

6. Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Considerate il seguente problema, che chiameremo HAM375: dato un grafo G con n vertici, trovare un ciclo che attraversa esattamente una volta $n - 375$ vertici del grafo (ossia tutti i vertici di G tranne 375).

- (a) Dimostrare che il problema HAM375 è in NP fornendo un certificato per il Sì che si può verificare in tempo polinomiale.

Sia n il numero di vertici del grafo che è istanza di HAM375. Un certificato per HAM375 è una sequenza ordinata di $n - 375$ vertici distinti. Occorre tempo polinomiale per verificare se ogni nodo è collegato al successivo e l'ultimo al primo.

- (b) Mostrare come si può risolvere il problema del circuito Hamiltoniano usando il problema HAM375 come sottoprocedura.

Dato un qualsiasi grafo G che è un'istanza del problema del circuito Hamiltoniano, costruiamo in tempo polinomiale un nuovo grafo G' che è un'istanza di HAM375, aggiungendo a G 375 vertici isolati (senza archi). Chiaramente G' ha un ciclo che attraversa esattamente una volta $n - 375$ vertici del grafo se e solo se G ha un ciclo Hamiltoniano.

5. “Colorare” i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate “colori”, ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. Il problema k COLOR è il problema di trovare una colorazione di un grafo non orientato usando k colori diversi.

- (a) Dimostrare che il problema 4COLOR (colorare un grafo con 4 colori) è in NP fornendo un certificato per il Sì che si può verificare in tempo polinomiale.

- (b) Mostrare come si può risolvere il problema 3COLOR (colorare un grafo con 4 colori) usando 4COLOR come sottoprocedura.

- (c) Per quali valori di k il problema k COLOR è NP-completo?

- Per nessun valore: k COLOR è un problema in P
- Per tutti i $k \geq 3$
- Per tutti i valori di k

Le risposte seguono:

- a) se i vertici del grafo sono numerati da 1 a n , allora una sequenza di lunghezza n dei 4 colori disponibili, dove il colore in posizione i della sequenza è associato al vertice i , è un certificato. Per verificare che la colorazione corrispondente al certificato ha risposta SI, basta verificare che il vertice i abbia colore diverso da tutti i vertici a cui è collegato e questa operazione è lineare nella taglia del grafo, visto che basta esaminare ogni arco del grafo 2 volte: una per ciascuno dei 2 vertici collegati dall'arco.

- b) Si riduce 3COLOR a 4COLOR come segue: dato un qualsiasi grafo G , istanza di 3COLOR, si aggiunge a G un vertice collegato a tutti i vertici di G . Il grafo G' così ottenuto è un'istanza di 4COLOR e infatti G è colorabile con 3 colori sse G' lo è con 4 colori. Per cui 4COLOR è almeno altrettanto intrattabile di 3COLOR.

- c) I problemi k COLOR con $k \geq 3$ sono NP-completi.

6. Il problema SETPARTITIONING chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . Sappiamo che questo problema è NP-completo.

Considerate il seguente problema, che chiameremo SUBSETSUM: dato un insieme di numeri interi S ed un valore obiettivo t , stabilire se esiste un sottoinsieme $S' \subseteq S$ tale che la somma dei numeri in S' è uguale a t .

- (a) Dimostrare che il problema SUBSETSUM è in NP fornendo un certificato per il Sì che si può verificare in tempo polinomiale.

Il certificato per SUBSETSUM è dato dal sottoinsieme S' . Occorre verificare che ogni elemento di S' appartenga anche ad S e che la somma dei numeri contenuti in S' sia uguale a t .

- (b) Mostrare come si può risolvere il problema SETPARTITIONING usando il problema SUBSETSUM come sottoprocedura.

Dato un qualsiasi insieme di numeri interi S che è un'istanza di SETPARTITIONING, costruiamo in tempo polinomiale un'istanza di SUBSETSUM. L'insieme di numeri interi di input rimane S , mentre il valore obiettivo t è uguale alla metà della somma degli elementi contenuti in S .

In questo modo, se S' è una soluzione di SUBSETSUM, allora la somma dei valori contenuti nell'insieme $S - S'$ sarà pari alla metà della somma degli elementi di S . Quindi ho diviso S in sue sottoinsiemi $S_1 = S'$ e $S_2 = S - S'$ tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 .

Viceversa, se S_1 e S_2 sono una soluzione di SETPARTITIONING, allora la somma dei numeri contenuti in S_1 sarà uguale alla somma dei numeri in S_2 , e quindi uguale alla metà della somma dei numeri contenuti in S . Quindi sia S_1 che S_2 sono soluzione di SUBSETSUM per il valore obiettivo t specificato sopra.

5. Il problema SETPARTITIONING chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . Sappiamo che questo problema è NP-completo.

Considerate la seguente variante del problema, che chiameremo QUASIPARTITIONING: dato un insieme di numeri interi S , stabilire se può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 meno 1.

- (a) Dimostrare che il problema QUASIPARTITIONING è in NP fornendo un certificato per il Sì che si può verificare in tempo polinomiale.

Il certificato è dato da una coppia di insiemi di numeri interi S_1, S_2 . Per verificarlo occorre controllare che rispetti le seguenti condizioni:

- i due insiemi S_1, S_2 devono essere una partizione dell'insieme S ;
- la somma degli elementi in S_1 deve essere uguale alla somma degli elementi in S_2 meno 1.

Entrambe le condizioni si possono verificare in tempo polinomiale.

- (b) Dimostrare che il problema QUASIPARTITIONING è NP-hard, mostrando come si può risolvere il problema SETPARTITIONING usando il problema QUASIPARTITIONING come sottoprocedura.

Dimostrare che QUASIPARTITIONING è NP-hard, usando SETPARTITIONING come problema di riferimento richiede diversi passaggi:

1. Descrivere un algoritmo per risolvere SETPARTITIONING usando QUASIPARTITIONING come subroutine. Questo algoritmo avrà la seguente forma: data un'istanza di SETPARTITIONING, trasformala in un'istanza di QUASIPARTITIONING, quindi chiama l'algoritmo magico black-box per QUASIPARTITIONING.
2. Dimostrare che la riduzione è corretta. Ciò richiede sempre due passaggi separati, che di solito hanno la seguente forma:
 - Dimostrare che l'algoritmo trasforma istanze “buone” di SETPARTITIONING in istanze “buone” di QUASIPARTITIONING.
 - Dimostrare che se la trasformazione produce un'istanza “buona” di QUASIPARTITIONING, allora era partita da un'istanza “buona” di SETPARTITIONING.
3. Mostrare che la riduzione funziona in tempo polinomiale, a meno della chiamata (o delle chiamate) all'algoritmo magico black-box per QUASIPARTITIONING. (Questo di solito è banale.)

Una istanza di SETPARTITIONING è data da un insieme S di numeri interi da suddividere in due. Una istanza di QUASIPARTITIONING è data anch'essa da un insieme di numeri interi S' . Quindi la riduzione deve trasformare un insieme di numeri S input di SETPARTITIONING in un altro insieme di numeri S' che diventerà l'input per la black-box che risolve QUASIPARTITIONING.

Come primo tentativo usiamo una riduzione che crea S' aggiungendo un nuovo elemento a S di valore 1, e proviamo a dimostrare che la riduzione è corretta:

\Rightarrow sia S un'istanza buona di SETPARTITIONING. Allora è possibile partizionare S in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . Se aggiungiamo il nuovo valore 1 ad S_1 otteniamo una soluzione per QUASIPARTITIONING, e abbiamo dimostrato che S' è una istanza buona di QUASIPARTITIONING.

\Leftarrow sia S' un'istanza buona di QUASIPARTITIONING. Allora è possibile partizionare S' in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 meno 1. Controlliamo quale dei due sottoinsiemi contiene il nuovo elemento 1 aggiunto dalla riduzione:

- se $1 \in S_1$, allora se tolgo 1 da S_1 la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2 . Abbiamo trovato una soluzione per SETPARTITIONING con input S .
- se $1 \in S_2$, allora se tolgo 1 da S_2 quello che succede è che la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2 meno 2. In questo caso abbiamo un

problema perché quello che otteniamo *non è una soluzione di SETPARTITIONING!*

Quindi il primo tentativo di riduzione non funziona: ci sono dei casi in cui istanze cattive di SETPARTITIONING diventano istanze buone di QUASIPARTITIONING: per esempio, l'insieme $S = \{2, 4\}$, che non ha soluzione per SETPARTITIONING, diventa $S' = \{2, 4, 1\}$ dopo l'aggiunta dell'1, che ha soluzione per QUASIPARTITIONING: basta dividerlo in $S_1 = \{4\}$ e $S_2 = \{2, 1\}$.

Dobbiamo quindi trovare un modo per “forzare” l'elemento 1 aggiuntivo ad appartenere ad S_1 nella soluzione di QUASIPARTITIONING. Per far questo basta modificare la riduzione in modo che S' contenga tutti gli elementi di S *moltiplicati per 3*, oltre all'1 aggiuntivo. Formalmente:

$$S' = \{3x \mid x \in S\} \cup \{1\}.$$

Proviamo a dimostrare che la nuova riduzione è corretta:

\Rightarrow sia S un'istanza buona di SETPARTITIONING. Allora è possibile partizionare S in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . Se moltiplichiamo per 3 gli elementi di S_1 ed S_2 , ed aggiungiamo il nuovo valore 1 ad S_1 otteniamo una soluzione per QUASIPARTITIONING, e abbiamo dimostrato che S' è una istanza buona di QUASIPARTITIONING.

\Leftarrow sia S' un'istanza buona di QUASIPARTITIONING. Allora è possibile partizionare S' in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 meno 1. Vediamo adesso che, a differenza della riduzione precedente, non è possibile che $1 \in S_2$: se così fosse, allora se tolgo 1 da S_2 quello che succede è che la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2 meno 2. Tuttavia, gli elementi che stanno in S_1 ed S_2 sono tutti quanti multipli di 3 (tranne l'1 aggiuntivo). Non è possibile che due insiemi che contengono solo multipli di 3 abbiano differenza 2. Quindi l'1 aggiuntivo non può appartenere a S_2 e deve appartenere per forza a S_1 . Come visto prima, se tolgo 1 da S_1 la somma degli elementi S_1 diventa uguale alla somma dei numeri in S_2 . Abbiamo trovato una soluzione per SETPARTITIONING con input S .

In questo caso la riduzione è corretta. Per completare la dimostrazione basta osservare che per costruire S' dobbiamo moltiplicare per 3 gli n elementi di S ed aggiungere un nuovo elemento. Tutte operazioni che si fanno in tempo polinomiale.

5. Il problema SETPARTITIONING chiede di stabilire se un insieme di numeri interi S può essere suddiviso in due sottoinsiemi disgiunti S_1 e S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . Sappiamo che questo problema è NP-completo.

Considerate la seguente variante del problema, chiamata 3WAYPARTITIONING: stabilire se un insieme di numeri interi S può essere suddiviso in tre sottoinsiemi disgiunti S_1, S_2 e S_3 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 che è uguale alla somma dei numeri in S_3 .

- Dimostrare che il problema 3WAYPARTITIONING è in NP fornendo un certificato per il *Sì* che si può verificare in tempo polinomiale.
- Dimostrare che il problema 3WAYPARTITIONING è NP-hard, mostrando come si può risolvere il problema SETPARTITIONING usando il problema 3WAYPARTITIONING come sottoprocedura.

- (a) Il certificato è dato da tre insiemi di numeri interi S_1, S_2, S_3 . Per verificarlo occorre controllare che rispetti le seguenti condizioni:
- i tre insiemi devono essere una partizione dell'insieme S ;
 - la somma degli elementi in S_1 deve essere uguale alla somma degli elementi in S_2 che deve essere uguale alla somma dei numeri in S_3 .
- (b) Dimostrare che 3WAYPARTITIONING è NP-hard, usando SETPARTITIONING come problema di riferimento richiede diversi passaggi:
1. Descrivere un algoritmo per risolvere SETPARTITIONING usando 3WAYPARTITIONING come subroutine. Questo algoritmo avrà la seguente forma: data un'istanza di SETPARTITIONING, trasformala in un'istanza di 3WAYPARTITIONING, quindi chiama l'algoritmo magico black-box per 3WAYPARTITIONING.
 2. Dimostrare che la riduzione è corretta. Ciò richiede sempre due passaggi separati, che di solito hanno la seguente forma:
 - Dimostrare che l'algoritmo trasforma istanze “buone” di SETPARTITIONING in istanze “buone” di 3WAYPARTITIONING.

- Dimostrare che se la trasformazione produce un'istanza “buona” di 3WAYPARTITIONING, allora era partita da un'istanza “buona” di SETPARTITIONING.
3. Mostrare che la riduzione funziona in tempo polinomiale, a meno della chiamata (o delle chiamate) all'algoritmo magico black-box per 3WAYPARTITIONING. (Questo di solito è banale.)

Una istanza di SETPARTITIONING è data da un insieme S di numeri interi da suddividere in due. Una istanza di 3WAYPARTITIONING è data anch'essa da un insieme di numeri interi S' . Quindi la riduzione deve trasformare un insieme di numeri S input di SETPARTITIONING in un altro insieme di numeri S' che diventerà l'input per la black-box che risolve 3WAYPARTITIONING.

Se $t = \sum_{x \in S} x$ è la somma dei numeri che appartengono ad S , la riduzione che crea S' aggiungendo un nuovo elemento a S di valore $t/2$ (metà della somma dei numeri che appartengono ad S). Dimostriamo che è corretta:

- \Rightarrow sia S un'istanza buona di SETPARTITIONING. Allora è possibile partizionare S in due sottoinsiemi S_1 ed S_2 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 . In particolare, sia S_1 che S_2 sommano a $t/2$. Se aggiungiamo il nuovo valore $t/2$ ad S_1 otteniamo una soluzione per 3WAYPARTITIONING, in cui i tre insiemi sono S_1, S_2 e $S_3 = \{t/2\}$, e abbiamo dimostrato che S' è una istanza buona di 3WAYPARTITIONING.
- \Leftarrow sia S' un'istanza buona di 3WAYPARTITIONING. Allora è possibile partizionare S' in tre sottoinsiemi S_1, S_2 ed S_3 tali che la somma dei numeri in S_1 è uguale alla somma dei numeri in S_2 che è uguale alla somma dei numeri in S_3 . Per come è definito S' , abbiamo che ognuno dei tre insiemi somma a $t/2$. Di conseguenza uno dei tre insiemi, che possiamo chiamare S_3 , contiene solamente il nuovo elemento $t/2$, mentre gli altri due, che chiamiamo S_1 e S_2 , formano una partizione degli elementi dell'insieme S di partenza. Di conseguenza, abbiamo trovato una soluzione per SETPARTITIONING con input S .

Per completare la dimostrazione basta osservare che per costruire S' dobbiamo aggiungere un nuovo elemento ad S , operazione che si riesce a fare in tempo polinomiale.

5. Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Un *circuito 1/3-Hamiltoniano* in un grafo G è un ciclo che attraversa esattamente una volta *un terzo* dei vertici del grafo. Il *problema del circuito 1/3-Hamiltoniano* è il problema di stabilire se un grafo contiene un circuito 1/3-Hamiltoniano.

- (a) Dimostrare che il problema del circuito 1/3-Hamiltoniano è in NP fornendo un certificato per il *Sì* che si può verificare in tempo polinomiale.
(b) Mostrare come si può risolvere il problema del circuito Hamiltoniano usando il problema del circuito 1/3-Hamiltoniano come sottoprocedura.

Per la parte (a) un certificato è una sequenza di $n/3$ nodi. E' facile (lineare) verificare se un tale sequenza di nodi forma un circuito hamiltoniano o no. Per la parte (b), mappiamo una qualsiasi istanza del problema del circuito Hamiltoniano che è formata da un grafo G in un'istanza del 1/3-circuito Hamiltoniano, composta da 3 copie di G completamente disgiunte tra loro. E' facile vedere che se il grafo con 3 copie di G ha un circuito Hamiltoniano su 1/3 dei nodi, lo ha su una delle 3 componenti e quindi su G . Se invece il grafo con 3 copie di G non ha un circuito Hamiltoniano su 1/3 dei nodi, allora non c'è circuito Hamiltoniano su G . Quindi abbiamo ridotto il problema del circuito Hamiltoniano a quello del 1/3 Hamiltoniano, dimostrando, assieme al punto (a), che esso è NP completo.

- 3.** Sia G un grafo non orientato, e si consideri il seguente problema¹:

$$L\text{PATH} = \{\langle G, s, t, k \rangle \mid G \text{ contiene un cammino semplice di lunghezza almeno } k \text{ da } s \text{ a } t\}$$

Un cammino semplice è un cammino nel grafo senza ripetizioni di vertici, con la possibile eccezione dei vertici iniziale e finale che possono coincidere. La sua lunghezza è data dal numero di archi che lo compongono.

Mostrare che $L\text{PATH}$ è NP-completo, usando il problema del circuito Hamiltoniano come problema NP-hard di riferimento.

¹ $L\text{PATH}$ è la *versione decisionale* del problema del cammino massimo tra i nodi s e t , in cui la lunghezza del cammino k diventa uno degli input del problema. In questo modo si trasforma un problema di ottimizzazione dove l'output è la lunghezza del cammino in un problema di decisione con risposta binaria vero/falso.

- 3.** Per mostrare che $L\text{PATH}$ è NP-completo, dimostriamo prima che $L\text{PATH}$ è in NP, e poi che è NP-Hard.

- $L\text{PATH}$ è in NP. Il cammino da s a t di lunghezza maggiore o uguale a k è il certificato. Il seguente algoritmo è un verificatore per $L\text{PATH}$:

$V =$ "Su input $\langle\langle G, s, t, k \rangle, c\rangle$:
1. Controlla che c sia una sequenza di vertici di G , v_1, \dots, v_m e che m sia minore o uguale al numero di vertici in G più uno. Se non lo è, rifiuta.
2. Se la sequenza è di lunghezza minore o uguale a k , rifiuta.
3. Controlla se $s = v_1$ e $t = v_m$. Se una delle due è falsa, rifiuta.
4. Controlla se ci sono ripetizioni nella sequenza. Se ne trova una diversa da $v_1 = v_m$, rifiuta.
5. Per ogni i tra 1 e $m - 1$, controlla se (v_i, v_{i+1}) è un arco di G . Se non lo è rifiuta.
6. Se tutti i test sono stati superati, accetta."

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. Ognuna delle fasi è un controllo sugli m elementi del certificato, e quindi richiede un tempo polinomiale rispetto ad m . Poiché l'algoritmo rifiuta immediatamente quando m è maggiore del numero di vertici di G più uno, allora possiamo concludere che l'algoritmo richiede un tempo polinomiale rispetto al numero di vertici del grafo.

- Dimostriamo che *LPATH* è NP-Hard per riduzione polinomiale da *HAMILTON* a *LSPATH*. La funzione di riduzione polinomiale f prende in input un grafo $\langle G \rangle$ e produce come output la quadrupla $\langle G, s, s, n \rangle$ dove s è un vertice arbitrario di G e n è uguale al numero di vertici di G . Dimostriamo che la riduzione polinomiale è corretta:

- Se $\langle G \rangle \in HAMILTON$, allora esiste un circuito Hamiltoniano in G . Dato un qualsiasi vertice s di G , possiamo costruire un cammino che parte da s e segue il circuito Hamiltoniano per tornare in s . Questo cammino attraversa tutti gli altri vertici di G prima di tornare in s e sarà quindi di lunghezza n . Di conseguenza $\langle G, s, s, n \rangle \in LSPATH$.
- Se $\langle G, s, s, n \rangle \in LSPATH$, allora esiste un cammino semplice nel grafo G che inizia e termina in s ed è di lunghezza maggiore o uguale a n . Un cammino semplice non ha ripetizioni, ad eccezione dei vertici iniziali e finali. Un cammino semplice da s ad s di lunghezza n deve attraversare tutti gli altri nodi una sola volta prima di tornare in s , ed è quindi un circuito Hamiltoniano per G . Cammini semplici più lunghi non possono esistere perché dovrebbero ripetere dei vertici. Quindi l'esistenza di un cammino semplice nel grafo G che inizia e termina in s ed è di lunghezza maggiore o uguale a n implica l'esistenza di un circuito Hamiltoniano in G , ed abbiamo dimostrato che $\langle G \rangle \in HAMILTON$.

La funzione di riduzione si limita ad aggiungere tre nuovi elementi dopo la codifica del grafo G : due vertici ed un numero, operazione che si può fare in tempo polinomiale.

3. Considera il problema di pianificare la disposizione dei posti a sedere per un matrimonio con n invitati. Per disporre gli invitati hai a disposizione k tavoli, che possono ospitare un numero arbitrario di invitati. Alcuni degli invitati sono amici tra di loro, altri sono rivali ed altri sono indifferenti l'uno all'altro. Il tuo obiettivo è di trovare una disposizione degli n invitati sui k tavoli che rispetti i seguenti requisiti:

- ogni invitato siede ad un solo tavolo;
- due invitati che sono amici devono sedere allo stesso tavolo;
- due invitati che sono rivali devono sedere a tavoli diversi.

Possiamo rappresentare l'input del problema con una tripla $\langle n, k, R \rangle$ dove:

- n è il numero di invitati
- k è il numero di tavoli
- R è una matrice $n \times n$ che descrive le relazioni tra gli invitati:

$$R[i, j] = \begin{cases} 1 & \text{se gli ospiti } i \text{ e } j \text{ sono amici} \\ -1 & \text{se gli ospiti } i \text{ e } j \text{ sono rivali} \\ 0 & \text{se gli ospiti } i \text{ e } j \text{ sono indifferenti} \end{cases}$$

e definire il seguente linguaggio:

$$\text{WSP} = \{ \langle n, k, R \rangle \mid \text{esiste una disposizione di } n \text{ ospiti su } k \text{ tavoli} \\ \text{che rispetta le relazioni tra invitati } R \}$$

- (a) Dimostra che WSP è un problema NP
- (b) Dimostra che WSP è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.¹

3. (a) WSP è in NP. La disposizione degli ospiti tra i tavoli è il certificato. Se gli ospiti sono numerati da 1 a n la possiamo rappresentare con un vettore T tale che $T[i]$ è il tavolo assegnato all'ospite i . Il seguente algoritmo è un verificatore per WSP:

V = “Su input $\langle \langle n, k, R \rangle, T \rangle$:

1. Controlla che T sia un vettore di n elementi dove ogni elemento ha un valore compreso tra 1 e k . Se non lo è, rifiuta.
2. Per ogni coppia i, j tale che $R[i, j] = 1$, controlla che $T[i] = T[j]$. Se il controllo fallisce, rifiuta.
3. Per ogni coppia i, j tale che $R[i, j] = -1$, controlla che $T[i] \neq T[j]$. Se il controllo fallisce, rifiuta.
4. Se tutti i test sono stati superati, accetta.”

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. La prima fase è un controllo sugli n elementi del vettore T , e quindi richiede un tempo polinomiale rispetto alla dimensione dell'input. La seconda e terza fase controllano gli elementi della matrice R , operazione che si può fare in tempo polinomiale rispetto alla dimensione dell'input.

(b) Dimostriamo che WSP è NP-Hard per riduzione polinomiale da 3-COLOR a WSP. La funzione di riduzione polinomiale f prende in input un grafo $\langle G \rangle$ e produce come output la tripla $\langle n, 3, R \rangle$ dove n è il numero di vertici di G e la matrice R è definita in questo modo:

$$R[i, j] = \begin{cases} -1 & \text{se c'è un arco da } i \text{ a } j \text{ in } G \\ 0 & \text{altrimenti} \end{cases}$$

Dimostriamo che la riduzione polinomiale è corretta:

- Se $\langle G \rangle \in 3\text{-COLOR}$, allora esiste un modo per colorare G con tre colori 1, 2, 3. La disposizione degli ospiti che fa sedere l'ospite i nel tavolo corrispondente al colore del vertice i nel grafo è corretta:
 - ogni invitato siede ad un solo tavolo;
 - per ogni coppia di invitati rivali i, j si ha che $R[i, j] = -1$ e, per la definizione della funzione di riduzione, l'arco (i, j) appartiene a G . Quindi la colorazione assegna colori diversi ad i e j , ed i due ospiti siedono su tavoli diversi;
 - per la definizione della funzione di riduzione non ci sono ospiti che sono amici tra di loro.
- Se $\langle n, 3, R \rangle \in \text{WSP}$, allora esiste una disposizione degli n ospiti su 3 tavoli dove i rivali siedono sempre su tavoli diversi. La colorazione che assegna al vertice i il colore corrispondente al tavolo dove siede l'ospite i è corretta: se c'è un arco tra i e j allora i due ospiti sono rivali e siederanno su tavoli diversi, cioè avranno colori diversi nella colorazione. Di conseguenza abbiamo dimostrato che $\langle G \rangle \in 3\text{-COLOR}$.

La funzione di riduzione deve contare i vertici del grafo G e costruire la matrice R di dimensione $n \times n$, operazioni che si possono fare in tempo polinomiale.

4. Fornisci un verificatore polinomiale per il seguente problema:

$$\text{DOUBLEHAMCIRCUIT} = \{ \langle G \rangle \mid G \text{ è un grafo non orientato che contiene un ciclo che visita ogni vertice esattamente due volte e attraversa ogni arco esattamente una volta} \}$$

V = “su input $\langle G, C \rangle$, dove G è un grafo ed il certificato C è una sequenza di vertici (v_1, \dots, v_k) :

1. Controlla che ogni elemento di C sia un vertice del grafo;
2. controlla che C sia un ciclo ($v_1 = v_k$);
3. controlla che ogni vertice del grafo compaia 2 volte in C , tranne v_1 che deve comparire 3 volte;
4. controlla che (v_i, v_{i+1}) sia un arco del grafo per ogni $i = 1, \dots, k-1$;
5. controlla che ogni arco compaia in C esattamente una volta;
6. se tutti i test sono superati accetta, altrimenti rifiuta.”

4. Una 5-colorazione di un grafo non orientato G è una funzione che assegna a ciascun vertice di G un “colore” preso dall'insieme $\{0, 1, 2, 3, 4\}$, in modo tale che per qualsiasi arco $\{u, v\}$ i colori associati ai vertici u e v sono diversi. Una 5-colorazione è *accurata* se i colori assegnati ai vertici adiacenti sono distinti e con differenza maggiore di 1 *modulo* 5.

Fornisci un verificatore polinomiale per il seguente problema:

$$\text{CAREFUL5COLOR} = \{ \langle G \rangle \mid G \text{ è un grafo che ammette una 5-colorazione accurata} \}$$

Se i vertici sono numerati da 1 a n , il certificato è un vettore C tale che $C[i]$ è il colore del vertice i .

V = “Su input $\langle G, C \rangle$, dove G è un grafo e C un vettore:

1. Controlla che C sia un vettore di n elementi dove ogni elemento ha valore in $\{0, 1, 2, 3, 4\}$.
2. Controlla che per ogni arco (i, j) la differenza tra $C[i]$ e $C[j]$ sia maggiore di 1 modulo 5.
3. Se tutti i test sono superati accetta, altrimenti rifiuta.”

4. (8 punti) Supponiamo che un impianto industriale costituito da m linee di produzione identiche debba eseguire n lavori distinti. Ognuno dei lavori può essere svolto da una qualsiasi delle linee di produzione, e richiede un certo tempo per essere completato. Il problema del bilanciamento del carico (LOADBALANCE) chiede di trovare un assegnamento dei lavori alle linee di produzione che permetta di completare tutti i lavori entro un tempo limite k .

Più precisamente, possiamo rappresentare l'input del problema con una tripla $\langle m, T, k \rangle$ dove:

- m è il numero di linee di produzione;
- $T[1 \dots n]$ è un array di numeri interi positivi dove $T[j]$ è il tempo di esecuzione del lavoro j ;
- k è un limite superiore al tempo di completamento di tutti i lavori.

Per risolvere il problema vi si chiede di trovare un array $A[1 \dots n]$ con gli assegnamenti, dove $A[j] = i$ significa che il lavoro j è assegnato alla linea di produzione i . Il tempo di completamento (o makespan) di A è il tempo massimo di occupazione di una qualsiasi linea di produzione:

$$\text{makespan}(A) = \max_{1 \leq i \leq m} \sum_{A[j]=i} T[j]$$

LOAD BALANCE è il problema di trovare un assegnamento con makespan minore o uguale al limite superiore k :

$$\begin{aligned} \text{LOADBALANCE} = & \{ \langle m, T, k \rangle \mid \text{esiste un assegnamento } A \text{ degli } n \text{ lavori} \\ & \text{su } m \text{ linee di produzione tale che } \text{makespan}(A) \leq k \} \end{aligned}$$

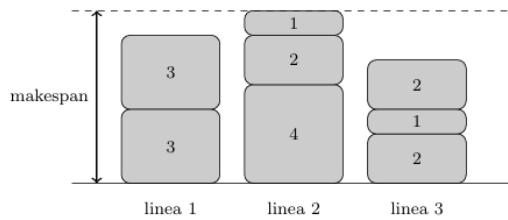


Figura 1: Esempio di assegnamento dei lavori $T = \{1, 1, 2, 2, 2, 3, 3, 4\}$ su 3 linee con makespan 7.

- (a) Dimostra che LOADBALANCE è un problema NP.
(b) Dimostra che LOADBALANCE è NP-hard, usando SETPARTITIONING come problema NP-hard di riferimento.

- (a) LOADBALANCE è in NP. L'array A con gli assegnamenti è il certificato. Il seguente algoritmo è un verificatore per LOADBALANCE:

V = "Su input $\langle \langle m, T, k \rangle, A \rangle$:

1. Controlla che A sia un vettore di n elementi dove ogni elemento ha un valore compreso tra 1 e m . Se non lo è, rifiuta.
2. Calcola $\text{makespan}(A)$: se è minore o uguale a k accetta, altrimenti rifiuta."

Per analizzare questo algoritmo e dimostrare che viene eseguito in tempo polinomiale, esaminiamo ogni sua fase. La prima fase è un controllo sugli n elementi del vettore A , e quindi richiede un tempo polinomiale rispetto alla dimensione dell'input. Per calcolare il makespan, la seconda fase deve calcolare il tempo di occupazione di ognuna delle m linee e poi trovare il massimo tra i tempi di occupazione, operazioni che si possono fare in tempo polinomiale rispetto alla dimensione dell'input.

- (b) Dimostriamo che LOADBALANCE è NP-Hard per riduzione polinomiale da SETPARTITIONING a LOADBALANCE. La funzione di riduzione polinomiale f prende in input un insieme di numeri interi positivi $\langle T \rangle$ e produce come output la tripla $\langle 2, T, k \rangle$ dove k è uguale alla metà della somma dei valori in T :

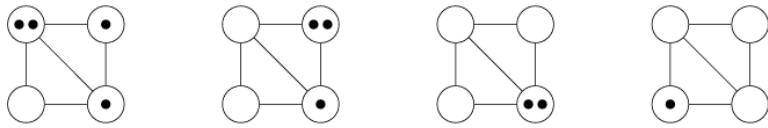
$$k = \frac{1}{2} \sum_{1 \leq i \leq n} T[i]$$

Dimostriamo che la riduzione polinomiale è corretta:

- Se $\langle T \rangle \in \text{SETPARTITIONING}$, allora esiste un modo per suddividere T in due sottoinsiemi T_1 e T_2 in modo tale che la somma dei valori contenuti in T_1 è uguale alla somma dei valori contenuti in T_2 . Nota che questa somma deve essere uguale alla metà della somma dei valori in T , cioè uguale a k . Quindi assegnando i lavori contenuti in T_1 alla prima linea di produzione e quelli contenuti in T_2 alla seconda linea di produzione otteniamo una soluzione per LOADBALANCE con makespan uguale a k , come richiesto dal problema.
- Se $\langle 2, T, k \rangle \in \text{LOADBALANCE}$, allora esiste un assegnamento dei lavori alle 2 linee di produzione con makespan minore o uguale a k . Siccome ci sono solo 2 linee, il makespan di questa soluzione non può essere minore della metà della somma dei valori in T , cioè di k . Quindi l'assegnamento ha makespan esattamente uguale a k , ed entrambe le linee di produzione hanno tempo di occupazione uguale a k . Quindi, inserendo i lavori assegnati alla prima linea in T_1 e quelli assegnati alla seconda linea in T_2 otteniamo una soluzione per SETPARTITIONING.

La funzione di riduzione deve sommare i valori in T e dividere per due, operazioni che si possono fare in tempo polinomiale.

4. Pebbling è un solitario giocato su un grafo non orientato G , in cui ogni vertice ha zero o più ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice v e aggiungere un ciottolo ad un vertice u adiacente a v (il vertice v deve avere almeno due ciottoli all'inizio della mossa). Il problema PEBBLEDESTRUCTION chiede, dato un grafo $G = (V, E)$ ed un numero di ciottoli $p(v)$ per ogni vertice v , di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.



Una soluzione in 3 mosse di PEBBLEDESTRUCTION.

Fornisci un verificatore per PEBBLEDESTRUCTION.

Un verificatore per PebbleDestruction può essere implementato come segue:

$V = \text{"Su input } \langle G, p, m \rangle, \text{ dove } G \text{ è un grafo, } p \text{ è una funzione che assegna ciottoli ai vertici, e } m \text{ è una sequenza di mosse:"}$

1. Verifica che G sia un grafo non orientato valido.
2. Verifica che $p(v) \geq 0$ per ogni vertice v .
3. Per ogni mossa in m : a. Verifica che il vertice di partenza abbia almeno 2 ciottoli. b. Verifica che il vertice di arrivo sia adiacente al vertice di partenza. c. Rimuovi 2 ciottoli dal vertice di partenza e aggiungi 1 ciottolo al vertice di arrivo.
4. Alla fine, verifica che tutti i vertici tranne uno abbiano 0 ciottoli.
5. Se tutte le verifiche sono passate, accetta. Altrimenti, rifiuta."

Questo verificatore opera in tempo polinomiale rispetto alla dimensione dell'input e verifica correttamente se una data sequenza di mosse risolve il problema PebbleDestruction.

4. (8 punti) "Colorare" i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate "colori", ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. Considera la seguente variante del problema 4-COLOR. Oltre al grafo G , l'input del problema comprende anche un *colore proibito* f_v per ogni vertice v del grafo. Per esempio, il vertice 1 non può essere rosso, il vertice 2 non può essere verde, e così via. Il problema che dobbiamo risolvere è stabilire se possiamo colorare il grafo G con 4 colori in modo che nessun vertice sia colorato con il colore proibito.

$\text{CONSTRAINED-4-COLOR} = \{\langle G, f_1, \dots, f_n \rangle \mid \text{esiste una colorazione } c_1, \dots, c_n \text{ degli } n \text{ vertici}$
 $\text{tale che } c_v \neq f_v \text{ per ogni vertice } v\}$

- Dimostra che CONSTRAINED-4-COLOR è un problema NP.
- Dimostra che CONSTRAINED-4-COLOR è NP-hard, scegliendo un opportuno valore di k e usando k -COLOR come problema NP-hard di riferimento.

4. Per Constrained-4-Color:

a) Dimostriamo che è in NP: Un verificatore V per Constrained-4-Color opera come segue: $V = \text{"Su input } \langle G, f_1, \dots, f_n, c_1, \dots, c_n \rangle:$

1. Verifica che $c_i \in \{1, 2, 3, 4\}$ e $c_i \neq f_i$ per ogni i .
2. Per ogni arco (u, v) in G , verifica che $c_u \neq c_v$.
3. Se tutte le verifiche passano, accetta. Altrimenti, rifiuta."

Questo verificatore opera in tempo polinomiale, quindi Constrained-4-Color è in NP.

b) Dimostriamo che è NP-hard: Riduciamo 3-Color a Constrained-4-Color. Data un'istanza G di 3-Color, costruiamo un'istanza G' di Constrained-4-Color come segue:

- G' ha gli stessi vertici e archi di G
- Per ogni vertice v in G' , fissiamo $f_v = 4$

Questa riduzione è corretta perché:

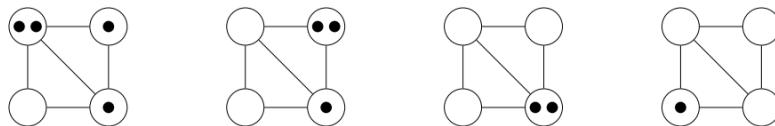
- Se G è 3-colorabile, allora G' è colorabile con 3 colori, rispettando il vincolo $f_v = 4$
- Se G' è colorabile rispettando i vincoli, allora G è 3-colorabile

La riduzione è chiaramente calcolabile in tempo polinomiale.

Quindi, Constrained-4-Color è NP-hard.

Combinando (a) e (b), concludiamo che Constrained-4-Color è NP-completo.

3. Pebbling è un solitario giocato su un grafo non orientato G , in cui ogni vertice ha zero o più ciottoli. Una mossa del gioco consiste nel rimuovere due ciottoli da un vertice v e aggiungere un ciottolo ad un vertice u adiacente a v (il vertice v deve avere almeno due ciottoli all'inizio della mossa). Il problema PEBBLEDESTRUCTION chiede, dato un grafo $G = (V, E)$ ed un numero di ciottoli $p(v)$ per ogni vertice v , di determinare se esiste una sequenza di mosse che rimuove tutti i sassolini tranne uno.



Una soluzione in 3 mosse di PEBBLEDESTRUCTION.

Dimostra che PEBBLEDESTRUCTION è NP-hard usando il problema del Circuito Hamiltoniano come problema NP-hard noto (un circuito Hamiltoniano è un ciclo che attraversa ogni vertice di G esattamente una volta).

Ridurremo il problema del Circuito Hamiltoniano a PebbleDestruction.

Data un'istanza $G = (V, E)$ del Circuito Hamiltoniano, costruiamo un'istanza di PebbleDestruction come segue:

- Il grafo è $G' = G$
- Per ogni vertice $v \in V$, $p(v) = 2$

Questa riduzione è corretta perché:

- Se G ha un circuito Hamiltoniano, allora esiste una sequenza di mosse in G' che rimuove tutti i ciottoli tranne uno: Segui il circuito Hamiltoniano, rimuovendo 2 ciottoli da ogni vertice e aggiungendone 1 al successivo.
- Se esiste una soluzione per PebbleDestruction in G' , allora G ha un circuito Hamiltoniano: La sequenza di mosse in G' corrisponde a un cammino che visita ogni vertice esattamente una volta, formando un circuito Hamiltoniano.

La riduzione è chiaramente calcolabile in tempo polinomiale. Quindi, PebbleDestruction è NP-hard.

- 3. (12 punti)** Una 3-colorazione di un grafo non orientato G è una funzione che assegna a ciascun vertice di G un “colore” preso dall’insieme $\{1, 2, 3\}$, in modo tale che per qualsiasi arco $\{u, v\}$ i colori associati ai vertici u e v sono diversi. Una 3-colorazione è *bilanciata* se ogni colore è associato ad esattamente $1/3$ dei vertici del grafo.

BALANCED-3-COLOR è il problema di trovare una 3-colorazione bilanciata:

$$\text{BALANCED-3-COLOR} = \{\langle G \rangle \mid G \text{ è un grafo che ammette una 3-colorazione bilanciata}\}$$

- (a) Dimostra che BALANCED-3-COLOR è un problema NP
- (b) Dimostra che BALANCED-3-COLOR è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

Soluzione.

- (a) Il certificato è un vettore c dove ogni elemento $c[i]$ è il colore assegnato al vertice i . Il seguente algoritmo è un verificatore V per BALANCED-3-COLOR:

V = “Su input $\langle \langle G \rangle, c \rangle$:

1. Controlla se c è un vettore di n elementi, dove n è il numero di vertici di G .
2. Controlla se $c[i] \in \{1, 2, 3\}$ per ogni i .
3. Controlla se per ogni arco (i, j) di G , $c[i] \neq c[j]$.
4. Controlla se ogni colore compare esattamente in $1/3$ degli elementi di c .
5. Se tutte le condizioni sono vere, *accetta*, altrimenti *rifiuta*.”

Ognuno dei passi dell’algoritmo si può eseguire in tempo polinomiale.

- (b) Dimostriamo che BALANCED-3-COLOR è NP-Hard per riduzione polinomiale da 3-COLOR a BALANCED-3-COLOR. La funzione di riduzione polinomiale f prende in input un grafo $\langle G \rangle$ e produce come output un grafo $\langle G' \rangle$. Se n è il numero di vertici di G , G' è costruito aggiungendo $2n$ vertici isolati a G . Un vertice è isolato se non ci sono archi che lo collegano ad altri vertici.

Dimostriamo che la riduzione è corretta:

- Se $\langle G \rangle \in 3\text{-COLOR}$, allora esiste un modo per colorare G con tre colori 1, 2, 3. Sia n il numero di vertici di G , e assumiamo che questa colorazione associa k_1 vertici al colore 1, k_2 vertici al colore 2 e k_3 vertici al colore 3. Posso costruire una colorazione bilanciata per il grafo G' come segue:
 - i colori dei vertici di G' che appartengono anche a G sono colorati come in G ;
 - $n - k_1$ vertici isolati sono colorati con il colore 1;
 - $n - k_2$ vertici isolati sono colorati con il colore 2;
 - $n - k_3$ vertici isolati sono colorati con il colore 3.

In questo modo ognuno dei tre colori compare in esattamente n vertici e la colorazione di G' è bilanciata.

- Se $\langle G' \rangle \in \text{BALANCED-3-COLOR}$, allora esiste una colorazione bilanciata di G' . Se elimino da G' i vertici isolati aggiunti dalla riduzione ottengo una 3-colorazione di G .

La funzione di riduzione deve contare i vertici del grafo G e aggiungere $2n$ vertici al grafo, operazioni che si possono fare in tempo polinomiale.

- 3. (12 punti)** Una 3-colorazione di un grafo non orientato G è una funzione che assegna a ciascun vertice di G un “colore” preso dall’insieme $\{1, 2, 3\}$, in modo tale che per qualsiasi arco $\{u, v\}$ i colori associati ai vertici u e v sono diversi. Una 3-colorazione è *sbilanciata* se esiste un colore che colora più di metà dei vertici del grafo.

UNBALANCED-3-COLOR è il problema di trovare una 3-colorazione sbilanciata:

$$\text{UNBALANCED-3-COLOR} = \{\langle G \rangle \mid G \text{ è un grafo che ammette una 3-colorazione sbilanciata}\}$$

- (a) Dimostra che UNBALANCED-3-COLOR è un problema NP
- (b) Dimostra che UNBALANCED-3-COLOR è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

Soluzione.

- (a) Il certificato è un vettore c dove ogni elemento $c[i]$ è il colore assegnato al vertice i . Il seguente algoritmo è un verificatore V per UNBALANCED-3-COLOR:

V = “Su input $\langle \langle G \rangle, c \rangle$:

1. Controlla se c è un vettore di n elementi, dove n è il numero di vertici di G .
2. Controlla se $c[i] \in \{1, 2, 3\}$ per ogni i .
3. Controlla se per ogni arco (i, j) di G , $c[i] \neq c[j]$.
4. Controlla se esiste un colore che compare in più di metà degli elementi di c .
5. Se tutte le condizioni sono vere, *accetta*, altrimenti *rifiuta*.”

Ognuno dei passi dell’algoritmo si può eseguire in tempo polinomiale.

- (b) Dimostriamo che UNBALANCED-3-COLOR è NP-Hard per riduzione polinomiale da 3-COLOR a UNBALANCED-3-COLOR. La funzione di riduzione polinomiale f prende in input un grafo $\langle G \rangle$ e produce come output un grafo $\langle G' \rangle$. Se n è il numero di vertici di G , G' è costruito aggiungendo $n + 1$ vertici isolati a G . Un vertice è isolato se non ci sono archi che lo collegano ad altri vertici. Dimostriamo che la riduzione è corretta:

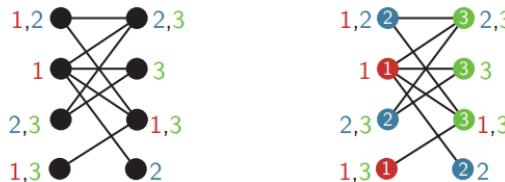
- Se $\langle G \rangle \in \text{3-COLOR}$, allora esiste un modo per colorare G con tre colori 1, 2, 3. Sia n il numero di vertici di G . Posso costruire una colorazione sbilanciata per il grafo G' come segue:
 - i colori dei vertici di G' che appartengono anche a G sono colorati come in G ;
 - i vertici isolati sono colorati tutti con il colore 1.
 In questo modo il colore 1 è assegnato ad almeno metà dei vertici di G' e la colorazione è sbilanciata.
- Se $\langle G' \rangle \in \text{UNBALANCED-3-COLOR}$, allora esiste una colorazione sbilanciata di G' . Se elimino da G' i vertici isolati aggiunti dalla riduzione ottengo una 3-colorazione di G .

La funzione di riduzione deve contare i vertici del grafo G e aggiungere $n + 1$ vertici al grafo, operazioni che si possono fare in tempo polinomiale.

- 3. (12 punti)** “Colorare” i vertici di un grafo significa assegnare etichette, tradizionalmente chiamate “colori”, ai vertici del grafo in modo tale che nessuna coppia di vertici adiacenti condivida lo stesso colore. Considera la seguente variante del problema chiamata LIST-COLORING. Oltre al grafo G , l’input del problema comprende anche una *lista di colori ammissibili* $L(v)$ per ogni vertice v del grafo.

Il problema che dobbiamo risolvere è stabilire se possiamo colorare il grafo G in modo che ogni vertice v sia colorato con un colore preso dalla lista di colori ammissibili $L(v)$.

$$\text{LIST-COLORING} = \{\langle G, L \rangle \mid \text{esiste una colorazione } c_1, \dots, c_n \text{ degli } n \text{ vertici tale che } c_v \in L(v) \text{ per ogni vertice } v\}$$



Esempio di istanza (a sinistra) e soluzione (a destra) di LIST-COLORING.

- (a) Dimostra che LIST-COLORING è un problema NP.
- (b) Dimostra che LIST-COLORING è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

(a) Dimostrazione che List-Coloring è in NP:

Un problema è in NP se esiste un verificatore polinomiale.

Verificatore per List-Coloring: Input: $\langle G, L, c \rangle$, dove G è un grafo, L è la lista dei colori ammissibili per ogni vertice, e c è una colorazione proposta

1. Per ogni vertice v di G , verifica che $c(v) \in L(v)$
2. Per ogni arco (u, v) di G , verifica che $c(u) \neq c(v)$
3. Se entrambe le condizioni sono soddisfatte, accetta. Altrimenti, rifiuta.

Questo verificatore opera in tempo polinomiale, quindi List-Coloring è in NP.

(b) Dimostrazione che List-Coloring è NP-hard:

Useremo una riduzione da 3-Color, che sappiamo essere NP-hard.

Sia $G = (V, E)$ un'istanza di 3-Color. Costruiamo un'istanza (G', L) di List-Coloring:

- $G' = G$
- Per ogni vertice $v \in V$, $L(v) = \{1, 2, 3\}$

Chiaramente, G ha una 3-colorazione se e solo se (G', L) ha una colorazione valida per List-Coloring.

Questa riduzione è polinomiale e preserva le soluzioni, quindi List-Coloring è NP-hard.

Poiché List-Coloring è sia in NP che NP-hard, è NP-completo.

4. (9 punti) Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo contiene un circuito Hamiltoniano è un problema NP-completo.

Considerate il seguente problema, che chiameremo HAM375: dato un grafo G con n vertici, trovare un ciclo che attraversa esattamente una volta $n - 375$ vertici del grafo (ossia tutti i vertici di G tranne 375).

- (a) Dimostra che HAM375 è un problema NP.
- (b) Dimostra che HAM375 è NP-hard.

(a) Dimostrazione che HAM375 è un problema NP:

Un problema è in NP se esiste un verificatore polinomiale.

Verificatore per HAM375: Input: $\langle G, C \rangle$, dove G è un grafo e C è un ciclo proposto

1. Verifica che C contenga esattamente $n - 375$ vertici di G
2. Verifica che ogni vertice in C appaia esattamente una volta
3. Verifica che per ogni coppia di vertici consecutivi in C , esista un arco in G che li collega
4. Se tutte le condizioni sono soddisfatte, accetta. Altrimenti, rifiuta.

Questo verificatore opera in tempo polinomiale, quindi HAM375 è in NP.

(b) Dimostrazione che HAM375 è NP-hard:

Useremo una riduzione dal problema del circuito Hamiltoniano (HAM), che sappiamo essere NP-hard.

Sia $G = (V, E)$ un'istanza di HAM con $|V| = n$. Costruiamo un'istanza G' di HAM375:

1. Inizia con $G' = G$
2. Aggiungi 375 nuovi vertici v_1, \dots, v_{375} a G'
3. Per ogni v_i , aggiungi archi da v_i a tutti i vertici di G

Dimostrazione della correttezza:

- Se G ha un circuito Hamiltoniano, allora G' ha un ciclo che attraversa tutti i vertici tranne i 375 nuovi (cioè il circuito Hamiltoniano originale).
- Se G' ha un ciclo che attraversa n vertici (tutti tranne 375), questo ciclo deve includere tutti i vertici di G , formando un circuito Hamiltoniano in G .

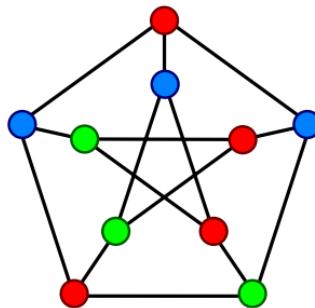
Questa riduzione è polinomiale e preserva le soluzioni, quindi HAM375 è NP-hard.

Poiché HAM375 è sia in NP che NP-hard, è NP-completo.

- 3. (12 punti)** Una 3-colorazione di un grafo non orientato G è una funzione che assegna a ciascun vertice di G un “colore” preso dall’insieme $\{R, G, B\}$, in modo tale che per qualsiasi arco $\{u, v\}$ i colori associati ai vertici u e v sono diversi. Una 3-colorazione è *equa* se per ogni coppia di colori, il numero di nodi associati a ciascun colore differisce di al più 1.

EQUITABLE-3-COLOR è il problema di trovare una 3-colorazione equa:

$$\text{EQUITABLE-3-COLOR} = \{\langle G \rangle \mid G \text{ è un grafo che ammette una 3-colorazione equa}\}$$



Esempio di colorazione equa con 4 vertici rossi, 3 vertici blu e 3 vertici verdi.

- Dimostra che EQUITABLE-3-COLOR è un problema NP
- Dimostra che EQUITABLE-3-COLOR è NP-hard, usando 3-COLOR come problema NP-hard di riferimento.

(a) Dimostrazione che EQUITABLE-3-COLOR è in NP:

Un problema è in NP se esiste un verificatore polinomiale.

Verificatore per EQUITABLE-3-COLOR: Input: $\langle G, c \rangle$, dove G è un grafo e c è una colorazione proposta

1. Verifica che c usi solo 3 colori
2. Per ogni arco (u, v) di G , verifica che $c(u) \neq c(v)$
3. Conta il numero di vertici n_i di ogni colore i
4. Verifica che $|n_i - n_j| \leq 1$ per ogni coppia di colori i, j
5. Se tutte le condizioni sono soddisfatte, accetta. Altrimenti, rifiuta.

Questo verificatore opera in tempo polinomiale, quindi EQUITABLE-3-COLOR è in NP.

(b) Dimostrazione che EQUITABLE-3-COLOR è NP-hard:

Useremo una riduzione da 3-COLOR, che sappiamo essere NP-hard.

Sia $G = (V, E)$ un'istanza di 3-COLOR. Costruiamo un'istanza G' di EQUITABLE-3-COLOR:

1. Inizia con $G' = G$
2. Aggiungi $3k$ nuovi vertici isolati a G' , dove $k = |V|$
3. Dividi questi $3k$ vertici in tre gruppi di k vertici ciascuno

Dimostrazione della correttezza:

- Se G ha una 3-colorazione, possiamo estenderla a G' colorando ciascun gruppo di k vertici aggiuntivi con uno dei tre colori. Questo produce una 3-colorazione equa di G' .
- Se G' ha una 3-colorazione equa, la restrizione di questa colorazione a G è una 3-colorazione valida di G .

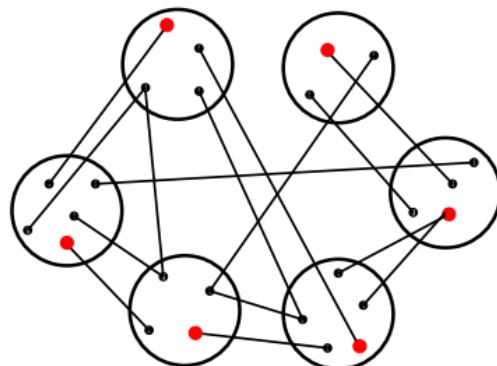
Questa riduzione è polinomiale e preserva le soluzioni, quindi EQUITABLE-3-COLOR è NP-hard.

Poiché EQUITABLE-3-COLOR è sia in NP che NP-hard, è NP-completo.

- 3. (12 punti)** La Rettrice dell'Università di Padova vuole costituire una commissione selezionando un membro per ogni dipartimento dell'ateneo. Sappiamo che alcuni dei docenti si detestano a vicenda. Per evitare scontri, la Rettrice non vuole avere membri della commissione che si detestano tra di loro. Se ogni dipartimento è un insieme D_i di docenti, e se I è la relazione di inimicizia tra docenti, una *buona commissione* è un insieme C di docenti tali che:

- ogni dipartimento ha esattamente un rappresentante in commissione;
- non esistono coppie di docenti che si detestano.

La figura seguente mostra un esempio di istanza del problema dove i cerchi sono i dipartimenti, i punti sono i docenti e gli archi collegano docenti che si detestano. I docenti evidenziati in rosso sono i componenti di una buona commissione.



Definiamo il linguaggio

$$\text{COMMITTEE} = \{\langle D_1, \dots, D_m, I \rangle \mid \text{esiste una buona commissione } C\}.$$

- (a) Dimostra che COMMITTEE è un problema NP.
- (b) Dimostra che COMMITTEE è NP-hard, usando 3SAT come problema NP-hard di riferimento.

(a) Dimostrazione che COMMITTEE è un problema NP:

1. Verificatore: data un'istanza $\langle D_1, \dots, D_n, I \rangle$ e un certificato C (sottoinsieme di docenti):

- Verifica che $|C| \leq 4$
- Per ogni dipartimento D_i , verifica che esiste almeno un docente in C
- Per ogni coppia di docenti in C , verifica che non si detestano. Questo verificatore opera in tempo polinomiale.

2. Il certificato ha dimensione polinomiale rispetto all'input. Quindi, COMMITTEE è in NP.

(b) Dimostrazione che COMMITTEE è NP-hard: Riduzione dal problema 3SAT (noto essere NP-completo).

Data una formula 3SAT ϕ con variabili x_1, \dots, x_n e clausole C_1, \dots, C_m :

1. Crea un dipartimento D_i per ogni variabile x_i
2. Crea un dipartimento E_j per ogni clausola C_j
3. Per ogni x_i , crea due docenti: x_i e $\neg x_i$
4. Per ogni $C_j = (l_1 \vee l_2 \vee l_3)$, crea tre docenti: $l_1 j$, $l_2 j$, $l_3 j$
5. I docenti si detestano se:
 - Rappresentano una variabile e la sua negazione
 - Rappresentano letterali diversi della stessa clausola
 - Rappresentano lo stesso letterale in clausole diverse

Questa costruzione garantisce che:

- ϕ è soddisfacibile se e solo se esiste una commissione valida
- La riduzione è calcolabile in tempo polinomiale

Poiché COMMITTEE è in NP e è NP-hard, è NP-completo.

4. (9 punti) Supponiamo di avere un sistema elettorale composto da n elettori, dove ogni elettore i ha un "peso" $W[i]$, corrispondente al numero di voti che rappresenta. Nel caso di una votazione a maggioranza semplice, una coalizione di elettori ha bisogno di un numero di voti strettamente superiore alla metà della somma dei pesi per vincere. L'elettore n è detto *pivot* se esiste una situazione in cui il voto dell'elettore "conta", ossia dove l'aggiunta dell'elettore n ai voti "sì" rende il "sì" la maggioranza, ma l'aggiunta ai voti "no" rende il "no" maggioranza. Formalmente, l'elettore n è un pivot se esiste una coalizione di elettori $C \subseteq \{1, \dots, n-1\}$ tale che

$$\sum_{j \in C} W[j] < \frac{1}{2} \sum_{j=1}^n W[j] \quad (C \text{ senza i voti di } n \text{ è minoranza})$$

$$\sum_{j \in C} W[j] + W[n] > \frac{1}{2} \sum_{j=1}^n W[j] \quad (C \text{ con i voti di } n \text{ è maggioranza})$$

e possiamo rappresentare il problema dell'elettore pivot con il linguaggio

$$PIVOT = \{\langle n, W \rangle \mid \text{l'elettore } n \text{ è un pivot}\}.$$

- (a) Dimostra che *PIVOT* è un problema NP.
- (b) Sappiamo che il linguaggio *SET-PARTITION* = $\{\langle S \rangle \mid S$ insieme di naturali, ed esistono $S_1, S_2 \subseteq S$ tali che $S_1 \cup S_2 = S, S_1 \cap S_2 = \emptyset, \sum_{x \in S_1} x = \sum_{y \in S_2} y\}$ è NP-completo. Dimostra che *PIVOT* è NP-hard, usando *SET-PARTITION* come problema NP-hard di riferimento.

Esempio: Supponiamo di avere 5 elettori, con pesi 4, 3, 3, 2, 1. La somma totale dei pesi è 13, quindi la maggioranza si ottiene con 7 voti. Il quinto elettore, con peso 1, è un pivot in coalizione con gli elettori di peso 4 e 2. La coalizione perde senza l'elettore pivot ma vince con lui.

(a) Un certificato per PIVOT è una coalizione $C \subseteq \{1, \dots, n-1\}$ tale che:

- $\sum_{j \in C} W[j] < (1/2) \sum_{j=1}^n W[j]$
- $\sum_{j \in C} W[j] + W[n] > (1/2) \sum_{j=1}^n W[j]$

Dato $\langle n, W \rangle$ e un certificato C , possiamo verificare in tempo polinomiale che:

1. $C \subseteq \{1, \dots, n-1\}$ (scorrendo C)
2. Le due disuguaglianze sono soddisfatte (calcolando le somme)

Quindi PIVOT \in NP.

(b) Riduciamo SET-PARTITION \leq_p PIVOT.

Data un'istanza $\langle S \rangle$ di SET-PARTITION, con $S = \{x_1, \dots, x_m\}$, costruiamo:

- $n = m+1$
- $W[i] = x_i$ per $i = 1, \dots, m$
- $W[n] = (1/2) \sum_{x \in S} x$

Allora:

- Se $\langle S \rangle \in$ SET-PARTITION, esiste $S_1 \subseteq S$ tale che $\sum_{x \in S_1} x = (1/2) \sum_{x \in S} x = W[n]$.
- Quindi $C = \{i \mid x_i \in S_1\}$ dimostra che $m+1$ è un pivot.
- Se $\langle n, W \rangle \in$ PIVOT, esiste $C \subseteq \{1, \dots, m\}$ che soddisfa le disuguaglianze.

Ponendo $S_1 = \{x_i \mid i \in C\}$, le disuguaglianze implicano che $\sum_{x \in S_1} x = (1/2) \sum_{x \in S} x$, quindi $\langle S \rangle \in$ SET-PARTITION.

La riduzione è in tempo polinomiale, quindi PIVOT è NP-hard.

3. (12 punti) In una delle storie delle Mille e una notte, Alì Babà, mentre viaggiava con il suo asino, trovò la grotta in cui i 40 ladroni avevano nascosto il loro bottino. Come cittadino rispettoso della legge, denunciò il fatto alla polizia, ma solo dopo aver tenuto il più possibile per sé. Il problema è che c'è troppo bottino e l'asino non può portarlo tutto: c'è un limite M al peso che l'asino può trasportare. Supponiamo che ognuno degli N oggetti rubati abbia un prezzo $P[i]$ e un peso $W[i]$. Alì Babà può caricare sull'asino un numero sufficiente di oggetti in modo che il prezzo totale sia almeno L ?

Formalmente, possiamo rappresentare il problema che Alì Babà deve risolvere con il linguaggio

$$ALIBABA = \left\{ \langle N, P, W, M, L \rangle \mid \text{esiste } B \subseteq \{1, \dots, N\} \text{ tale che } \sum_{j \in B} W[j] \leq M \text{ e } \sum_{j \in B} P[j] \geq L \right\}.$$

- (a) Dimostra che ALIBABA è un problema NP.
- (b) Sappiamo che il linguaggio

$$SUBSET-SUM = \left\{ \langle S, t \rangle \mid S \text{ insieme di naturali, ed esiste } S' \subseteq S \text{ tale che } \sum_{x \in S'} x = t \right\}$$

è NP-completo. Dimostra che ALIBABA è NP-hard, usando SUBSET-SUM come problema NP-hard di riferimento.

(a) Per dimostrare che ALIBABA è un problema NP, forniamo un verificatore polinomiale V:

V = "Su input $\langle \langle N, P, W, M, L \rangle, B \rangle$:

1. Controlla se B è un sottoinsieme di $\{1, \dots, N\}$. Se no, rifiuta.
2. Calcola $\sum_{j \in B} W[j]$ e verifica se è $\leq M$. Se no, rifiuta.
3. Calcola $\sum_{j \in B} P[j]$ e verifica se è $> L$. Se no, rifiuta.
4. Accetta."

(b) Dimostrazione che ALIBABA è NP-hard:

Riduzione da SUBSET-SUM a ALIBABA: Data un'istanza $\langle S, t \rangle$ di SUBSET-SUM, costruiamo un'istanza di ALIBABA:

- $N = |S|$
- $W[i] = S[i]$ per ogni i
- $P[i] = S[i]$ per ogni i
- $M = t$
- $L = t$

Questa riduzione è corretta perché:

- Esiste $S' \subseteq S$ tale che $\sum_{x \in S'} x = t$ se e solo se esiste $B \subseteq \{1, \dots, N\}$ tale che $\sum_{i \in B} W[i] \leq M$ e $\sum_{i \in B} P[i] \geq L$

La riduzione è calcolabile in tempo polinomiale.

Poiché ALIBABA è in NP e è NP-hard, è NP-completo.

3. (12 punti) Date m sostanze nutritive ed un menu di n alimenti che forniscono tali sostanze nutritive, si desidera determinare se esiste un piccolo insieme di alimenti che forniscono tutti i nutrienti di cui avete bisogno. Se ogni alimento $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di nutrienti, una *dieta valida* è un insieme T di alimenti che, nel loro insieme, forniscono tutti gli m nutrienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$DIET = \{\langle S_1, \dots, S_n, k \rangle \mid \text{esiste una dieta valida } T \text{ di dimensione } |T| = k\}.$$

- (a) Dimostra che DIET è un problema NP.
- (b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T . Sappiamo che il linguaggio *VERTEX-COVER* = $\{\langle G, k \rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che DIET è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

3. a) DIET è in NP. Ecco un verificatore polinomiale:

V = "Su input $\langle \langle S_1, \dots, S_n, k \rangle, T \rangle$:

1. Verifica che T sia un sottoinsieme di $\{1, \dots, n\}$ con $|T|=k$. Se no, rifiuta.
2. Verifica che ogni elemento di $\{1, \dots, n\}$ compaia in qualche S_i con $i \in T$. Se no, rifiuta.
3. Accetta."

Chiaramente V impiega tempo polinomiale.

(b) DIET è NP-hard per riduzione da VERTEX-COVER. $f(\langle G, k \rangle) = \langle S_1, \dots, S_n, k \rangle$ dove n è il numero di vertici di G e $S_i = \{j \mid (i, j)$ è un arco di $G\}$.

Se $\langle G, k \rangle \in \text{VERTEX-COVER}$ allora esiste un vertex cover C di dimensione k, quindi prendendo gli insiemi S_i con $i \in C$ si ottiene una dieta valida, perché ogni elemento (vertice) compare in qualche S_i (è coperto da C). Viceversa, se $f(\langle G, k \rangle) \in \text{DIET}$ allora esiste una dieta valida di k insiemi S_i , e i vertici corrispondenti formano un vertex cover per G. Quindi f è una riduzione polinomiale corretta e DIET è NP-hard.

Essendo DIET sia in NP che NP-hard, è NP-completo.

- 3. (12 punti)** Un ristorante ha necessità di acquistare m ingredienti per preparare tutti i piatti che offre ai suoi clienti. Dato un insieme di n fornitori che forniscono tali ingredienti, si desidera determinare se esiste un piccolo insieme di fornitori che consentano di acquistare tutti gli ingredienti di cui il ristorante ha bisogno. Se ogni fornitore $i = 1, \dots, n$ fornisce un insieme $S_i \subseteq \{1, \dots, m\}$ di ingredienti, una *fornitura valida* è un insieme T di fornitori che, nel loro insieme, forniscono tutti gli m ingredienti: $\bigcup_{i \in T} S_i = \{1, \dots, m\}$. Definiamo il linguaggio

$$\text{SUPPLY} = \{\langle S_1, \dots, S_n, k \rangle \mid \text{esiste una fornitura valida } T \text{ di dimensione } |T| = k\}.$$

- (a) Dimostra che *SUPPLY* è un problema NP.
- (b) Una copertura di vertici di un grafo G è un insieme di vertici T tale che ogni arco del grafo ha almeno una estremità su un vertice in T . Sappiamo che il linguaggio $\text{VERTEX-COVER} = \{\langle G, k \rangle \mid G \text{ ha una copertura di vertici di dimensione } k\}$ è NP-completo. Dimostra che *SUPPLY* è NP-hard, usando *VERTEX-COVER* come problema NP-hard di riferimento.

(a) *SUPPLY* è in NP. Un verificatore polinomiale è: V = "Su input $\langle \langle S_1, \dots, S_n, k \rangle, T \rangle$:

1. Verifica che T sia una fornitura valida di dimensione k . Se no, rifiuta.
2. Per ogni $i = 1, \dots, n$, verifica che $\sum_{j \in T} S_i[j] \leq \{1, \dots, n\}$. Se no, rifiuta.
3. Accetta." Chiaramente V impiega tempo polinomiale.

(b) *SUPPLY* è NP-hard per riduzione da *VERTEX-COVER*. Sia $f(\langle G, k \rangle) = \langle S_1, \dots, S_n, k \rangle$ dove:

- n è il numero di vertici di G
- $S_i[j] = 1$ se (i, j) è un arco di G , $S_i[j] = 0$ altrimenti Allora: Se $\langle G, k \rangle \in \text{VERTEX-COVER}$, esiste un vertex cover C di dimensione k, e prendendo la fornitura $T = C$ abbiamo che $\sum_{j \in T} S_i[j] = 1$ per ogni vertice $i \in \{1, \dots, n\}$, perciò $f(\langle G, k \rangle) \in \text{SUPPLY}$. Viceversa, se $f(\langle G, k \rangle) \in \text{SUPPLY}$, esiste una fornitura valida T di dimensione k tale che ogni vertice i ha una adiacenza con un vertice in T . Quindi T è un vertex cover per G e $\langle G, k \rangle \in \text{VERTEX-COVER}$.

Quindi f riduce *VERTEX-COVER* a *SUPPLY* in tempo polinomiale, e *SUPPLY* è NP-hard. Essendo *SUPPLY* sia NP che NP-hard, è NP-completo.