e*), podem desxifrar en temps polinòmic sense conèixer la clau secreta *d* factoritzant *N* i aplicant el mateix procés per a trobar la clau secreta *d*. Primer factoritzem *N* en els seus dos factors primers (polinòmic si P = NP) i calculem phi(N) = (p – 1) * (q – 1). Aleshores per a trobar la *d* tal que *e*d = 1 (mod phi(N))* executem l'algorisme EXTENDED-EUCLID amb entrada (*e*, N), que retornarà en el segon element de la tupla *d* (invers multiplicatiu).

Desxifrem computant *c**d mod N = plaintext.*

(b) Si RSA és fàcilment vulnerable llavors P = NP?

No necessàriament, donat que no està demostrat que el problema de l'RSA (donat el missatge encriptat i la clau pública trobar el missatge original) sigui tant difícil com factoritzar. Si considerem el problema de trobar la clau privada *d* llavors sí que seria igual de difícil que factoritzar *[Coron and May : Deterministic Polynomial Time Equivalence of Computing the RSA Secret Key and Factoring]*

EXERCICI 7 – ESPIANT RSA

Si Eve aconsegueix la clau privada *d* i la pública (*N, e = 3)*, com pot calcular Eve eficientment *p, q?*

Sabent *e = 3* i *d*, tenim que *3 * d = 1 (mod phi(N))*. És a dir, que existeix una *k > 0* tal que 3 * d = *k * phi(N) + 1*. Això és equivalent a dir 3 * d = k * (p – 1) * (q – 1) + 1. Donat que d < phi(N) (recordem que podem tenir *0<=d<phi(N)* en temps polonòtcc) tenim que *k* pot prendre un valor de com a màxim 2. Podem provar per els dos casos (k = 1 i k = 2) i aïllar (p – 1) * (q – 1) = phi(N).

També podem calcular fàcilment *p + q* sabent que phi(N) = (p – 1) * (q – 1) = pq – (p + q) + 1 = N – (p + q) + 1 → p + q = N – phi(N) + 1. Si aquesta suma és negativa, això és indicatiu que no hem escollit correctament la *k*.

Una vegada coneixem (p – 1) * (q – 1) i la seva suma p + q, podem trobar *p* i *q* calculant els zeros de la eqüació quadràtica $X^2 – (p + q)*X + pq$. Podem veure fàcilment p.ex. que $p^2 – (p+q)*p + pq = 0$. Trobar els zeros d'aquesta funció quadràtica és polinòmic i molt eficient.

EXERCICI 8 – NO INCENTIU DE CANVI (NCG)

Considerem el problema “EstaBé”, on donat un joc de creació de xarxes, un vector d’estratègies *s* i un jugador *u* , decidir si no hi ha cap estratègia *s’* perc al jugador *u* tal que C(*s’*) < C(s).

Demostreu que el problema “EstaBé” és **co-NP-complet.**

Primerament veurem que “EstáBé” pertany a **co-NP:**

El complementari “NoEstàBé” : decidir si, donat un NCG, vector d’estratègies i jugador *u*, existeix alguna estratègia amb millor cost per a *u*. Aquest problema és equivalent a decidir si, donat un NCG, vector d’estratègies, jugador *u* i cost màxim *K*, existeix una estratègia amb cost < *K*.

Aquest problema pertany a la classe **NP**, doncs el testimoni és una estratègia *s’* (representada com un subconjunt d’enllaços que ha comprat *u*, polinòmic respecte el joc [gamma]) i el verificador computa *α*|s’| + SUM [shortests paths from u].* Podem obtenir la suma de totes les distàncies de *u* als altres vèrtexs mitjançant Dijkstra (linial).

Seguidament hem de veure que “EstàBé” és **co-NP-complet** veient que “NoEstàBé” és **NP-complet**. Per a fer-ho reduirem DOMINATING-SET a NO-ESTA-BE.

Volem transformar entrades (G, *k*) (existeix un subconjunt de vèrtexs dominant amb cardinalitat màxima *k ?*) a entrades per a “NoEstàBé” (donat el cost d’una estratègia *r* , existeix una estratègia *s’* amb cost < *r* ?).

Primerament, definim el joc com a J = ({1 … n}, α) on |V(G)| = *n* i un 1 < α < 2 qualsevol. El cost a millorar *r* serà *r = α*k +2*n – k.* Notem que totes aquestes construccions tenen temps polinòmic.

Si existeix un DS (anomenarem D) de mida màxima *k*, llavors l’estratègia on el jugador *u* compra tots els enllaços cap a tots els vèrtexs de D té cost < *r*. Efectivament, el cost és com a màxim α*#D (enllaços) + 2*n (distància a tots els nodes) – k (restem 1 u. de distància als membres de D) <= *r*. Existeix aquesta estratègia.

Si existeix una estratègia *s’* amb cost <= α*k + 2n – k, llavors cerquem tots els nodes *v* que estiguin a distància > 2 de *u*. Veiem que podem afegir una aresta (*u, v)* a la nostra estratègia. Donat que α < 2, això sempre decrementarà el cost total i per tant no ens sortim del límit. Això ho farem per a tots els nodes a distància > 2 iterativament.

Sigui *D* el conjunt de nodes a distància 1 de *u*, aquests nodes conformen un conjunt dominant. I donat que *cost(s) = α |U| +2n - |U| <= α*k +2n -k*, tenim que #U <= k. Ergo, si existeix alguna estratègia amb cost < *r* aquest nou graf que hem construit conté aquest conjunt dominant de mida <= *k*.