Algoritmo Corretto con Dimostrazione

- Algoritmo
  - Idea dell'algoritmo
  - Fase 1 (Preprocessing)
  - Fase 2
  - Fase 3
  - Fase 4
  - Check Finale
  - Costo totale algoritmo
- Correttezza dell'algoritmo
  - 1. Fase 1: Verifica della connettività temporale
  - 2. Fase 2: Ricerca della foglia più profonda con timestamp minimo
  - 3. Fase 3: Calcolo dell'EA minimo
  - 4. Fase 4: Calcolo tempo massimo per visitare il sottoalbero
    - 1. Correttezza dell'aggiornamento a livello locale
    - 2. Propagazione bottom-up
  - 5. Check finale
  - Conclusione
- Codice dell'algoritmo

Algoritmo

L'algoritmo è diviso in più fasi :

- **Preprocessing (Fase 1)**
- **Fase 2**
- **Fase 3**
- **Fase 4**
- **Check Finale**

Idea dell'algoritmo

Fase 1 : Prepcrocessing

Verifico se partendo da root, posso visitare tutti i nodi dell'albero (se non è possibile, l'albero non è temporalmente connesso).
Per farlo basta fare una visita DFS che controlla se ogni nodo rispetta la condizione di crescita incrementale dei timestamp.

Fase 2 : Calcolo foglia più profonda con timestamp minimo nei sottoalberi sinistro e destro

Per trovare la foglia con timestamp minimo nei sottoalberi sinistro e destro,
posso utilizzare la funzione ***find_leaf_min_timestamp*** definita di seguito.
Questa funzione calcola la foglia con timestamp minimo in un albero binario.

Fase 3 : Calcolo dell'$EA_{\min}$ partendo dalla foglia più profonda fino alla radice

Questa fase è simmetrica, ovvero vale per entrambi i sottoalberi, quindi ci concentreremo solo su un sottoalbero.
A questo punto, applico l'algoritmo per calcolare l'EA, usando la struttura dati del professore, ovvero quella che impiega tempo $O(\log M \cdot \log L)$
Se la funzione ritorna $-\infty$, sia da un lato che dall'altro, allora ritorno subito False (l'albero non è temporalmente connesso)

Fase 4 : Calcolo tempo massimo di visita di un sottoalbero

Questa fase è simmetrica, come la fase 3.

Con un approccio bottom-up, possiamo calcolare il tempo massimo di visita per un sottoalbero.

Il tempo massimo è calcolato così:

1. Si parte dalla profondità massima dell'albero, si controllano tutti i timestamp degli archi di quel livello e si prende il timestamp minimo fra tutti i massimi
2. Risalendo di livello mi porto l'informazione del livello precedente, e riapplico la verifica precedente
3. Ripeto 1 e 2 fino a che non raggiungo la radice, a quel punto al nodo radice aggiungo l'informazione $t_{\max}$, che mi indentifica il tempo massimo per visitare il sottoalbero

Check Finale

La condizione da verificare è la seguente :

$$EA_{\min,sx} \leq t_{\max,dx} \wedge EA_{\min,dx} \leq t_{\max,sx}$$

Se questa condizione viene verificata, allora possiamo affermare che l'albero è temporalmente connesso.

Fase 1 (Preprocessing)

L'idea è quella di far partire dalla **root** una visita DFS Temporale, dove per DFS Temporale si intende una DFS che visita il nodo solo se esso rispetta la condizione di crescita temporale dei timestamps.

Se partendo dalla root con questa visita riesco a toccare tutti i nodi, allora ritorno True e procedo a fase 2, altrimenti ritorno False ed esco.

Se ritorno False in questa fase posso concludere subito che l'albero non è temporalmente connesso, in quanto esiste almeno un nodo che sicuramente non può connettersi temporalmente alla radice.

Quanto costa questa visita? : La visita DFS Temporale ha un costo lineare nel numero di timestamps totali, ovvero

$$O(M)$$

Fase 2

La fase 2 prevede la ricerca delle foglie, nel sottoalbero sx e dx, che si trovano nella massima profondità che hanno timestamp minimo.

Quanto costa questa fase? : La ricerca di queste foglie ha un costo lineare nel numero di timestamp, e quindi impiega tempo

$$O(M)$$

Fase 3

La seconda fase prevede un'approccio diverso.

Partiamo da un qualunque sottoalbero, e procedendo **bottom-up** mi calcolo l'earliest-arrival time **minimo** fino al nodo root.

Per $EA_{\min}$ intendiamo l'earliest-arrival che parte dalla foglia più profonda con timestamp minimo, e che da li arriva fino a root

Quanto costa questa fase? : La ricerca bottom-up dell'earliest-arrival time minimo impiega tempo

$$O(\log M \cdot \log L)$$

Con $M = \text{num. timestamps}, L = \text{num. timestamps max su arco}$

Fase 4

La quarta fase prevede l'uso di un approccio bottom-up per calcolare il tempo massimo per visitare il sottoalbero.

La spiegazione è descritta in precedenza.

Il costo di questa fase è

$$O(M)$$

Check Finale

Il check finale si occupa solo di controllare se la condizione

$$EA_{\min,sx} \leq t_{\max,dx} \land EA_{\min,dx} \leq t_{\max,sx}$$

viene verificata o no.

Il costo è costante, ovvero $O(1)$

Costo totale algoritmo

La fase di preprocessing dei valori costa

$$O(M)$$

La fase 2 costa

$$O(M)$$

La fase 3 costa

$$O(\log M \cdot \log L)$$

Per questa fase il costo va raddoppiato, in quanto vale la simmetria del problema.

La fase 4 costa

$$O(1)$$

Di conseguenza il costo totale dell'algoritmo risulta essere

$$O(M) + O(M) + O(M) + O(\log M \log L) + O(1) \implies O(M) + O(\log M \log L)$$

Possiamo notare che l'algoritmo risulta essere lineare nel numero di timestamp, e anche nel caso peggiore il costo computazionale sarà

$$O(M)$$

Correttezza dell'algoritmo

La correttezza della dimostrazione e dell'algoritmo dipende dal fatto che ogni fase sia implementata in modo rigoroso e che i vincoli posti siano sufficienti per garantire la proprietà di **connessione temporale**.
Consideriamo i punti critici

---

1. Fase 1: Verifica della connettività temporale

L'algoritmo di visita **DFS Temporale** esplora solo i nodi i cui timestamp sugli archi sono maggiori/uguali al tempo attuale di visita. Infatti alla fine della dfs avremo un'insieme di nodi definito in questo modo

$$V(root) = \{v \in V : t_v \geq t_{root}\}$$

Se la visita riesce a visitare tutti i nodi partendo da root, procediamo verso la fase 2, altrimenti ritorniamo False, e affermiamo che l'albero non è temporalmente connesso.

Questo lo possiamo affermare perchè, se dalla root esiste almeno un nodo che non può essere raggiunto, allora significa che su un determinato arco i timestamp sono strettamente minori del tempo di visita attuale.

Questo implica che sul percorso da root al nodo, ovvero $P_{root \to u}$,esiste un arco che va a rompere la sequenza crescente dei timestamp, e di conseguenza questo implica che root e u non possono connettersi temporalmente.

---

2. Fase 2: Ricerca della foglia più profonda con timestamp minimo

Questa fase si concentra sul ricercare le foglie più profonde dei rispettivi sottoalberi, tale che la foglia che viene presa è quella con timestamp minimo fra tutte le foglie in profondità massima.

Perchè cercare questo tipo di foglia? Perchè così facendo, possiamo sfruttare queste foglie per far partire il calcolo dell'$EA_{\min}$ fino al nodo root.

L'osservazione che comanda questa dimostrazione è la seguente :

> **Osservazione chiave**
>
> Valgono le due condizioni :
>
> 1. Se $\nexists EA_{\min} \implies \nexists EA_{\max}$
> 2. Se $\nexists EA_{\max} \implies \text{non è detto che } \nexists EA_{\min}$

Queste due condizioni si possono dimostrare in modo logico :

1. Se l'$EA_{\min}$ non esiste, allora significa che se parto dalla foglia più profonda con timestamp minimo non riesco a raggiungere l'altro nodo, che nel nostro caso è la root. Questo implica che anche se parto dalla foglia più profonda con timestamp massimo, non riuscirò comunque a raggiungere la root.
2. Se l'$EA_{\max}$ non esiste, significa che se parto dalla foglia più profonda con timestamp massimo non riesco a raggiungere l'altro nodo, che nel nostro caso è la root. Questo però non implica che sicuramente non esiste $EA_{\min}$. Infatti, semplicemente possiamo

avere la casistica in cui l'$EA_{\max}$ non esiste a causa di una sequenza non crescente tra due nodi, partendo dal nodo con timestamp massimo, ma avere allo stesso tempo l' $EA_{\min}$ partendo sempre da quel nodo.

---

3. Fase 3: Calcolo dell'EA minimo

In questa fase ci occupiamo di calcolare l'$EA_{\min}$ partendo dalla foglia in fase 2 fino alla radice.

L'algoritmo che fa ciò si basa sulla struttura dati del paper del professore, e la dimostrazione di correttezza è spiegata li

---

4. Fase 4: Calcolo tempo massimo per visitare il sottoalbero

1. Correttezza dell'aggiornamento a livello locale

Per ogni nodo NN:

- Se N è una **foglia**, il tempo massimo per visitarlo è semplicemente il massimo $t_{\max} = \max(\omega(N))$ dei suoi timestamp. Questo è corretto perché una foglia non ha figli da considerare.
- Se N ha figli L (sinistro) e R (destro), il tempo massimo per visitare il sottoalbero dipende da:
  - Il tempo massimo $t_{\max,L}$ per visitare il sottoalbero sinistro.
  - Il tempo massimo $t_{\max,R}$ per visitare il sottoalbero destro.
  - Per garantire che entrambi i figli siano raggiungibili, $t_{\max}$ deve essere il **minimo** tra $t_{\max,L}$ e $t_{max,R}$, poiché il percorso verso entrambi deve essere valido temporalmente.

Correttezza locale: Questo approccio assicura che si selezioni sempre un tempo che permetta di visitare sia il sottoalbero sinistro che il destro.

---

2. Propagazione bottom-up

Il calcolo viene effettuato risalendo dai nodi foglia fino alla radice:

- Ogni passo garantisce che $t_{\max}$ sia valido per il sottoalbero considerato.

- Poiché ogni nodo calcola $t_{\max}$ in base ai figli, il risultato propagato verso la radice rappresenta il tempo massimo per visitare l'intero sottoalbero.

Correttezza globale: La propagazione bottom-up garantisce che ogni sottoalbero soddisfi i vincoli temporali richiesti, e il risultato finale rappresenta il tempo massimo per visitare tutto il sottoalbero.

5. Check finale

Nell'ultima fase si fa un semplice controllo tra gli $EA_{\min}$ e i t_{max} dei rispettivi sottoalberi

Se vale che l'$EA_{\min,sx} > t_{\max,dx}$, allora significa che il tempo minimo che ci vuole per risalire il sottoalbero sinistro verso la radice non combacia con il tempo massimo per visitare il sottoalbero destro, quindi significa che i nodi del sottoalbero sinistro non potranno mai visitare i nodi nel sottoalbero destro.

Ovviamente la stessa cosa vale per l'$EA_{\min,dx}$ e $t_{\max,sx}$.

Conclusione

Seguendo tutte le fasi dell'algoritmo, andiamo a coprire tutte queste casistiche :

1. I nodi di un sottoalbero sono temporalmente connessi fra loro (fase 1+fase3)
2. I nodi di un sottoalbero possono visitare i nodi dell'altro sottoalbero (fase 3+fase4)

E di conseguenza l'algoritmo è corretto, e ritorna correttamente True o False

Codice dell'algoritmo

is_temporaly_connected

```python
from bisect import bisect_left

class Node:
    def __init__(self, value, weight=[]):
        self.value = value
        self.weight = weight
        self.left = None
        self.right = None

def build_temporal_tree(node, parent=None, tree=None):
    if tree is None:
        tree = {}

    if node is None:
        return tree

    if parent is not None:
```

```
        if node.value not in tree:
            tree[node.value] = []
        tree[node.value].append((parent.value, node.weight))

    build_temporal_tree(node.left, node, tree)
    build_temporal_tree(node.right, node, tree)

    return tree

def print_temporal_tree(tree):
    for node, edges in tree.items():
        print(f"{node}: {edges}")

def EA_min_query_function(tree, u, v, t_start):
    current_node = u
    current_time = t_start

    while current_node != v:
        neighbors = tree.get(current_node, [])
        parent = None
        times = []

        for neighbor, timestamps in neighbors:
            if neighbor == v or neighbor == neighbors[0][0]:
                parent = neighbor
                times = timestamps
                break

        if not parent or not times:
            return float("-inf")

        idx = bisect_left(times, current_time)
        if idx == len(times):
            return float("-inf")

        max_valid_time = min(times[idx:])
        current_time = max_valid_time
        current_node = parent

    return current_time

def find_node_with_max_weight(node, depth=0):
    """
    Trova la foglia più profonda con il timestamp minimo alla sua profondità.

    Args:
        node: Nodo corrente.
        depth: Profondità corrente del nodo.

```

```
    Returns:
        Una tupla (timestamp minimo, nodo corrispondente, profondità).
    """
    if node is None:
        return float("inf"), None, -1  # Nessun nodo

    # Se è una foglia, restituisci il timestamp minimo
    if node.left is None and node.right is None:
        min_timestamp = min(node.weight, default=float("inf"))
        return min_timestamp, node, depth

    # Ricorsione sui figli
    left_min, left_node, left_depth = find_node_with_max_weight(node.left, depth + 1)
    right_min, right_node, right_depth = find_node_with_max_weight(node.right, depth + 1)

    # Scegli la foglia più profonda; in caso di parità, quella con il timestamp minimo
    if left_depth > right_depth:
        return left_min, left_node, left_depth
    elif right_depth > left_depth:
        return right_min, right_node, right_depth
    else:  # Stessa profondità
        if left_min <= right_min:
            return left_min, left_node, left_depth
        else:
            return right_min, right_node, right_depth

def calculate_maximum_earliest_arrivals(tree, left_leaf, right_leaf, root, max_timestamps, EA_query_function):
    """
    Calcola il minimo Earliest Arrival (EA minimo) per i percorsi dalle foglie sinistra e destra alla radice.

    Args:
        tree: struttura dati che rappresenta l'albero temporale.
        left_leaf: nodo foglia più profondo tutto a sinistra.
        right_leaf: nodo foglia più profondo tutto a destra.
        root: nodo radice dell'albero.
        max_timestamps: tuple con i timestamp massimi per le foglie sinistra e destra.
        EA_query_function: funzione che implementa la query EA dalla struttura del paper.

    Returns:
        - Un dizionario contenente EA_sx e EA_dx se validi, oppure False se almeno uno è infinito.
    """
    # Timestamp massimo per le foglie
```

```
    max_t_sx, max_t_dx = max_timestamps

    # Calcola EA per il percorso dalla foglia sinistra alla radice
    EA_sx = EA_query_function(tree, left_leaf.value, root.value, max_t_sx)

    # Calcola EA per il percorso dalla foglia destra alla radice
    EA_dx = EA_query_function(tree, right_leaf.value, root.value, max_t_dx)

    # Controlla se uno dei risultati è infinito
    if EA_sx == float("-inf") or EA_dx == float("-inf"):
        return False  # Non c'è percorso temporalmente connesso

    # Salva le informazioni nella radice
    root_info = {
        "Min_EA_sx": EA_sx,
        "Min_EA_dx": EA_dx
    }
    return root_info

def verify_temporal_connectivity(node, current_time=float("-inf")):
    """
    Verifica se l'albero è temporalmente connesso usando una visita DFS.

    Args:
        node: Nodo corrente dell'albero.
        current_time: Timestamp corrente minimo per rispettare la connettività temporale.

    Returns:
        True se l'albero è temporalmente connesso, False altrimenti.
    """
    if node is None:
        return True  # Nodo vuoto è sempre connesso

    # Controlla se esiste un timestamp valido
    valid_timestamps = [t for t in node.weight if t >= current_time]
    if not valid_timestamps:
        return False  # Non esiste un percorso temporale valido

    # Procedi verso i figli con il timestamp minimo richiesto
    next_time = min(valid_timestamps)
    left_connected = verify_temporal_connectivity(node.left, next_time)
    right_connected = verify_temporal_connectivity(node.right, next_time)

    return left_connected and right_connected

```

```python
def calculate_max_timestamp_bottom_up(node):
    """
    Calcola il timestamp massimo per visitare un intero sottoalbero usando un approccio bottom-up.

    Args:
        node: Nodo corrente.

    Returns:
        Il timestamp massimo utilizzabile per il sottoalbero.
    """
    if node is None:
        return float("inf")  # Nodo vuoto non impone restrizioni

    # Calcola ricorsivamente i timestamp massimi per i figli
    left_max = calculate_max_timestamp_bottom_up(node.left)
    right_max = calculate_max_timestamp_bottom_up(node.right)

    # Calcola il massimo utilizzabile per il nodo corrente
    node_max = max(node.weight, default=float("-inf"))

    return min(left_max, right_max, node_max)

def algoritmo(root):
    print("--------FASE 1--------")
    # Fase 1: Verifica della connettività temporale
    if not verify_temporal_connectivity(root.left):
        return "L'albero non è temporalmente connesso."

    print("\nCheck Fase 1 OK")

    # Fase 2: Trova le foglie più profonde con timestamp minimo nei sottoalberi
    min_left, left_node, _ = find_node_with_max_weight(root.left)
    min_right, right_node, _ = find_node_with_max_weight(root.right)

    print("\n--------FASE 2--------")
    print(f"\nFoglia più profonda sottoalbero sinistro: {left_node.value if left_node else None}, Timestamp: {min_left}")
    print(f"Foglia più profonda sottoalbero destro: {right_node.value if right_node else None}, Timestamp: {min_right}")

    print("\n--------FASE 3--------")
    # Fase 3: Calcolo dell'EA minimo partendo dalle foglie trovate in Fase 2
    tree = build_temporal_tree(root)
    print("\nStruttura dell'albero temporale:")
    print_temporal_tree(tree)

```

```python
        EA_sx = EA_min_query_function(tree, left_node.value, root.value, min_left) if left_node else float("-inf")
        EA_dx = EA_min_query_function(tree, right_node.value, root.value, min_right) if right_node else float("-inf")

        print(f"EA minimo dal nodo sinistro: {EA_sx}")
        print(f"EA minimo dal nodo destro: {EA_dx}")

        print("\n--------FASE 4--------")
        # Fase 4: Calcolo dei timestamp massimi per entrambi i sottoalberi
        t_max_sx = calculate_max_timestamp_bottom_up(root.left)
        t_max_dx = calculate_max_timestamp_bottom_up(root.right)

        print(f"\nTimestamp massimo sottoalbero sinistro: {t_max_sx}")
        print(f"Timestamp massimo sottoalbero destro: {t_max_dx}")

        # Check finale
        if EA_sx <= t_max_dx and EA_dx <= t_max_sx:
            print("\nCheck Fase 4 OK")
            return "\nL'albero è temporalmente connesso."
        else:
            print("\nCheck Fase 4 NO")
            return "\nL'albero non è temporalmente connesso."
```