Path Problems On Temporal Graphs

Astrazione

Lo `Shortest Path` è un problema fondamentale sui grafi.

Il concetto di shortest path classico è insufficiente, o addirittura imperfetto nei grafi temporali, dato che le informazioni temporali determinano l'ordine delle attività lungo qualsiasi percorso.

Calcolare questi `percorsi temporali` è impegnativo poiché i sottopercorsi di un percorso "corto" potrebbero non essere "corti" in un grafico temporale.

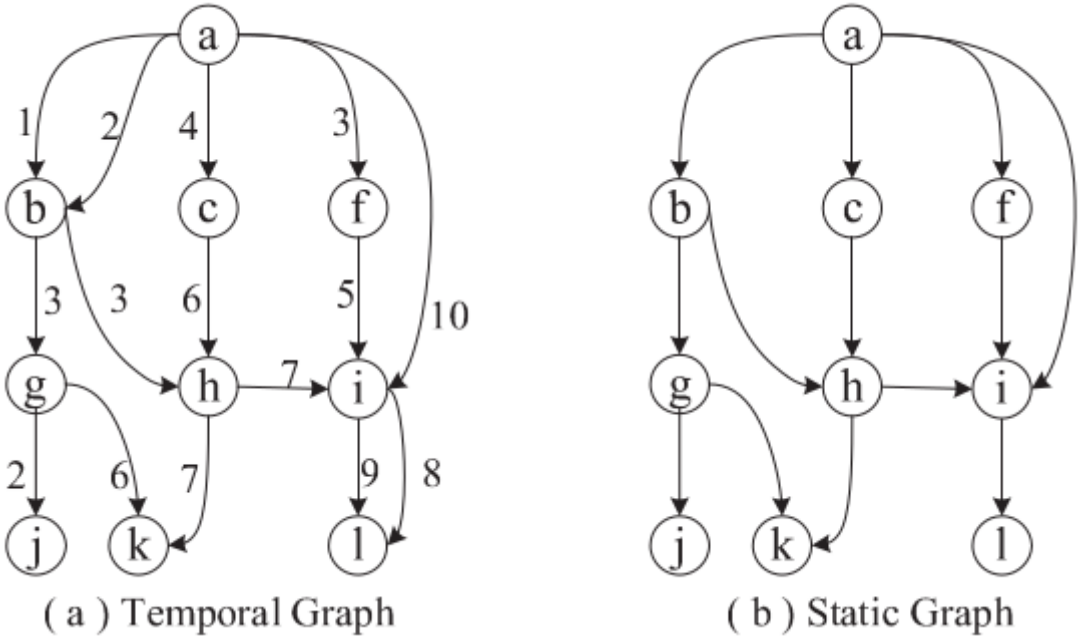

Figura 1

1. Introduzione

Molti grafi nel mondo reale sono **Grafi Temporali**, dove ogni vertice comunica con un'altro vertice ad un'istanza di tempo specifica.

Esempio :

Assumiamo che $\mathrm{Figura}\ 1(a)$ mostri un network di trasporto aereo, allora i due nodi a e b indicano che esiste un volo da a a b nel giorno 1 e nel giorno 2, infatti i numeri 1 e 2 sugli archi rappresentano il tempo di partenza dei voli.

Ci sono vari esempi di grafi temporali nella vita reale, come:

- Chiamate tra persone
- Social Network
- etc...

Ogni grafo temporale ha la sua versione `statica`, infatti i grafi temporali sono spesso condensati in quelli che si chiamano `grafi statici`, perchè questa versione è molto più semplice da gestire.

Esempio

Per calcolare le Componenti Fortemente Connesse ($SCCs$) di un grafo statico abbiamo un'algoritmo lineare, mentre per calcolare quelle di un grafo temporale **NON** esiste un'algoritmo polinomiale.

> ⚠ **Osservazione** >
>
> Si può notare che alcuni percorsi nel grafo statico potrebbero non essere un percorso completo nel grafo temporale.

Per esempio, $\langle a, b, g, j \rangle$ è un percorso in $\text{Figura } 1(b)$, ma $\langle a, b, g, j \rangle$ in $\text{Figura } 1(a)$ è problematico perchè g ha solo un volo per j al giorno 2 ma non lo possiamo raggiungere prima del giorno 4.

L'esempio soora indica che un grafo statico condensato può presentare informazioni fuorvianti relative al grafo temporale originale, e quindi è essenziale mantenere le informazioni temporali nei grafi.

Esistono 4 tipi di percorsi temporali, che collettivamente si chiamano **Minimum temporal paths** (percorsi temporali minimi), dato che loro danno il valore minimo per differenti misure:

1. `earliest-arrival path` : percorso che fornisce l'earliest arrival time partendo da una sorgente x fino a un target y
2. `latest-departure path` : percorso che fornisce il latest departure time partendo da sorgente x fino a un target y entro un certo tempo
3. `fastest path` : il percorso attraverso il quale si va da x a y con il tempo trascorso minimo
4. `shortest path` : il percorso più breve da x a y in termini di tempo di attraversamento complessivo necessario sull'arco

> ⚠ **Osservazione** >
>
> Notare che uno shortest path potrebbe non essere necessariamente un fastest path
> Notare anche che un fastest path potrebbe non essere un earliest-arrival path

Per calcolare questi percorsi non possiamo usare l'approccio Greedy che di norma viene usato nel calcolo dei percorsi per grafi statici, per esempio l'algoritmo di Dijkstra, perchè quelli sono basati sulla proprietà che un sottopercorso di uno shortest path è anch'esso uno shortest path, cosa non necessariamente vera quando calcoliamo uno dei 4 `Minimum temporal paths`.

2. Notazione dei Grafi Temporali

2.1 Definizione di un grafo temporale

`Notazione di un grafo temporale`

Sia $G = (V, E)$ un grafo temporale, con V denotiamo l'insieme dei nodi, e con E l'insieme degli archi.

Un'arco $e \in E$ è una **quadrupla** (u, v, t, λ), dove:

- $u, v \in V$
- t è il **tempo di inizio (starting time)**
- λ è il **tempo di traversata (traversal time)** per andare da $u \to v$ partendo al tempo t
- $t + \lambda$ è il **tempo di fine**

Denotiamo lo starting time di e con $t(e)$ e il traversal time con $\lambda(e)$

Se gli archi non sono diretti, allora lo starting time e il traversal time di un arco sono gli stessi da $u \to v$ e da $v \to u$.

Vediamo alcuni esempi:

- **Chiamata o Sistema di Messaggistica** : ogni vertice rappresenta una persona, e un'arco (u, v, t, λ) indica che il vertice u comunica con v al tempo t, e il tempo di connessione è λ.
- Social Network : ogni vertice rappresenta una persona, e un'arco (u, v, t, λ) può essere un'interazione tra u e v al tempo t che prende tempo λ.

2.2 Insiemi di archi temporali

Denotiamo con $\prod(u, v)$ l'insieme degli archi temporali da u verso v, mentre il numero di questi archi temporali è $\pi(u, v)$, infatti vale

$$\pi(u, v) = \left|\prod(u, v)\right|$$

Definiamo anche il **massimo** numero di archi temporali da u a v , $\forall u, v \in V$, come

$$\pi = \max\{\pi(u, v) : (u, v) \in (V \times V)\}$$

Il valore di π può essere molto grande per molti grafi temporali reali.

2.2.1 Uguaglianza di due archi temporali

In un grado temporale $G = (V, E)$, dati due archi temporali
$e_1 = (u_1, v_1, t_1, \lambda_1) \in E, e_2 = (u_2, v_2, t_2, \lambda_2) \in E$, abbiamo che

$$e_1 = e_2 \iff (u_1 = u_2 \land v_1 = v_2 \land t_1 = t_2 \land \lambda_1 = \lambda_2)$$

Se noi condensiamo archi temporali in archi statici, otteniamo il corrispondente grafo statico, definito come $G_s = (V_S, E_s)$, dove $V_s = V$ e $E_s = (u, v) : (u, v, t, \lambda) \in E$.
Come possiamo notare, la condensazione **rimuove** tutte le informazioni temporali dagli archi di E e combina tutti gloi archi che hanno stessi nodi di inizio e fine in un singolo arco.

2.2.2 Vicinato di un nodo

Definiamo il numero di vertici di G e G_s come $n = |V| = |V_s|$, e il numero di archi di G come $M = |E|$, mentre quelli di G_s sono $m = |E_s|$

Definiamo l'insieme dei **vicini-uscenti (out-neighbors)** di un vertice u in G o G_s come

$$\Gamma_{\text{out}}(u, G) = \Gamma_{\text{out}}(u, G_s) = \{v : (u, v, t, \lambda) \in E\} = \{v : (u, v) \in E_s\}$$

Definiamo il **grado-uscente (out-degree)** di un vertice u in G come

$$d_{\text{out}}(u, G) = \sum_{v \in \Gamma_{\text{out}}(u,G)} \pi(u, v)$$

e quello di u in G_s come

$$d_{\text{out}}(u, G_s) = |\Gamma_{\text{out}}(u, G_s)|$$

L'insieme dei **vicini-entranti (in-neighbors)** e il **grado-entrante (in-degree)** di un vertice u in G o G_s sono definiti in modo completamente simmetrico

$$\Gamma_{\text{in}}(u, G) = \Gamma_{\text{in}}(u, G_s) = \{v : (v, u, t, \lambda) \in E\} = \{v : (v, u) \in E_s\}$$

$$d_{\text{in}}(u, G) = \sum_{v \in \Gamma_{\text{in}}(u,G)} \pi(v, u)$$

$$d_{\text{in}}(u, G_s) = |\Gamma_{\text{int}}(u, G_s)|$$

3. Definizione dei Percorsi Temporali

Un `Percorso Temporale` P in un grafo temporale G è una sequenza di vertici

$$P = \langle v_1, v_2, \ldots, v_k, v_{k+1} \rangle$$

dove $(v_i, v_{i+1}, t_i, \lambda_{i)} \in E$ è l'i-esimo arco temproale di P per $1 \leq i \leq k$, e $(t_i + \lambda_i) \leq t_{i+1}$ per $1 \leq i \leq k$.
Da notare che nell'ultimo arco $(v_k, v_{k+1}, t_k, \lambda_k)$ di P, non mettiamo la condizione su $(t_k + \lambda_k)$ dato che t_{k+1} non è definito per il percorso P.

Infatti, la quantità $(t_k + \lambda_k)$ è l'**ending time** di P, ed è definita come $end(P)$.
Definiamo anche lo **starting time** di P, definito come $start(P) = t_1$.
Definiamo poi **durata** e **distanza** di P, rispettivamente

$$dura(P) = end(P) - start(P)$$
$$dist(P) = \sum_{i=1}^{k} \lambda_i$$

Esempio

Un esempio di percorso temporale è $P = \langle (a.\, f, 3, 1), (f, i, 5, 1), (i, l, 8, 1) \rangle$ nel grafo temporale G in $\text{Figura } 1(a)$.

Abbiamo :

- $start(P) = 1$
- $end(P) = 8 + 1 = 9$
- $dura(P) = 9 - 3 = 6$
- $dist(P) = 1 + 1 + 1 = 3$

3.1 Definizione formale dei Minimum Temporal Paths

Minimum Temporal Paths >

Dato un grafo temporale G, una sorgente x e un target y entrambi in G, e un intervallo di tempo $[t_\alpha, t_\omega]$, sia

$\mathbb{P}(x, y, [t_\alpha, t_\omega]) = \{P : P \text{ è un percorso temporale da } x \to y \text{ dove } start(P) \geq t_\alpha \wedge\ end(P) \leq$

Definiamo i seguenti 4 tipi di percorsi temporali da $x \to y$ entro l'intervallo $[t_\alpha, t_\omega]$ che hanno il minimo valore per differenti misure, così collettivamente chiamati : **Minimum temporal paths** :

- **Earliest-arrival path** : $P \in \mathbb{P}(x, y, [t_\alpha, t_\omega])$ è un earliest-arrival path se

$$end(P) = \min\{end(P') : P' \in \mathbb{P}(x, y, [t_\alpha, t_\omega])\}$$

- **Latest-departure path** : $P \in \mathbb{P}(x, y, [t_\alpha, t_\omega])$ è un latest-departure path se

$$start(P) = \max\{start(P') : P' \in \mathbb{P}(x, y, [t_\alpha, t_\omega])\}$$

- **Fastest path** : $P \in \mathbb{P}(x, y, [t_\alpha, t_\omega])$ è un fastest path se

$$dura(P) = \min\{dura(P') : P' \in \mathbb{P}(x, y, [t_\alpha, t_\omega])\}$$

- **Shortest path** : $P \in \mathbb{P}(x, y, [t_\alpha, t_\omega])$ è uno shortest path se

$$dist(P) = \min\{dist(P') : P' \in \mathbb{P}(x, y, [t_\alpha, t_\omega])\}$$

Notare che se l'intervallo di tempo $[t_\alpha, t_\omega]$ non è esplicitamente specificato per i minimum temporal paths, allora viene semplicemente impostasto come $[t_\alpha = 0, t_\omega = \infty]$.
Comunque, noi potremmo non essere sempre interessati all'intera storia temporale del grafo e quindi lasciare che l'utente specifichi l'intervallo $[t_\alpha, t_\omega]$, dando così più flessibilità ed applicabilità.

Una definizione più completa di minimum temporal paths è stata proposta qua. [1]

3.2 Single-Source Minimum Temporal Paths (SSMTP)

Definizione del problema >

Dato un grafo temporale $G = (V, E)$, un vertice $x \in V$, e un intervallo di tempo $[t_\alpha, t_\omega]$, il problema del **SSMTP** è quello di cercare :

1. l'**earliest-arrival path** da x ad ogni $v \in V$, oppure
2. il **latest-departure path** da ogni $v \in V$ ad x, oppure
3. il **fastest path** da x ad ogni $v \in V$, oppure
4. lo **shortest path** da x ad ogni $v \in V$

rispettivamente entro l'intervallo di cui sopra.

Sia P il **percorso temporale minimo** (minimum temporal path) ad essere calcolato. Per semplificare il tutto, nella presentazione dell'algoritmo per `SSMTP` vengono riportati solamente :

1. earliest-arrival time $end(P)$
2. latest-departure time $start(P)$
3. durata del fastest path $dura(P)$
4. distanza dello shortest path $dist(P)$

I seguenti **lemmi** danno alcune proprietà dei percorsi temporali minimi.

Lemma 1.
Un sottopercorso-prefisso di un earliest-arrival path potrebbe non essere un earliest-arrival path

Lemma 2.
Un sottopercorso-postfisso di un latest-departure path potrebbe non essere un latest-departure path

Lemma 3.
Un sottopercorso di un fastest path potrebbe non essere un fastest path

Lemma 4.
Un sottopercorso di uno shortest path potrebbe non essere uno shortest path

4. Algoritmi One-Pass per calcolare i Percorsi Temporali Minimi

4.1 Rappresentazione di un Grafo Temporale come Stream

La rappresentazione **edge stream** di un grafo temporale G è semplicemente una sequenza di tutti gli archi di G che arrivano nell'ordine in cui ciascun arco viene creato/collezionato (es. gli archi sono ordinati secondo il loro starting time).

Se due archi temporali sono creati/collezionati allo stesso momento, il loro ordine sarà arbitrario.

Esempio

Se G ha i seguenti archi, $\{(v_1, v_2, 2, 5), (v_2, v_4, 4, 1), (v_3, v_2, 1, 1)\}$, allora lo stream edge di G apparirà come : $\{(v_3, v_2, 1, 1), (v_1, v_2, 2, 5), (v_2, v_4, 4, 1)\}$

Lo stream edge è un formato naturale con cui viene generato e collezionato un grafo temporale.

Il seguente lemma mostra una proprietà di un percorso temporale in connessione con la rappresentazione **edge stream**.

Lemma 5.
Sia $P = \langle v_1, v_2, \ldots, v_k, v_{k+1} \rangle$ un percorso temporale in G, dove $(v_i, v_{i+1}, t_i, \lambda_i) \in E$ è l'i-esimo arco temproale di P per $1 \leq i \leq k$. Per ogni e_i e e_j in P, se $i < j$ allora e_i viene prima di e_j nella rappresentazione edge stream di G.

Dimostrazione Lemma 5
Per definizione di percorso temporale, abbiamo che $t_i + \lambda_i \leq t_{i+1}$ per $1 \leq i \leq k$, e quindi $t_{i+1} > t_i$ come $\lambda_i > 0$. Quindi, gli starting times degli archi $e_1, e_2, \ldots, e_k$ sono strettamente in ordine ascendente,e quindi e_i viene prima di e_j nella rappresentazione edge stream di G. □

4.2 Earliest-Arrival Paths

Algorithm 1: Computing earliest-arrival time

Input : A temporal graph $G = (V, E)$ in its edge stream representation, source vertex x, time interval $[t_\alpha, t_\omega]$
Output : The earliest-arrival time from x to every vertex $v \in V$ within $[t_\alpha, t_\omega]$

```
1 Initialize t[x] = t_α, and t[v] = ∞ for all v ∈ V \ {x};
2 foreach incoming edge e = (u, v, t, λ) in the edge stream do
3     if t + λ ≤ t_ω and t ≥ t[u] then
4         if t + λ < t[v] then
5             t[v] ← t + λ;
6     else if t ≥ t_ω then
7         Break the for-loop and go to Line 8;
8 return t[v] for each v ∈ V;
```

Algoritmo 1

Il classico algoritmo di Dijkstra per calcolare il single-source shortest-path è basato sul fatto che un sottopercorso-prefisso di uno shortest path è anch'esso uno shortest path.

Comunque, accordandoci con il Lemma 1, il sottopercorso-prefisso di un' earliest-arrival path è anch'esso un earliest-arrival path.

Questo sembra implicare che la strategia greedy per far crescere lo shortest-paths che è applicata nell'algoritmo di Dijkstra non può essere applicata per calcolare gli earliest-arrival paths, sebbene la seguente osservazione dimostri il contrario.

Lemma 6
Sia $\mathbb{P}$ il set degli earliest-arrival paths da x a un vertice v_k nell'intervallo $[t_\alpha, t_\omega]$. Se $\mathbb{P} \neq \emptyset$, allora esiste $P = \langle X, v_1, \ldots, v_k \rangle \in \mathbb{P}$ tale che ogni sottopercorso-prefisso $P_i = \langle x, v_1, \ldots, v_i \rangle$ è un earliest-arrival path da x a v_i nell'intervallo $[t_\alpha, t_\omega], 1 \leq i \leq k$.

Dimostrazione lemma 6
Dato un qualunque earliest-arrival paths $P \in \mathbb{P}$, se nessuno dei suoi sottopercorsi-prefisso sono earliest-arrival path, noi possiamo sempre costruire un percorso $\tilde{P}$ in questo modo.
Attraversiamo P in ordine inverso e troviamo il primo vertice v_i tale che il sottopercorso-prefisso corrisspondente P_i non è un earliest-arrival path da $x \to v_i$. Allora, deve esistere un'altro percorso $\tilde{P}_i$ che è un earliest-arrival path da $x \to v_i$. Sostituiamo P_i con $\tilde{P}_i$ in P.
Il nuovo percorso $\tilde{P}$ è comunque un percorso temporale valido perchè $end(\tilde{P}_i) < end(P_i)$.
Inoltre, $\tilde{P}$ è un earliest-arrival path da $x \to v_k$ perchè $end(\tilde{P}) = end(P)$.
Questo processo continua fino a quando ogni sottopercorso-prefisso è un earliest-arrival path e il risultante $\tilde{P} \in \mathbb{P}$, il che prova il lemma. □

Basandoci sul Lemma 6, possiamo applicare la strategia greedy per far crescere gli earliest-arrival paths in un modo silime all'algoritmo di Dijkstra.
Comunque, questo approccio necessità di una coda con priorità minima, risultando in un algoritmo che impiega tempo $O(m \log(\pi) + m \log(n))$ e spazio $O(M + n)$.

Comunque, per i grafi temporali, il Lemma 5 implica che il grafo in input può essere nella sua rappresentazione edge stream,ed è possibile calcolare gli earliest-arrival paths con una sola scansione del grafo.

Usiamo un array $t[v]$ per tenere traccia degli earliest-arrival time da x verso ogni vertice $v \in V$ che è stato visto nello stream.
Come dice il Lemma 5, se esiste un percorso temporale P da $x \to v$ tale che tutti gli archi in P sono stati visti nello stream, allora $t[v] = end(P) = t + \lambda$ come aggiornato in Linea 5.

La condizione $t + \lambda < t[v]$ in linea 4, si assicura che $t[v]$ sarà aggiornato con il più piccolo $end(P)$ per ogni P da $x \to v$ nell'intervallo $[t_\alpha, t_\omega]$.

Noi scannerizziamo il grafo G in tempo lineare e per ogni arco entrante $e = (u, v, t, \lambda)$ nello stream, controlliamo se e soddisfa il vincolo temporale di un percorso temporale all'interno dell'intervallo $[t_\alpha, t_\omega]$, per esempio, se $t + \lambda \leq t_\omega$ e $t \geq t[u]$.

Se si, facciamo crescere il percorso temporale estendendolo a v usando l'arco e. Durante il processo, aggiorniamo $t[v]$ quando necessario come discusso prima. Il processo termina quando incontriamo il primo arco nello stream che ha starting time maggiore/uguale a t_ω (Linee 6-7).

Il seguente lemma mostra che quando l'Algoritmo 1 termina, $t[v]$ riporta in modo corretto l'earliest-arrival time da $x \to v$

Lemma 7
Per ogni vertice $v \in V$, se l'earliest-arrival path da $x \to v$ entro l'intervallo $[t_\alpha, t_\omega]$ esiste, allora $t[v]$ ritornato dall'Algoritmo 1 è il corrispondente earliest-arrival time; altrimento $t[v] = \infty$.

Dimostrazione Lemma 7
Supponiamo che l'earliest-arrival path da $x \to v$ entro l'intervallo $[t_\alpha, t_\omega]$ esiste.
Allora, grazie al Lemma 6, esiste un earliest-arrival path da $x \to v$, detto $P_i = \langle x = v_1, v_2, \dots, v_k, v_{k+1} = v \rangle$, tale che ogni sottopercorso-prefisso di P è un earliest-arrival path da x a qualche vertice v_i di P.
Sia $t_e[v_i]$ l'earliest-arrival time da $x \to v_i$, per $1 \leq i \leq k+1$.
Siano $e_1, e_2, \dots, e_k$ gli archi in P, con $e_i = (u_i, v_{i+1}, t_i, \lambda_i)$ per $1 \leq i \leq k$. Allora abbiamo che $t_i \geq t_e[v_i]$ e $t_i + \lambda_i = t_e[v_i]$ per $1 \leq i \leq k$.
Dimostriamo che l'Algoritmo 1 calcola $t[v_i] = t_e[v_i]$, per $1 \leq i \leq k+1$ per induzione su i
Quando $i = 1, x = v_i, t[x] = t_e[x] = t_\alpha$ è inizializato in linea 1 dell'algoritmo, e $t[x]$ non sarà più aggiornato.
Ora assumiamo che per $i = j$, con $j < k+1, t[v_j] = t_e[v_j] = t_{j-1} + \lambda_{j-1}$ quando processiamo e_{j-1}. Consideriamo $i = j+1$ e vogliamo dimostrare che $t[v_{j+1}] = t_e[v_{j+1}]$.
Grazie al Lemma 5 sappiamo che e_j viene dopo di e_{j-1} nello stream.
Quindi, quando l'algoritmo legge e_j, abbiamo le due casistiche seguenti riguardanti il valore di $t[v_{j+1}]$.

1. $t[v_{j+1}] = t_e[v_{j+1}]$. In queso caso, la linea 5 non verrà processata a causa della condizione in linea 4 e $t[v_{j+1}]$ darà il corretto earliest-arrival time da x a v_{j+1}
2. $t[v_{j+1}] > t_e[v_{j+1}]$. In questo caso, $t[v_{j+1}]$ è aggiornato a $t_e[v_{j+1}] = t_j + \lambda_j$ in linea 5, e non sarà più aggiornato sempre per la condizione in linea 4

In entrambi i casi abbiamo che $t[v_{j+1}] = t_e[v_{j+1}]$, e per induzione $t[v_j] = t_e[v_j]$ per $1 \leq i \leq k+1$
Quindi, sde l'earliest-arrival path da $x \to v$ non esisite, allora non essite nessun percorso temporale da $x \to v$ e $t[v]$ rimarrà $\infty, \square$

Il seguente teorema dimostra il risultato principale

Teorema 1
L'algoritmo 1 calcola correttamente l'earliest-arrival time da un vertice sorgente x a ogni vertice $v \in V$ nell'intervallo $[t_\alpha, t_\omega]$ usando tempo $O(n + M)$ e spazio $O(n)$

Dimostrazione
La correttezza è dimostrata dal Lemma 7

4.3 Latest-Departure Paths

Aggiungere foto algoritmo 2

L'algoritmo per calcolare il latest-departure time da ogni vertice a un vertice target $x \in G$ è sostanzialmente identico all'algoritmo 1, in quanto leggiamo lo stream in ordine inverso Infatti l'algoritmo 2 è simmetrico all'algoritmo 1

Il seguente lemma mostra che l'algoritmo calcola correttamente il latest-departure time

Lemma 8
Per ogni vertice $v \in V$, se il latest-departure path da $v \to x$ nell'intervallo $[t_\alpha, t_\omega]$ esiste, allora $t[v]$ ritornato dall'algoritmo 2 è il corrispondente latest-departure time; altrimenti $t[v] = -\infty$

Il seguente teorema dimostra il risultato principale

Teorema 2
L'algoritmo 2 calcola correttamente il latest-departure time da ogni vertice v a un vertice target x nell'intervallo $[t_\alpha, t_\omega]$ usando tempo $O(n + M)$ e spazio $O(n)$

4.4 Fastest Paths

4.4.2 Algoritmo One-Pass con Time Bound Migliore

4.4.2 Algoritmo Linear-Time per Casi Specifici

4.5 Shortest Paths

5. Un approccio alla trasformazione di un grafo

5.1 Earliest-Arrival Paths

5.2 Latest-Departure Paths

5.3 Fastest Paths

5.4 Shortest Paths

5.5 Analisi della Complessità

6. Applicazioni dei percorsi temporali

6.1 Temporal Closeness Centrality

6.2 Top-k Nearest Neighbors

1. B.-M. B. Xuan, A. Ferreira, and A. Jarry. Computing shortest, fastest, and foremost journeys in dynamic networks. Int. J. Found. Comput. Sci., 14(2):267–285, 2003. ↩