# Esercizi1

## Es. 1

### Lemma

$INDIPENDENT\_SET \leq_p VERTEX\_COVER$

### Dimostrazione

1. Abbiamo un **oracolo** per $INDIPENDENT\_SET$
2. Consideriamo un'istanza di $VETREX\_COVER$
   $G = (V, E), k$
3. Trasformare questa istanza in una equivalente istanza di $INDIPENDENT\_SET$
   $G' = (V', E'), k'$
4. $G' = G$
   $k' = n - k$
   $n = |V|$

## Es. 2

Dato un universo $U$ di elementi e una collezione $S_1, S_2, \cdots, S_n$ di sottoinsiemi di $U$ e un intero $k$, esiste una collezione di almeno $k$ di questi sottoinsiemi $t.c.$ nessuno di loro si interseca con un altro?

$SET\_PACKING$ è una generalizzazione di $INDIPENDENT\_SET$.

$INDIPENDENT\_SET \leq_p SET\_PACKING$.

Esercizio: dimostralo.

1. Istanza di $INDIPENDENT\_SET$
   $G = (V, E), k$
2. Fornire un'istanza equivalente di $SET\_PACKING$
   $U = E$
   $S_i =$ (unione degli archi incidenti su $v_i$)
   $k' = k$
3. Sia $S$ un sottoinsieme indipendente $|S| \geq k$
   $z \geq k$
   $S = v_1, v_2, \cdots, v_z$
4. $S_1 \cap S_2 \cdots S_z = \emptyset$

L'unione è $U$ ma l'intersezione è $\emptyset$

$\Leftarrow$ Assumiamo che esista una collezione di sottoinsiemi che non si intersecano $U$
$(S_1, S_2, \cdots, S_m)$
$m \geq k' \geq k$

$\Rightarrow S = v_1, v_2, \cdots, v_m = S_1, S_2, \cdots, S_m = U$

# Es. 3

1. Sia $G = (V, E)$ un grafo. Un'istanza del problema Clique può essere rappresentato come segue:
   Rappresentiamo ogni vertice come $v_i \in V$.
   $\forall v_i \in V$ allochiamo $n$ bit t.c. $n = |V|$.
   Sia $0 \leq j \leq n$
   Ogni bit di $v_i$ indica se $\exists v_j | (v_i, v_j) \in E$. Se è vero allora quel bit vale 1, 0 altrimenti.
2. Il parametro più significativo è il numero di nodi $n$ in quanto ci vogliono $n^2$ nodi per rappresentare il problema in binario.
3. Dato un grafo $G = (V, E)$ ed un intero $k$ stabilire se è presente all'interno del grafo una Clique composta da almeno $k$ nodi.
4. Il problema decisionale è una specializzazione del problema Clique, quindi va bene la stessa rappresentazione. Bisogna aggiungere il parametro k

# Es. 4

Un algoritmo $V$ è un verificatore efficiente per un problema $X$ se

1. $V$ è un algoritmo polinomiale che prende in input due stringhe, $s$ e $c$
2. Esiste una funzione polinomiale $p()$ tale che per ogni stringa $s$, si ha che
   $s \in X \iff \exists c \mid |c| \leq p(|s|)$ e $V(s, c) = si$.

$V$ deve controllare se $c$ è una soluzione al problema decisionale $X$ per una determinata codifica $s$ (l'input) in tempo polinomiale

# Es. 5

Supponiamo l'esistenza di un oracolo $O$ in grado di risolvere $3 - SAT$.

Trasformiamo l'istanza del problema $SAT$ in un'istanza valida del problema $3 - SAT$.

L'istanza di $SAT$ è composta da $k$ clausole $C$ e $n$ letterali $X$. Il problema $3 - SAT$ è composto da clausole composte da esattamente $3$ letterali distinti.

Se una clausola del problema $SAT$ è composto da meno di $3$ letterali, allora dovremo aggiungere delle variabili $z_1, z_2, z_3, z_4$ t.c.

1. $z_1$ e $z_2$ verranno utilizzate per riempire la clausola che assumeranno valore falso $(0)$
2. $z_3$ e $z_4$ renderanno $z_1$ e $z_2$ sempre false.

Se la clausola è composta da $3$ letterali, lasciala invariata.

Se una clausola del problema $SAT$ è composto da più di $3$ letterali, allora sia $k$ il numero di letterali che compongono la clausola $C = l_1 \vee, \dots, \vee l_k$.

Inizializziamo $k-3$ nuove variabili e sostituiamo $C$ con $k-3$ nuove clausole.

$(l_1 \vee l_2 \vee z_1), (l_3 \vee \neg z_1 \vee z_2), \ldots, (l_{k-2} \vee \neg z_{k-4} \vee z_{k-3}), (l_{k-1} \vee l_k \vee$

Se $C$ è vera allora è possibile assegnare valori alle variabili $z$ in modo che le clausole così ottenute siano tutte vere.
Sia $l_i$ il primo letterale per cui $C$ diventa vera. A questo punto basta impostare
$z_j = True, j < i$
$z_j = False, j \geq i$

Se $C$ è falsa allora non è possibile ottenere un assegnamento alle variabili $z$ t.c. $C$ sia vera.
Se $C$ è falsa $\Rightarrow$ tutti i letterali hanno valore $False$.
Affinché tutte le clausole siano vere anche le ultime due devono essere vere ma questo può accadere solo se nella clausola precedente $z_{k-4}$ è $False$. A questo punto si deve retrocedere fino alla prima clausola in cui per essere vera $z_1$ deve avere valore $True$ ma per costruzione alla seconda clausola gli sarà assegnato il valore $False$.

La costruzione può essere fatta in tempo polinomiale rispetto al numero di letterali presenti in ogni clausola.

A questo punto basta eseguire l'oracolo $O$ sul problema del $3-SAT$ e il risultato fornito dall'oracolo sarà l'output del problema $SAT$.

https://cse.iitkgp.ac.in/~palash/2018AlgoDesignAnalysis/SAT-3SAT.pdf

# Es. 6

I problemi in NP sono quei problemi per i quali è facile fornire un verificatore che ciò che viene presentato come una soluzione del problema è effettivamente una soluzione.

algoritmo $V$ è un verificatore efficiente per un problema $X$ se

- $V$ è un algoritmo polinomiale che prende in input due stringhe, $s$ e $c$
- esiste una funzione polinomiale $p()$ tale che per ogni stringa $s$, si ha che $s \in X$ se e solo se esiste una stringa $c\ t.c.$
  - $|c| \leq p(|s|)$
    $V(s, c) =$ si.

Siano $G$ e $G'$ due grafi cui vogliamo dimostrare l'isomorfismo $t.c.$

- $G = (V, E)$
- $G' = (V', E')$
  Siano $V = v_1, v_2, \ldots, v_n$ n nodi del grafo $G$.
  Siano $V' = w_1, w_2, \ldots, w_m$ m nodi del grafo $G'$.
  Costruiamo un Verificatore $V\ t.c.$
- $s$ è la codifica dei grafi $G$ e $G'$.

- $c$ è il certificato e contiene una permutazione corretta $\pi$ del grafo $G'$ $t.c.$ $G$ è isomorfo a $G'$.
- Crea un associazione tra i nodi es. $w_i$ -> $\pi(v_i)$
- Se tutti gli archi di $G$ trovano una corrispondenza in $G'$ e viceversa, allora il grafo è isomorfo
  - $(\pi(u), \pi(v)) \in E' \leftrightarrow (u, v) \in E$

Il Verificatore avrà tempo polinomiale in base al numero di archi.

# Es. 7

Un numero polinomiale di volte altrimenti sarebbe un problema in NP.

# Es. 8

- Indipendente:
  (1,3,5,7,10)
- Ricoprente:
  (2,4,6,8,9)

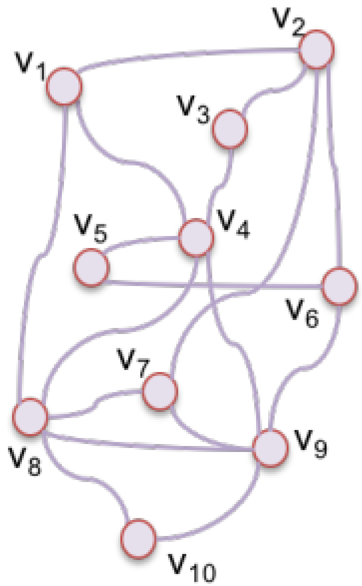


Figura 1.21: Grafo Esercizio 8

# Es. 9

La Figura 1.6 mostra un esempio con $n = 9$, $U = \{a, b, c, d, e, f, g, h, i\}$, $m = 7$ e $S_1 = \{a, b, c\}$, $S_2 = \{d, e\}$, $S_3 = \{g, h, i\}$, $S_4 = \{d, f, g, i\}$, $S_5 = \{d, f\}$, $S_6 = \{a, d, f\}$, e $S_7 = \{e, c, h\}$. Un insieme ricoprente è $\{S_1, S_4, S_7\}$.

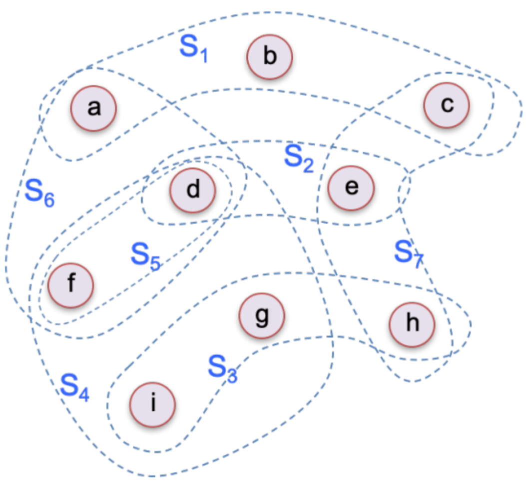


Figura 1.6: Grafo di esempio per SetCover

$S = \{S_1, S_4, S_7\}$ è una soluzione per SetCover.

Allora la soluzione $V/S = \{S_2, S_3, S_5, S_6\}$ dovrebbe essere una soluzione per SetPacking.

Però $S_5$ e $S_2$ condividono un nodo e questo va contro la definizione dii SetPacking.

# Es. 10

$4SAT \leq_p INDIPENTENTSET$
bisogna ridurre clique a indipendent_set

1. Abbiamo un oracolo che risolve il problema $INDIPENTENTSET$.
2. Consideriamo un'istanza di $4SAT$.
3. Trasformiamo tale istanza in una equivalente per il problema $INDIPENTENTSET$.

Sia $\phi$ una formula Booleana data in input per il problema $4SAT$.
Possiamo trasformare il problema in input in uno per $INDIPENTENTSET$ costruendo un un grafo $G = (V, E)$ t.c.

- $V$ = L'insieme di tutti i letterali di tutte le clausole.
- $E$ = L'insieme degli archi è costruito in modo che se due letterali sono nella stessa clausola, allora esiste un arco che li connette. In più ogni letterale $l_i$ è collegato ad ogni altro letterale $-l_i$ delle altre clausole.

E' possibile costruire l'IndipendentSet se:

- Solo un letterale per ogni clausola è all'interno dell'IndipendentSet
- Non è possibile avere un letterale e lo stesso complementato all'interno dell'IndipendentSet.

# Es. 11

- 

# Es. 12

- https://www.cs.tau.ac.il/~bchor/CM09/Compute14.pdf

# Es. 13

$Bipartito \iff 2 - GraphColoring$

$\rightarrow$ Bisogna stabilire se un grafo $G = (V, E)$ è $2$-colorabile se $G$ è bipartito.
Per il problema del GraphColoring:
$\forall (u, v) \in E$, $f(u) \neq f(v)$

Sia $V = V_1 \cup V_2$

Siano $u \in V_1$ e $v \in V_2$ t.c. $(u, v) \in E$. Per definizione $f(u) \neq f(v)$
Basta assegnare quindi a $u$ e $v$ due colori diversi.

Consideriamo ora $u' \in V_1$ e $v \in V_2$ t.c. $(u', v) \in E$ e $u \neq u'$
Siccome $u$ e $u'$ sono in $V_1$, per definizione non ci sarà

nessun arco che li può collegare qualsiasi siano i due nodi. Se così non fosse e $(u, u') \in E$ allora avremo che $f(u) \neq f(v) \neq f(u')$ il che è impossibile perché abbiamo solo due colori a disposizione. Per questo l'unico vincolo è dato da $f(v)$. Avremo inoltre che i nodi di $V_1$ avranno un colore e quelli di $V_2$ un altro.

$\leftarrow$ Se su un grafo $G = (V, E)$ è possibile eseguire l'algoritmo di $2 - GraphColoring$ allora significa che avremo due insiemi di nodi $V_1$ e $V_2$ t.c. i nodi di ciascun insieme sono colorati allo stesso modo. A questo punto però siccome i nodi in $V_1$ e $V_2$ non hanno archi tra di loro proprio perché abbiamo assunto in grafo $2 - Colorabile$ ma questa è proprio la definizione di grafo bipartito.

Per verificare se un grafo è bipartito basta controllare per ogni arco che un nodo faccia parte di $V_1$ e l'altro di $V_2$ o viceversa. (Oppure eseguire l'algoritmo di 2-GraphColoring)

# Es. 14

Perchè nella costruzione se esiste un ciclo hamiltoniano $C$ allora se $C$ passa per $c_j$ arrivando da $v_{i,3j}$ deve necessariamente lasciare $c_j$ andando a $v_{i,3j+1}$, altrimenti non potrebbe più visitare il nodo di transito $v_{i,3j+2}$. In modo simmetrico, se visita $c_j$ provenendo da $v_{i,3j+1}$ dovrà lasciarlo andando a $v_{i,3j}$, altrimenti il nodo di transito $v_{i,3j-1}$ non potrà più essere visitato. In altre parole l’unico modo per visitare i nodi $c_j$ è quello di seguire le deviazioni costruite sui cammini (e per questo il nodo di transito svolge un ruolo importante)