Problem set 1
Esercizio 4

Alessandro Straziota

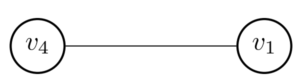
Esercizio 4. Data una lista di n interi positivi $d_1, d_2, \ldots, d_n$, vogliamo determinare in modo efficiente se esiste un grafo $G = (V, E)$ in cui i gradi dei nodi sono esattamente $d_1, \ldots, d_n$. Ossia, se indichiamo con $V = \{v_1, \ldots, v_n\}$ l'insieme dei nodi, allora per ogni $i = 1, 2, \ldots, n$ il grado del nodo v_i deve essere esattamente d_i. Il grafo non deve contenere *self-loop* (archi che hanno entrambi gli estremi nello stesso nodo) o archi *multipli* fra la stessa coppia di nodi.

1. Dare un esempio di d_1, d_2, d_3, d_4 in cui $d_i \leqslant 3$ per ogni i e $d_1 + d_2 + d_3 + d_4$ è pari, ma non esiste nessun grafo con sequenza dei gradi (d_1, d_2, d_3, d_4);
2. Supponiamo che $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n$ e che esiste un grafo $G = (V, E)$ con sequenza di gradi $(d_1, \ldots, d_n)$. Dimostrare che deve esistere anche un grafo con questa sequenza di gradi e in cui in più i vicini del nodo v_1 sono esattamente $v_2, v_3, \ldots, v_{d_1+1}$;
3. Usando il punto precedente, progettare un algoritmo che dati $d_1, \ldots, d_n$ decide se esiste un grafo con questa sequenza di gradi. L'algoritmo deve avere tempo polinomiale in n.

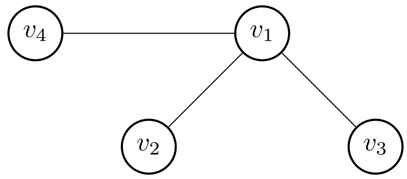
Punto 1 Un esempio possibile è $d_1 = d_2 = d_3 = 3$ e $d_4 = 1$. Innanzitutto la loro somma è apri, infatti $d_1 + d_2 + d_3 + d_4 = 10$. Ora supponiamo che esista il grafo $G = (V, E)$ in questione. Poniamo l'insieme dei nodi $V = \{v_1, v_2, v_3, v_4\}$ con rispettivi gradi d_1, d_2, d_3, d_4. Innanzitutto è corretto che la somma dei gradi di un grafo sia pari, infatti

$$\sum_{v \in V} \delta(v) = 2 \cdot |E|$$

Da questo possiamo dedurre che $|E| = 5$. Ora bisogna cercare un modo corretto di disporre 5 archi su un grafo di 4 nodi che rispetti però i vincoli sui gradi d_1, d_2, d_3, d_4.
Partiamo dal nodo v_4, essendo l'unico con grado 1 avrà un solo arco incidente e sarà adiacente a uno solo tra i nodi v_1, v_2, v_3 (per esempio v_1 tanto è indifferente dato che hanno tutti e tre lo stesso grado).



Adesso dato che v_1 ha grado 3 deve essere collegato necessariamente a v_2 e v_3, visto che G non può contenere *self-loop* ne *archi multipli* fra una stessa coppia di nodi.



A questo punto è evidente che non ci sono modi di disporre i restanti 2 archi senza violare uno dei vincoli posti per G. Per esempio potremmo aggiungere un arco tra v_2 e v_3 ma rimarremmo comunque in un vicolo cieco, nel quale il grado di v_2 e v_3 è ancora minore di 3 e saremo inoltre incapaci di poter aggiungere il quinto arco mancante. *Scacco matto*!

Punto 2 Dato G con sequenza di gradi $d_1, d_2, ..., d_n$ tali che $d_1 \geq d_2 \geq ... \geq d_n$ verrà mostrato per costruzione che esiste anche un grafo G' con la stessa sequenza di gradi, in cui in più i vicini di v_1 sono $v_2, v_3, ..., v_{d_1+1}$. Più precisamente G' si ricava ristrutturando G.

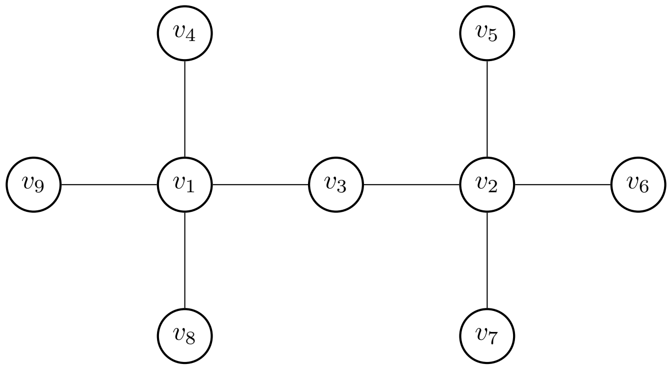
Innanzitutto supponiamo che in G i vicini di v_1 non siano esattamente $v_2, v_3, ..., v_{d_1+1}$ (altrimenti saremo nella situazione in cui $G \equiv G'$) e soprattutto supponiamo che $d_1 > 0$ (altrimenti non avrebbe senso cercare di costruire G'). Questo implica che esiste almeno un nodo v_i (con $i \leq d_1 + 1$) che non è vicino di v_1. Dato che $\delta_G(v_1) = d_1$ allora deve esistere necessariamente un altro nodo v_k (con $k > d_1 + 1$) vicino di v_1.
Un'altra osservazione importante da fare è che sicuramente v_i ha almeno un vicino a sua volta (ossia $\delta_G(v_i) = d_i > 0$) infatti:

- $d_k > 0$ in quanto v_k è vicino di v_1
- $i \leq d_1 + 1$
- $k > d_1 + 1$
- $d_1 \geq d_2 \geq ... \geq d_n$

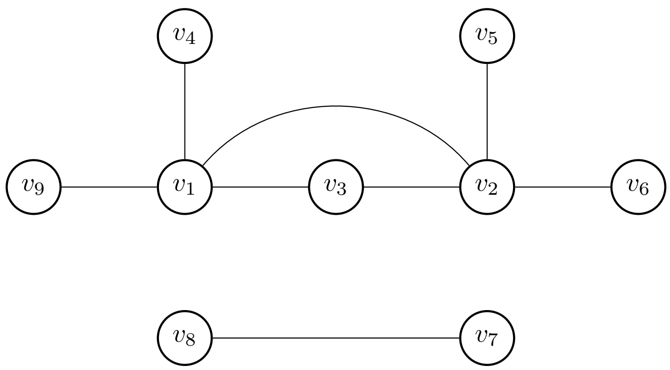
possiamo concludere che anche v_i ha almeno un vicino, in quanto $d_i \geq d_k \geq 1$.

Ricapitolando: v_1 ha grado d_1 e tra i suoi vicini esiste un certo nodo v_k (con $k > d_1 + 1$) che in G' vorremmo sostituire con un certo nodo v_i (con $i \leq d_1 + 1$), che a sua volta ha anch'esso almeno un vicino (diciamo v_h).
Ciò che viene in mente è quindi di "staccare" v_k da v_1, staccare v_h da v_i, "attaccare" v_k a v_h e attaccare v_i a v_1. È facile notare che **il grado dei nodi rimane inalterato** e che abbiamo ottenuto l'effetto desiderato, ovvero v_i è ora vicino di v_1. Alla luce di questo si potrebbe iterare questo passaggio finché non otterremo che i vicini di v_1 sono esattamente i nodi da v_2 a v_{d_1+1}.

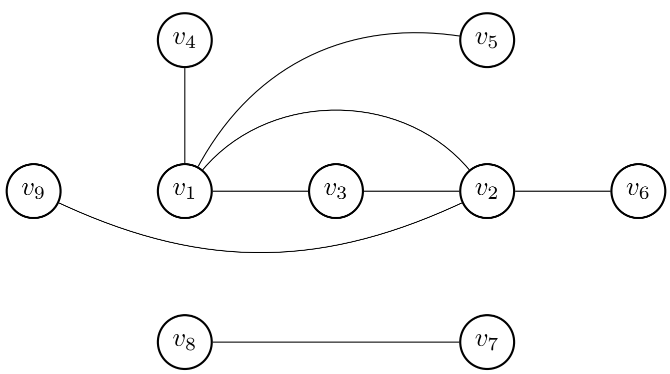
Esempio: Consideriamo il grafo G in figura con la seguente sequenza di gradi $(4, 4, 2, 1, 1, 1, 1, 1, 1)$ e costruiamo il grafo G' secondo il procedimento sopra esposto.



Al primo passo stacco v_2 da v_7 e v_1 da v_8, poi rendo vicino v_2 a v_1 e v_7 a v_8.



Al secondo passo invece stacco v_2 da v_5 e v_1 da v_9, poi rendo vicino v_5 a v_1 e v_9 a v_2.



Punto 3 Prima di procedere a un algoritmo che decide se esiste un grafo $G = (V, E)$ in cui i gradi dei nodi sono esattamente $d_1, d_2, ..., d_n$ è doveroso analizzare i casi in cui potrebbe non esistere quel grafo. Innanzitutto quando la somma $d_1 + d_2 + ... + d_n$ è dispari sicuramente il grafo non esiste, in quanto la somma dei gradi dei nodi di un grafo è **sempre pari**. Poi, dato che il grafo non deve avere *self-loop* ne *archi multipli* allora la somma $d_1, d_2, ..., d_n$ deve essere **al più** $n(n-1)$, in quanto nel caso limite in cui ci troviamo con un grafo completo il numero di archi è esattamente $\dfrac{n(n-1)}{2}$.
Un altro caso in cui non esiste il grafo in questione è quando esiste un valore d_i tra quelli in input il cui valore è strettamente maggiore di $n-1$; questo perché altrimenti tale nodo con almeno n vicini avrà almeno un arco riflessivo oppure un arco multiplo con un dei suoi vicini.
Ancora, nel caso in cui il grafo G sia composto da un solo nodo esso dovrà necessariamente avere come grado 0 (sempre perché non si possono avere archi riflessivi).
Queste situazioni però da sole non bastano per dire che il grafo G in questione non esiste, infatti come abbiamo visto nel **Punto 1** esistono dei casi in cui $d_1, d_2, ..., d_n$ non violano i precedenti vincoli, però comunque è evidente che il grafo non esista.

Perciò sfruttiamo quanto scoperto nel **Punto 2**:
Data una sequenza di interi positivi $d_1, d_2, ..., d_n$ supponiamo che esista un grafo G tale che la sequenza dei gradi dei nodi corrisponde esattamente a $d_1, d_2, ..., d_n$. Supponiamo inoltre che la sequenza $d_1, d_2, ..., d_n$ sia **ordinata in modo non crescente** ($d_1 \geq d_2 \geq ... \geq d_n$). Sappiamo che se esiste G allora esiste anche un grafo G' con la stessa sequenza dei gradi dei nodi, dove però i vicini del nodo v_1 sono $v_2, ..., v_{d_1+1}$.
Se questo è vero allora supponiamo di rimuovere il nodo v_1 da G': otterremo un nuovo grafo G'' dove però i nodi $v_2, ..., v_{d_1+1}$ avranno il loro grado abbassato di uno ($\delta_{G''}(v_i) = \delta_{G'}(v_i) - 1 = d_i - 1, \forall 2 \leq i \leq d_1 + 1$).
Se esiste G'' sappiamo che possiamo ricavarci un nuovo grafo G''' con la medesima sequenza dei gradi dei nodi di G'' $(d_2, d_3, ..., d_n)$ dove però v_2 avrà come vicini i nodi $v_3, v_4, ..., v_{d_2+2}$. Ancora, possiamo rimuovere da G''' il nodo v_2 e ricavare un nuovo grafo dove tutti i vecchi vicini di v_2 avranno il loro grado diminuito di 1.
Così facendo, se è vero che G esiste, arriveremo a un grafo G^* con **un** solo nodo, il quale deve necessariamente avere grado 0.
Contrariamente, se è falso che G esiste, potremmo incorrere in due situazioni:

1. Ci ritroviamo con un grafo G^* con un solo nodo il quale grado risulta essere maggiore di 0
2. Ci ritroviamo con un grafo G^* con un nodo il quale grado risulta essere minore di 0

Alla luce di questa analisi sui possibili casi in cui G può o non può esistere verrà descritto un algoritmo che:

Data una sequenza di n interi positivi $d_1, d_2, ..., d_n$ ritorna *True* se esiste un grafo $G = (V, E)$ in cui i gradi dei nodi sono esattamente $d_1, d_2, ..., d_n$, *False* altrimenti.

```
    input  : Una sequenza di n interi positivi d_1, d_2, ..., d_n
    output: True se esiste un grafo G = (V, E) in cui i gradi dei nodi sono
            esattamente d_1, d_2, ..., d_n, False altrimenti
 1  if n = 1 ∧ d_1 > 0 then
 2      return False;
 3  s ← ∑_i d_i;
 4  if s mod 2 ≠ 0 ∨ s > n(n − 1) then
 5      return False;
 6  if ∃d_i ≥ n then
 7      return False;
 8  Ordina la sequenza d_1, d_2, ..., d_n in senso non crescente, tale che
      d_1 ≥ d_2 ≥ ... ≥ d_n;
 9  while la lunghezza della sequenza dei numeri è maggiore di 1 do
10      Sia k il primo elemento della sequenza;
11      Rimuovi il primo elemento della sequenza e decrementa di uno tutti
          i k valori successivi all'elemento appena rimosso;
12      if ci sono meno di k elementi rimasti nella sequenza then
13          return False;
14      Ordina nuovamente la sequenza d_1, d_2, ..., d_n in senso non crescente;
15  if l'ultimo elemento rimasto è uguale a 0 then
16      return True;
17  else
18      return False;
```

Possiamo concludere che l'*algoritmo* appena descritto è corretto perché per costruzione rispetta tutte le possibili situazioni che possono capitare, alla luce di quanto analizzato in precedenza.

Inoltre possiamo affermare che l'algoritmo computa in un tempo *polinomiale in* n dato che:

1. A `Riga 3` per ricavare la somma di tutti gli elementi della sequenza eseguo $n - 1$ addizioni;

2. A `Riga 6` per verificare se esiste un valore maggiore n devo controllare tutti gli elementi della serie;

3. A `Riga 8` so che posso ordinare una sequenza di n numeri in tempo $O(n \log n)$;

4. Il ciclo `while` invece so che dura al più $n-1$ iterazioni, ognuna delle quali mi costa:
    - k decrementi dei valori della sequenza, dove però k è al più $n-1$
    - un ordinamento di una serie di numeri, dove però tale serie sarà al più lunga $n-1$
    - altre poche operazioni costanti
5. Il resto delle operazioni sono costanti

È evidente che la parte più costosa dell'algoritmo nel caso peggiore è il ciclo `while`, però si può verificare che il ciclo ha un costo computazionale di

$$O(n \cdot (n + n \log n)) \in O(n^2 \log n) \in POLY(n)$$