1) Dato un grafo si consideri il problema di colorare i nodi o di bianco o di nero in modo che, per ogni nodo, almeno metà dei suoi nodi adiacenti sia di colore diverso dal suo.

1a) Dimostrare che il seguente algoritmo risolve il problema e trovarne la complessità computazionale (tenendo anche conto della struttura dati necessaria ad eseguire l'algoritmo): se esiste un nodo con più nodi adiacenti del suo stesso colore lo si cambi di colore altrimenti stop.

Soluzione: L'algoritmo consiste in una iterazione di passi in cui in ogni passo si verifica se la condizione è verificata in ogni nodo e, se non è verificata in un nodo, il colore di quel nodo viene cambiato. Per iniziare, l'algoritmo ha bisogno che tutti i nodi siano colorati (in modo arbitrario). Dobbiamo far vedere che l'algoritmo necessariamente termina. Sia $T(k)$ il numero di archi al passo k i cui estremi sono di colore diverso. Siccome il cambiamento di colore di un nodo si effettua solo se più della metà dei nodi adiacenti al nodo in esame è dello stesso colore, il cambiamento di colore fa aumentare $T(k)$. Quindi $T(k+1) > T(k)+1$. Siccome $T(k) \leq m$ (con m numerio di archi del grafo) e $T(0) \geq 0$ (se ad esempio tutti i nodi sono dello stesso colore inizialmente) si vede che il numero di iterazioni è $O(m)$.

Si tratta ora di valutare quanto lavoro è richiesto per verificare la condizione. Sia n_i il numeo di nodi adiacenti al nodo i e sia t_i il numero di nodi adiacenti al nodo i di colore diverso da i. Conviene mantenere in memoria i valori n_i e $s_i := t_i - n_i/2$. Inizialmente (se si colorano i nodi tutti dello stesso colore) $s_i = -n_i/2$. L'algoritmo termina quando $s_i \geq 0$ per ogni i. Sia Q una lista di nodi candidati al cambiamento di colore. Inizialmente Q contiene tutti i nodi. L'algoritmo rimuove il primo nodo della lista di Q e verifica se $s_i < 0$. Se no si passa al nodo successivo. Se si, si cambia il colore di i e si aggiorna $s_i := -s_i$ e $s_j := s_j + 1$ se j è un nodo adiacente a i di colore uguale al colore originale di i e $s_j := s_j - 1$ in caso contrario e se $s_j = -1$ si aggiunge j alla lista Q. Poi si passa al nodo successivo di Q. L'esplorazione dei nodi adiacenti a i comporta un costo di n operazioni nel caso peggiore e quindi il numero globale di operazioni è $O(n\,m)$. Bisogna ancora tener conto di quante volte viene interrogata la lista Q globalmente (vi possono essere anche nodi per i quali l'algoritmo ha reso non negativo il valore s_i). Nel caso peggiore ad ogni iterazione la coda Q viene accresciuta di n elementi. Quindi si arriva anche in questo caso ad una complessità di $O(n\,m)$.

1b) Dimostrare con un controesempio che l'algoritmo *non* risolve il problema collegato di massimizzare il numero di archi i cui nodi siano di colore diverso. Dimostrare che questo secondo problema è **NP**-difficile.

Soluzione: Si consideri il grafo in figura. L'algoritmo potrebbe eseguire le due iterazioni indicate, mentre l'ottimo è dato dall'ultima figura.

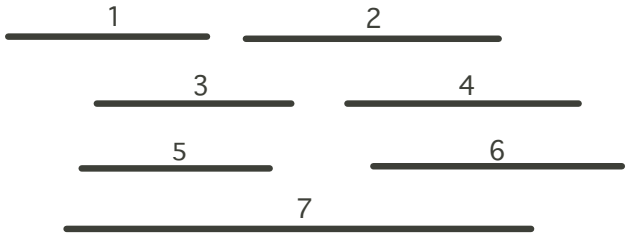
Per dimostrare la **NP**-completezza della versione ricognitiva, cioè, 'dato K esiste una colorazione per cui vi sono almeno K archi con estremi di colore diverso?', basta notare che si tratta del problema MAXCUT, di cui è nota la **NP**-completezza.

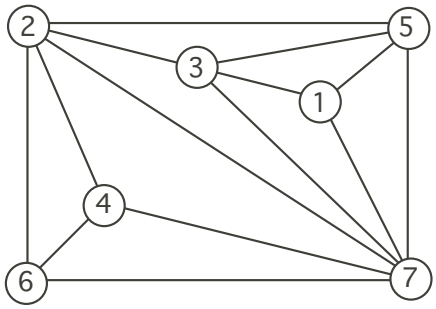
2) Sia dato un grafo non orientato completo. Si orientino arbitrariamente gli archi. Dimostrare, in modo costruttivo, che esiste nel grafo orientato così ottenuto un cammino hamiltoniano (orientato).

Soluzione: L'algoritmo procee in modo ricorsivo: dato un cammino orientato di k nodi ed un nodo i non appartenente al cammino, crea un nuovo cammino orientato con i k nodi ed il nodo i. I k nodi siano per semplicità $1, 2, \ldots, k$. Se c'è l'arco $(i, 1)$ allora si aggiunge il nodo i in testa ai k nodi. Se invece c'è l'arco (k, i) si aggiunge il nodo i in coda ai k nodi. Altrimenti sia h un nodo per cui esistono gli archi (h, i) e $(i, h + 1)$ (tale nodo deve esistere a questo punto). Basta allora prendere il cammino $1 \rightsquigarrow h \rightarrow (h + 1) \rightsquigarrow k$. Come cammino iniziale basta prendere un arco qualsiasi.

Si noti che, se si applica il teorema al campionato di calcio, con i nodi che rappresentano le squadre e gli archi l'esito degli incontri fra le squadre (in caso di parità secondo tutti i criteri, si orienti arbitrariamente l'arco), si ottiene il risultato controintuitivo che esiste una successione di tutte le squadre in cui la prima è meglio della seconda, la seconda è meglio della terza e così via. Non necessariamente la prima squadra è quella che vince il campionato.

3) Siano assegnati n intervalli $I_1, \ldots, I_n$ in R. Si definisca il *grafo degli intervalli* come un grafo in cui ogni nodo è associato ad un intervallo ed esiste un arco (i, j) se e solo se $I_i \cap I_j \neq \emptyset$ (si veda un esempio in figura).





– si determini un algoritmo che fornisca (in tempo polinomiale) la coloratura minima di un grafo degli intervalli;

– si determini un algoritmo che fornisca (in tempo polinomiale) tutte le clique massimali di un grafo degli intervalli (una clique K è massimale se $K \cup \{i\}$ non è una clique per ogni $i \notin K$).

Soluzione: Siano gli intervalli $I_i := [a_i, b_i]$, $i := 1, \ldots, n$. Si definisca la seguente funzione $f : R \to Z$

$$f(x) := |\{i : x \geq a_i\}| - |\{i : x > b_i\}|$$

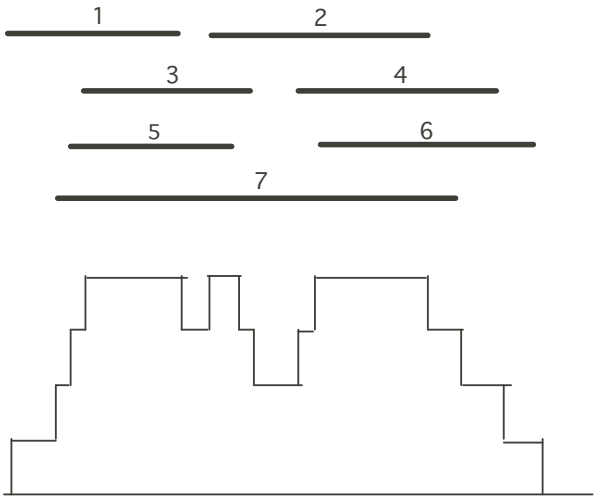
Per definizione (i, j) è un arco del grafo se $I_i \cap I_j \neq \emptyset$, ovvero se $\max\{a_i\,,\, a_j\} \leq \min\{b_i\,,\, b_j\}$. Allora, se nel grafo esiste la clique $\{i, j, k\}$, si ha $\max\{a_i\,,\, a_j\} \leq \min\{b_i\,,\, b_j\}$, $\max\{a_i\,,\, a_k\} \leq \min\{b_i\,,\, b_k\}$ e $\max\{a_j\,,\, a_k\} \leq \min\{b_j\,,\, b_k\}$, da cui $\max\{a_i\,,\, a_j\,,\, a_k\} \leq \min\{b_i\,,\, b_j\,,\, b_k\}$ cioè $I_i \cap I_j \cap I_k \neq \emptyset$.

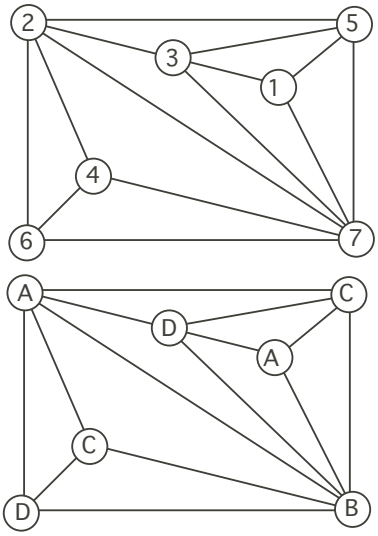
Allora la funzione f 'conta' gli indici per cui sia $x \geq a_i$ che $x \leq b_i$. Per ogni x è quindi definita una clique e per ogni clique esiste un x che la definisce. Il valore della massima clique è pertanto dato da $\chi = \max f(x)$.

Per colorare il grafo basta ordinare i valori a_i e b_i e scandirli dal minimo al massimo. In un passo generico dell'algoritmo i colori 'impegnati' sono marcati e quelli disponibili sono quindi non marcati. Inzialmente nessun colore è marcato. Se il generico numero da scandire è a_i allora ci sono $\chi - f(a_i - \varepsilon)$ colori disponibili e si può usare uno di questi per colorare il nodo i. Quindi tale colore viene marcato. Se il generico numero da scandire è b_i allora il colore con cui era stato colorato il nodo i diventa nuovamente disponibile e tale colore viene smarcato.

È immediato a questo punto notare che i massimi locali di $f(x)$ forniscono le clique massimali. Come si vede si tratta di un grafo in cui il numero cromatico è uguale alla massima clique. Si può dimostrare infatti che il grafo degli intervalli è un grafo perfetto.

Entrambi gli algoritmi richiedono $O(n \log n)$ per ordinare i dati e $O(n)$ per scandirli. Si veda la figura.


1
2
3
4
5
6
7


2
5
3
1
4
6
7

A
C
D
A
C
D
B