**NP-P**

NP- hard

P

NP completi

NP

**Definire le classi: P, NP e indicare le relazioni di contenimento fra le classi**La classe **P** è la classe dei linguaggi L che data una stringa x possiamo controllare la sua appartenenza a L, mediante una MdT *deterministica* **in tempo polinomiale;**La classe **NP** è la classe dei linguaggi L che data una stringa x possiamo controllare la sua appartenenza a L, mediante una MdT *non deterministica* **in tempo polinomiale;  
Sappiamo che P è contenuto in NP;**  
**Dati due problemi P e Q, come posso dimostrare che P è non più difficile di Q?**Per stabilire che P è più difficile di Q devo trovare una riduzione del tipo P <= p Q  
Bisogna mappare ogni istanza di P in Q e deve accadere che la soluzione di Q per quell’istanza coincida con la soluzione di P;  
**Definire il problema decisionale PL-{0,1} e mostrare che esso è NP-hard**PL-{0,1}  Programmazione lineare a variabili 0 1  
x1, …. , xn variabili  
Sistemi di disequazioni linari con le x1, … , xn variabili  
Per mostrare che è NP hard SAT <= p PL-{0,1}  
Prendo un’istanza generica di Sat e la trasformo in un’istanza generica di PL-{0,1}   
es: x1 v \_x2 v x3 v \_x4  x1 + (1-x2) + x3 + (1-x4) > 0  
*la clausola è soddisfatta se e solo se è soddisfatta questa disequazione  
Per il teorema di Levin-Cook*  
**Cosa scopriremo sul problema P vs NP, se si dimostrasse che:  
*Esiste un lower bound esponenziale per un problema in NP di cui non si sa se è NP-completo*** P != NP  
***Un problema NP-completo ammette upper bound polinomiale*** P = NP  
***Esiste un problema in NP (non noto essere NP completo) che ammette upper bound polinomiale*** nulla  
***Esiste un NP-hard che è riducibile a un problema in P*** P = NP  
-Formula vuota: **formula soddisfatta**  
**Definire la classe dei problemi NP-completi e indicare le relazioni di contenimento rispetto alle classi P e NP**La classe dei problemi NP-completi è rappresentata dai linguaggi alfa tali che:  
-alfa £ NP  
-per ogni alfa’ £ NP vale che alfa’ <=p alfa  
**Dati due problemi P e Q, come posso dimostrare che P è almeno tanto difficile quanto Q**Bisogna dimostrare che Q <=p P ovvero bisogna mappare le istanze di Q su istanze di P la soluzione di ogni istanza in P deve essere soluzione dell’istanza in Q  
**Definire la classe NP senza impiegare il concetto di non determinismo**La classe dei linguaggi NP è l’insieme di tutti i linguaggi L per cui esiste una funzione c(x) detta certificato e un polinomio P per cui se x appartiene a L allora |c(x)| <= p(|x|) (lunghezza certificato polinomiale nella lunghezza del modulo di x)  
Esiste una Mdt deterministica che con ingresso x e c(x) certifica che x appartiene ad L e ha complessità temporale polinomiale in |x| e |c(x)|  
**Confrontare le due definizioni di NP, giustificandone l'equivalenza**Sono equivalenti perché le decisioni non deterministiche della MDTND corrispondono al certificato che informa la MdtDet che verifica la soluzione.  
**Se per un problema della classe NP viene individuato un lower bound polinomiale, cosa**   
**possiamo dedurre in merito alla questione P vs NP?** Non possiamo concludere un cazzo  
**Se per un problema della classe NP viene individuato un lower bound esponenziale, cosa**   
**possiamo dedurre in merito alla questione P vs NP** P = NP  
**Definire la classe dei problemi NP-completi e indicare le relazioni di contenimento rispetto alle classi menzionate al punto**Un problema è NP completo se appartiene a NP e se per ogni A’ che appartiene a NP A’ è <=p del problema  
Per dimostrare la NP completezza dimostriamo che il problema è NP con un opportuno certificato e poi dimostriamo che è NP-hard.  
**Dimostrare un possibile risultato (a piacere) che permetterebbe di stabilire che P != NP**Uso una Mdt deterministica che usa spazio polinomiale  P != da NP perché se così non fosse avremmo anche per NP tempo polinomiale, invece occupa tempo esponeziale.  
**GRAMMATICHE E LINGUAGGI**

**Illustrare la gerarchia di Chomsky delle grammatiche; indicare per ciascuna categoria la proprietà o caratteristica a vostro giudizio maggiormente rilevante.**Tipo unrestricted  senza restrizioni  Il tipo *unrestricted è riconosciuto da una Mdt;*  
Tipo context sensitive  se abbiamo *alfa  beta* con |alfa| < |beta|  Il tipo context *sensitive è riconosciuto da una Mdt linear bounded;*  
Tipo context free  se |alfa| = 1  
X  *alfa* con X £ Vn, *alfa* £ V\* e V = (Vt U Vn)  
Il tipo context *free* da un automa a pila  
Tipo regolare  se abbiamo produzioni del tipo *A  aB B  b con A, B £ Vn e a,b £ Vt*L’epsilon produzione non altera il linguaggio:  
*se esiste un epsilon produzion sull’assioma allora eps £ Linguaggio  
se abbiamo un eps-prod su un non terminale diverso dall’assioma la possiamo eliminare avendo cura di visitare tutte le produzioni in cui questo non terminale compare nelle parti destre e aggiungendo un’altra produzione che ne tiene conto*  
Il tipo *regular* da un ASF che può essere deterministico o non:  
L(ASFd) = { x £ SIGMA\* | delta(q0, x) £ F}  
L(ASFnd) = { x £ SIGMA\* | deltan(q0, x) int F != 0}  
Per tutti i tipi **tranne per il due (context free)** gli automi NON DETERMINISTICI hanno lo **stesso potere computazionale di quelli** DETERMINISTICI.  
**Definire la grammatica LL(1)**Una grammatica è LL(1) se per ogni simbolo NON terminale A, e per ogni coppia di produzioni A  alfa e A  B vale che FIRST(alfa) int FIRST(B) = 0;  
**Dimostrare che G: S  SS| aSb | ab non è LL(1) E modificala tale che lo sia**First(aSb) = a, First(ab) = a; != 0  non è LL(1)  
Per modificarla bisogna lavorare sui prefissi comuni e eliminare eventuali ricorsioni sinistre  
**Definire il concetto di grammatica ambigua fornendo e discutendo un esempio. Discutere brevemente la necessità di avere grammatiche non ambigue per definire linguaggi di programmazione.**Una grammatica si dice ambigua se data x £ L(G) possiamo generare due o più parse tree differenti.  
ESEMPIO la grammatica per calcolare la somma di espressioni  
E  E + E | E \* E | x | y | z  
Nei linguaggi di programmazione avere una grammatica non ambigua è essenziale, perché dato un programma il parse tree deve essere unico così che il compilatore non avrà dubbio su quale istruzione sostituire.  
**Definire i linguaggi di tipo 2, perché vengono chiamati context free**Sono i linguaggi generati da una grammatica di tipo 2 con produzioni del tipo X  alfa con X £ Vn, alfa £ V\*  
Le grammatiche di Chomsky si dividono in sensibili al contesto –> aXb  awb  
Libere dal contesto X  w indipendentemente dalla presenza (contesto) di a e b  
**!Definire!formalmente!i!concetti!di   
• Linguaggio!di!tipo!0,!di!tipo!2,!di!tipo!3.   
• Fornire!inoltre!un!esempio!di!linguaggio!di!tipo!3**Tipo 3 sono linguaggi regolari, caratterizzati da una pentupla (SIGMA, k, F, qo stato iniziale, delta f di trans}  
delta ha il compito di cambiare lo stato determina quindi se l’ASF che riconosce le gramm di tipo 3 è det o no;  
Delta D:  
K X SIGMA  K  
Delta ND:  
KXSIGMA  2^(K)  
{alfa £ Vn and |alfa| = 1, beta £ Vt (|beta| = 1) or beta £ Vt°Vn (|beta| = 2)}  
Tipo 2  context free poiché libere dal contesto  
abbiamo |alfa| = 1 ^ |alfa| £ Vn;  
Riconosciuto da automa a pila  
Tipo zero  unrestricted   
Macchina di touring;  
**Definire il concetto di espressione regolare**E’ un’espressione atomica a, b, c, eps, LAMBDA, con a,b,c £ SIGMA  
se e1 e e2 sono E.R allora e1\*e2 è E.R e vale L(e1)\*L(e2)  
se e1 e e2 sono E.R allora e1+e2 è E.R e vale L(e1)+L(e2)  
se e1 e e2 sono E.R allora e1\* è E.R e vale L(e1)\*  
**spiegare perché è interessante studiare la decidibilità dei problemi di riconoscimento dei linguaggi e perché è interessante farlo su un modello di calcolo elementare come la macchina di Touring.**La riconoscibilità dei linguaggi ci offre una maniera standard per modellare i problemi di decisione, usiamo la Mdt perché è una macchina semplice e grazie alla tesi di Church-Touring ha validità generale, possiamo fare ciò che faremmo su un modello di calcolo più complicato.  
input  ALGORITMO  si|no il linguaggio in questione è il linguaggio delle stringhe x tale che la risposta sia Si.   
**MACCHINE DI TOURING  
Si descriva il funzionamento di una macchina di Turing non deterministica. Sotto quali aspetti tale macchina è "superiore" a quella deterministica?**La MdTNd è definita come la MdTd cambia solo la funzione di transizione che invece che dipendere da una sola variabile dipende da più variabili;  
Può andare in più stati contemporaneamente, leggere diversi caratteri e spostarsi a sx, dx o non spostarsi.  
E’ superiore in termini di costo temporale, può eseguire algoritmi esponenziali in tempo polinomiale. In termini di capacità di calcolo e di spazio ha le stesse prestazioni della MdtDet (come gli ASF)  
**Si descriva il funzionamento di una macchina di Turing a due nastri. Sotto quali aspetti tale macchina è "superiore" a quella a nastro singolo?**Una macchina di Touring a due nastri è formata appunto da due nastri, un’unità di controllo. I due nastri hanno una testina di lettura e scrittura ciascuno con la quale leggono e scrivono sul nastro. Le testine si possono spostare a destra, e a sinistra di un passo alla volta oppure possono rimanere immobili.  
In potenza di calcolo e di spazio la MdT a due nastri è equivalente ala MdT a un nastro, ciò che cambia è il costo Temporale. Per simulare una MdT multinastro in una mononastro bisogna usare un algoritmo che ha costo polinomiale e un alfabeto che ha costo esponenziale.  
**Quali sono gli ingressi e il risultato calcolato dalla macchina di Turing universale? Perché il risultato è considerato importante?**La macchina di Touring universale prende la descrizione di una macchina generica M e un input x che è l’input da dare a M e fornisce in output l’output di M su x;  
E’ importante perché emula il comportamento di un computer, compilatore/interprete, dove cM è il programma e x l’input da dare al programma  
**Enunciare e discutere la tesi di Church-Turing, chiarendo perché viene formulata come tesi e non come teorema.**Qualunque algoritmo in qualuque modello di calcolo può essere realizzato ediante un algoritmo sulla macchina di touring, questa tesi ci dice che se dimostriamo qualcosa sulla Mdt allora vale sempre in tutti gli altri modelli.  
**Cosa succede quando ho una Mdt con un seminastro**strategia per simulare comunque il comportamento della macchina  
Mdt normale  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 0 1 2 3 4 5  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Mdt seminastro \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 0 -1 1 -2 2 ….  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**ALTRO  
Teorema di Savitch**PSPACE == NPSPACE **Definire la nozione di token e descrivere i processi di analisi lessicale e sintattica spiegando quali sono gli input su cui operano e quali sono gli output attesi  
il sigma tipicamente è diverso dall’alfabeto umano è composti da cose più complicate e l’analisi lessicale serve a questo**

Input Analisi Ouutput  
  
token ANALISI SINTATTICA albero di derivazione

caratteri ANALISI LESSICALE token **Illustrare la notazione O(.) – O grande - e fornire la definizione di problema computazionalmente "facile".**Si dice che f appartiene a O(g) se f è al più g  ovvero f(x) <= g(x) **Nell’analisi di complessità e, in particolare nella notazione O(), si ignorano le costanti moltiplicative e additive. Discutere questa scelta indicandone i vantaggi e gli svantaggi.**Vantaggi  permette di semplificare il calcolo poiché essendo un tipo di notazione asintotica possiamo trascurare l’uso delle costanti  risultato in maniera più semplici  
Svantaggio  ne risente di precisione proprio perché è un calcolo approssimato **Cosa si intende per problema non decidibile (o non calcolabile)? Fornire un esempio di problema non decidibile.**Non esiste un algoritmo che risponda SI quando bisogna rispondere si e NO quando bisogna rispondere NO  
Il problema ha soluzione solo che non si trova un algoritmo che la determini  
Halting problem, problema della fermata.

**DPLL**  
**Illustrare l'algoritmo DPLL, spiegando quale problema risolve.**Il DPLL è un algoritmo che risolve un problema SAT nell forma CNF cioè una formula così fatta F = and (e1 or e2 or … e3)  
L’algotirmo usa due strategie per semplificare la formula:  
-Unit-propagation: Prende in esame una alla volta le clausole unitarie, viene soddisfatta quindi posso eliminarla se appare affermata, se apparte negata posso eliminare solo il singolo termine;  
-Pure literal elimination: un letterale che appare *sempre* o positivo o negativo viene chiamato puro, e può essere quindi eliminato.  
Se l’algortimo non può usare nessuna di queste due semplificazioni torna indietro e ricomincia da capo **backtracking o bruteforce.**Se durante l’algoritmo ci ritroviamo una clausola vuota o una formula vuota:  
-Clausola vuota: condizione di arresto se l’algorirmo ha provato tutte le scelte **formula insoddisfatta  
Problema del vertex cover**Dato G = (V, E) ci chiediamo se esiste k con |V’| <= k che copre tutti gli archi?  
3SAT <=p VC  la riduzione si applica mettendo un arco per ogni letterale e un triangolo (3SAT) per ogni clausola. K = 2\*m + n dove m = numero di traingoli (clausole), numero di archi (letterali)  
**Cosa è un problema SAT**Un problema sat è un problema di soddisfacibilità dove data la formula del calcolo proposizionale si vuole sapere se esiste un’assegnazione di valori di verità tale da rendere la formula vera.  
CNF: forma normale congiuntiva [ ( … or …) and ( … ) ]  
DNF: forma normale disgiuntiva [ ( … and …) or ( …. ) ]