Dimostrazione Algoritmo Completa

Algoritmo

Versione alberi binari

L'algoritmo è diviso in due fasi

- Preprocessing
- Check finale

La **fase di preprocessing** è la fase che calcola, con approccio bottom-up, l'$EA_{\max}$ e il $T_{\max}$ di ogni sottoalbero fino alla radice. Ogni volta che risalgo di livello, propago le informazioni dai figli di u fino a u , e combino le informazioni che ho ottenuto con i valori sul nodo u.

Quando l'algoritmo risale alla radice, per ogni sottoalbero avremo calcolato correttamente i valori EA e $T_{\max}$.

I valori EA e $T_{\max}$ sono definiti così :

- $EA_{\max}$: $\max_{f:\text{ f è foglia}} EA$ da $f \in T_v$ fino al padre di v
- $T_{\max}$: Istante di tempo t tale che se arrivo al padre di v a tempo $\leq t$ allora riesco a visitare tutto T_v
- T_v : sottoalbero radicato nel nodo v
  E vengono calcolati dall'algoritmo in questo modo :
- Il valore dell'EA è uguale al massimo dei minimi timestamp di ogni livello
- Il valore del $T_{\max}$ è uguale al minimo dei massimi timestamp di ogni livello

Una volta eseguita la fase 1, verranno ritornati due dizionari, uno per l'EA e uno per il $T_{\max}$
Usando poi questi dizionari, passiamo in fase 2 per il check della temporal connectivity

Pseudocodice del preprocessing

Algorithm Procedura Preprocessing

Require: L_v : Lista timestamp arco entrante in v

Require: Dizionario D_{EA},Dizionario D_{Tmax}
procedure PREPROCESSING(Albero T)
if v è Nullo **then**
return $-\infty, \infty, D_{EA} = \emptyset, D_{Tmax} = \emptyset$
end if
if v è foglia **then**
return $L_v[1], L_v[n], D_{EA}, D_{Tmax}$
end if
min_{sx}, max_{sx} = Preprocessing($sx(v)$)
min_{dx}, max_{dx} = Preprocessing($dx(v)$)
$EA = \max(min_{sx}, min_{dx})$
$Tmax = \min(max_{sx}, max_{dx})$
NextEA = BinarySearch(L_v, EA)
NextTime = BinarySearch($L_v, Tmax$)
if NextEA $= -1 \vee$ NextTime$=-1$ **then**
return ∞, ∞
end if
minTime = $\min(Tmax, L_v[n])$
$D_{EA}(v)$=NextEA *//Aggiungo al dizionario EA la coppia (nodo v:NextEA) [chiave,valore]*
$D_{Tmax}(v)$=minTime *//Stessa cosa del dizionario EA*
return NextTime,minTime,D_{EA}, D_{Tmax}
end procedure

La **fase di check finale** è la fase che si occupa di vedere se l'albero rispetta la condizione di connettività temporale, ovvero

$$EA_{sx} \le T_{\max,dx} \wedge EA_{dx} \le T_{\max,sx} \quad (1)$$

Se usando i valori ottenuti in fase 1 questa condizione viene verificata per ogni sottoalbero, allora posso affermare che l'albero è temporalmente connesso, altrimenti se almeno un sottoalbero non mi verifica la condizione, affermo che l'albero non è temporalmente connesso.

Pseudocodice fase 2

Algorithm Procedura Check Temporal Connectivity

procedure CHECKTEMPORALCONNECTIVITY(D_{EA}, D_{Tmax})
Controllo in modo ricorsivo la condizione di temporal connectivity per ogni sottoalbero, usando man mano i valori all'interno dei due dizionari.
Check = False
for all nodo v **do**
$EA(v), Tmax(v) = D_{EA}.get(v), D_{Tmax}.get(v)$
if $EA(v) \le Tmax(v)$ **then**
Check=True
else
Check=False
Se Check diventa False, significa che un sottoalbero non rispetta la condizione, quindi esco subito dal ciclo e ritorno Check
return False
end if
end for

```
    return Check
  end procedure
```

L'algoritmo completo sarà quindi il seguente

Algorithm Algoritmo per Alberi Binari

```
Require: Dizionario D_EA,Dizionario D_Tmax
  procedure ALGORITMO(Albero T)
    D_EA, D_Tmax =Preprocessing(T)
    Check = CheckTemporalConnectivity(D_EA, D_Tmax)
    if Check = True then
      return Albero Temporalmente Connesso
    else
      return Albero Non Temporalmente Connesso
    end if
  end procedure
```

Versione alberi non binari

Mettere algoritmo in due fasi

- preprocessing
- check finale

Pseudocode qui

Algorithm Algoritmo per Alberi Non Binari

Algorithm Procedura Preprocessing

La **fase di check finale** è la fase che si occupa di vedere se l'albero rispetta la condizione di connettività temporale, ovvero

$$EA \leq T_{\max}, \forall EA, T_{\max} \quad (2)$$

Se usando i valori ottenuti in fase 1 questa condizione viene verificata per ogni sottoalbero, allora posso affermare che l'albero è temporalmente connesso, altrimenti se almeno un sottoalbero non mi verifica la condizione, affermo che l'albero non è temporalmente connesso.

Algorithm Procedura Check Temporal Connectivity

Dimostrazione

La dimostrazione verrà fatta per alberi non binari, in quanto per gli alberi binari basta minimizzare tutto a un fattore 2

Abbiamo che la fase 1 impiega tempo $\Theta(N \log(M))$, in quanto per ogni nodo calcola EA e $T_{\max}$, sfruttando l'ordinamento degli archi.

Quindi per ogni nodo, le informazioni corrette vengono propagante pagando $\log(M)$

Vediamo la fase 2:
Per ogni sottoalbero, viene effettuata la seguente verifica

Consideriamo un nodo u con i suoi figli :

- $\forall\ EA(v)$ con v figlio di u eseguiamo le seguenti operazioni
  - Elimino dal dizionario D_{Tmax} il $T_{\max}(v)$ corrispondente all'$EA(v)$ appena preso, mi costa $\log(\Delta_u)$
  - Trovo il minimo $T_{\max}$ tra tutti i figli v_i di u, mi costa $\log(\Delta_u)$
  - Eseguo il check tra EA_v e $T_{\max,\text{minimo}}$ e costa $O(1)$
  - Riaggiungo il valore $T_{\max}(v)$ eliminato prima nel dizionario corrispondente, costo $\log(\Delta_u)$

Adesso, preso

- δ_u = num. di figli del nodo u
- Δ_u = num. di valori $T_{\max}$ del nodo u
  Abbiamo che il costo totale dell'algoritmo per il nodo u è il seguente :
  $$\delta_u \log(\Delta_u)$$
  Ora, per ogni nodo $u \in T$, il costo totale dell'algoritmo di check sarà
  $$N\sum_i^N \delta_i \log(\Delta_i) \implies N\delta \log(\Delta)$$
  e ora, dato che $\delta \leq N$ e $\Delta \leq M$, il costo diventerà $N^2 \log(M)$
  Quindi, abbiamo che l'algoritmo impiega :
  $$\begin{aligned} \text{Tempo} &= \underbrace{\Theta(N\log(M))}_{\text{Preprocessing}} + \underbrace{O(N^2\log(M))}_{\text{Check Temporal Connectivity}} = O(N^2\log(M)) \\ \text{Spazio} &= \Theta(N) \end{aligned}$$

Alberi Binari

Per quanto riguarda gli alberi binari, la dimostrazione è la stessa, semplicemente il tutto viene abbassato di un fattore 2.

Infatti il costo della fase 2 sarà semplicemente $N\log(M)$, in quanto il valore δ sarà uguale a 2, $\forall\ u \in T$

Il costo totale sarà sempre

$$\text{Tempo} = \underbrace{\Theta(N\log(M))}_{\text{Preprocessing}} + \underbrace{O(N\log(M))}_{\text{Check Temporal Connectivity}} = \Theta(N\log(M))$$
$$\text{Spazio} = \Theta(N)$$

Ottimizzazione dell'algoritmo

Possiamo notare che, a meno di costanti motliplicative, le due fasi dell'algoritmo possono essere unite in un unico algoritmo, che mentre calcola i valori $EA, T_{\max}$ bottom-up riesce anche ad effettuare il controllo di connettività temporale fra tutti i sottoalberi relativi ad un nodo interno u, $\forall\, u \in T$

I due pseudocodici sono i seguenti

Alberi Binari

Algorithm Algoritmo Completo

Require: L_v : Lista timestamp arco entrante in v
  procedure DFS-EA-TMAX(Albero T,nodo v)
    if v è Nullo **then**
      return $-\infty, \infty$
    end if
    if v è foglia **then**
      return $L_v[1], L_v[n]$
    end if
    min_{sx}, max_{sx} = DFS-EA-Tmax($sx(v)$)
    min_{dx}, max_{dx} = DFS-EA-Tmax($dx(v)$)
    if $(min_{sx} > max_{dx}) \vee (min_{dx} > max_{sx})$ **then**
      return ∞, ∞
    end if
    $EA = \max(min_{sx}, min_{dx})$
    $Tmax = \min(max_{sx}, max_{dx})$
    NextEA = BinarySearch(L_v, EA)
    NextTime = BinarySearch($L_v, Tmax$)
    if (NextEA $= -1$)$\vee$(NextTime$=-1$) **then**
      return ∞, ∞
    end if
    minTime = $\min(Tmax, L_v[n])$
    return NextEA,minTime
  end procedure

Possiamo notare che nella versione ottimizzata, per gli alberi binari non c'è bisogno di mantenere in memoria i due dizionari, di conseguenza il costo temporale rimane invariato ma il costo spaziale passa da $O(N)$ a $O(1)$

Una possibile implementazione in Python è la seguente

```python
def dfs_EA_tmax(nodo):

    if nodo is None:
```

```python
            return float("-inf"),float("inf")
        if nodo.left == None and nodo.right == None:
            return nodo.weight[0],nodo.weight[-1]

        min_sx,max_sx = dfs_EA_tmax(nodo.left)
        min_dx,max_dx = dfs_EA_tmax(nodo.right)

        if min_sx>max_dx and min_dx>max_sx:
            return float("inf"),float("inf")

        EA = max(min_sx,min_dx)
        t_max_visita = min(max_sx,max_dx)

        nextEA = binary_search(nodo.weight,EA)
        nextTimeMax = binary_search_leq(nodo.weight,t_max_visita)
        if nextEA == -1 or nextTimeMax == -1:

            exit("Errore: EA o tempo max visita non trovati")
        minTime = min(t_max_visita,nextTimeMax)

        return nextEA,minTime
```

L'algoritmo completo sarà quindi

Algorithm Algoritmo

procedure ALG(Albero T,radice $root$)
- $EA_{sx}, T_{max,sx} =$ DFS-EA-Tmax$(T, sx(root))$
- $EA_{dx}, T_{max,dx} =$ DFS-EA-Tmax$(T, dx(root))$
- **if** $EA_{sx} = \infty \vee EA_{dx} = \infty$ **then**
  - **return** Albero non è temporalmente connesso
- **end if**
- **if** $EA_{sx} \leq T_{max,dx} \wedge EA_{dx} \leq T_{max,sx}$ **then**
  - **return** Albero è temporalmente connesso
- **else**
  - **return** Albero non è temporalmente connesso
- **end if**

end procedure

Alberi Non Binari

Per gli alberi non binari lo pseudocodice è il seguente

Algorithm Algoritmo Completo

Require: L_v : Lista timestamp arco entrante in v
Require: Dizionario D_{Nodi}
Require: Dizionario $D_{SottoAlberi}$
Require: Dizionario D_{EA},Dizionario D_{Tmax}

procedure DFS-EA-TMAX(Albero T,nodo v)
- **if** v è Nullo **then**
  - **return** $D_{Nodi} = \emptyset$

end if
if $child(v) = \emptyset$ **then**
return $D_{Nodi}(v) = (L_v[1], L_v[n])$
end if
$D_{SottoAlberi}(v) = \emptyset$
for all figlio u di v **do**
$update(D_{SottoAlberi}(v)$,DFS-EA-TMax$(u)$
$D_{EA}(v), D_{Tmax}(v){=}D_{SottoAlberi}(u)$
end for
for all $EA_v \in D_{EA}(v)$ **do**
$delete(D_{Tmax}(v), T_{\max,v})$
$\text{minTime} = FindMin(D_{Tmax}(v))$
if $EA_v > minTime$ **then**
return $D_{Nodi}(v) = (\infty, \infty)$
end if
$insert(D_{Tmax}(v), T_{\max,v})$
end for
$EA = \max(D_{EA})$
$Tmax = \min(D_{Tmax})$
$\text{NextEA} = \text{BinarySearch}(L_v, EA)$
$\text{NextTime} = \text{BinarySearch}(L_v, Tmax)$
$\text{minTime} = \min(Tmax, L_v[n])$
$D_{SottoAlberi}(v) = (NextEA, minTime)$
return $D_{SottoAlberi}$
end procedure

Una possibile implementazione in Python di questo pseudocodice potrebbe essere la seguente

```
def dfs_EA_tmax_spazioN_NonBinary(root):
    # Caso base: nodo nullo
    if root is None:
        return {}

    # Caso base: foglia
    if not root.children:
        return {root.value: (root.weight[0], root.weight[-1])}

    # Variabili per raccogliere i valori EA e Tmax per ogni sottoalbero
    sottoalberi = {}

    # Calcolo ricorsivo per ogni figlio
    ea_vals = []
    t_max_vals = []

    for child in root.children:
        sottoalberi.update(dfs_EA_tmax_spazioN_NonBinary(child))
        ea, t_max = sottoalberi[child.value]
        ea_vals.append(ea)
        t_max_vals.append(t_max)
```

```python

        min_tmax = min(t_max_vals)
        pos_min = t_max_vals.index(min_tmax)
        #first_ea = ea_vals[pos_min]
        for i in range(len(ea_vals)):
            if ea_vals.index(ea_vals[i]) == pos_min:
                continue
            elif ea_vals[i] > min_tmax:
                return {root.value: (float("inf"), float("inf"))}

        # Calcolo EA e Tmax per il nodo corrente
        EA = max(ea_vals)
        t_max_visita = min(t_max_vals)

        nextEA = binary_search(root.weight, EA)
        nextTimeMax = binary_search_leq(root.weight, t_max_visita)  # Binary search per trovare il predecessore
        minTime = min(t_max_visita, nextTimeMax)

        # Aggiornamento del nodo corrente nei risultati
        sottoalberi[root.value] = (nextEA, minTime)

        return sottoalberi
```

L'algoritmo completo sarà quindi il seguente

Algorithm Algoritmo

Require: Dizionario D_{EA}, Dizionario D_{Tmax}

  procedure ALG(Albero T,radice $root$)

    $D_{Risultati}$ = DFS-EA-Tmax($T, root$)

    for all Figlio u di $root$ **do**

      $D_{EA}, D_{Tmax} = D_{Risultati}(u)$

    end for

    Check=True

    for all $EA_{v_i} \in D_{EA}(root)$ **do**

      $delete(D_{Tmax}(root), T_{\max,v_i})$

      minTime $= FindMin(D_{Tmax}(root))$

      if $EA_{v_i} > minTime$ **then**

        Check=False

      end if

      $insert(D_{Tmax}(root), T_{\max,v_i})$

    end for

    if Check=True **then**

      return Albero temporalmente connesso

    else

      return Albero non temporalmente connesso

    end if

  end procedure

In questo caso, l'ottimizzazione si trova solo sulla parte del codice, perchè sia il costo temporale che spaziale rimane invariato

Costo temporale $O(\Delta N \log(M))$

Osservazione sull'ordinamento degli archi

Fino ad ora abbiamo fatto l'assunzione che i timestamp sugli archi fossero ordinati in partenza, ma nella realtà nessuno ci conferma se è effettivamente così oppure no.

Nel caso in cui i timestamp degli archi non siano ordinati, si può effettuare una procedura di preprocessing in cui in tempo $O(M \log(M))$ si possono ordinare tutti i timestamp.

Questo vale sia per alberi binari che non binari.

Il costo totale quindi cambierà in questo modo :

- Alberi Binari :

$$O(N \log(M)) + O(M \log(M)) = O(M \log(M)), \quad M = \Omega(N)$$

- Alberi Non Binari

$$O(N^2 \log(M)) + O(M \log(M)) = O(M \log(M)), \quad M = \Omega(N^2) = \Omega(N)$$