Computabilità ed Algoritmi
Esame di Algoritmi
6 Settembre 2018

Tempo a disposizione: 2 h 00 min

1. *Racetrack* è un gioco che simula una corsa automobilistica con carta e matita. Si gioca su una pista disegnata su un foglio di carta millimetrata. I giocatori scelgono alternativamente una sequenza di punti della griglia che rappresentano il movimento di un'auto sulla pista, soggetti a determinati vincoli spiegati di seguito.

Ogni auto ha una posizione e una velocità, entrambe con coordinate X e Y intere. Un sottoinsieme dei quadretti della griglia è contrassegnato come area di partenza e un altro sottoinsieme viene contrassegnato come area di arrivo. La posizione iniziale di ogni auto è scelta dal giocatore da qualche parte nell'area di partenza; la velocità iniziale di ogni auto è sempre $(0, 0)$. Ad ogni passo, il giocatore può modificare ciascun componente della velocità di al massimo di 1 unità. La nuova posizione dell'auto viene quindi determinata aggiungendo la nuova velocità alla posizione precedente dell'auto. La nuova posizione deve essere all'interno della pista; altrimenti, l'auto esce dalla pista e il giocatore perde la gara. La gara termina quando la prima vettura raggiunge una posizione all'interno dell'area di arrivo.

Supponete che la pista sia rappresentata da una matrice binaria $n \times n$, dove ogni 0 rappresenta un punto della griglia all'interno della pista, ogni 1 rappresenta un punto della griglia al di fuori della pista, la "linea di partenza" consiste di tutti i bit 0 nella colonna 1 e la "linea di arrivo" consiste di tutti i bit 0 nella colonna n (le posizioni nella matrice sono numerate da 1 a n).

Considerate il problema di trovare il numero minimo di mosse necessarie per spostare un'auto dalla linea di partenza al traguardo in una pista rappresentata come spiegato sopra. Trasformate l'input in un grafo e applicate uno degli algoritmi visti a lezione per risolverlo.

Fornite 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*?

Chiamiamo M la matrice che rappresenta la pista, e modelliamo il problema con un grafo diretto non pesato $G = (V, E)$:

- I vertici di G sono quadruple (x, y, vx, vy) che rappresentano posizione e velocità dell'auto: la coppia (x, y) è la posizione e (vx, vy) la velocità. Il grafo comprende solo i vertici con posizioni e velocità valide: $M[x, y] = 0$ (la posizione è all'interno della pista), $|vx| < n$, $|vy| < n$. In prima approssimazione possiamo limitare le velocità con la dimensione n della matrice. Infatti, se aggiungiamo o togliamo n ad una coordinata di posizione otteniamo un valore che è fuori dalla matrice.
- Gli archi corrispondono alle mosse valide. C'è un arco da (x, y, vx, vy) a (x', y', vx', vy') se e solo se valgono le seguenti condizioni:
  - $x' = x + vx'$ e $y' = y + vy'$ (la nuova posizione si ottiene sommando le nuove velocità alla posizione precedente);
  - $vx' \in \{vx - 1, vx + 1\}$ e $vy' = \{vy - 1 + vy + 1\}$ (ciascuna componente della velocità varia di al massimo di 1 unità).
- I vertici sono associati a posizione e velocità dell'auto come specificato. Gli archi non hanno informazione aggiuntiva.
- Il problema da risolvere è trovare il cammino minimo (nel numero degli archi) da uno dei vertici corrispondenti alla posizione iniziale dell'auto, cioè quelli del tipo $(1, y, 0, 0)$ ad uno dei vertici corrispondenti alla linea di arrivo, cioè quelli del tipo (n, y, vx, vy).
- Poiché gli archi sono privi di peso, la soluzione ingenua del problema effettua una serie di visite in ampiezza del grafo (BFS) che partono da tutte le posizioni iniziale. Per ottenere una soluzione più efficiente si può aggiungere un nodo "super-sorgente" v_0 al grafo, collegato da un arco con tutte le posizioni iniziali. In questo modo basta una sola visita in ampiezza con sorgente v_0 per risolvere il problema.
- Il tempo impiegato per costruire il grafo è proporzionale alla sua dimensione. Da ogni vertice

partono al più nove archi. Il numero dei vertici dipende dai valori che le variabili x, y, vx, vy possono assumere. Le variabili x e y variano nell'intervallo $[1, n]$, mentre vy e vy variano nell'intervallo $[-n+1, n-1]$. Di conseguenza, il grafo possiede $O(n^4)$ vertici $O(n^4)$ archi. Durante la costruzione, ogni volta che si aggiunge un vertice o un arco al grafo si deve accedere alla matrice M per stabilire se la posizione è all'interno della matrice. L'accesso alla matrice ha costo $O(1)$, quindi la costruzione del grafo ha un costo asintotico di $O(n^4)$. La complessità di BFS è $O(|V| + |E|) = O(n^4)$. Quindi l'intero algoritmo impiega tempo $O(n^4 m)$.

Nota: l'analisi di complessità precedente si può migliorare osservando che le velocità possono solo aumentare o diminuire di 1 ad ogni mossa. Quindi l'auto esce sicuramente dalla matrice quando la somma $1 + 2 + \cdots + v_{max}$ è maggiore o uguale a n. Quindi abbiamo che

$$n \leq \frac{v_{max}(v_{max} - 1)}{2} \leq \frac{{v_{max}}^2}{2}$$

Di conseguenza, possiamo limitare i valori di velocità all'intervallo $[-\sqrt{2n}, \sqrt{2n}]$, ottenendo un numero di vertici $O(n^3)$ per il grafo, e abbassando di conseguenza la complessità totale dell'algoritmo di soluzione del gioco.

2. *Connettività di una rete sociale.* Dato un social network contenente n membri e un file di log contenente m timestamp che identificano i momenti in cui le coppie di membri hanno formato amicizie, progettare un algoritmo per determinare il primo momento in cui tutti i membri sono connessi da catene di amicizie. Supponente che il file di registro sia ordinato per data e ora e che l'amicizia sia una relazione simmetrica. L'algoritmo deve avere un tempo di esecuzione $O(m \log n)$ o migliore e deve utilizzare uno spazio aggiuntivo $O(n)$.

L'algoritmo utilizza una struttura dati per insiemi disgiunti per costruire le componenti connesse del grafo delle amicizie. Supponiamo inoltre che ad ogni insieme disgiunto sia associato un attributo *size* che ritorna il numero di elementi nell'insieme. A questo punto, per risolvere il problema basta scorrere il file di log seguendo l'ordine dei timestamp, eseguendo una operazione di UNION sulle coppie di membri. Quando il valore di *size* dell'insieme ottenuto dall'unione diventa uguale a n l'algoritmo termina e ritorna il valore dell'ultimo timestamp. Lo pseudocodice seguente implementa l'algoritmo:

```
for u in 0, ..., n − 1 do MAKE-SET(u)
for (u, v, time) in logfile do        ▷ il logfile è formato da triple, dove i primi due valori sono membri della rete sociale e l'ultimo è il timestamp
    UNION(u, v)
    if size[FIND-SET(u)] == n then
        return time
return ∞                              ▷ grafo non connesso
```

L'algoritmo esegue n operazioni di MAKE-SET ed al più m operazioni di UNION e FIND-SET. La complessità dipende quindi dall'implementazione scelta per gli insiemi disgiunti. Utilizzando path compression e union by rank otteniamo un costo ammortizzato $O(\alpha(n))$ per UNION e FIND-SET(,) dove $\alpha(n)$ è una funzione che cresce molto più lentamente del logaritmo. MAKE-SET ha costo $O(1)$. Di conseguenza la complessità totale dell'algoritmo è $O(n+m\alpha(n))$. Supponendo che il numero di timestamp m nel file di log sia maggiore del numero n di membri della rete sociale, possiamo semplificare l'espressione a $O(m\alpha(n))$.

Lo spazio aggiuntivo richiesto dall'algoritmo è dato da un puntatore *parent* e dagli attributi *size* e *rank* per ognuno degli n elementi nella collezione. Quindi lo spazio aggiuntivo totale è limitato a $O(n)$.

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.)

Dato un grafo non orientato e pesato G, e due vertici u, v del grafo, il problema del cammino massimo (LongestPath) chiede di trovare il cammino semplice (senza ripetizioni di vertici) da u a v di peso massimo. Dimostrate che LongestPath è NP-hard.

Dimostriamo che LongestPath è NP-hard con una riduzione del problema del circuito Hamiltoniano, che sappiamo essere NP-hard.

Dato un grafo non orientato e non pesato $G = (V, E)$ di input per HamiltonianCircuit, lo trasformiamo in un grafo pesato $G' = (V, E)$ assegnando ad ogni arco $\{u, v\}$ peso $w[u, v] = 1$. E' facile dimostrare che G ha un circuito Hamiltoniano se e solo se il cammino più lungo che parte da un vertice arbitrario u e ritorna ad u ha peso $n - 1$. Poiché tutti gli archi di G' hanno peso 1, il peso di un cammino corrisponde al numero degli archi che lo compongono. Di conseguenza, un circuito Hamiltoniano avrà sempre peso $n - 1$, mentre qualsiasi cammino semplice che parte da u e ritorna in u di peso $n - 1$ deve passare per tutti i vertici del grafo (e quindi è un circuito Hamiltoniano).