Algoritmi

1 Fastest Path

Algorithm 1: Fastest Path

Input: Un *temporal graph* $G = (V, E)$ nella sua rappresentazione *edge-stream*, un nodo sorgente s
Output: I *Fastest Path* da nodo s a ogni nodo $v \in V$

```
1  esr = [ ]                                                    // edge-stream residuo
2  foreach v ∈ V do
3      L_v = null
4  Inizializza f[s] = 0 e f[v] = ∞ ∀v ∈ V \ {s}
5  foreach e = (u, v, t_e, λ_e) do
6      if u == s then
7          if (t, t) ∉ L_u then
8              L_u.insert((t, t))
9      a'[u] = max{a[u] : (s[u], a[u]) ∈ L_u, a[u] ≤ t}
10     s[v] = s'[u]
11     a[v] = t + λ
12     if s[v] ∈ L_v then
13         L_v.update((s[v], a[v]))
14     else
15         if la coppia (s[v], a[v]) non è dominata then
16             L_v.insert((s[v], a[v]))
17             esr[i] = e
18     Rimuovi elementi dominati
19     if a[v] − s[v] < f[v] then
20         f[v] = (a[v] − s[v], i − 1)
21 return f
```

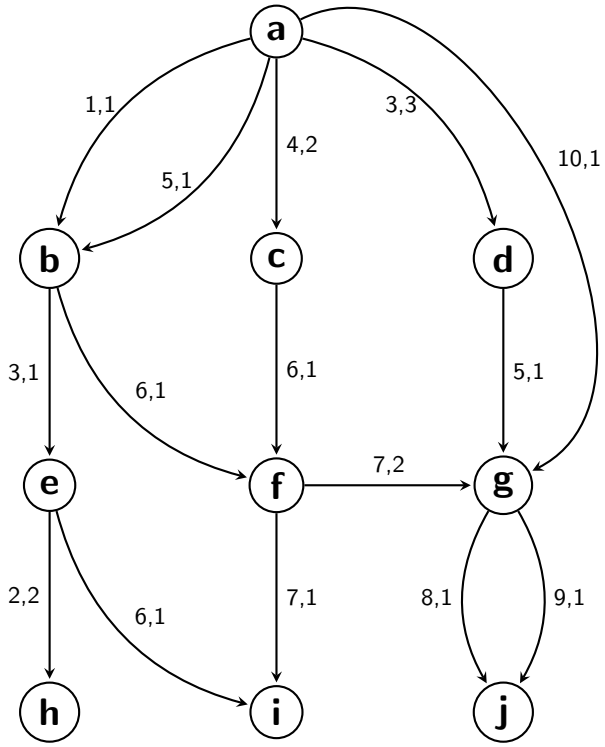
Algorithm 2: Fastest Path Oracle

Input: Un *temporal graph* residuo *esr* dei *Fastest Path* da un nodo sorgente s a tutti in nodi $v \in V$, nella sua rappresentazione *edge-stream*, l'array f con i puntatori agli archi dove iniziare la visita per ogni nodo, un nodo target t
Output: Il *Fastest Path* dal nodo s al nodo t

```
1  if f[t][1] = null then
2      return "No esiste path"
3  path = []
4  start = f[t][1]
5  path = [t]
6  at = ∞
7  i = 1 while snoninpath do
8      :
9  if esr[start][1] == t and esr[start][2] + esr[start][3] ≤ at then
10     path[i] = esr[start][0]
11     i = i + 1
12     t = esr[start][0]
13     at = esr[start][2]
14     start = start − 1
15 return path
```

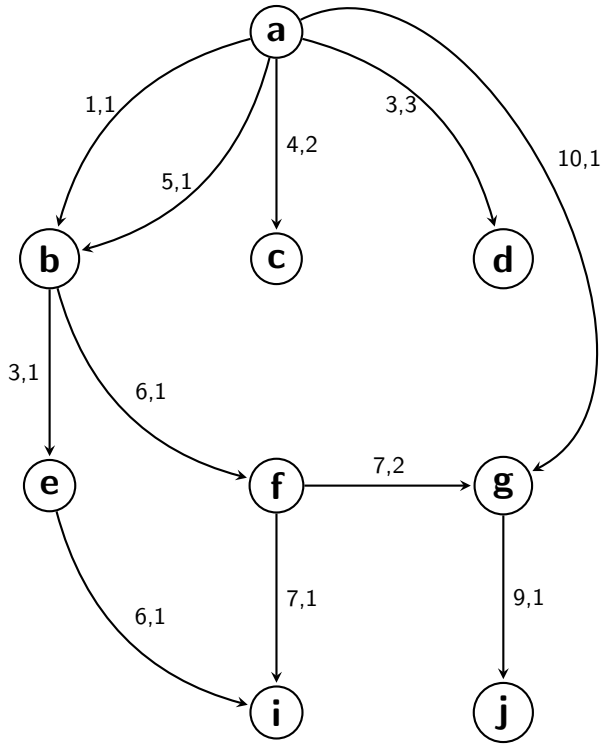
2 Esempi

Temporal Graph



Edge Stream
a, b, 1, 1
e, h, 2, 2
b, e, 3, 1
a, d, 3, 3
a, c, 4, 2
a, b, 5, 1
d, g, 5, 1
b, f, 6, 1
e, i, 6, 1
c, f, 6, 1
f, i, 7, 1
f, g, 7, 2
g, j, 8, 1
g, j, 9, 1
a, g, 10, 1

Grafo Residuo:



***Edge Stream* Residuo:**
a, b, 1, 1
b, e, 3, 1
a, d, 3, 3
a, c, 4, 2
a, b, 5, 1
b, f, 6, 1
e, i, 6, 1
f, i, 7, 1
f, g, 7, 2
g, j, 9, 1
a, g, 10, 1

3 Dimostrazione Algoritmo 1

L' Algoritmo 1 costruisce un *edge stream* residuo che contiene tutti i *Fastest Path*, poiché vengono esclusi solo gli archi che producono un *path* dominato quando vengono scansionati.

4 Dimostrazione Algoritmo 2

Supponiamo per assurdo che l'algoritmo ritorni un *path* p_1 con *starting time* e *arrival time* rispettivamente (s_1, a_1) che non sia un *Fastest Path*, pertanto esiste un *path* p_2 con *starting time* e *arrival time* rispettivamente (s_2, a_2) tale che $a_2 - s_2 < a_1 - s_1$ che invece è un *Fastest Path*. Questo implica due casi.

Caso 1: Il *path* è dominato

- Un arco $(u, v, t, \lambda) \in p_1$ al momento della scansione rende il percorso p_1 dominato e non viene inserito $\rightarrow$ assurdo.
- Un arco $(u, v, t_1, \lambda_1) \in p_1$ al momento della scansione non rende il percorso p_1 dominato, ma successivamente viene inserito un arco $(w, v, t_2, \lambda_2) \in p_2$[1] che lo rende tale: poiché (u, v, t_1, λ_1) viene prima nell'ordinamento dell'*Edge Stream* residuo, l'arco $(w, v, t_2, \lambda_2) \in p_2$ verrà visto prima e la costruzione del *path* da parte dell'algoritmo continuerebbe a partire da questo $\rightarrow$ assurdo.

Caso 2: Il *path* non è dominato

- Se $a_1 < s_2$ tutti gli archi di p_1 precederanno gli archi di p_2 e pertanto l'algoritmo, che inizia la visita dall'ultimo arco che compone un *Fastest Path* avrebbe ritornato $p_2 \rightarrow$ assurdo;
  Se $a_2 < s_1$ tutti gli archi di p_2 precederanno gli archi di p_1 e pertanto l'algoritmo, che inizia la visita dall'ultimo arco che compone un *Fastest Path* avrebbe ritornato $p_2 \rightarrow$ assurdo.
- Sia che risulti $s_1 < s_2$ e $s_2 \leq a_1 < a_2$ sia che risulti $s_2 < s_1$ e $s_1 \leq a_2 < a_1$, la visita partirebbe dall'urtimo arco che compone il *path* p_2 procedendo a ritroso sullo stream, pertanto l'algoritmo avrebbe ritornato p_2 e non $p_1 \rightarrow$ assurdo.[2]

[1] il nodo v può essere a qualsiasi livello del percorso

[2] Qualora nei percorsi ci siano dei nodi comuni, quello che succede guardando gli archi entranti è la seguente cosa: se l'arco che rende il proprio *path* dominante viene dopo nello *stream*, l'algoritmo lo vede prima e procede lungo il percorso cui appartiene, se viene prima nello stream, l'arco del *path* dominato non viene inserito nello *stream* residuo.

5 Shortest Path

Algorithm 3: Shortest Path

Input: Un *temporal graph* $G = (V, E)$ nella sua rappresentazione *edge-stream*, un nodo sorgente s
Output: Gli *Shortest Path* da nodo s a ogni nodo $v \in V$

```
1  esr = [ ]                                                    // edge-stream residuo
2  foreach v ∈ V do
3      L_v = null
4  Inizializza f[s] = 0 e f[v] = ∞ ∀v ∈ V \ {s}
5  foreach e = (u, v, t_e, λ_e) do
6      if u == s then
7          if (0, t) ∉ L_u then
8              L_u.insert((0, t))
9      a'[u] = max{a[u] : (d[u], a[u]) ∈ L_u, a[u] ≤ t}
10     d[v] = d'[u] + λ
11     a[v] = t + λ
12     if a[v] ∈ L_v then
13         L_v.insert((d[v], a[v]))
14     Rimuovi elementi dominati
15     if d[v] < f[v] then
16         f[v] = d[v]
17 return f
```

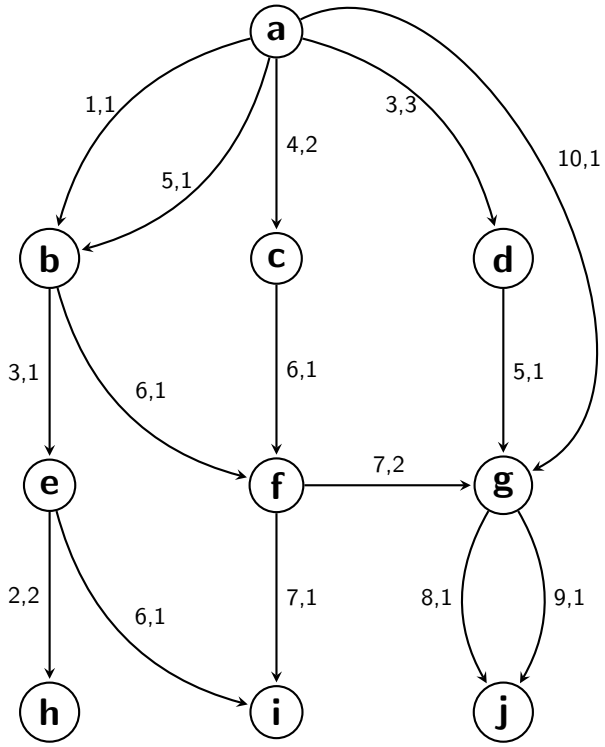
Algorithm 4: Shortest Path Oracle

Input: Un *temporal graph* residuo *esr* degli *Shortest Path* da un nodo sorgente s a tutti in nodi $v \in V$, nella sua rappresentazione *edge-stream*, l'array f con i puntatori agli archi dove iniziare la visita per ogni nodo, un nodo target t
Output: Lo *Shortest Path* dal nodo s al nodo t

```
1  if f[t][1] = null then
2      return "No esiste path"
3  path = []
4  start = f[t][1]
5  path = [t]
6  at = ∞
7  i = 1 while snoninpath do
8      :
9  if esr[start][1] == t and esr[start][2] + esr[start][3] ≤ at then
10     path[i] = esr[start][0]
11     i = i + 1
12     t = esr[start][0]
13     at = esr[start][2]
14     start = start − 1
15 return path
```

6 Esempi e dimostrazione

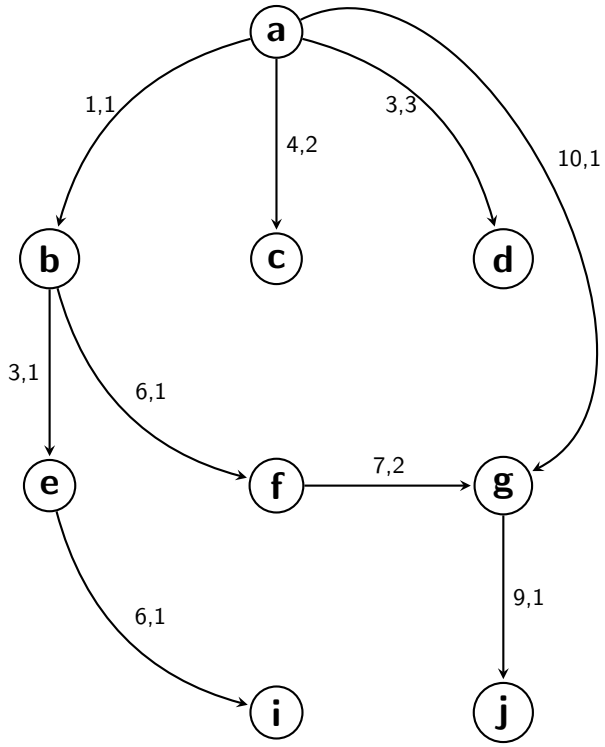
Temporal Graph



Edge Stream

a, b, 1, 1
e, h, 2, 2
b, e, 3, 1
a, d, 3, 3
a, c, 4, 2
a, b, 5, 1
d, g, 5, 1
b, f, 6, 1
e, i, 6, 1
c, f, 6, 1
f, i, 7, 1
f, g, 7, 2
g, j, 8, 1
g, j, 9, 1
a, g, 10, 1

Grafo Residuo



***Edge Stream* Residuo**

a, b, 1, 1
b, e, 3, 1
a, d, 3, 3
a, c, 4, 2
b, f, 6, 1
e, i, 6, 1
f, g, 7, 2
g, j, 9, 1
a, g, 10, 1