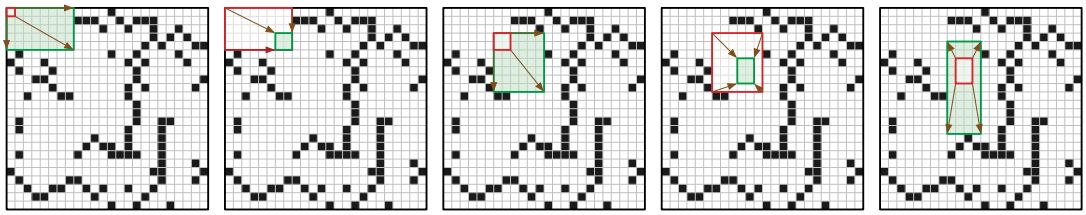
Computabilità ed Algoritmi
Esame di Algoritmi
21 Febbraio 2019

Tempo a disposizione: 2 h 00 min

1. *Rectangle Walk* è un rompicapo che si gioca su una griglia $n \times n$ di quadrati bianchi e neri. Il giocatore sposta un rettangolo attraverso questa griglia, rispettando le seguenti condizioni:

- Il rettangolo deve essere allineato con la griglia: le coordinate dei vertici del rettangolo devono essere numeri interi.
- Il rettangolo deve rimanere all'interno della griglia $n \times n$ e deve contenere almeno una cella della griglia.
- Il rettangolo non deve contenere quadrati neri.
- In una singola mossa, il giocatore può sostituire il rettangolo r corrente con qualsiasi rettangolo r' che contiene r o è contenuto in r.

All'inizio della partita il rettangolo è un quadrato 1×1 posto nell'angolo in alto a sinistra della griglia. L'obiettivo del giocatore è di raggiungere il quadrato 1×1 nell'angolo in basso a destra usando il minor numero possibile di mosse.



Le prime cinque mosse di una partita a Rectangle Walk.

Considera il problema di trovare il numero minimo di mosse necessarie per risolvere il rompicapo, o di riportare correttamente che il rompicapo non ha soluzione. L'input del problema è costituto da una matrice binaria $M[1 \ldots n, 1 \ldots n]$ dove $M[i, j] = 1$ indica un quadrato nero e $M[i, j] = 0$ indica un quadrato bianco. Per semplicità, puoi assumere che i quadrati di partenza e di arrivo siano bianchi (ossia, che $M[1, 1] = 0$ e $M[n, n] = 0$). Nell'esempio mostrato nella figura, il tuo algoritmo dovrebbe ritornare il valore 18.

Fornisci le seguenti informazioni:

- Quali sono i vertici? Cosa rappresenta ciascun vertice?
- Quali sono gli archi? Sono orientati o non orientati?
- Se i vertici e/o gli archi hanno dei valori associati, quali sono questi valori?
- Quale problema si deve risolvere sul grafo?
- Quale algoritmo si può usare per risolvere il problema?
- Qual è il tempo di esecuzione dell'intero algoritmo, *incluso* il tempo di creazione del grafo, *in funzione dei parametri di input originali*?

Modelliamo il problema con un grafo non orientato $G = (V, E)$:

- I vertici di G sono quadruple (x_1, y_1, x_2, y_2) che rappresentano le coordinate dei vertici in alto a sinistra e in basso a destra del rettangolo. I valori delle coordinate variano tra 0 e n. Il grafo comprende solo le quadruple che rappresentano rettangoli che non contengono quadrati neri della griglia.
- Gli archi corrispondono alle mosse valide, e non sono pesati. C'è un arco da (x_1, y_1, x_2, y_2) a (x'_1, y'_1, x'_2, y'_2) se e solo se $(x_1, y_1, x_2, y_2) \neq (x'_1, y'_1, x'_2, y'_2)$ e vale una delle seguenti condizioni:
  - $x'_1 \leq x_1$, $y'_1 \leq y_1$, $x_2 \leq x'_2$ e $y_2 \leq y'_2$ (il primo rettangolo è contenuto nel secondo)
  - $x'_1 \geq x_1$, $y'_1 \geq y_1$, $x_2 \geq x'_2$ e $y_2 \geq y'_2$ (il secondo rettangolo è contenuto nel primo)
- I vertici sono associati ai possibili rettangoli validi come specificato. Gli archi non sono pesati.
- Il problema da risolvere è trovare il cammino minimo da $(0, 0, 1, 1)$ (rettangolo iniziale) al vertice $(n-1, n-1, n, n)$ che rappresenta il rettangolo di arrivo.
- L'algoritmo da usare per risolvere il problema è BFS.
- Il numero dei vertici del grafo dipende dai valori che le variabili x_1, y_1, x_2, y_2 possono assumere. Siccome le coordinate dei vertici del rettangolo variano tra 0 e n, allora ci sol al più

$O(n^4)$ vertici nel grafo. Per sapere se un vertice va inserito oppure no devo controllare la presenza di quadrati neri all'interno del rettangolo, operazione che costa $O(n^2)$. Quindi, costruire i vertici del grafo richiede tempo $O(n^6)$. Da ogni vertice possono partire fino a $O(n^4)$ archi: il grafo delle mosse è un grafo denso con $O(n^4)$ vertici e $O(n^8)$ archi. La complessità dell'algoritmo BFS è $O(|E| + |V|) = O(n^4 + n^8) = O(n^8)$. Costruire il grafo richiede meno tempo di BFS, quindi la complessità totale della soluzione proposta è $O(n^8)$.

2. Dopo esserti trasferito in una nuova città, decidi percorrere a piedi il percorso da casa tua al tuo nuovo luogo di lavoro. Per consentirti di fare un buon esercizio quotidiano, il percorso deve essere costituito da un primo tratto in salita seguito da un secondo tratto in discesa, oppure solo da un percorso in salita, o solo da un percorso in discesa. (Farai lo stesso percorso per tornare a casa, in modo da fare comunque esercizio in un modo o nell'altro.) Vuoi trovare il percorso più breve che soddisfi queste condizioni, in modo che tu possa arrivare al lavoro in orario.

L'input del tuo problema consiste in un grafo G non orientato, i cui vertici rappresentano le intersezioni stradali e gli archi rappresentano i segmenti stradali, assieme al vertice di partenza s e al vertice di destinazione t. Ogni vertice v è associato ad un valore $h(v)$, che è l'altezza di quell'intersezione sul livello del mare, mentre ogni arco $\{u, v\}$ è associato ad valore $l(u, v)$, che è la lunghezza di quel segmento di strada.

(a) Descrivi un algoritmo per trovare il percorso più breve in salita-discesa, oppure solo in salita o solo in discesa, da s a t. Assumi che i vertici del grafo abbiano altezze tutte diverse tra di loro.
(b) Dimostra che l'algoritmo proposto è corretto.
(c) Analizza la complessità dell'algoritmo.

L'input del problema è un grafo non orientato: questo significa che ogni arco può essere percorso in entrambe le direzioni. Data l'assunzione che i vertici del grafo hanno altezze tutte diverse tra loro, non esistono archi "in piano" e ogni arco può essere percorso in salita oppure in discesa. Una delle soluzioni più semplici, e tuttavia ottimale in termini di complessità asintotica, è di utilizzare due chiamate alla versione modificata dell'algoritmo di Dijkstra che percorre gli archi solamente in salita:

```
function ModifiedDijkstra(G, s)
    InitSSSP(s)
    Q = PriorityQueue(V)
    while not Empty(Q) do
        u = ExtractMin(Q)
        for all v ∈ Adj[u] do
            if h(u) < h(v) and d[u] + l(u, v) < d[v] then
                Relax(u, v)
                DecreaseKey(Q, v, d[v])
```

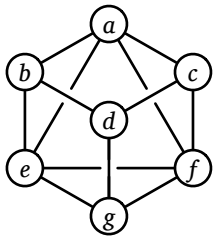
La prima chiamata ha come nodo sorgente s, e ritorna i vettori $d_s[u]$ e $\pi_s[u]$ con le informazioni sui cammini minimi in salita che iniziano da s. La seconda chiamata usa come nodo sorgente il nodo destinazione t, e ritorna i vettori $d_t[u]$ e $\pi_t[u]$. Siccome gli archi non sono orientati, allora ogni cammino in salita da t ad un qualsiasi altro vertice v del grafo è anche un cammino in discesa da v a t. Di conseguenza, dato un vertice $v \in V$, il valore $d_s[v] + d_t[v]$ è uguale alla lunghezza del cammino più corto in salita-discesa da s a t che ha v come nodo di altezza massima. Per trovare il cammino più corto in salita-discesa da s a t basta quindi scorrere tutti i nodi del grafo per trovare il nodo dove $d_s[v] + d_t[v]$ è minimo. Usando le informazioni contenute in $\pi_s[v]$ e $\pi_d[v]$ è possibile costruire la lista dei nodi che compongono il cammino. Notate che per il nodo sorgente s abbiamo che $d_s[s] = 0$ mentre $d_t[s]$ contiene la lunghezza del cammino in discesa più breve da s a t. Viceversa, per il nodo destinazione t abbiamo che $d_t[t] = 0$ mentre $d_s[t]$ contiene la lunghezza del cammino in salita più breve da s a t. Quindi iterare tra tutti i nodi come descritto sopra permette di trovare anche il cammino solamente in salita o solamente in discesa.

La complessità computazionale dell'algoritmo è data dalla somma delle complessità delle due chiamate a ModifiedDijkstra della complessità dell'iterazione per trovare il nodo con $d_s[v] + d_t[v]$ e del tempo necessario per costruire il cammino a partire da $\pi_s[v]$ e $\pi_d[v]$. Se implementato con una heap binaria, l'algoritmo di Dijkstra modificato richiede tempo $O((|V| + |E|) \log |V|)$, mentre la ricerca del vertice minimo e la costruzione del cammino richiedono tempo $O(|V|)$. La complessità di ModifiedDijkstra domina rispetto ad $O(|V|)$, e di conseguenza abbiamo dimostrato che l'algoritmo proposto ha complessità asintotica $O((|V| + |E|) \log |V|)$.

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

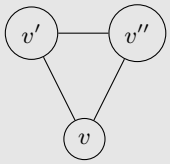
- Scegli un problema Y che sai essere NP-hard.
- Descrivi un algoritmo per risolvere Y usando un algoritmo per X come subroutine. Tipicamente questo algoritmo ha la seguente forma: data un'istanza di Y, trasformala in un'istanza di X, quindi chiama l'algoritmo magico black-box per X.
- Dimostra che l'algoritmo 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 *Doppio Circuito Hamiltoniano* in un grafo non orientato G è un ciclo che visita ogni vertice di G esattamente due volte. Dimostra che determinare se un grafo possiede un doppio circuito Hamiltoniano è un problema NP-Hard.



Questo grafo contiene il doppio circuito Hamiltoniano
$a \to b \to d \to g \to e \to b \to d \to c \to f \to a \to c \to f \to g \to e \to a$.

Dimostriamo che il problema è NP-hard con usando il problema del circuito Hamiltoniano standard come problema di riferimento. Dato un grafo $G = (V, E)$ non orientato, costruiamo un grafo H aggiungendo due vertici v' e v'' per ogni vertice $v \in V$, collegati tra loro dagli archi $\{v, v'\}, \{v', v''\}$ e $\{v'', v\}$:



Dimostriamo che esiste un circuito Hamiltoniano in G se e solo se esiste un doppio circuito Hamiltoniano in H:

$\Rightarrow$ Supponiamo che $v_1 v_2 \dots v_n v_1$ sia un circuito Hamiltoniano per G. Possiamo costruire un doppio circuito Hamiltoniano per H sostituendo ogni vertice v_i con il cammino:

$$\dots \to v_i \to v_i' \to v_i'' \to v_i' \to v_i'' \to v_i \to \dots$$

$\Leftarrow$ Viceversa, supponiamo che H abbia un doppio circuito Hamiltoniano D. Considerate un qualsiasi vertice v del grafo originale G: il circuito D deve visitare v esattamente due volte. Queste due visite dividono D in due cicli, ognuno dei quali visita v esattamente una volta. Qualsiasi cammino da v' o v'' a qualsiasi altro vertice in H deve passare attraverso v. Quindi, uno dei due cicli visita solo i vertici v, v' v''. Se rimuoviamo da D i vertici che non appartengono a G, otteniamo un ciclo in G che visita una sola volta ogni vertice in G, ossia un circuito Hamiltoniano.

Dato qualsiasi grafo G, possiamo costruire chiaramente il grafo H corrispondente in tempo polinomiale.