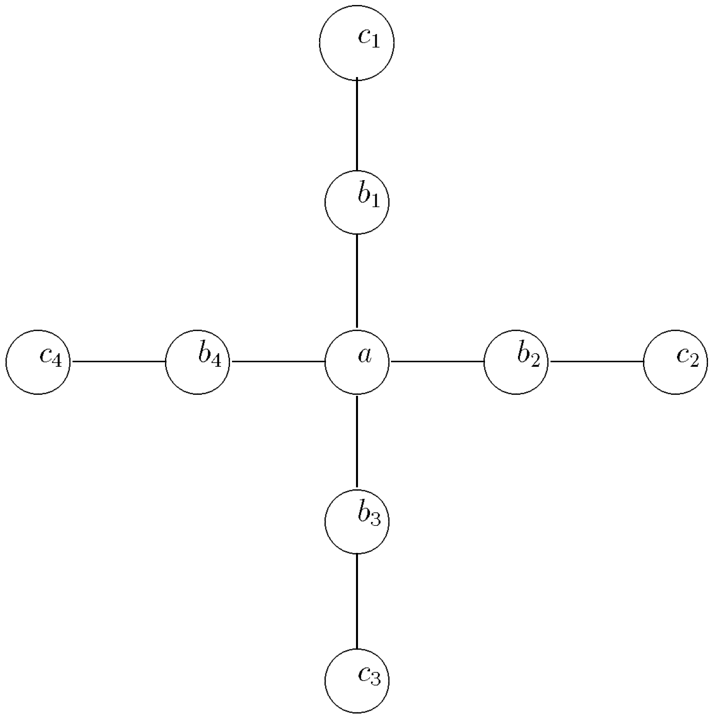
Esercizio 1 Si consideri il seguente problema. Dato un grafo non diretto $G = (V, E)$ trovare un insieme dominante di cardinalità minima. Un insieme di nodi è *dominante* se ogni nodo del grafo o appartiene all'insieme o è adiacente ad un nodo dell'insieme. Si propone il seguente algoritmo.

INPUT Un grafo non diretto $G = (V, E)$
  $N \leftarrow V$
  $D \leftarrow \emptyset$
  WHILE $N \neq \emptyset$ DO
    Sia u un nodo in N con il massimo numero di adiacenti in N
    $D \leftarrow D \cup \{u\}$
    Elimina da N il nodo u e i nodi ad esso adiacenti
  ENDWHILE
OUTPUT D.

1. Dimostrare che l'algoritmo produce sempre un insieme dominante.
2. Dire se l'algoritmo risolve il problema e in caso affermativo dimostrare la correttezza dell'algoritmo mentre in caso negativo fornire un controesempio.
3. Assumendo che il grafo venga rappresentato tramite liste di adiacenza, descrivere una implementazione dell'algoritmo con complessità asintotica $O((|V|+|E|)\log|V|)$. Spiegare perchè l'implementazione ha una tale complessità.

Soluzione Esercizio 1

1. Supponiamo per assurdo che su un certo grafo di input G l'algoritmo non produca un insieme dominante. Sia w un nodo di G che non è dominato da D, cioè, non è in D e non è adiacente a nessun nodo di D. L'algoritmo termina solamente quando N è diventato vuoto, quindi ci deve essere stata una iterazione del WHILE in cui w viene eliminato da N. Siccome w non è in D deve essere che w era adiacente al nodo u che in quella iterazione è stato scelto e inserito in D: questo è in contraddizione con l'assunto iniziale.
2. L'algoritmo non risolve il problema!
   Si consideri ad esempio il grafo G riportato in basso. Nella prima iterazione del WHILE il nodo a che ha il massimo numero di adiacenti viene inserito in D, i nodi $\{b_1, b_2, b_3, b_4\}$ adiacenti ad a vengono cancellati da N dove restano quindi i soli nodi $\{c_1, c_2, c_3, c_4\}$ non adiacenti tra loro. Seguiranno a questo punto 4 iterazioni del WHILE in ciascuna delle quali uno di questi nodi verrà inserito in D. La soluzione $\{a, c_1, c_2, c_3, c_4\}$ prodotta dall'algoritmo ha cadinalità 5 mentre è ovvio che l'insieme $\{b_1, b_2, b_3, b_4\}$ di cardinalità 4 è un insieme dominante per G.



3. L'idea principale per una implementazione efficiente dell'algoritmo sta nell'uso di un heap H per mantenere i nodi di N "ordinati" rispetto al grado in N e quindi il nodo in cima all'heap è uno di quelli che hanno il massimo grado in N. Poi è necessario un vettore P tale che, per ogni nodo u, $P[u]$ è la posizione di u in H e se u non è in H allora $P[u] = -1$. L'insieme D può essere rappresentato tramite una semplice lista. Ad ogni iterazione del WHILE l'estrazione dall'heap costa $\mathrm{O}(\log |V|)$. L'aggiunta di u a D ha costo costante. L'eliminazione di u e dei suoi adiacenti da N, cioè da H, può essere fatta scorrendo la lista degli adiacenti di u nel grafo e per ogni adiacente v, se $P[v] \neq -1$ si procede all'eliminazione di v da H, conoscendo la sua posizione $P[v]$. Inoltre, bisogna anche aggiornare il grado dei nodi adiacenti a v. Per questo si scorre la lista degli adiacenti a v e per ogni adiacente che appartiene ad H si decrementa il grado e conseguentemente si aggiorna la sua posizione in H. Ogni eliminazione ed ogni aggiornamento di grado ha costo $\mathrm{O}(\log |V|)$. Tenendo conto che ogni nodo viene eliminato una sola volta e che ogni arco del grafo può dar luogo ad al più un aggiornamento di grado, il costo di tutte le eliminazioni e tutti gli aggiornamenti è limitato da $\mathrm{O}((|V| + |E|) \log |V|)$. Quindi la complessità asintotica di questa implementazione è $\mathrm{O}((|V| + |E|) \log |V|)$.

Esercizio 2 Si consideri il seguente problema. Abbiamo una valigia che regge fino ad un peso p intero positivo, n tipi di oggetti e per ogni tipo $i = 1, 2, \ldots, n$ una coppia di interi positivi (p_i, v_i) dove p_i è il peso e v_i il valore di oggetti di tipo i; vogliamo riempire la valigia massimizzando la somma dei valori degli oggetti nella valigia ma senza sfondarla. Per ogni tipo è disponibile una fornitura illimitata di oggetti. Ad esempio, se la valigia regge fino a 19 e ci sono 3 tipi di oggetti $(5, 20)$, $(3, 9)$, $(2, 5)$ una soluzione ottima, di valore 70, è riempire la valigia con tre oggetti di tipo 1 e due di tipo 3.

1. Descrivere un algoritmo di programmazione dinamica che calcola il *valore* di una soluzione ottima per il problema.

2. Dimostrare la correttezza dell'algoritmo o almeno spiegare perché è corretto.

3. Modificare l'algoritmo cosicché produca anche una soluzione ottima, cioè, un vettore SOL tale che $SOL[i]$ è il numero di oggetti di tipo i presi.

4. Valutare la complessità dell'algoritmo proposto.

Soluzione Esercizio 2

1. Definiamo $S(h)$, con $0 \leq h \leq p$, come il valore massimo che si può ottenere disponendo degli n tipi di oggetti e di una valigia che regge fino ad un peso h.

   I valori di S sono calcolati ricorsivamente come segue:
   $S(0) = 0$, mentre per $h > 0$ si ha

   $$S(h) = \begin{cases} 0 & \text{se gli oggetti hanno tutti un peso superiore ad } i \\ \max_{1 \leq j \leq n,\, p_j \leq h}\{S(h - p_j) + v_j\} & \text{altrimenti.} \end{cases}$$

   Il valore della soluzione ottima è $S(p)$, ecco lo pseudo-codice dell'algoritmo che la calcola:

   VALIGIA: INPUT il numero n di oggetti, il peso p che la valigia regge, i vettori P e V con il peso ed il valore degli oggetti
   $S[0] \leftarrow 0$
   FOR $h = 1$ to p DO
   $\quad S[h] \leftarrow 0$
   $\quad$ FOR $j = 1$ to n DO
   $\quad\quad$ IF $h \geq P[j]$ AND $S[h] < S[h - P[j]] + V[j]$ THEN $S[h] \leftarrow S[h - P[j]] + V[j]$
   $\quad$ ENDFOR
   ENDFOR
   OUTPUT $S[p]$

2. È sufficiente far veder che i valori $S(h)$ sono calcolati in modo corretto. Ciò può essere dimostrato per induzione su h. Per $h = 0$ il calcolo è banalmente corretto. Supponiamo che sia corretto per ogni $0 \leq k < h$ e dimostriamolo corretto per h. Se tutti gli oggetti hanno peso maggiore di h allora il calcolo è corretto, cioè, $S(h) = 0$. Altrimenti, dimostriamo che $S(h) = \max_{1 \leq j \leq n,\, p_j \leq h}\{S(h-p_j)+v_j\}$. Infatti, se $p_j \leq h$ allora per ipotesi induttiva $S(h - p_j)$ è il valore ottimo per una valigia che regge fino a $h - p_j$, quindi $S(h-p_j)+v_j$ è un valore possibile per una valigia che regge fino ad h. Ciò prova che $S(h) \leq \max_{1 \leq j \leq n,\, p_j \leq h}\{S(h-p_j)+v_j\}$. Volendo provare la disuguaglianza opposta, consideriamo una soluzione ottima SOL^* per una valigia che regge fino ad h, cioè il valore di SOL^* è $S(h)$. Sia ℓ un tipo di oggetto che fa parte di SOL^*, allora SOL^* consiste in ℓ e una soluzione ottima per una valigia che regge $h - p_\ell$. Per ipotesi induttiva il valore di quest'ultima soluzione è $S(h - p_\ell)$. Ne segue che $S(h) = S(h - p_\ell) + v_\ell$ e ovviamente $p_\ell \leq h$. Dunque $S(h) \geq \max_{1 \leq j \leq n,\, p_j \leq h}\{S(h - p_j) + v_j\}$.

3. Per produrre anche il vettore SOL con il numero di ciascuno dei diversi oggetti presenti nella soluzione ottima, definiamo un vettore T di p elementi dove $T[h]$ contiene l'indice j di un oggetto tale che $S(h) = S(h - p_j) + v_j$ e contiene il valore 0 se la soluzione non contiene alcun oggetto (i.e. ha valore 0). In questo modo ripercorrendo all'indietro il vettore T sarà possibile ottenere gli oggetti scelti. Più precisamente modifichiamo l'algoritmo come segue:

   VALIGIA1: INPUT il numero n di oggetti, il peso p che la valigia sopporta, i vettori P e V con il peso ed il valore degli oggetti
   $S[0] \leftarrow 0$
   FOR $h = 1$ to p DO
       $S[h] \leftarrow 0$
       $T[h] \leftarrow 0$
       FOR $j = 1$ to n DO
           IF $h > P[j]$ AND $S[h] < S[h - P[j]] + V[j]$ THEN
               $S[h] \leftarrow S[h - P[j]] + V[j]$
               $T[h] \leftarrow j$
           ENDIF
       ENDFOR
   ENDFOR
   FOR $j = 1$ to n DO $SOL[j] \leftarrow 0$
   $h \leftarrow p$
   WHILE $T[h] > 0$ DO
       $SOL[T[h]] \leftarrow SOL[T[h]] + 1$
       $h \leftarrow h - P[T[h]]$
   ENDWHILE
   OUTPUT $S[p]$, SOL

4. Nell'algoritmo sono presenti due FOR annidati, il primo viene eseguito p volte ed il secondo n volte. Seguono poi, un FOR eseguito n volte ed un WHILE eseguito al più p volte. La complessità dell'algoritmo è quindi $O(np)$.