Computabilità ed Algoritmi
Esempio di Esame di Algoritmi
25 Maggio 2018

Tempo a disposizione: 1 h 30 min

1. *Scale e Serpenti* è un gioco da tavolo tradizionale, nato in India e diffuso soprattutto nei paesi di lingua inglese (il nome originale è "snakes and ladders"). Si gioca su una scacchiera quadrata di dimensione $n \times n$ con le celle numerate consecutivamente da 1 a n^2, iniziando dalla cella in basso a sinistra e proseguendo riga per riga dal basso verso l'altro, alternando la numerazione delle righe verso sinistra e verso destra. Alcune coppie di celle, poste su righe diverse, sono collegate da "serpenti" che portano verso il basso, o da "scale" che portano verso l'alto, come nell'esempio sottostante. *Ogni casella può essere l'estremo di al massimo un serpente o una scala.*

Il gioco inizia posizionando una pedina nella cella 1. Ad ogni mossa, la pedina può avanzare di un numero di celle fino ad un massimo di k (tipicamente 6). Se al termine della mossa la pedina si trova nella cella *superiore* di un serpente, allora è *obbligata* a scivolare giù lungo il serpente fino alla cella inferiore. Se invece si trova nella cella *inferiore* di una scala, allora *può* arrampicarsi verso l'alto fino a raggiungere la cella superiore.

Descrivere ed analizzare un algoritmo efficiente per calcolare il numero minimo di mosse necessarie per raggiungere la cella finale.

Soluzione: Modelliamo il problema con un grafo diretto e non pesato $G = (V, E)$:

- I vertici di G sono le caselle della scacchiera, numerati da 1 a n^2;
- Gli archi corrispondono alle mosse valide. Per ogni vertice x e numero di caselle $j \in \{1, \ldots, k\}$ si aggiungono gli archi come segue:
  - (x, y) se $x + j$ è la cella superiore di un serpente con cella inferiore y;
  - $(x, x + j)$ e (x, y) se $x + j$ è la cella inferiore di una scala che porta a y;
  - $(x, x + j)$ in tutti gli altri casi.
- I vertici sono associati alle caselle come specificato. Non serve aggiungere informazione agli archi.
- Il problema da risolvere è trovare il cammino minimo da 1 a n^2.
- L'algoritmo da usare per risolvere il problema è BFS.
- Il tempo impiegato per costruire il grafo è proporzionale alla sua dimensione. Il grafo ha n^2 vertici e al più $2kn^2$ archi. Quindi la sua costruzione ha costo $O(kn^2)$. La complessità di BFS è $O(|V| + |E|) = O(n^2 + 2kn^2) = O(kn^2)$. Quindi l'algoritmo complessivamente impiega tempo $O(kn^2)$.

2. Un *albero ad anello* è un grafo orientato e pesato costruito aggiungendo ad ogni foglia di un albero binario un arco che porta alla radice. Tutti gli archi hanno peso non negativo.

(a) Quanto tempo impiega l'algoritmo di Dijkstra per calcolare il percorso più breve da un vertice u ad un vertice v in un albero ad anello con n nodi?

Soluzione: Un albero ad anello con n nodi ha al più $2n$ archi. Quindi l'algoritmo di Dijkstra impiega tempo $O((|V| + |E|) \log |V|) = O(n \log n)$ per trovare il cammino minimo.

(b) Descrivere e analizzare un algoritmo più veloce.

Soluzione: Se G è l'albero ad anello ed r è la radice, l'algoritmo per trovare il cammino minimo da u a v procede come segue:

1. Risale lungo l'albero partendo dal vertice v, finché non si trova u oppure si arriva alla radice.
2. Se si raggiunge prima u, allora abbiamo trovato l'unico cammino semplice da u a v e abbiamo terminato.
3. Se si raggiunge prima la radice, allora il cammino minimo da u a v è composto dal cammino minimo da u alla radice più il cammino dalla radice a v. Se Left(u) e Right(u) identificano rispettivamente il figlio sinistro ed il figlio destro di u, allora la seguente procedura trova il cammino minimo da u a r:

```
function MinToRoot(G, u)
    if (u, r) ∈ E then
        return w(u, r), [u, r]
    wleft, pathleft = MinToRoot(G, Left(u))
    wright, pathright = MinToRoot(G, Right(u))
    if w(u, Left(u)) + wleft < w(u, Right(u)) + wright then
        w = w(u, Left(u)) + wleft
        path = [u] + pathleft
    else
        w = w(u, Right(u)) + wright
        path = [u] + pathright
    return w, path
```

La complessità del punto 1. dell'algoritmo è lineare rispetto al numero di nodi n nell'albero. La complessità del punto 3. dipende dal tempo impiegato da MinToRoot. Poiché la procedura effettua una visita in profondità dell'albero, fermandosi quando arriva ad una foglia, allora la sua complessità è $O(n)$. Quindi l'algoritmo proposto impiega tempo $O(n)$ per trovare il cammino minimo da u a v.

3. Dimostrare che un problema X è NP-hard richiede diversi passaggi:

- Scegli un problema Y che sai essere NP-hard (perché l'hai visto a lezione).
- Descrivi un algoritmo per risolvere Y usando un algoritmo per X come subroutine. Tipicamente questo algoritmo ha la seguente forma: data un'istanza di Y, trasformala in un'istanza di X, quindi chiama l'algoritmo magico black-box per X.
- Dimostra che l'algoritmo è corretto. Ciò richiede sempre due passaggi separati, che di solito hanno la seguente forma:
  - Dimostra che il tuo algoritmo trasforma istanze "buone" di Y in istanze "buone" di X.
  - Dimostra che il tuo algoritmo trasforma istanze "cattive" di Y in istanze "cattive" di X. Equivalentemente: Dimostra che se la tua trasformazione produce un'istanza "buona" di X, allora era partita da un'istanza "buona" di Y.
- Mostra che il tuo algoritmo per Y funziona in tempo polinomiale, a meno della chiamata (o delle chiamate) all'algoritmo magico black-box per X. (Questo di solito è banale.)

Un circuito Hamiltoniano in un grafo G è un ciclo che attraversa ogni vertice di G esattamente una volta. Stabilire se un grafo arbitrario contiene un circuito Hamiltoniano è un problema NP-hard.

Un *circuito toniano* in un grafo G è un ciclo che attraversa almeno *la metà dei vertici* di G esattamente una volta. Dimostrare che stabilire se un grafo contiene un circuito toniano è un problema NP-hard.

Soluzione: Il problema NP-hard scelto per la riduzione è quello del circuito Hamiltoniano. Sia $G = (V, E)$ un grafo arbitrario e supponiamo che G abbia n vertici. Costruiamo un grafo H aggiungendo n nuovi vertici isolati a G (senza nessun arco aggiuntivo). Dimostriamo che G possiede un circuito Hamiltoniano se e solo se H ha un circuito toniano.

- $\Rightarrow$ Supponiamo che G possiede un circuito Hamiltoniano C. Quindi C visita esattamente la metà dei vertici di H, e di conseguenza è un circuito toniano per H.
- $\Leftarrow$ Supponiamo che H abbia un ciclo toniano C. Questo ciclo non può visitare nessuno dei nuovi vertici, quindi deve giacere interamente all'interno del sottografo G. Poiché G contiene esattamente la metà dei vertici di H, il ciclo C deve visitare ogni vertice di G, e quindi è un ciclo hamiltoniano in G.

Per concludere, dato G per costruire H dobbiamo aggiungere gli n nuovi vertici. Questa operazione si può eseguire in tempo polinomiale (più precisamente in tempo $O(n)$).