Esercizi per il Corso di Algoritmica 2 (a.a. 2013/14)

Roberto Grossi
Dipartimento di Informatica, Università di Pisa
grossi@di.unipi.it

24 novembre 2013

Sommario

Vengono raccolti i problemi discussi in classe e collegati agli argomenti a lezione. Tali problemi vengono approfonditi e osservati da più punti di vista, anche sbagliati, in quanto l'errore è funzionale all'apprendimento di situazioni complesse. La motivazione risiede nel fatto che interessa sviluppare il percorso che conduce alla soluzione (piuttosto che la soluzione stessa), sotto la guida del docente in base alle idee proposte dagli studenti. La soluzione non viene qui fornita per i motivi suddetti: è preferibile venire a ricevimento dal docente.

1. [Sottografi K_4 e C_4] Descrivere un algorimo per elencare e contare, dato un grafo non orientato, tutti i suoi sottografi di tipo K_4 (grafo completo di 4 vertici) e C_4 (ciclo di vertici che non sono collegati da altri archi tra di loro). Valutare e motivare la complessità della soluzione proposta.

2. [Clique massimali] Una clique massimale è un sottoinsieme S dei vertici del grafo tale che S induce un grafo completo mentre per ogni altro vertice $u \notin S$ vale che $S \cup \{u\}$ non induce un grafo completo (massimalità). Descrivere un algoritmo che elenchi tutte le clique massimali di un grafo non orientato con un costo che dipende dalla dimensione dell'uscita (output-sensitive): se C è l'insieme delle clique massimali del grafo con n vertici e m archi, il costo temporale deve essere limitato superiormente da $|C| \times (m + n)^{O(1)}$.

3. [Cammini tra due vertici] Prendendo il metodo della partizione binaria visto a lezione per elencare tutti i cammini semplici tra due vertici distinti s e t di un grafo non orientato, mostrare come modificarlo in modo da garantire che ogni chiamata ricorsiva conduca ad almeno un nuovo cammino elencato (vogliamo quindi evitare chiamate ricorsive che non generino alcun cammino da elencare). Valutare e motivare la complessità della soluzione proposta.

4. [Cammini più brevi tra due vertici] Descrivere un algorimo per elencare tutti i cammini più brevi (shortest path) tra due vertici distinti s e t di un grafo non orientato (volendo, pesato o meno). Valutare e motivare la complessità della soluzione proposta.

5. [Minimal feedback vertex set] Nell'algoritmo per elencare i minimal feedback vertex set (MFVS) visto a lezione, dimostrare che il meta-grafo risultante è fortemente connesso, ossia, per ogni due meta-vertici esiste sempre un cammino diretto che li collega, dove ogni arco è rappresentato dalla funzione successore μ: questa proprietà serve a garantire che da qualunque meta-vertice partiamo, riusciamo a raggiungere tutti gli altri meta-vertici. Dimostrare se la funzione successore μ sia iniettiva o meno; in base a ciò, mostrare come elencare tutti i MFVS attraverso una visita di tutti i meta-vertici senza però costruire e memorizzare esplicitamente il meta-grafo. Valutare e motivare la complessità della soluzione proposta. Suggerimento: non riuscendo a dimostrare se μ sia iniettiva o meno, descrivere come generare tutti i MFVS ipotizzando che (i) μ sia iniettiva e (ii) μ non sia iniettiva (qui usare un dizionario e una coda di priorità per memorizzare i MFVS/meta-vertici).

6. [Cammino/ciclo euleriano] Descrivere un algoritmo per trovare, in tempo lineare, un cammino o un ciclo euleriano in un grafo non orientato e connesso (suggerimento: utilizzare il fatto che nei vertici di grado pari è possibile sempre uscire una volta che si è entrati per trovare un cammino parziale, che viene esteso attraverso cicli che devono essere collegati al cammino perché il grafo è connesso).

7. [Ricoprimento minimo di vertici] Per il problema del minimum vertex cover (MVC), trovare un contro-esempio che mostri come la strategia greedy di scegliere sempre il vertice di grado massimo e iterando sul grafo residuo ottenuto cancellando tale vertice e i suoi vicini, non garantisca una 2-approssimazione. Generalizzare tale argomento per mostrare che tale strategia greedy non garantisce una r-approssimazione per una qualunque costante prefissata $r > 1$.

8. [Approssimazione per MAX-SAT] Per il problema MAX-SAT della soddisfacibilità di una formula booleana, si consideri il seguente algoritmo di approssimazione per massimizzare il numero di clausole soddisfatte in una data formula: Sia F la formula data, $x_1, x_2, \ldots, x_n$ le variabili booleane in essa contenute, e $c_1, c_2, \ldots c_m$ le sue clausole. Scegli i valori booleani casuali $b_1, b_2, \ldots, b_n$, ossia ciascun $b_i \in \{0, 1\}$ $(1 \leq i \leq n)$. Calcola il numero m_0 di clausole soddisfatte dall'assegnamento tale che $x_i := b_i$ $(1 \leq i \leq n)$. Calcola il numero m_1 di clausole soddisfatte dall'assegnamento tale che $x_i := \bar{b_i}$ $(1 \leq i \leq n)$, dove $\bar{b_i}$ indica la negazione di b_i. Se $m_0 > m_1$, restituisci l'assegnamento $x_i := b_i$ $(1 \leq i \leq n)$; altrimenti, restituisci l'assegnamento $x_i := \bar{b_i}$ $(1 \leq i \leq n)$.

   Dimostrare che il suddetto algoritmo è una r-approssimazione per MAX-SAT, indicando anche il valore di $r > 1$ (e motivando l'utilizzo di tale valore). Discutere se, in generale, la scelta di $b_1, b_2, \ldots, b_n$ possa influenzare o meno il valore di r, motivando le argomentazioni addotte. Facoltativo: creare un'istanza di MAX-SAT in cui il suddetto algoritmo ottiene un costo che è r volte più piccolo del costo ottimo per una data scelta dei valori di b_1, b_2 $\ldots$, b_n.

9. [Approssimazione per knapsack] Nell'algoritmo greedy di approssimazione per il problema dello zaino (knapsack) visto a lezione, mostrare che quando vale la condizione $\bar{p}_j = b$ (vedere Teorema 2.1 nel capitolo disponibile sul sito) tale algoritmo ottiene la solizione ottima.

10. [Approssimazione per MAX-CUT] Il problema MAX-CUT è NP-hard ed è definito come segue per un grafo non orientato $G = (V, E)$. Una partizione di nodi $(C, V - C)$ con $C \subseteq V$ si chiama "cut" o *taglio*. Un arco $e = (v, w)$ con $v \in C$ e $w \in V - C$ si chiama arco di taglio (ricordando che (v, w) e (w, v) denotano lo stesso arco in un grafo non orientato). Il numero di archi di taglio definisce la dimensione del cut $(C, V - C)$. Poiché cambiando taglio, può cambiare la sua dimensione, il problema richiede di trovare il taglio di dimensione *massima* e quindi gli archi di taglio corrispondenti. Dimostrare che il seguente algoritmo randomizzato è una 2-approssimazione in valore atteso, ossia che il numero medio di archi di taglio così individuati è in media almeno la metà di quelli del taglio massimo. *(1)* Per ogni nodo $v \in V$, lancia una moneta equiprobabile: se viene testa, inserisci v in C; altrimenti (viene croce), inserisci v in $V - C$. *(2)* Inizializza T all'insieme vuoto. Per ogni arco $(v, w) \in E$, tale che $v \in C$ e $w \in V - C$, aggiungi (v, w) all'insieme T. Restituisci C e T come soluzione approssimata.

11. [Famiglia di funzioni hash uniformi] Mostrare che la famiglia di funzioni hash $H = \{h(x) = ((ax + b)\% p)\% m\}$ è (quasi) "pairwise independent", dove $a, b \in [m]$ con $a \neq 0$ e p è un numero primo sufficientemente grande $(m + 1 \leq p \leq 2m)$. La pairwise independent è la k-wise independent vista a lezione dove $k = 2$.

12. [Count-min sketch: estensione] Estendere l'analisi vista a lezione permettendo di incrementare e decrementare i contatori con valori arbitrati. Invece di avere le operazioni $F[i]{++}$ e $F[i]{--}$, l'elemento generico dello strem contiene una coppia (i, v) dove i è una item e v è un intero qualsiasi: l'operazione diventa $F[i] = F[i] + v$. Notare che l'incremento e il decremento unitari di $F[i]$ possono essere ora visti come $(i, 1)$ e $(i, -1)$.

13. [Count-min sketch: prodotto scalare] Mostrare e analizzare come utilizzare il paradigma del count-min sketch per approssimare il prodotto scalare $\sum_{k=1}^{n} F_a[k] * F_b[k]$.

14. [Count-min sketch: interval query] Mostrare e analizzare come utilizzare il paradigma del count-min sketch per rispondere in modo approssimato alle interval query (i, j) per calcolare $\sum_{k=i}^{j} F[k]$.