ALGORITMI - Complessità **Anno Accademico 2014-15**

Appello 2 - Luglio 2015

Nome e Cognome: Matricola:

Importante: Usare solo i fogli forniti dal docente, riportando Nome Cognome e Matricola su ogni foglio. Scrivere in modo **leggibile** e **giustificare** ogni affermazione. Risposte non giustificate saranno valutate **0**. Per ogni algoritmo proposto va provata la correttezza e se necessario al problema, la complessità.
Si chiede di risolvere 3 esercizi:
(i) uno a scelta tra 1 e 2, (ii) uno a scelta tra 3 e 4, (iii) uno a scelta tra 5 e 6.

Esercizio 1 - 15 punti

Definiamo il seguente problema

HalfCycle (HalfC)
Input: Un grafo diretto $G = (V, E)$
Output: *yes* se e solo il ciclo piú lungo in G è di lunghezza esattamente $\lfloor |V|/2 \rfloor$.

Si dimostri che
(i) HalfC è $\mathcal{NP}$-hard
(ii) HalfC è co-$\mathcal{NP}$-hard
(iii) se $\mathcal{PH} \neq$ co$-\mathcal{NP}$ allora HalfC non è $\mathcal{NP}$-completo.

Esercizio 2 - 15 punti

Dati due problemi $\mathbb{A}$ e $\mathbb{B}$ definiti sullo stesso insieme di istanze, denotiamo con $\mathbb{A} \cap \mathbb{B}$ il problema le cui istanze *yes* sono tutte e solo le istanze che risultano essere contemporaneamente *yes* per $\mathbb{A}$ e $\mathbb{B}$.

Denotiamo con $\mathbb{A} \setminus \mathbb{B}$ il problema le cui istanze *yes* sono tutte e solo le istanze che risultano essere contemporaneamente *yes* per $\mathbb{A}$ e *no* per $\mathbb{B}$.

Definiamo le classi di problemi

$$\mathcal{DP}_0 = \{\mathbb{C} = \mathbb{A} \cap \mathbb{B} \mid \mathbb{A} \in \mathcal{NP}, \mathbb{B} \in \text{co}-\mathcal{NP}\}$$

$$\mathcal{DP}_1 = \{\mathbb{C} = \mathbb{A} \setminus \mathbb{B} \mid \mathbb{A}, \mathbb{B} \in \mathcal{NP}\}$$

Con riferimento alla definizione del problema HalfC data nell'esercizio precedente, dimostrare che
(i) $\mathcal{DP}_0 = \mathcal{DP}_1$
(ii) HalfC $\in \mathcal{DP}_1$
(iii) $\mathcal{DP}_1 = \mathcal{NP} \cup$ co$-\mathcal{NP}$ se e solo se $\mathcal{NP} =$ co$-\mathcal{NP}$.

Nota: Per la (iii) si possono usare come assunzioni (i) e (ii) dell'esercizio precedente.

Esercizio 3 - 15 punti

Definiamo il seguente problema

Facility Location Problem (FLP)
Input: Un grafo $G = (V, E)$ e per ogni vertice v un valore b_v, ed un valore di soglia B.
Due giocatori $P1$ e $P2$ si alternano nello scegliere un nuovo nodo di V rispettando la condizione che i nodi scelti formino sempre un insieme indipendente.
$P2$ vince se alla fine i nodi da lui selezionati hanno valore totale almeno pari a B
Output: *yes* se e solo $P2$ puó vincere.

Dimostrare che
(i) FLP $\in \mathcal{PSPACE}$.
(ii) se per ogni problema $\mathbb{A} \in \mathcal{PSPACE}$ risulta che $\mathbb{A} \leq_L$ allora FLP $\notin \mathcal{NL}$. Dove per $\leq_L$ intendiamo una riduzione log-space.

Bonus 10 punti. FLP è $\mathcal{PSPACE}$-completo

Esercizio 4 - 15 punti

Si fornisca la definizione esatta di riducibilità log-space tra problemi e se ne dimostri dettagliatamente la proprietà di transitività .

Esercizio 5 - 20 punti

Fornire la seguente L-riduzione

Max-3-Sat $\leq_L$ Max-2-Sat

Considerato quanto visto a lezione per Max-3-Sat, e la L-riduzione fornita, cosa possiamo concludere riguardo all'inapprossimabilità di Max-2-Sat? (Fornire una costante α tale che Max-3-Sat non è α-approssimabile se $\mathcal{P} \neq \mathcal{NP}$)

Nota: Si può assumere che nella formula Max-3-Sat ogni clausola abbia esattamente 3 letterali e che per Max-2-Sat ogni clausola possa avere uno o due letterali.

Esercizio 6 - 20 punti

Si consideri la seguente variante del problema dello Scheduling a minimo makespan visto a lezione

Scheduling with Different Machines (Sc2M)
Input: Un insieme di n job, m processori lenti e k processori veloci.
ogni job j, impiega tempo t_j se eseguito su un processore lento e tempo $\frac{1}{2}t_j$ su un processore veloce.
Output: Assegnamento dei job ai processori in modo da minimizzare il makespan, cioè il massimo tempo di esecuzione tra tutti i processori.

Si dimostri che Sc2M $\in \mathcal{APX}$

Suggerimento: Si può considerare l'assegnamento *greedy* usato dall'algoritmo visto a lezione e analizzarlo in maniera analoga per valutare l'approssimazione garantita dall'algoritmo che si ottiene.

ALGORITMI - Complessità **Anno Accademico 2014-15**

Appello 1 - Giugno 2015

Nome e Cognome: Matricola:

Importante: Usare solo i fogli forniti dal docente, riportando Nome Cognome e Matricola su ogni foglio. Scrivere in modo **leggibile** e **giustificare** ogni affermazione. Risposte non giustificate saranno valutate **0**. Per ogni algoritmo proposto va provata la correttezza e se necessario al problema, la complessità.
Si chiede di risolvere 3 esercizi:
(i) uno a scelta tra 1 e 2, (ii) uno a scelta tra 3 e 4, (iii) uno a scelta tra 5 e 6.

Esercizio 1 - 15 punti

Sia $\mathcal{F} = \{A_1, \ldots, A_m\}$ una famiglia 3-regolare di sottinsiemi di $V = \{v_1, \ldots, v_n\}$: per ogni $i = 1, \ldots, m$, vale che $A_i \subseteq V$ ed $|A_i| = 3$ (ha esattamente tre elementi).

Diciamo che $\mathcal{F}$ è *2-colorabile* se possiamo colorare gli elementi di V di bianco o di nero in modo tale che ognuno degli insiemi di $\mathcal{F}$ contenga sia un elemento nero che un elemento bianco.

Definiamo il seguente problema

HYPERGRAPH 2-COLOURING (HG-2COL)
Input: Un insieme $V = \{v_1, \ldots, v_n\}$ e una famiglia 3-regolare $\mathcal{F} = \{A_1, \ldots, A_m\}$ di sottinsiemi di V
Output: *yes* se e solo $\mathcal{F}$ è 2-colorabile.

Mostrare che HG-2COL è $\mathcal{NP}$-completo.
Suggerimento: Una possibile riduzione è da NAE-3-SAT. Fare attenzione a come vengono correlati i valori di verità delle variabili della formula con i colori dei corrispondenti elementi dell'insieme V nell'istanza di HG-2COL.

Esercizio 2 - 15 punti (+ bonus 5 punti)

Diciamo che un grafo $G = (V, E)$ è ricopribile con k clique se esistono k insiemi di vertici $A_1, \ldots, A_k$ tali che: (i) ogni A_i *induce* una clique, cioè i vertici in A_i sono a due a due adiacenti; (ii) gli insiemi $A_1, \ldots, A_k$ partizionano i vertici di V, cioè $A_1 \cup A_2 \cup \cdots \cup A_k = V$ e $A_i \cap A_j = \emptyset$, per ogni $1 \leq i < j \leq k$.

Si dimostri che il seguente problema è $\mathcal{NP}$-completo:

CLIQUE COVER
Input: Un grafo $G = (V, E)$ ed un intero k
Output: *yes* se e solo G è ricopribile con $\leq k$ clique

Sugg.: Una possibile riduzione è da GRAPH-K-COLOURING. In tal caso può essere utile osservare che in una *colorazione propria* i vertici dello stesso colore costituiscono ....
Bonus 5 punti. Si provi che ammettendo che gli insiemi $A_1, \ldots, A_k$ possano avere intersezione non-nulla, il problema rimane lo stesso. Ovvero, **provare che** *se esiste una* Clique Cover *in cui per qualche* A_i, A_j *vale* $A_i \cap A_j \neq \emptyset$ *allora esiste una* Clique Cover della stessa taglia *senza intersezioni non-nulle.*

Esercizio 3 - 15 punti

Si considerino le seguenti classi di problemi

$$\mathbf{SPACE}(\text{polylog}(n)) = \cup_{c>0}\mathbf{SPACE}(\log^c n)$$
$$\mathbf{NSPACE}(\text{polylog}(n)) = \cup_{c>0}\mathbf{NSPACE}(\log^c n).$$

Siano dati due problemi $\mathbb{A} \in \mathbf{SPACE}(\text{polylog}(n))$ e $\mathbb{B} \in \mathbf{NSPACE}(\text{polylog}(n))$. Cosa possiamo affermare circa il tempo necessario a risolvere un'istanza di tali problemi? E quale è la relazione tra le due classi di problemi? Precisamente, **si chiede di rispondere ai seguenti 4 punti**—Tutte le risposte vanno motivate/dimostrate!

(i) Si fornisca una funzione $f(n)$ tale che $\mathbb{A} \in \mathbf{TIME}(f(n))$?
(ii) Si fornisca una funzione $g(n)$ tale che $\mathbb{B} \in \mathbf{TIME}(g(n))$?
(iii) Possiamo affermare che $\mathbf{NSPACE}(\text{polylog}(n)) \subseteq \mathbf{SPACE}(\text{polylog}(n))$?
(iv) Possiamo affermare che $\mathbf{SPACE}(\text{polylog}(n)) \subseteq \mathbf{NSPACE}(\text{polylog}(n))$?

Esercizio 4 - 15 punti

(i) Si fornisca l'**enunciato** del teorema di Savitch
(ii) Sia $\mathbf{NSPACETIME}(f(n), g(n))$ la classe di problemi di decisione che possono essere risolti nondeterministicamente da un algoritmo che usa spazio di memoria $O(f(n))$ ed impiega tempo $O(g(n))$, dove n denota la taglia dell'input.
Si dimostri che vale la seguente inclusione: $\mathbf{NSPACETIME}(n, n) \subseteq \mathbf{SPACE}(n \log n)$
Sugg.: può essere utile ricordare il modo in cui abbiamo dimostrato il teorema di Savitch.

Esercizio 5 - 20 punti

Si definisca MAX-3$^{\leq}$SAT la variante di MAX-3SAT in cui ogni clausola può avere ≤ 3 (quindi anche meno di 3) e MAX-3$_{\neq}$SAT la variante di MAX-3SAT in cui ogni clausola ha esattamente 3 letterali ma non possiamo avere la stessa variabile più di una volta nella stessa clausola.
(i) Si fornisca una L-riduzione da MAX-3$^{\leq}$SAT a MAX-3$_{\neq}$SAT.
(ii) Sulla base del punto (i) e dei risultati di inapprossimabilità forniti a lezione—che, rispetto alla notazione di questo esercizio, sono stati dati per MAX-3$_{\neq}$SAT—si dimostri il seguente lemma:
Lemma 1. Se $\mathcal{NP} \neq \mathcal{P}$ allora MAX-3$^{\leq}$SAT $\notin \mathcal{PTAS}$.

Sugg.: Per la riduzione si trasformino le clausole sottodimensionate aggiungendo variabili nuove.

Esercizio 6 - 20 punti

Si consideri il seguente problema di massimizzazione: Sono dati tre insiemi A, B, C della stessa dimensione $|A| = |B| = |C| = n$ e con mutua intersezione vuota $A \cap B = A \cap C = B \cap C = \emptyset$ ed un insieme di m triple $\mathcal{T} = \{t_i = (a_i, b_i, c_i) \mid i = 1, \ldots m\}$, tali che per ogni i abbiamo $a_i \in A$, $b_i \in B$, $c_i \in C$.
Lo scopo è selezionare il massimo numero di triple tali che nessuna coppia di triple contiene uno stesso elemento.
(i) Si fornisca un algoritmo che garantisce una 3-approssimazione.
(ii) Il problema di decidere se esiste una scelta delle triple che copre tutti gli elementi, noto come 3-DIMENSIONAL MATCHING è $\mathcal{NP}$-completo. Usando questo risultato e quanto studiato al corso si dimostri che non può esistere un $FPTAS$ per il problema di massimizzazione se $\mathcal{NP} \neq \mathcal{P}$.

Esercizio 7 - Bonus - 7 punti

Si dimostri la seguente implicazione $\mathcal{BPP} \subseteq \mathcal{RP} \Rightarrow \mathcal{BPP} \subseteq \mathcal{ZPP}$

ALGORITMI - Complessità **Anno Accademico 2015-16**

Prova in Itinere 1 - Midterm 1

Nome e Cognome: Matricola:

Importante: Usare solo i fogli forniti dal docente, riportando Nome Cognome e Matricola su ogni foglio. Scrivere in modo **leggibile** e **giustificare** ogni affermazione. Risposte non giustificate saranno valutate **0**

Si chiede di risolvere 4 esercizi:
(i) <u>entrambi</u> 1 e 2, (ii) <u>uno a scelta</u> tra 3 e 4, (iii) <u>uno a scelta</u> tra 5 e 6.

Esercizio 1 - 5 punti

Si consideri il seguente problema:

> CIRCUIT-SAT-20 (C-SAT-20)
> **Input**: Un circuito Booleano C con 20 input
> **Output**: *yes* se e solo se esiste un input per il quale l'output del circuito è 1.

(i) Si dimostri che C-SAT-20 è in $\mathcal{P}$.
(ii) Sia $\mathcal{NPC}$ l'insieme dei problemi $\mathcal{NP}$-completi. Siano $\mathbb{A}, \mathbb{B}, \mathbb{C}$ tre problemi decisionali per i quali valgono le seguenti relazioni ($\leq$ indica una Karp-riduzione)

$$\mathbb{A} \leq \mathbb{B}, \qquad \mathbb{B} \leq \mathbb{C} \qquad \mathbb{A} \in \mathcal{NPC} \qquad \mathbb{C} \in \mathcal{NPC}$$

Si dimostri che $\mathbb{B} \in \mathcal{NPC}$.

Esercizio 2 - 10 punti

Dato un problema di decisione $\mathbb{A}$ e una funzione $f(n) \geq n$ sia $\text{PAD}(\mathbb{A}, f)$ il seguente problema di decisione:

> $\text{PAD}(\mathbb{A}, f)$
> **Input**: $(\mathbf{x}, 1^{f(|\mathbf{x}|)-|\mathbf{x}|})$, dove $\mathbf{x}$ è un'istanza di $\mathbb{A}$
> **Output**: *yes* se e solo se $\mathbf{x}$ è un'istanza *yes* per $\mathbb{A}$.

(i) Si dimostri che $\mathbb{A} \in \mathbf{TIME}(n^2)$ se e solo se $\text{PAD}(\mathbb{A}, n^2) \in \mathbf{TIME}(n)$.
(ii) Si dimostri che $\mathbb{A} \in \mathbf{NTIME}(n^2)$ se e solo se $\text{PAD}(\mathbb{A}, n^2) \in \mathbf{NTIME}(n)$.
(iii) Si dimostri che se $\mathbf{TIME}(n) = \mathbf{NTIME}(n)$ allora $\mathbf{TIME}(n^2) = \mathbf{NTIME}(n^2)$.

Esercizio 3 - 15 punti

Si assuma che per ogni problema $\mathbb{A} \in \mathcal{NP}$ esista una riduzione log-space al problema 3-SAT e che valga anche 3-SAT $\leq_L$ NAE-3SAT, dove $\leq_L$ denota una riduzione log-space.

Sia GRAPH-3-COL il problema di determinare se un grafo ammette una 3-colorazione. Si dimostri che se GRAPH-3-COL $\in \mathcal{NL}$ allora $\mathcal{NP} = \mathcal{NL}$.

Esercizio 4 - 15 punti

Si consideri il seguente problema

> NO-TRIANGLE
> **Input**: un grafo non-orientato $G = (V, E)$
> **Output**: *yes* se e solo se in G non esistono tre vertici a due a due collegati da un arco, cioè formanti un triangolo

Assumendo che $\mathcal{L} \neq \mathcal{NL}$, si argomenti per ognuna delle seguenti affermazioni se essa risulta vera o falsa
(i) NO-TRIANGLE è in $\mathcal{NL}$.
(ii) NO-TRIANGLE è $\mathcal{NL}$-completo.
(iii) NO-TRIANGLE è in $\mathcal{L}$.

Esercizio 5 - 15 punti

Si dica per ciascuna delle seguenti affermazioni se essa risulta vera o falsa. Si motivi opportunamente ogni risposta. Risposte non motivate non verranno valutate.

È possibile usare l'assunzione $\mathcal{NP} \neq$ co-$\mathcal{NP}$, ma laddove la si usi va esplicitamente detto che la si sta utilizzando.

1. <u>Se $\mathbb{A}$ è $\mathcal{NP}$-completo e ed esiste una riduzione log-space da $\mathbb{A}$ a $\mathbb{B}$ allora $\mathbb{B}$ è $\mathcal{NP}$-completo.</u>
2. Siano $\mathbb{A}, \mathbb{B}$ due problemi decisionali, e sia $k > 1$ una costante ed $f()$ una funzione che trasforma istanze di $\mathbb{A}$ in istanze di $\mathbb{B}$ tale che per ogni istanza $\mathbf{x}$ di $\mathbb{A}$ vale che $|f(\mathbf{x})| = O(|\mathbf{x}|^k)$.
   <u>Se $\mathbb{B} \in \mathbf{TIME}(n^k)$ allora $\mathbb{A} \in \mathbf{TIME}(n^k)$.</u>
3. Siano $\mathbb{A}, \mathbb{B}$ due problemi decisionali definiti sullo stesso insieme di istanze I. Sia $\mathbb{C}$ un terzo problema definito sull'insieme di istanze I e tale che $\mathbf{x} \in I$ è un'istanza *yes* per $\mathbb{C}$ se e solo se $\mathbf{x}$ è *yes* per $\mathbb{A}$ **e** $\mathbf{x}$ è *yes* per $\mathbb{B}$.
   <u>Se $\mathbb{A}$ e $\mathbb{B}$ sono $\mathcal{NP}$-completi allora $\mathbb{C}$ è $\mathcal{NP}$-completo.</u>

Esercizio 6 - 15 punti

Un insieme A di vertici di un grafo $G = (V, E)$ si dice essere un *insieme indipendente* se nessuna coppia di vertici in A è unita da un arco. Si consideri il seguente problema

> LARGEST INDEPENDENT SET
> **Input**: un grafo $G = (V, E)$ non orientato
> **Output**: *yes* se e solo se il più grande insieme indipendente in G ha taglia esattamente $\lfloor |V|/2 \rfloor$.

(i) Fornire una classe, più in basso possibile nella gerarchia polinomiale, che contiene il suddetto problema. Motivare la risposta.
(ii) La seguente variante del suddetto problema risulta essere $\mathcal{NP}$-completa. Lo si dimostri. (Suggerimento: per la riduzione si può usare INDEPENDENT SET).

> MIDSIZE INDEPENDENT SET
> **Input**: un grafo $G = (V, E)$ non orientato
> **Output**: *yes* se e solo se il più grande insieme indipendente in G ha taglia $\geq \lfloor |V|/2 \rfloor$.

ALGORITMI - Complessità **Anno Accademico 2014-15**

Prova in Itinere 1 - Soluzioni degli esercizi

Esercizio 1 - 8 punti

Una delle definizioni date a lezione per $\mathcal{NP}$ è in termini di predicati logici:

$$\mathcal{NP} = \{A(x) = \exists w : B(x,w),\ B \in \mathcal{P} \text{ e } |w| = O(poly(|x|))\}$$

Secondo la stessa prospettiva consideriamo la seguente classe definita in termini logici

$$\mathcal{OP} = \{A(x) = \exists w : B(x,w),\ B \in \mathcal{P} \text{ e } |w| = O(log(|x|))\}.$$

Si dimostri che $\mathcal{OP} = \text{co}-\mathcal{OP}$.

Soluzione.

Intuizione: Se il verificatore, nel tempo che ha a disposizione, può creare tutti i possibili certificati, allora l'esistenza o meno di un certificato è inutile e l'unica limitazione che rimane è quella relativa alle risorse (di tempo) che il verificatore può usare. Quindi nel caso specifico quello che conta è che si può risolvere il problema in tempo polinomiale. Formalmente:

$$\mathbb{A} \in \mathcal{OP} \iff \exists B(.,.) : (A(x) = 1 \iff \exists w : B(x,w) = yes) \tag{1}$$

Abbiamo anche che, data la natura polinomiale di B e la taglia di w,

$$\exists w : B(x,w) = yes \iff B'(x) = yes \tag{2}$$

dove $B'(.)$ è un algoritmo che costruisce ogni w, chiama $B(x,w)$ e restituisce yes se almeno per uno dei w vale $B(x,w) = yes$. Si noti che poich´ $B(.,.)$ è polinomiale, e $|w| = O(\log(|x|))$ allora $B'(.)$ è polinomiale. Quindi

$$\mathbb{A} \in \mathcal{OP} \iff \exists B'(.) : (A(x) = 1 \iff B'(x) = yes) \tag{3}$$

dove $B'(.)$ è polinomiale, quindi $\mathbb{A} \in \mathcal{P}$. Quindi abbiamo $\mathcal{OP} = \mathcal{P} = \text{co}-\mathcal{P} = \text{co}-\mathcal{OP}$.

Esercizio 2 - 8 punti

Un nostro amico è un pò confuso circa la definizione di co-$\mathcal{NP}$. Ci ha chiesto di chiarirgli se le seguenti affermazioni sono vere o false

1. se $\mathcal{P} \neq \text{co}-\mathcal{NP}$ allora 3-Sat $\not\leq$ 2-Sat (non può esistere una riduzione da 3-Sat a 2-Sat).
2. Possiamo usare il seguente algoritmo non deterministico per risolvere Tautologia

   > L'algoritmo usando l'istruzione non-deterministica **goto both** crea tutti gli assegnamenti per la formula ϕ da verificare. Quindi per ogni assegnamento esiste un ramo di computazione. Ogni ramo di computazione verifica se l'assegnamento creato soddisfa ϕ ed in tal caso dà in output SI e termina, altrimenti termina senza fornire output.

   Il nostro amico è convinto che questo algoritmo funzioni e si chiede come mai esso non dimostra che $\mathcal{NP} = \text{co-}\mathcal{NP}$.

Soluzione.

1. Proviamo la contronominale:

   Sappiamo che 2-Sat $\in \mathcal{P}$ e 3-Sat è NPC. Quindi

   $$\text{3-Sat} \leq \text{2-Sat} \Rightarrow \mathcal{NP} = \mathcal{P} \Rightarrow \text{co}-\mathcal{NP} = \text{co} - \mathcal{P} = \mathcal{P}.$$

   Il che mostra che se $\text{co}-\mathcal{NP} \neq \mathcal{P}$ allora non può valere 3-Sat $\leq$ 2-Sat.

2. Per la seconda, basta osservare che l'algoritmo proposto dal nostro amico risponde SI se esiste almeno un assegnamento che soddisfa la formula, mentre un'istanza di Tautologia è SI se e solo se ogni assegnamento soddisfa la formula.

Esercizio 3 - 12 punti

Si consideri la seguente variante di 3-Sat:

> Majority 3-Sat (Maj-3-Sat)
> **Input**: una formula CNF ϕ in cui ogni clausola ha 3 letterali
> **Output**: yes se e solo se esiste un assegnamento che rende veri almeno due letterali in ogni clausola

Quale delle due seguenti affermazioni vale? (in parentesi una possibile strada per provarle)

- Maj-3-Sat è in $\mathcal{P}$ (mostrare una riduzione usando 2-Sat).
- Maj-3-Sat è $\mathcal{NP}$-completo (mostrare una riduzione usando 3-Sat).

Soluzione. Vale la prima, in quanto possiamo mostrare la riduzione Maj-3-Sat $\leq$ 2-Sat.

Sia $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ un'istanza di Maj-3-Sat, con $C_i = \ell_1^{(i)} \vee \ell_2^{(i)} \vee \ell_3^{(i)}$, dove $\ell_j^{(i)}$ denota il j-mo letterale della i-ma clausola.

Osservazione:

> per ogni fissato assegnamento, la clausola $C_i = \ell_1^{(i)} \vee \ell_2^{(i)} \vee \ell_3^{(i)}$ ha almeno due letterali posti a vero **se e solo se** non esistono due letterali posti a falso, ovverossia **se e solo se** per tutte le coppie di tali letterali almeno uno è posto a vero, ovverossia **se e solo se** risulta vera la seguente formula 2-Sat:
>
> $$D_i = (\ell_1^{(i)} \vee \ell_2^{(i)}) \wedge (\ell_1^{(i)} \vee \ell_3^{(i)}) \wedge (\ell_2^{(i)} \vee \ell_3^{(i)})$$

Ne segue che esiste un assegnamento che rende veri almeno due letterali di ogni clausola di ϕ **se e solo se** esiste un assegnamento che soddisfa la seguente formula 2-CNF

$$\psi = D_1 \wedge D_2 \wedge \cdots \wedge D_m$$

dove ogni D_i è la tripla di clausole (2-CNF) ottenuta da C_i.

Quindi la funzione che mappa ϕ (input a MAJ 3-SAT) in ϕ è una riduzione da MAJ 3-SAT a 2-SAT nel senso che ogni istanza SI è mappata in un istanza SI ed ogni istanza NO è mappata in un'istanza NO.

Inoltre, visto che per ogni clausola, cioè per ogni tripla di letterali in ϕ abbiamo 6 letterali in ψ segue anche che la riduzione è polinomiale.

Esercizio 4 - 12 punti

NSA sostiene di essere capace di individuare i gruppi terroristici controllando le interconnessioni tra le persone che spia. Dicono che prima di un attacco terroristico ogni partecipante contatta o viene contattato da ogni altro partecipante all'attacco. NSA pensa che sia quindi giusto studiare il seguente problema:

MASSIMA INTERCONNESSIONE
Input: n individui, l'insieme di coppie di individui che hanno comunicato tra loro, e un numero $k \leq n$
Output: *yes* se e solo se esiste un gruppo C di almeno k individui tale che
ogni individuo di C ha comunicato con ogni altro individuo di C

Provare che il problema MASSIMA INTERCONNESSIONE è $\mathcal{NP}$-completo. (Una possibile riduzione è da INDEPENDENT SET).

Soluzione. Usiamo MAX-INT per MASSIMA INTERCONNESSIONE.
MAX-INT $\in \mathcal{NP}$

Infatti, dato un insieme C di individui (certificato) possiamo verificare in tempo polinomiale che $|C| \geq k$ e che ogni ndividuo in C ha comunicato con ogni altro individuo di C. La prima verifica costa $O(|C|) = O(n)$ la seconda verifica consiste nel verificare che ogni coppia in $|C|$ è nell'input, che fatto esaustivamente costa al più $O(|x||C|^2)$, dove $|\mathbf{x}|$ è la taglia dell'input $\mathbf{x}$ al problema MAX-INT.

MAX-INT è $\mathcal{NP}$-hard. Questo lo proviamo mostrando IND-SET $\leq$ MAX-INT.

Data un'istanza $(G = (V, E), k)$ per IND-SET costruiamo l'istanza $\mathbf{x}^G$ per MAX-INT in cui abbiamo un individuo i_v per ogni vertice $v \in V$ e per ogni coppia di vertici $u, v \in V$ tali che $(u, v) \notin E$ abbiamo che la coppia i_u, i_v ha comunicato. Inoltre il parametro k è lo stesso dell'istanza di IND-SET.

Osserviamo che un insieme di vertici $I \subseteq V$ è indipendente (cioè per ogni coppia di vertici $u, v \in I$ abbiamo che $(u, v) \notin E$) se e solo se ogni coppia tra gli individui corrispondenti ai vertici di I ha comunicato.

Segue che esiste un insieme indipendente I in G tale che $|I| \geq k$ se e solo se il corrispondente insieme di individui nell'istanza di MAX-INT è a massima interconnessione, cioè esiste un insieme a massima interconnessione in $\mathbf{x}^G$ di taglia almeno k.

Possiamo costruire $\mathbf{x}^G$ in tempo $O(|V|^2|E|)$: 1. scorriamo tutti i veritici ed aggiungiamo un individuo per vertice; 2. enumeriamo le $O(|V|^2)$ possibili coppie di vertici e per ognuna di esse, se la coppia non corrisponde ad un arco (cosa che possiamo sicuramente verificare in $O(|E|)$ aggiungiamo la corrispondente coppia di individui all'istanza $\mathbf{x}^G$. Quindi la costruzione è polinomiale, il ché conclude la dimostrazione della riduzione.

Esercizio 5 - 10 punti

Supponiamo di voler controllare se in una stringa di parentesi aperte e chiuse di 2 possibili tipi: " [" e " (", le parentesi risultano correttamente accoppiate e annidate.

Per esempio la stringa [ ] [( )] [[ ] ( )] è corretta, mentre la stringa [[([ )]]] e la stringa [( ) [ ] non sono corrette (nella prima gli annidamenti sono sbagliati, nella seconda manca una parentesi chiusa).

PARTE 1. Mostrare che il problema è in L.

(ii) Mostrare che se aggiungiamo la limitazione che l'input può essere letto solo da sinistra a destra allora ogni algoritmo che risolve il problema richiede $\Theta(n)$ bit di memoria (dove n è la lunghezza della stringa input)

Soluzione. Diciamo $\mathbf{x} = x_1 \ldots, x_n$ l'input, dove $x_i \in \{[,], (,)\}$. Cerchiamo ora di caratterizzare le stringhe corrette:

Definizione: Data una parentesi aperta x_i definiamo *la chiusura di* x_i come la prima parentesi chiusa alla sua destra (sia essa x_j, per qualche $j > i$) tale che tra i e j il numero di aperture di parentesi è uguale al numero di chiusure di parentesi.

Osservazione 1: In una stringa corretta: (i) per ogni parentesi aperta esiste la sua chiusura; (ii) per ogni parentesi aperta la sua chiusura è una parentesi dello stesso tipo.

Questa condizione è facile da verificare con qualche contatore. Scorriamo la stringa mantenendo l'indice della parentesi da controllare in un contatore i. Ogni volta che incontriamo una parentesi aperta (cioè $x_i \in \{$"[", "("$\}$) usiamo un contatore q per trovare la chiusura di x_i. Poniamo $q = 1$ e scorrendo la stringa a partire da $i + 1$ in avanti per ogni parentesi chiusa decrementiamo q e per ogni parentesi aperta incrementiamo q. Sia k il primo indice su cui otteniamo $q = 0$. Allora x_k è la chiusura di x_i Se k diventa $n+1$ ritorniamo NO in quanto x_i non ha una chiusura. Ci servono tre contatori: i, $j = i+1, \ldots,$ e q tutti da $\log n$ bit. A questopunto possiamo controllare se x_k è dello stesso tipo di x_i o meno. Se no, ritorniamo NO; se si, ricominciamo a scorrere incrementando i fino alla prossima parentesi aperta e ripetiamo.

Chiaramente se la stringa viola la condizione di Osservazione 1 lo riconosceremo e correttamente riporteremo NO.

Definizione 2: Data una parentesi chiusa x_i definiamo *l'apertura di* x_i come la prima parentesi aperta alla sua sinistra (sia essa x_j, per qualche $j < i$) tale che tra j ed i il numero di aperture di parentesi è uguale al numero di chiusure di parentesi.

Osservazione 2: In una stringa corretta: (i) per ogni parentesi chiusa esiste la sua apertura; (ii) per ogni parentesi chiusa la sua apertura è una parentesi dello stesso tipo.

Questa condizione è facile da verificare con qualche contatore. Scorriamo la stringa da destra a sinistra mantenendo l'indice della parentesi da controllare in un contatore i. Ogni volta che incontriamo una parentesi chiusa (cioè $x_i \in \{$"]", ")"$\}$) usiamo un contatore q per trovare l'apertura di x_i. Poniamo $q = 1$ e scorrendo la stringa a partire da $i - 1$ all'indietro per ogni parentesi chiusa incrementiamo q e per ogni parentesi aperta decrementiamo q. Sia k il primo indice su cui otteniamo $q = 0$. Allora x_k è l'apertura di x_i Se k diventa < 1 ritorniamo NO in quanto x_i non ha un'apertura. Ci servono tre contatori: i, $j = i - 1, \ldots,$ e q tutti da $\log n$ bit. A questo punto possiamo controllare se x_k è dello stesso tipo di x_i o meno. Se no, ritorniamo NO; se si, ricominciamo a scorrere decrementando i fino alla prossima parentesi chiusa e ripetiamo.

Chiaramente se la stringa viola la condizione di Osservazione 2 lo riconosceremo e correttamente riporteremo NO.

È ovvio che se la stringa è corretta se e solo se ogni parentesi aperta ha la sua chiusura dello stesso tipo e ogni parentesi chiusa ha la sua apertura dello stesso tipo. Quindi le verifiche di cui sopra correttamente permettono di riconoscere se la stringa è propriamente formata o meno. E usano solo $O(\log n)$ bit quindi il problema è in L.

Lo Pseudocodice (quasi-C) di una possibile implementazione. Usiamo l'array `par` per codificare mediante la funzione `encode(x)` una parentesi x in un numero tra 0 e 3 in modo che per parentesi complementari la somma delle codifiche sia un numero pari, inoltre le parentesi aperte sono un numero tra 0 e 1 e le parentesi chiuse un numero tra 2 e 3.

Inoltre usiamo `dir` $\in \{+1, -1\}$ per definire se cerchiamo una chiusura (+1) o un'apertura (-1). Quindi data una parentesi x abbiamo che `encode(x) + 2dir` appartiene a $\{0, 1, 2, 3\}$ se e solo se x è una parentesi aperta (quando `dir = +1`) e x è una parentesi chiusa quando `dir = -1`.

$par[] = \{$"[", "(", "]", ")"$\}$

Procedure `encode`(x)

1: **for** $i = 0$ **to** 3 **do**
2: **if** $x = par[i]$ **return** i

Procedure `find-open-close`(i, dir)

1: $j = i + dir$, $q = 1$
2: **while** $q \neq 0$ **and** $j \in \{1, \ldots, n\}$ **do**
3: **if** $0 \leq$ `encode`$(x_j) + 2dir \leq 3$ **then** q++ **else** $q - -$
4: **if** $j \notin \{1, \ldots, n\}$ **or** `encode`$(x_j) +$ `encode`$(x_i) \bmod 2 \neq 0$ **then return** 0 **else return** 1

Procedure `Check-parenthesization`(**x** = x[1],... x[n])

for $i = 0$ **to** n **do**
if `encode`$(\mathtt{x_i}) \leq 1$ **then**
if `find-open-close`$(i, +1) = 0$ **return** NO
for $i = n$ **dowto** 1 **do**
if `encode`$(\mathtt{x_i}) \geq 2$ **then**
if `find-open-close`$(i, -1) = 0$ **return** NO
return SI

Esercizio 5 -Parte 2. Mostrare che se aggiungiamo la limitazione che l'input può essere letto solo da sinistra a destra allora ogni algoritmo che risolve il problema richiede $\Theta(n)$ bit di memoria (dove n è la lunghezza della stringa input)

Supponiamo che l'input sia fatto da $n/2$ parentesi aperte e $n/2$ parentesi chiuse. L'osservazione fondamentale è che un qualsiasi algoritmo che non può tornare indietro sulla stringa input, dopo aver scorso la prima metà dell'input dovrà in qualche modo "ricordare la sequenza di parentesi aperte altrimenti non potrà sapere se le parentesi chiuse che incontra *matchano* o meno con quanto c'è nella prima parte della stringa.

Ma per ricordare una sequenza di $n/2$ alternative (tonda o quadra) servono almeno $n/2$ bit.

Alternativamente possiamo ragionare come segue: ognuno dei possibili $2^{n/2}$ input (per la prima parte) deve indurre un differente cammino dell'algoritmo nello spazio dei suoi stati quindi un qualche stato diverso.

> Lo stato quando la prima metà della stringa è $(((\ldots(($ deve essere diverso dallo stato quando la prima metà è $(((\ldots([$ altrimenti l'algoritmo risponderebbe allo stesso modo per $(((\ldots(()$ e $(((\ldots([)$ sbagliando in uno dei due casi.
>
> Analogamente lo stato quando la prima metà della stringa è $(((\ldots(($ deve essere diverso dallo stato quando la prima metà è $(((\ldots[($ altrimenti l'algoritmo risponderebbe allo stesso modo per $(((\ldots(())$ e $(((\ldots[()]$ sbagliando in uno dei due casi.
>
> Ed ancora ci devono essere altri due diversi stati quado la prima metà della stringa è $(((\ldots([$ e quando la prima metà è $(((\ldots[[$ altrimenti l'algoritmo risponderebbe allo stesso modo per $(((\ldots([])$ e $(((\ldots[[]]$ sbagliando in uno dei due casi.

> In generale per ogni stringa di parentesi aperte ci deve essere uno stato diverso. Altrimenti prendiamo la prima parentesi in cui differiscono a partire da n/2 andando indietro e possiamo creare due input diversi sui quali, data l'uguaglianza degli stati, il programma funzionerà allo stesso modo, quindi sbagliando.

Ma se il programma che usiamo per risolvere il problema usa solo $o(n)$ bit di memoria di lavoro allora ci saranno al più $2^{o(n)}$ stati possibili e quindi ci sarà un qualche input per il quale l'algoritmo si comporterà esattamente come su un altro input che differisce dal primo su una qualche parentesi aperta, e coincide sulla corrispondente parentesi chiusa e quindi esiste un input per il quale l'output sarà errato.

Esercizio 6 - 10 punti

A lezione abbiamo provato che se $\mathcal{P} = \mathcal{NP}$ allora vale anche EXP = NEXP.

Provare che per una qualsiasi costante $c > 0$ se NTIME($2^{2^{O(n^c)}}$) $\neq$ TIME($2^{2^{O(n^c)}}$) allora $\mathcal{P} \neq \mathcal{NP}$.

Soluzione.

Proviamo che se $\mathcal{P} = \mathcal{NP}$ allora vale anche NTIME($2^{2^{O(n^c)}}$) = TIME($2^{2^{O(n^c)}}$).

Come a lezione utilizziamo la tecnica del padding.

Sia $\mathbf{x}$ un'istanza di un problema $\mathbb{A} \in$NTIME($2^{2^{O(n^c)}}$). Allora esiste un algoritmo A^N non deterministico che risolve $\mathbf{x}$ in tempo $2^{2^{k|\mathbf{x}|^c)}}$.

Definiamo il problema $\mathbb{B}$ per il quale un'istanza $\mathbf{y}$ è SI se e solo se $\mathbf{y} = (\mathbf{x}, 1^{2^{2^{k|n|^c}}})$ ed $\mathbf{x}$ è un'istanza SI per $\mathbb{A}$. Per tale problema esiste un algoritmo non deterministico B^N tale che controlla semplicemente se $\mathbf{y}$ è separabile in due parti $\mathbf{x}$ e $1^{2^{2^{k|\mathbf{x}|^c}}}$ e quindi simula A^N su $\mathbf{x}$ e ritorna lo stesso output.

Chiaramente B^N risolve un'istanza $\mathbf{y}$ di $\mathbb{B}$ in tempo $O(|\mathbf{y}|)$, quindi $\mathbb{B} \in \mathcal{NP} = \mathcal{P}$, per l'ipotesi. Quindi, esiste per $\mathbb{B}$ un algoritmo deterministico B^D che per ogni istanza $\mathbf{y}$ dice SI sse $\mathbf{y}$ è un istanza SI e lavora in tempo $O(k'|y|^{c'})$ dove k' e c' sono costanti.

Consideriamo quindi l'algoritmo deterministico A^D per $\mathbb{A}$ che lavora come segue: su input $\mathbf{x}$ costruisce un padded input $\mathbf{y} = (\mathbf{x}, 1^{2^{2^{k|\mathbf{x}|^c}}})$ in tempo $O(2^{2^{k|\mathbf{x}|^c}})$. Quindi dà in output $B^D(\mathbf{y}) = B^N(\mathbf{y}) = A^N(\mathbf{x})$, in tempo (il running time di B^D) $O(k'|\mathbf{y}|^{c'}) = O(k'(2^{2^{k|\mathbf{x}|^c}})^{c'}) = O(2^{2^{k|\mathbf{x}|^c}})$. Quindi l'output di A^D è corretto e dato il bound sul tempo abbiamo anche provato che $\mathbb{A} \in$TIME($2^{2^{O(n^c)}}$).

Poiché la prova vale per ogni $\mathbb{A} \in$ NTIME($2^{2^{O(n^c)}}$) segue che NTIME($2^{2^{O(n^c)}}$) $\subseteq$ TIME($2^{2^{O(n^c)}}$) che insieme all'ovvia inclusione TIME($2^{2^{O(n^c)}}$) $\subseteq$ NTIME($2^{2^{O(n^c)}}$) dimostra che le due classi sono uguali (sotto l'ipotesi $\mathcal{P} = \mathcal{NP}$.)

ALGORITMI - Complessità **Anno Accademico 2014-15**

Prova in Itinere 2 - Soluzioni

Nome e Cognome: Matricola:

Importante: Usare solo i fogli forniti dal docente, riportando Nome Cognome e Matricola su ogni foglio. Scrivere in modo **leggibile** e **giustificare** ogni affermazione. Risposte non giustificate saranno valutate **0**. Per ogni algoritmo proposto va provata la correttezza e se necessario al problema, la complessità.
Si chiede di risolvere 3 esercizi:
(i) uno a scelta tra 1 e 2, (ii) uno a scelta tra 3 e 4, (iii) uno a scelta tra 5 e 6.

Esercizio 1 - 15 punti

Dimostrare le seguenti implicazioni

(i) $\mathcal{P} \neq \mathcal{BPP} \Rightarrow \mathcal{P} \neq \mathcal{NP}$

(ii) $\mathcal{NP} \subseteq \text{co}-\mathcal{RP} \Rightarrow \mathcal{ZPP} = \mathcal{NP}$

Soluzione.

(i) Dimostriamo che se $\mathcal{P} = \mathcal{NP}$ allora $\mathcal{P} = \mathcal{BPP}$.

Ricordiamo la gerarchia polinomiale e la definizione di $\mathcal{PH} = \bigcup_{i \geq 1} \Sigma_i^P$. Sappiamo che $\mathcal{BPP} \subseteq \mathcal{PH}$, in quanto abbiamo dimostrato che $\mathcal{BPP} \subseteq \Sigma_2^P$. Sappiamo anche che $\mathcal{P} \subseteq \mathcal{BPP}$.

Se $\mathcal{P} = \mathcal{NP}$ allora la gerarchia polinomale collassa e quindi

$$\mathcal{P} \subseteq \mathcal{BPP} \subseteq \mathcal{PH} = \mathcal{P}$$

da cui segue che $\mathcal{P} = \mathcal{BPP}$.

(ii) Sappiamo che $\mathcal{ZPP} = \text{co}-\mathcal{RP} \cap \mathcal{RP}$. (1)

Inoltre $\mathcal{RP} \subseteq \mathcal{NP}$ in quanto la sequenza di bit che l'algoritmo probabilistico usa è un certificato per l'istanza e l'algoritmo stesso è un verificatore.

Quindi abbiamo anche $\text{co}-\mathcal{RP} \subseteq \text{co}-\mathcal{NP}$, (2)
in quanto

$$\mathbb{A} \in \text{co}-\mathcal{RP} \Rightarrow \overline{\mathbb{A}} \in \mathcal{RP} \Rightarrow \overline{\mathbb{A}} \in \mathcal{NP} \Rightarrow \mathbb{A} \in \text{co}-\mathcal{NP}.$$

Usando la premessa dell'implicazione da dimostrare e quanto osservato per la (2) otteniamo

$$\mathcal{RP} \subseteq \mathcal{NP} \subseteq \text{co}-\mathcal{RP} \subseteq \text{co}-\mathcal{NP}$$

Se $\mathcal{NP} \subseteq \text{co}-\mathcal{NP}$ allora $\mathcal{NP} = \text{co}-NP$. Analogamente se $\mathcal{RP} \subseteq \text{co}-\mathcal{RP}$ allora $\mathcal{RP} = \text{co}-RP$. Quindi dalla sequenza di inclusioni segue che le quattro classi sono uguali. In particolare, otteniamo $\mathcal{NP} = \mathcal{RP} = \text{co}-\mathcal{RP} = \mathcal{RP} \cap \text{co}-\mathcal{RP} = \mathcal{ZPP}$.

Esercizio 2 - 15 punti

Siano $\mathbb{P}_1, \mathbb{P}_2, \mathbb{P}_3$ tre problemi decisionali per i quali un'istanza è definita da un grafo e siano π_1 e π_2 due proprietà che un grafo può avere o meno.

Per $i = 1, 2$ un istanza G è *yes* per $\mathbb{P}_i$, se G presenta la proprietà π_i. Inoltre, un'istanza G è *yes* per $\mathbb{P}_3$ se G presenta contemporaneamente le proprietà π_1 e π_2.

Dimostrare che se $\mathbb{P}_1, \mathbb{P}_2 \in \mathcal{BPP}$ allora vale anche $\mathbb{P}_3 \in \mathcal{BPP}$.

Soluzione. Usiamo la definizione di $\mathcal{BPP}$, ed implicitamente la possibilità di amplificare le probabilità.

Nel seguito indichiamo con n la taglia del grafo in input ai tre problemi. Usiamo 1 per *yes* e 0 per *no* in relazione alla natura di un'istanza e all'output degli algoritmi per il relativo problema su quell'istanza

$\mathbb{P}_1 \in \mathcal{BPP}$ significa che esiste un algoritmo A_1 che prende in input un'istanza G ed una sequenza di bit $\mathbf{r}_1 \in \{0,1\}^{m_1}$ con $m1 = poly(n)$ e tale che

- se $\mathbb{P}_1(G) = 1$ allora $Pr[A_1(G, \mathbf{r}_1) = 1] \geq 10/11$
- se $\mathbb{P}_1(G) = 0$ allora $Pr[A_1(G, \mathbf{r}_1) = 1] \leq 1/11$

Analogamente, $\mathbb{P}_2 \in \mathcal{BPP}$ significa che esiste un algoritmo A_2 che prende in input un'istanza G ed una sequenza di bit $\mathbf{r}_2 \in \{0,1\}^{m_2}$ con $m2 = poly(n)$ e tale che

- se $\mathbb{P}_2(G) = 1$ allora $Pr[A_2(G, \mathbf{r}_2) = 1] \geq 10/11$
- se $\mathbb{P}_2(G) = 0$ allora $Pr[A_2(G, \mathbf{r}_2) = 1] \leq 1/11$

Consideriamo l'algoritmo B per $\mathbb{P}_3$ che prende in input un grafo G ed una stringa random $\mathbf{r}$ di taglia $m_1 + m_2$ e dà la prima parte di $\mathbf{r}$ insieme a G in pasto all'algoritmo A_1 e la seconda parte di $\mathbf{r}$ insieme a G in pasto all'algoritmo A_2. Quindi B risponde 1 se e solo se entrambi gli algoritmi A_1 e A_2 rispondono 1.

Abbiamo che, detta $\mathbf{r}_1$ i primi m_1 bit di $\mathbf{r}$ e $\mathbf{r}_2$ i restanti m_2 bit di $\mathbf{r}$, le due scelte sono indipendenti e random, quindi:

- se $\mathbb{P}_3(G) = 1$ allora $\mathbb{P}_1(G) = 1$ e $\mathbb{P}_2(G) = 1$. Ne segue che $Pr[B(G, \mathbf{r}) = 1] = Pr[A_1(G, \mathbf{r}_1) = 1] \times Pr[B(G, \mathbf{r}_2) = 1] \geq 100/121$
- se $\mathbb{P}_3(G) = 0$ allora $\mathbb{P}_1(G) = 0$ o $\mathbb{P}_2(G) = 0$ o entrambi. Quindi vale che

$$\begin{aligned} Pr[B(G, \mathbf{r}) = 1] &\leq Pr[(A_1(G, \mathbf{r}_1) = 1 \mid \mathbb{P}_1 = 0) \vee (A_2(G, \mathbf{r}_2) = 1 \mid \mathbb{P}_2 = 0)] \\ &\leq Pr[A_1(G, \mathbf{r}_1) = 1 \mid \mathbb{P}_1 = 0] + Pr[A_2(G, \mathbf{r}_2) = 1 \mid \mathbb{P}_2 = 0] \\ &\leq 2/11 \end{aligned}$$

Abbiamo mostrato quindi che l'algoritmo B fornisce la risposta esatta con probabilità d'errore limitata ad una costante inferiore a $1/2$, sia per istanze *yes* che per istanze *no*. Inoltre B è polinomiale, perchè lo sono A_1 ed A_2 e B usa un numero di random bit polinomiale in n. Quindi $\mathbb{P}_3 \in \mathcal{BPP}$.

Esercizio 3 - 15 punti

Si consideri il seguente problema:

2-Player Alternating HamCycle Construction (2P-HamCycle)
Input: Un grafo diretto $G = (V, E)$ ed un vertice iniziale s già marcato
Output: *yes* se e solo se Player 2 vince nel seguente gioco:
Player 1 comincia marcando un vertice che ha un arco entrante da s.
Quindi i giocatori si alternano marcando un vertice non ancora marcato
e tale che esiste un arco entrante dall'ultimo vertice marcato dall'altro giocatore.
Player 2 vince se e solo se alla fine la sequenza definisce un ciclo Hamiltoniano:
tutti i vertici risultano marcati e dall'ultimo vertice marcato ad s esiste un arco.

Mostrare che 2P-HamCycle è $\mathcal{PSPACE}$-completo.

La prova va dettagliata.

Soluzione. A lezione abbiamo dimostrato che 2P-HamPath è $\mathcal{PSPACE}$-completo, dove 2P-HamPath differisce da 2P-HamCycle solo per il fatto che nel primo per la vittoria di Player 2 non si richiede che tra l'ultimo vertice marcato ed il vertice s esista un arco diretto.

Per provare che 2P-HamCycle è $\mathcal{PSPACE}$-hard possiamo quindi procedere in due modi, fondamentalmente equivalenti

- modifichiamo la riduzione 2P-Sat $\leq$ 2P-HamPath vista a lezione aggiungendo un arco da ogni nodo clausola al nodo s. Nella riduzione vista, quando Player 2 vince si termina in un nodo clausola e con la modifica da questo esiste un arco verso il primo nodo, quindi l'istanza è una vittoria anche per il problema 2P-HamCycle. Se invece non si ottiene un cammino hamiltoniano chiaramente non si ottiene nemmeno un ciclo hamiltoniano.

- usiamo la riduzione 2P-HamPath $\leq$ 2P-HamCycle. Data un'istanza $(G = (V, E), s)$ per 2P-HamPath costruiamo l'istanza $(G = (V, E'), s)$ per 2P-HamCycle dove $E' = E \cup \{v \to s \mid v \in V \setminus \{s\}\}$.

  In pratica modifichiamo il grafo aggiungendo archi da ogni vertice v (diverso da s) verso s. L'osservazione è che iniziando il gioco in s (che viene quindi subito marcato) questi archi non possono essere "usati" mai, nel senso che nessuno dei due giocatori può usarli per tornare a s. Essi però garantiscono che se la sequenza di marcature definisce un cammino Hamiltoniano allora esso può essere esteso, con uno di questi archi, ad un ciclo Hamiltoniano.

  Quindi se Player 2 vince in 2P-HamPath su G allora vince anche in 2P-HamCycle su G'. Se Player 2 vince in 2P-HamCycle su G' poiché abbiamo aggiunto solo archi che possono essere usati per chiudere il ciclo, senza l'ultimo arco il ciclo hamiltoniano è un cammino Hamiltoniano in G quindi Player 2 con la stessa strategia vince 2P-HamPath in G. La riduzione è chiaramente polinomiale.

Per dimostrare che 2P-HamCycle $\in \mathcal{PSPACE}$ osserviamo che ogni partita può andare avanti per al più n mosse. Possiamo quindi enumerare tutte le possibili partite una per volta ricordando solo le mosse fissate nella partita che stiamo valutando. Per ogni $i = 1, \ldots, n$ definiamo $vinceP2(m_1, \ldots, m_{i-1})$ come la funzione Booleana che vale 1 se e solo se Palyer 2 vince in una partita che comincia con le mosse $m_1, \ldots, m_i$.

Se i è pari allora il prossimo a muovere è Player 1 ed abbiamo che Player 2 vince sse per ogni scelta m_{i+1} della prossima mossa da parte di $Player1$ poi Player 2 può vincere.

Se i è dispari allora il prossimo a muovere è Player 2 ed abbiamo che Player 2 vince se e solo se esiste una scelta m_{i+1} della prossima mossa da parte di $P2$ che induce una nuova posizione dalla quale P2 può vincere.

Quindi ricordando che le mosse sono in realtà scelte di vertici non marcati, quindi non tra $m_1, \ldots, m_{i-1}$, abbiamo che

$$vinceP2(m_1, \ldots, m_{i-1}) = \begin{cases} \bigwedge_{\substack{m_{i+1} \in V \setminus \{m_1, \ldots, m_{i-1}\} \\ (m_{i-1} \to m_i) \in E}} vinceP2(m_1, \ldots, m_{i-1}, m_i) & \text{se } i \text{ è pari} \\ \bigvee_{\substack{m_{i+1} \in V \setminus \{m_1, \ldots, m_{i-1}\} \\ (m_{i-1} \to m_i) \in E}} vinceP2(m_1, \ldots, m_{i-1}, m_i) & \text{se } i \text{ è dispari} \end{cases}$$

Il valore che ci interessa è $vinceP2(\emptyset)$ che usando la ricorsione può essere calcolato in spazio polinomiale, in quanto bisogna ricordare i nodi già marcati (al più n) e nel calcolare $\bigwedge$ e $\bigvee$ basta un bit. Ci sono al più n livelli di ritorsione.

Esercizio 4 - 15 punti

Si consideri il seguente problema:

Monotone Q-Sat (M-QSat)
Input: Una formula Booleana in forma normale congiunta $\phi(x_1, \ldots, x_n)$ tale che n è pari e ogni variabile in ϕ appare o solo in forma negata o solo in forma non-negata.
Output: *yes* se e solo se la formula quantificata $\Phi = \exists x_1 \forall x_2 \exists x_3 \ldots \forall x_n \phi(x_1, \ldots x_n)$ è vera.

Dimostrare la seguente implicazione

$$\mathcal{P} \neq \mathcal{NP} \Rightarrow \text{2-Player-Sat} \not\leq \text{M-QSat},$$

dove $\leq$ indica una riduzione polinomiale.
Suggerimento: Individuare la classe di complessità più adeguata per M-QSat.

Soluzione. Il problema M-QSat è in $\mathcal{P}$, come dimostreremo in seguito.

Da questo e dalla $\mathcal{PSPACE}$-hardness di 2-Player-Sat segue che se 2-Player-Sat si riducesse polinomialmente a M-QSat allora avremmo $\mathcal{PSPACE} = \mathcal{P}$. E visto che $\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}$ seguirebbe anche $\mathcal{NP} = \mathcal{P}$, quindi sotto l'ipotesi $\mathcal{P} \neq \mathcal{NP}$ deve valere 2-Player-Sat $\not\leq$ M-QSat,

Resta da dimostrare che M-QSat è in $\mathcal{P}$. Cominciamo con l'assumere che in ϕ tutte le variabili siano positive (non-negate). In tal caso, ponendo tutte le variabili dispari a vero e tutte le variabili pari a falso otteniamo un assegnamento base: chiamiamolo $\mathbf{x}^* = x_1^*, x_2^*, \ldots, x_n^*$. Supponiamo che $\phi(\mathbf{x}^*) = V$ allora possiamo concludere che si tratta di un'istanza *yes*. Infatti, per ogni modifica dei valori delle variabili di indice pari da falso a vero il valore delle formula può solo aumentare (dove intendiamo falso $<$ vero) in quanto potremmo al più aumentare il numero dei letterali veri in qualche clausola. Questo implica che esiste la scelta delle varaibili dispari come in $\mathbf{x}^*$ tale che per ogni possibile scelta delle altre variabili, e quella in $\mathbf{x}^*$ è la peggiore possibile, la formula risulta vera

Se al contrario $\phi(\mathbf{x}^*) = F$ allora possiamo concludere che si tratta di un'istanza *no*. Infatti, anche se cambiamo qualcuna (o anche tutte) delle variabili di indice dispari da vero a falso, il valore di verità della formula può solo diminuire o rimanere invariato. Quindi non esiste una scelta per le variabili dispari tale che per ogni scelta delle variabili pari ϕ è vera, in particolare la scelta delle variabili pari come in $\mathbf{x}^*$ la rendono falsa.

Se non tutte le variabili sono in forma positiva, lo stesso ragionamento vale semplicemente fissando $\mathbf{x}^*$ come l'assegnamento che per ogni variabile dispari, rende vera ogni occorrenza dell'unico tipo di letterale corrispondente a quella variabile; e che per ogni variabile pari, rende falsa ogni occorrenza dell'unico tipo di letterale corrispondente a quella variabile. Possiamo farlo perchè c'è un unico letterale per ogni variabile.

Esercizio 5 - 20 punti

Sia MAX-3SAT(k) la variante di MAX-3SAT in cui la formula in input ϕ soddisfa il seguente vincolo addizionale: ogni variabile appare in ϕ un numero di volte non superiore a k.

Sia VC(d) la variante di Min VertexCover in cui il grafo G in input soddisfa il seguente vincolo addizionale: il grado massimo dei vertici di G non è superiore a d.

1. Si fornisca una L-riduzione da MAX-3SAT(k) a VC(k+1).

   Suggerimenti: (i) Come base per la riduzione può essere utile considerare la costruzione del grafo vista nella riduzione da Max 3-Sat ad Independent Set. (ii) Se la formula ϕ ha m clausole allora una soluzione ottima ne soddisfa almeno $m/2$, quindi $m \leq 2OPT(\phi)$.

2. Usando 1. e la Proposizione 1 qui in basso, si dimostri la seguente implicazione

$$\mathcal{NP} \neq \mathcal{P} \Rightarrow \text{ per ogni } d \geq 4 \text{ VC}(d) \notin \mathcal{PTAS}.$$

Proposizione 1. *Per $k \geq 3$ se $\mathcal{NP} \neq \mathcal{P}$ allora* MAX-3SAT*(k) $\notin \mathcal{PTAS}$.*

Soluzione. Una L-riduzione è una quadrupla (f, g, α, β), dove f trasforma un'istanza di MAX-3SAT(k) in un'istanza di VC(k+1), g trasforma soluzioni per l'istanza trasformata in soluzioni per l'istanza di partenza, e α e β sono costanti che definiscono la relazione lineare tra gli ottimi delle due istanze e le differenze assolute tra gli ottimi e le soluzioni coinvolte (la definizione formale è nelle slide del corso).

La funzione f. Prende in input una formula 3-CNF e produce un grafo G come segue: per ogni clausola $C = (\ell_1 \vee \ell_2 \vee \ell_3)$ crea tre vertici che indichiamo con $\ell_1^C, \ell_2^C, \ell_3^C$. Tale vertici sono collegati a formare una clique da tre. Inoltre vertici derivanti da clausole diverse vengono collegati se e solo se corrispondono a letterali che sono l'uno la negazione dell'altro (quindi a scelte di variabili incompatibili tra loro).

Si noti che se una variabile appare $\leq k$ volte in ϕ ci saranno k vertici in G associati ad essa, in forma positiva o negata. Inoltre ognuno di questi vertici sarà adiacente ad al più altri $k-1$ vertici relativi alla stessa variabile ed ai due altri vertici nel suo stesso triangolo. Quindi ogni vertice avrà grado al più $2 + k - 1 = k + 1$, come richiesto da un'istanza di VC($k+1$).

Questa è la riduzione che abbiamo usato per 3-Sat $\leq$ Independent Set. Quindi possiamo facilmente mostrare che per ogni t esiste un assegnamento che soddisfa t clausole se e solo se esiste un independent set in G di taglia t e tale independent set ha al più un vertice in ogni triangolo. Poichè il complemento di un indipendent set è un vertex cover, abbiamo che: esiste un assegnamento che soddisfa t clausole se e solo se esiste un vertex cover in G di taglia $3m - t$.

In particolare, indicando con $OPT(\phi)$ l'assegnamento che massimizza il numero di clausole soddisfatte e $OPT(G)$ is vertex cover minimo nel grafo $G = f(\phi)$ abbiamo che $OPT(G) = 3m - OPT(\phi)$. Dal suggerimento, sappiamo che $m \leq 2OPT(\phi)$ quindi la precedente relazione diventa

$$OPT(G) = 3m - OPT(\phi) \leq 5OPT(\phi)$$

da cui possiamo porre $\alpha = 5$.

Inoltre definiamo g come la funzione che dato un vertex cover A per $G = f(\phi)$ lo mappa su un assegnamento che corrisponde a rendere veri i letterali corrispondenti ai vertici nel complemento del vertex cover. Tale assegnamento esiste perchè il complemento di A è un independent set, quindi non contiene vertici relativi a letterali incompatibili. Fissando il valore di tali letterali, le restanti

variabili le fissiamo arbitrariamente. Tale assegnamento soddisfa almeno le clausole corrispondenti ai triangoli da cui abbiamo preso vertici per l'indipendente set, e possiamo prenderne al più uno per triangolo. Quindi indicando con $s(f(\phi))$ la taglia del vertex cover A e con $s(g(\phi, A))$ il numero di clausole soddisfatte da tale assegnamento ottenuto con la trasformazione g abbiamo

$$s(g(\phi, s^{VC}(f(\phi)))) \geq 3m - s(f(\phi)).$$

Ne segue che

$$\begin{aligned} |OPT(\phi) - s(g(\phi, s^{VC}(f(\phi))))| &= OPT(\phi) - s(g(\phi, s^{VC}(f(\phi)))) \\ &= 3m - OPT(G) - s(g(\phi, s^{VC}(f(\phi)))) \\ &\leq 3m - OPT(G) - 3m + s(f(\phi)) \\ &= |OPT(G) - s(f(\phi))| \end{aligned}$$

che ci da le seconda relazione necessaria alla L-riduzione, se poniamo $\beta = 1$.

Abbiamo provato che Max-3-Sat$(k) \leq_L$ VC$(k+1)$. Quindi vale anche Max-3-Sat$(k) \leq_{PTAS}$ VC$(k+1)$.

Proviamo ora l'implicazione al punto 2., ragionando per assurdo. Supponiamo che $\mathcal{NP} \neq \mathcal{P}$ ma che esista un $d \geq 4$ tale che VC$(d) \in \mathcal{PTAS}$.

Allora questo insieme a quanto provato, cioè Max-3-Sat$(d-1) \leq_{PTAS}$ VC(d), implicherebbe che Max-3-Sat$(d-1) \in \mathcal{PTAS}$ per un $d \geq 4$, cioè, ponendo $d' = d-1$, Max-3-Sat$(d') \in \mathcal{PTAS}$ per un $d' \geq 3$, cosa che risulterebbe in contraddizione con la Proposizione 1. Quindi deve valere l'implicazione al punto 2.

Esercizio 6 - 20 punti

1. Si dimostri che per ogni $r > 1$ se esiste un algoritmo polinomiale A che fornisce una r-approssimazione per Max Clique allora esiste anche un algoritmo polinomiale B che fornisce una $\sqrt{r}$-approssimazione per Max Clique.

   Suggerimento: Si utilizzi la trasformazione del grafo come nell'analogo risultato visto a lezione per Independent Set.

2. Sulla base del risultato dimostrato al punto 1. e della Proposizione 2 in basso, si dimostri la seguente affermazione

   Se $\mathcal{NP} \neq \mathcal{P}$ *allora* Max Clique $\notin \mathcal{APX}$.

Proposizione 2. *Se* $\mathcal{NP} \neq \mathcal{P}$ *allora* Max Clique *non ammette un algoritmo polinomiale* 2*-approssimante.*

Soluzione. Sia $G = (V, E)$ un'istanza di Max Clique. Consideriamo il grafo $G[G]$ definito come a lezione (vedi risultati di inapprossimabilità di Independent Set). Si ha che per ogni clique K per $G[G]$ possiamo ottenere una clique di taglia $|K|^{1/2}$ per G In particolare $OPT(G[G]) = OPT(G)^2$ dove $OPT(G[G])$ è la taglia di una massima clique in $G[G]$ e $OPT(G)$ è la taglia di una massima clique in G.

Quindi possiamo definire un algoritmo B che dato un grafo G costruisce—in tempo polinomiale—il grafo $G[G]$ ed usa A su di esso. L'algoritmo A fornisce una clique K in $G[G]$ di taglia k tale che $k \geq OPT(G[G])/r = OPT(G)^2/r$. Data la costruzione di $G[G]$ e le proprietà citate, da K possiamo ottenere una clique K_G in G tale che

$$|K_G| \geq |K|^{1/2} \geq \left(\frac{OPT(G)^2}{r}\right) = \frac{OPT(G)}{r^{1/2}}$$

. Quindi B è un algoritmo che garantisce una $\sqrt{r}$-approssimazione come richiesto. Il punto 1. è dimostrato

Per il punto 2., supponiamo che esista una costante $q > 2$ ed un algoritmo q-approssimante per Max Clique. Allora per il punto 1. esisterebbe un algoritmo B_1 che risulta $q^{1/2}$-approssimante. Riapplicando il punto 1. a tale algoritmo, concluderemmo che esiste un algoritmo B_2 che risulta $q^{1/4}$-approssimante, e quindi un B_3 che risulta $q^{1/8}$-approssimante etc.. In particolare per ogni costante $c \geq 1$ esisterebbe un algoritmo polinomiale—in quanto useremmo la trasformazione polinomiale al punto 1. un numero costante di volte—B_c che risulta $q^{(1/2)^c}$-approssimante per Max Clique. Quindi con un numero costante di ripetizioni del punto 1. possiamo ottenere un algoritmo con approssimazione migliore di 2 violando la proposizione.

Per la precisione, ponendo $\tilde{c} = \lceil \log_2 \log_q 1/2 \rceil$, allora $q^{(1/2)^{\tilde{c}}} \leq 2$, quindi avremmo un algoritmo polinomiale $B_{\tilde{c}}$ che garantisce un'approssimazione 2 in contraddizione con la Proposizione.

Computational Complexity **Academic Year 2015-16**

Exercise Set 1

1. Discuss the following statement "$\mathcal{NP} = co\text{-}\mathcal{NP}$ *implies* $\mathcal{NP} = \mathcal{P}$".

2. Write down a polynomial time non-deterministic algorithm for 3-SAT. Use your favourite C language with the additional non-deterministic instruction *goto-both*.

3. Show that if for a problem A there is a non-deterministic polynomial time algorithm that, on each computation thread, uses $O(\log n)$ `goto-both` then A is in $\mathcal{P}$

4. Show that if a problem A is $\mathcal{NP}$-complete then the complement of A, denoted by $\overline{A}$ is in co-$\mathcal{NP}$ and for each problem B in co-$\mathcal{NP}$ we have $B \leq \overline{A}$.

5. Show that if there exists an $\mathcal{NP}$-complete problem A that is in co-$\mathcal{NP}$ then co-$\mathcal{NP} = \mathcal{NP}$

6. Show that if $\mathcal{P} = \mathcal{NP}$ then the following problem is $\mathcal{NP}$-complete:

   ZERO OR ONE
   Input: A single bit b
   Output: *yes* iff $b = 1$

7. Provide a formal proof that the following problem is in co-$\mathcal{NP} \cap \mathcal{NP}$.

   SMALL FACTOR
   Input: Two integers q and r
   Output: *yes* iff q has a prime factor smaller than r

8. Using any of the definitions of $\mathcal{NP}, \mathcal{P}$ and co-$\mathcal{NP}$ seen in class, provide a formal proof of the following statements ($\leq$ denotes a Karp-reduction). Let PROB1 and PROB2 and PROB3 be three decision problems.

   (i) If PROB1 $\in \mathcal{NP}$, and PROB2 $\leq$ PROB1 then PROB2 $\in \mathcal{NP}$.

   (ii) Assume PROB1 is $\mathcal{NP}$-complete, PROB3 $\in$ co-$\mathcal{NP}$, PROB2 $\in \mathcal{P}$ and PROB1 $\leq$ PROB2. Show that these assumptions imply the following two statements

   (a) PROB3 $\leq$ PROB1;

   (b) PROB1 $\leq$ PROB3.

Computational Complexity **Academic Year 2015-16**

Exercise Set 2

1. Prove that INDEPENDENT SET and VERTEX COVER are in $\mathcal{P}$ if the graph in input is 2-colorable, or, equivalently, if the input graph is bipartite.

2. You are given a graph $G = (V, E)$, with weights w_e on its edge $e \in E$. The weights can be negative or positive. The ZWC-Problem is to decide whether there is a simple cycle in G so that the sum of the edge weights on this cycle is exactly 0. Prove that this problem is $\mathcal{NP}$-complete.

3. Consider the VERTEX COVER problem, asking whether a graph G with n vertices has a vertex cover of size k or less.

   (i) Show that if $k = O(1)$ then this problem is in $\mathcal{P}$. (ii) Design and analyze an algorithm that takes $O(2^k n^c)$ time for some small constant c not depending on k. (*Hint*: for an edge (u, v) either u or v must be in the vertex cover.)

4. Consider the following variant of k-SAT

   Show that THRESHOLD k-SAT is $\mathcal{NP}$-complete even when $k = 2$.

   THRESHOLD k-SAT
   Input: A k-SAT formula ϕ and an integer t
   Output: *yes* iff there exists a satisfying assignment for ϕ where $\leq t$ variables are true

   Hint: for $k \geq 3$ you might try to reduce from k-SAT. For $k = 2$ you might try to reduce from VERTEX COVER.

5. Let S be a set of 10 *distinct* integers taken from the set $\{0, 1, \ldots, 100\}$. Show that there are two *disjoint* subsets $A \subset S$ and $B \subset S$ with $A \cap B = \emptyset$ such that the sum of the elements in A is equal to the sum of the elements in B (it is not required that $A \cup B = S$). *Hint*: How many possible subsets do you have? What is the number of possible values for the sum of the elements in a subset?

   (ii) Do you think it is easy to find such pair of sets A and B? Motivate your answer. Can you find a generalization of these questions into a computational problem?

6. Let L be a set of finite strings. Let L^* be the set of strings which can be written as the concatenation of sequences (of any length) of strings in L. That is, each word v in L^* can be written as $v = w_1 w_2 \ldots w_k$ (for some k) where $w_i \in L$ (for each $i = 1, \ldots, k$).

   Let DECIDE-L be the problem of telling whether a given string is in L or not. Let DECIDE-L^* be the problem of telling whether a given string is in L^* or not.

   Prove that

   (i) if DECIDE-$L \in \mathcal{NP}$ then DECIDE-$L^* \in \mathcal{NP}$

   (ii) if DECIDE-$L \in \mathcal{P}$ then DECIDE-$L^* \in \mathcal{P}$

Computational Complexity **Academic Year 2015-16**

Exercise Set 3

1. Show that $\mathcal{NP}$ = co-$\mathcal{NP}$ *if and only if* 3-Sat and Tautology are polynomial-time reducible to one another.

2. Let $G = (V, E)$ be an undirected graph. Let k^* be the size of the minimum vertex cover of G.

   A set of edges of G such that no two of them share a vertex is called a matching of G. A matching is called *maximal* if for any edge e of G which is not in the matching, there is an edge in the matching that shares a vertex with e.

   Show how to construct a maximal matching M in polynomial time. Show that the smallest vertex cover for G is at least as large as $|M|$. Let A be the set of vertices incident to the edges in M. Show that A is a vertex cover and $k^* \leq |A| \leq 2k^*$.

3. Suppose that you are given a graph G and a number k and you are told that either (i) the smallest vertex cover of G is of size at most k or (ii) it is of size at least $3k$ (in other words the size of the smallest vertex cover is guaranteed not to be a number in the set $\{k+1, k+2, \ldots, 3k-1\}$).

   Show a polynomial time algorithm that can distinguish between the two possible cases, namely in polynomial time it must decide whether the smallest vertex cover of G has size $\leq k$ or it has size $\geq 3k$.

   Since Vertex Cover is $\mathcal{NP}$-hard, why such a polynomial time algorithm does not show that $\mathcal{P} = \mathcal{NP}$?

4. Consider the following variant of Graph 3-Colouring. In addition to the graph G, we are given for each vertex v a forbidden color f_v. For instance, we might be told that vertex 1 cannot be *red* and vertex 2 cannot be *green* and so on. The question is whether G has a proper coloring (i.e., no edge with the two vertices of the same color) such that for each $v \in V$, vertex v is not colored with the color forbidden f_v. In the following description we use numbers $\{1, 2, 3\}$ to represent the three possible colors. Also, for $i = 1, \ldots, n$, the color of ith vertex is c_i.

   Constrained Graph 3-Colouring
   Input: A graph G with n vertices and a list $f_1, \ldots, f_n \in \{1, 2, 3\}$
   Output: *yes* iff G has a proper 3-colouring $c_1, \ldots, c_n$ where $c_v \neq f_v$ for all v

   (i) Show that the problem is in $\mathcal{P}$ by reducing it to another problem in $\mathcal{P}$

   (ii) What can we say about Constrained Graph 4-Colouring? Is it also in $\mathcal{P}$ or is it $\mathcal{NP}$-complete? Motivate your answer with the appropriate reduction.

5. Consider the following problem

   Vertex Alliances
   Input: A graph $G = (V, E)$, a positive integere k and for each vertex $v \in V$ a number $a_v \geq 1$
   Output: *yes* iff there is a set of vertices $A \subseteq D$ such that
   $|A| \leq k$ and for each $v \in V \setminus A$ there are at least a_v neighbors of v which are in A.

   Show that Vertex Alliance is $\mathcal{NP}$-complete by reducing from Vertex Cover. You can try to produce an instance of Vertex Alliance where each vertex v has an associated number $a_v = 1$.

6. The degree of a vertex v in a graph $G = (V, E)$ is defined as the number of vertices $u \neq v$ such that $(u, v) \in E$, and it is denoted by $d(v)$. In words, the degree of a vertex is the number of edges incident to it.

   Show that in any finite graph, the number of vertices with odd degree cannot be odd. For instance, there cannot be a graph where there are 3 vertices with degree $3, 5, 5$ respectively and the remaining vertices all have even degree.

7. Let $G = (V, E)$ be an undirected connected graph and for each $e \in E$ let $w(e)$ denote a weight associated to the edge e. A *spanning tree* for G is a set A of $|V| - 1$ edges of E such that the graph $T = (V, A)$ is a tree. The weight of the spanning tree T is the sum of the weights of its edges. Consider the following three problems and argue whether they belong to the class $\mathcal{P}$ or whether they are $\mathcal{NP}$-complete

   Min-Spanning-Tree
   Input: A graph $G = (V, E)$, a non-negative weight $w(e)$ for each $e \in E$, a value $k \geq 0$
   Output: *yes* iff there is a spanning tree for G of weight $\leq k$

   Max-Spanning-Tree
   Input: A graph $G = (V, E)$, a non-negative weight $w(e)$ for each $e \in E$, a value $k \geq 0$
   Output: *yes* iff there is a spanning tree for G of weight $\geq k$

   Exact-Spanning-Tree
   Input: A graph $G = (V, E)$, a non-negative weight $w(e)$ for each $e \in E$, a value $k \geq 0$
   Output: *yes* iff there is a spanning tree for G of weight exactly equal to k

Computational Complexity **Academic Year 2015-16**

Exercise Set 4

1. Show that the following problem is $\mathcal{NP}$-complete by reducing from SubSetSum.

   Partition
   Input: A set of numbers $Y = \{y_1, \ldots, y_N\}$
   Output: *yes* iff there exists a subset of the numbers whose sum is exactly half of the total sum of all N numbers, i.e., iff $\exists A \subset Y : \left(\sum_{y \in A} y\right) = \frac{1}{2}\left(\sum_{y \in Y} y\right)$

   Hint:*For* Partition *consider the instance obtained from the instance* $(X = \{x_1, \ldots, x_n\}, S)$ *of* SubSetSum *by adding to* X *one element of value* $2S$ *and one element of value equal to the total sum on* X.

2. In one of the definitions seen in class, the class $\mathcal{NP}$ is defined in terms of properties which are expressible as follows:

   $$\mathcal{NP} = \{A(x) = \exists w : B(x, w),\ B \in \mathcal{P} \text{ e } |w| = O(poly(|x|))\}$$

   According to the same perspective, let us define the following complexity class

   $$\mathcal{OP} = \{A(x) = \exists w : B(x, w),\ B \in \mathcal{P} \text{ e } |w| = O(log(|x|))\}.$$

   Prove that $\mathcal{OP} = \text{co}-\mathcal{OP}$.

3. A friend of ours is a bit confused about the definition of the class co-$\mathcal{NP}$. He asked us to argue about the following two issues

   - co-$\mathcal{NP}$ is the complement of $\mathcal{NP}$
   - Consider the following non-deterministic algorithm to solve Tautology?

     Use the non-deterministic instruction **goto both** to create all the possible assignments for the input Boolean formula; then for each assignment, if it is not satisfying say NO and terminate, otherwise say YES.

     Our friend is asking us how come the existence of such a polynomial non-deterministic algorithm is not enough to prove that NP = co-NP.

   Argue whether any of the above statements is false.

4. Consider the following variant of Independent Set

   Exact Independent Set
   Input: A graph $G = (v, E)$ and an integer k
   Output: *yes* iff G's largest independent set has size k

   How can you express the property $A(x)$ that x is a *yes*-instance?

   Can you conclude that Exact Independent Set $\in \Pi_2 P$?

   Can you conclude that Exact Independent Set $\in \Sigma_2 P$?

5. Consider the following variant of Reachability. The input is a graph $G = (V, E)$ and the initial and final position of k coins, i.e., vertices $s_1, \ldots, s_k$ for the starting positions and vertices $t_1, \ldots, t_k$ for the final positions. The task is to decide whether it is possible to move each coin from its initial position to its final position. Each step of the procedure can move one coin from its present position to an adjacent vertex, but we can never have two coins on the same vertex at the same time.

   (i) Show that this problem is in NL.

   (ii) Try to give a deterministic algorithm for the problem and analyze its space complexity as a function of $n = |V|$ and k.

6. Define NSPACETIME$(f(n), g(n))$ as the class of problems that can be solved nondeterministically by an algorithm that uses $O(f(n)$ memory and runs in time $O(g(n))$, where n denotes the size of the input. First explain why this is *a priori* a stronger property than being both in NSPACE$(f(n))$ and in NTIME$(g(n))$. In other words explain why

   $$\text{NSPACETIME}(f(n), g(n)) \subseteq \text{NSPACE}(f(n)) \cap \text{NTIME}(g(n))$$

   might be in general a proper inclusion.

Computational Complexity **Academic Year 2015-16**

Exercise Set 5

1. A directed graph is strongly connected if for every pair of vertices u, v there is both a directed path from u to v and a directed path from v to u. Let Strong Connectedness be the problem of deciding whether a given graph is strongly connected.

   (i) Show that Strong Connectedness is in NL.

   (ii) Show a log-*space reduction* from Reachability to Strong Connectedness.

2. Show that 2-Sat is NL-complete

   Hint: You can show that Reachability can be log-space reduced to the complement of 2-Sat. Then, you can use the consequences of the Immerman-Szelepcsnyi Theorem to complete the proof.

3. Show that if 3-Sat $\in \mathcal{P}$ then deciding whether two graphs G_1 and G_2 are isomorphic can be done in polynomial time.

   Recall that two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic iff there is a function $f : v \in V(G_1) \mapsto f(v) \in V(G_2)$ (that maps vertices in G_1 to vertices in G_2) such that $(u, v) \in E_1$ if and only if $(f(u), f(v)) \in E_2$.

4. Define the classes $\Sigma_2 P$ and $\Pi_2 P$ and show that

   $$\Sigma_2 P \cup \Pi_2 P \subseteq \Sigma_3 P.$$

5. Consider the following problem

   Station placement
   Input: A graph $G = (V, E)$ and a positive integer k
   Output: *yes* iff there is a set of vertices $D \subseteq V$ such that $|D| \leq k$ and for each $v \in V \setminus D$ there is some vertex $u \in D$ which is adjacent to v.

   Show that Station placement is $\mathcal{NP}$-complete by reducing from Vertex Cover. You can try to modify the graph G' in an instance $(G' = (V', E'), k')$ of the Vertex Cover problem by adding a vertex w for each edge $(u, v) \in E'$ and connecting it to u and v. Argue that the new instance is *yes* for Station placement if and only if the original instance was *yes* for Vertex Cover; you will have to properly define the value of the parameter k for the Station placement instance.

6. Give a polynomial time algorithm that solves the Station placement problem for the special case where the graph is a tree.

7. Consider the following variant of the Independent Set problem

   Extended Independent Set
   Input: A graph $G = (V, E)$ and a positive integer k
   Output: *yes* iff there is a set of vertices $I \subseteq V$ such that $|I| \geq k$ and each pair of vertices $u, v \in I$ are at distance at least 3 from each other, i.e., there is no edge $(u, v) \in E$ and there is no vertex w such that $(u, w) \in E$ and $(w, v) \in E$.

   Show that Extended Independent Set is $\mathcal{NP}$-complete.

8. Show how we could use a polynomial time algorithm that solves the Independent Set problem to solve in polynomial time the following optimization variant of the coloring problem

   Minimum Graph Colouring
   Input: A graph $G = (V, E)$
   Output: a proper coloring of the vertices of G that uses the minimum possible number of colors

   Hint: The core part of the question is to show that the polynomial solution to the decision variant of an $\mathcal{NP}$-complete problem can be used to solve in polynomial time the search and optimization variant of the problem.

9. You want to develop techniques for the coordination of a group of robots. Each robot has a radio transmitter that it uses to communicate with a base station. However, if two robots get too close to each other then the transmitters interfere. So it is important to plan the motion of the robots guaranteeing that each of them reaches the desired destination but without ever coming too close to another robot.

   We can model this problem as follows. We have an undirected graph $G = (V, E)$ representing the floor plan where the robots move and there are two robots initially located at nodes a and b. Robot at node a has to reach a given node c and robot initially at node b has to reach a given node d. Each robot in one step can move from the node where it is to an adjacent node. We want to plan the series of steps for each robot so that at any time step, they don't occupy two nodes which are at distance $\leq k$ for a given parameter k.

   We can assume that the distance between the two starting nodes a and b as well as the distance between the two destination nodes c and d is at least k.

   - (i) Show that the above problem is in $\mathcal{P}$, by providing a polynomial algorithm that given the graph $G = (V, E)$, the initial positions of the two robots $a, b \in V$, the final positions $c, d \in V$ and the minimum distance k, provides a complete schedule of the robots movements which never takes them at a distance $\leq k$.
   - (ii) Is the problem in NL?

10. Consider the following problem and show that it is $\mathcal{NP}$-complete by reducing from Vertex Cover.

    Set Cover
    Input: A set $U = \{u_1, \ldots, u_n\}$ and a family of subsets of U, i.e., $S_1, \ldots, S_m \subseteq U$, and a parameter k.
    Output: *yes* iff there are k of the subsets in the input such that their union equals U.

11. Consider the following variant of the Sat problem

    Sat-k
    Input: A CNF formula ϕ such that each variable appears at most k times
    Output: *yes* iff there exists a satisfying assignment

    Ex: $\phi = (\overline{x_1} \vee x_2 \vee x_4 \vee x_7) \wedge (\overline{x_4} \vee x_2) \wedge (x_5 \vee x_6 \vee x_2 \vee x_1) \wedge (\overline{x_3} \vee \overline{x_1} \vee x_4)$ is an instance of Sat-3 (because each variable appears at most 3 times) but it is not an instance of Sat-2 because x_1, x_2 and x_4 appear 3 times in the formula (in negated o non-negated form).

    Show that Sat-2 $\in \mathcal{P}$ and Sat-3 is $\mathcal{NP}$-complete.

    What about Sat-k for $k > 3$?

Computational Complexity **Academic Year 2015-16**

Exercise Set 6

1. Show that any $\mathcal{PSPACE}$-hard problem is also $\mathcal{NP}$-hard.

2. Show that, if every $\mathcal{NP}$-hard problem problem is also $\mathcal{PSPACE}$-hard, then $\mathcal{PSPACE} = \mathcal{NP}$.

3. Consider the following problem

   2-Player Alternating HAMPATH Construction (2P-HamPath)

   Input: A directed graph $G = (V, E)$ and an initial vertex s
   Output: *yes* iff Player 2 has a winning strategy in the following game:
   Player 1 starts with a vertex with an incoming edge from s.
   Players then alternate choosing a vertex never chosen before and
   with an incoming edge from the last vertex chosen by the other player.
   Player 2 wins if in the end the path so constructed is Hamiltonian.
   Player 1 wins if at some point no new vertex can be added to the path and it's not Hamiltonian.

   Show that 2P-HamPath is $\mathcal{PSPACE}$-complete by reducing from 2-Player Sat.

4. Consider the following Cat&Mouse game: The game is played on an undirected graph $G = (V, E)$. One node $h \in V$, fixed at the beginning, is called the hole. The node occupied by the cat is named c and the node occupied by the mouse is named m. The player take turns moving to a vertex adjacent to the one where they are currently standing. The Cat wins if at any point it manages to occupy the same node as the mouse. The mouse wins if it manages to reach the *hole* before being caught by the cat.

   Let us define the problem HappyMouse where an instance is given by $\mathbf{x} = (G, h, c, m)$, defining the graph G, the location of the *hole* and the initial locations of the cat and the mouse respectively. The instance is *yes* if and only if there is a winning strategy for the mouse.

   Analyze the complexity of the game.

5. Consider the problem Geography we analyzed in class. Show that the problem is also $\mathcal{PSPACE}$-complete if we allow vertices to be visited more than once but edges can only be used once.

   Hint: You can use almost the same reduction as for the variant seen in class.

6. Consider the following generalization of binary search. We first agree with a friend on a number n and then we have to identify the number $x \in \{1, \ldots, n\}$ secretly chosen by our friend by asking questions of the form: "Is $x \in A$?", where A is a subset of $\{1, \ldots, n\}$. If we have no constraint on the choice of the sets used in the questions, we know that there is a way to find out the secret number with $\lceil \log n \rceil$ questions.

   Now, suppose that the set of possible questions is also fixed beforehand, i.e., it is some family of sets $\mathcal{F} = \{T_1, \ldots, T_m\}$ we have agreed upon. Then, it is, in general, much more difficult to find an optimal strategy, i.e., a strategy that finds the secret number using the minimum possible number of questions.

   Example: Suppose that $n = 16$ and the family of available sets/questions are

   $$\mathcal{F} = \{\{1\}, \{2\}, \ldots \{8\}, \{1, \ldots, 8\}, \{1, 2, 3, 9, 10, 13, 14\}, \{7, 9, 11, 13, 15\}, \{9, 10, 11, 12\}\}.$$

   You can show that in the worst case (the worst possible choice of the secret number) we need 6 questions.

We can represent a strategy as a tree (called a decision tree) where each node represents a question, taken from $\mathcal{F}$, and the two edges branching out represent the outcome of the test. Therefore, such a tree represents the strategy where we start asking the question at the root, if the answer is yes we repeat starting at the root of the right subtree and if the answer is no we repeat starting at the root of the left subtree. In this process the subset in which the secret number can be reduces at each question until it becomes a singleton. When that happens we have a leaf labelled with the only possible secret number for which we would reach this leaf.

The height of the tree, i.e., the largest number of internal nodes on a path from the root to a leaf coincides with the worst case number of questions asked when we use this strategy.

Let us consider the following related problem:

Decision Tree

Input: A ground set $U = \{x_1, \ldots x_n\}$ and a family of binary test $\mathcal{F} = \{T_1, \ldots, T_k\}$, and a parameter h.
Output: *yes* iff there is a strategy that guarantees to identify a secret number chosen from U with at most h questions.

Show that Decision Tree is $\mathcal{NP}$-complete by reducing from Set Cover (see exercise 9 from Set 5).

7. Let us consider the following problem:

   Traveling Salesman Problem (TSP)

   Input: An edge-weighted complete directed graph $G = (V, E)$ and a parameter $k \geq 0$.
   Complete and edge weighted means that for any $u, v \in V$ there exists a directed edge $e = u \to v$ and the edge $e \in E$ has an associated non-negative weight or length or distance $\ell(e)$.
   Output: *yes* iff there exists a tour that touches each vertex exactly once and such that the sum of the weights of the edges traversed is $\leq k$.

   Show that TSP is $\mathcal{NP}$-complete by reducing from Hamiltonian Cycle.

   In the definition of Traveling Salesman Problem there is no constraint on the "distances". Let $d_{u,v}$ denote the cost of using the edge from vertex u to vertex v. The metric variant of the TSP is defined by adding the following additional constraints:

   - $d_{u,v} = d_{v,u}$ (symmetry) i.e., the cost of going from vertex u to vertex v is the same as the cost of going from vertex v to vertex u.
   - $d_{u,v} \leq d_{u,w} + d_{w,v}$ (triangular inequality), i.e., if instead of going directly from u to v we first go to w, then the cost of the resulting tour cannot decrease.

   Show that this variant of the problem is also $\mathcal{NP}$-complete.