Esercitazione 08/05/2017

The Party Alice vuole organizzare la festa del secolo. Ha n amici e per ogni coppia di amici sa se questi si conoscono o meno. Per essere una festa di successo, sia k un dato intero, Alice deve invitare un sottoinsieme dei suoi amici tali che ogni invitato conosce almeno altri k invitati (per non sentirsi fuori contesto) e non conosce almeno k invitati (per poter fare nuove amicizie). Dare un algoritmo (ed una dimostrazione della sua ottimalità) che massimizza il numero di invitati di Alice.

Soluzione Cercare di trovare una condizione sufficiente per determinare se un amico sarà invitato alla festa è complicato, mentre è più semplice ragionare riguardo alle condizioni necessarie, che invece ci permettono di escludere alcuni amici dalla festa con sicurezza. Gli amici di Alice, e le relazioni tra essi possono essere rappresentati come un grafo $G(V, E)$. Sia $\bar{G}(\bar{V}, \bar{E})$ il grafo complemento di G ($\bar{V} = V$ e $u \in \bar{E}$ se e solo se $u \notin E$). Sia S un qualsiasi sottografo di G e $n_S(u)$ il numero di vicini di u in S. Affinché S sia una soluzione **feasible** deve valere che

$$\forall u \in S, n_S(u) \geq k \text{ e } n_{\bar{S}}(u) \geq k.$$

La chiave dell'esercizio sta nel seguente lemma:

Lemma 1. *Sia G un qualsiasi grafo, u un nodo di G tale che $n_G(u) < k$ e $S \subset G$ allora $n_S(u) < k$.*

Proof. Gli archi di S sono anch'essi un sottoinsieme degli archi di G quindi il numero di vicini di u in S non può essere maggiore dei vicini che ha in G. □

Siamo pronti a definire e dimostrare un algoritmo greedy per il problema di Alice.

Algorithm 1: *greedyAlg*

```
Data: G(V, E)
Result: S ⊂ G
1 begin
2     S = G
3     while S non è feasible do
4         Scegli u ∈ S tale che n_S(u) < k or n_S(u) > n − k − 1
5         S = S − {u}
6     return S
```

L'ottimalità dell'algoritmo deriva dal fatto che S non contiene tutti e soli i nodi che non sarebbero potuti stare in nessuna soluzione feasible.

Fractional Knapsack Problem La tua ditta di trasporti possiede un camion che può trasportare al più $k \in \mathbb{R}$ chilogrammi. Bisogna decidere come riempire questo camion, scegliendo tra gli n prodotti disponibili. Ogni prodotto i pesa p_i e vale v_i. Trattandosi di prodotti divisibili, è possibile caricare sul camion una qualsiasi frazione $x_i \in [0, 1]$ del prodotto i, ovviamente a patto che $\sum_i x_i \cdot p_i \leq k$. Dare un algoritmo (e una dimostrazione della sua ottimalità) che massimizzi il valore del carico $\sum_i x_i \cdot v_i$.

Soluzione L'algoritmo greedy corretto ordinare tutti i prodotti per valori $\frac{v_i}{p_i}$ decrescenti e inserisce i prodotti sul camion fino ad arrivare al peso k. Eventualmente l'ultimo prodotto inserito può essere frazionato.

Algorithm 2: *greedyAlg2*

```
   Data: (v_1, p_1), ... (v_n, p, n)
   Result: x_1, ..., x_n
1  begin
2      spazio_libero = k
3      i=0
4      Ordina i prodotto per valori v_i/p_i decrescenti
5      while spazio_libero > 0 do
6          if p_i ≤ spazio_libero then
7              x_i = 1
8          else
9              x_i = spazio_libero /p_i
10         i = i+1
```

Utilizziamo ora l'Exchange Argument per dimostrare l'ottimalità dell'algoritmo.

Proof. Sia $X = x_1, x_2, \ldots, x_n$ la soluzione ordinata restituita dal nostro algoritmo, e sia $O = \{o_1, o_2, \ldots, o_n\}$ una soluzione ottima anch'essa ordinata per $\frac{v_i}{p_i}$ decrescenti. Sia h il primo indice in cui $x_h \neq o_h$. Per costruzione il nostro algoritmo sceglierà x_h il più grande possibile, quindi abbiamo che $x_h < o_h$. Allora definiamo una nuova soluzione O' tale che $\forall j \leq h, o'_j = o_j = x_j$ (se prima O e X compievano le stesse scelte fino al passo $h-1$ adesso O' compie le stesse scelte di X fino al passo h).

Le scelte della nuova soluzione O' per gli indici $j > h$ vanno modificate perché incrementando la frazione di prodotto h scelta rispetto a O abbiamo aggiunto un peso uguale a $(x_h - o_h)p_h$. Sottraiamo allora dai successivi $j > h$ una frazione δ_j tale che $\delta_j < o_j$ e

$$\sum_{j>h} \delta_j p_j = (x_h - o_h)p_h \tag{1}$$

Dimostriamo ora che non peggioriamo la soluzione.

$$\begin{aligned}
\sum_i o'_i \cdot v_i - \sum_i o_i \cdot v_i &= (x_h - o_h)v_h - \sum_{j>h} \delta_j v_j \\
&= p_h(x_h - o_h)\frac{v_h}{p_h} - \sum_{j>h} \delta_j p_j \frac{v_j}{p_j} \\
&> p_h(x_h - o_h)\frac{v_h}{p_h} - \frac{v_h}{p_h}\sum_{j>h} \delta_j p_j \qquad (2) \\
&= 0 \qquad (3)
\end{aligned}$$

Dove in (2) abbiamo usato che $\forall j > h$ allora $\frac{v_h}{p_h} > \frac{v_j}{p_j}$ e in (3) abbiamo usato (1). Adesso è sufficiente argomentare che è possibile prendere una soluzione ottima generica O e trasformarla iterativamente nella soluzione restituita dal nostro algoritmo. □