Algoritmo Corretto con Dimostrazione

Algoritmo

Due versioni :

- una che usa spazio costante ma paga $O(N \log(M))$ per ogni sottoalbero
- una che usa spazio $O(N)$ ma paga $O(N \log(M))$ sempre, quindi fa una passata per tutto il sottoalbero

Versioni

Versione spazio costante

Algorithm Is Temporaly Connected

procedure DFS-EA-TMAX(v)

```
    if v è Nullo then
        return −∞, ∞
    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 not (min_sx ≤ max_dx ∨ min_dx ≤ max_sx) then
        return ∞, ∞
    end if
    EA = max(min_sx, min_dx)
    Tmax = min(max_sx, max_dx)
    NextTime = BinarySearch(L_v, EA)
    if NextTime = −1 then
        return ∞, ∞
    end if
    return NextTime,min(Tmax, L_v[n])
end procedure
```

Variabili :

- L_v : lista di timestamp associati all'arco entrante in v
- min, max sia sx, dx sono rispettivamente il timestamp minimo e massimo per il sottoalbero radicato nel nodo
  - Se il nodo è foglia, questi valori saranno semplicemente il tempo minimo e massimo dell'arco entrante nel nodo
  - Se il nodo è interno, questi valori indicano i tempi minimi e massimi per il sottoalbero radicato nel nodo
- I valori min e max servono per calcolare l'$EA_{\max}$ e $T_{\max}$, valori che poi verrano propagati dal basso verso l'alto. In questo modo, una volta arrivati alla radice dell'albero originale, avremo i valori per quanto riguarda l'$EA_{\max}$ dal basso verso l'alto, e per quanto riguarda il $T_{\max}$, ovvero il tempo massimo di visita del sottoalbero
  - Questi valori vengono calcolati per ogni possibile sottoalbero, in quanto la propagazione parte dal basso verso l'alto
- La condizione espressa nella riga dell'IF ci assicura che i nodi di un sottoalbero sono temporalmente connessi fra loro, questo sempre per ogni sottoalbero fino a risalire la radice
- Il valore $NextTime$ indica il prossimo timestamp da prendere per continuare la propagazione dell'$EA_{\max}$, se tale valore non esiste, ovvero se ci troviamo a guardare un'arco tale che $\forall t \in L_v, t < EA_{\max}$, allora ritorniamo ∞, e di conseguenza affermiamo che non è possibile trovare un'EA bottom-up, che implica che risalendo l'albero, un nodo del livello $i - esimo$ non si potrà connettere con gli altri nodi del livello $(i-1) - esimo$ e cosi via

Questa procedura viene poi applicata all'algoritmo di partenza, ovvero l'algoritmo

Algorithm Algoritmo

procedure ALG($root$)
  $EA_{sx}, T_{max,sx}$ = DFS-EA-Tmax(sx($root$))
  $EA_{dx}, T_{max,dx}$ = DFS-EA-Tmax(dx($root$))
  if $EA_{sx} = \infty \vee EA_{dx} = \infty$ **then**
    return False
  end if
  if $EA_{sx} \leq T_{max,dx} \wedge EA_{dx} \leq T_{max,sx}$ **then**
    return True
  else
    return False
  end if
end procedure

Codice python algoritmo

```
def algoritmo(root):

    ea_sx,t_max_sx = dfs_EA_tmax(root.left)
    ea_dx,t_max_dx = dfs_EA_tmax(root.right)
    print("------------------------------------------------")
    print(f"EA e tempo max visita sx della radice {root.value} : {ea_sx,t_max_sx}")
    print(f"EA e tempo max visita dx della radice {root.value} : {ea_dx,t_max_dx}")

    if ea_sx == float("inf") or ea_dx == float("inf"):
        return False

    # Ogni controllo del caso per alberi non binari
    if ea_sx <= t_max_dx and ea_dx <= t_max_sx:
        return True
    else:
        return False
```

Codice Python visita

Algoritmo 2

```
def dfs_EA_tmax_spazio1(root):

    if root is None:
        return float("-inf"),float("inf")
    if root.left == None and root.right == None:
        print(f"EA e tempo max visita per il sottoalbero radicato nel nodo {root.value} (foglia) : {root.weight[0],root.weight[-1]}")
        return root.weight[0],root.weight[-1]

    min_sx,max_sx = dfs_EA_tmax_spazio1(root.left)
```

```python

    min_dx,max_dx = dfs_EA_tmax_spazio1(root.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)
    print(f"EA e tempo max visita per il sottoalbero radicato nel nodo {root.value} (nodo interno) : {EA,t_max_visita}")
    k = binary_search(root.weight,EA)
    nextTimeMax = binary_search_leq(root.weight,t_max_visita)
    if k == -1 or nextTimeMax == -1:

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

    return k,minTime
```

Versione con spazio lineare

Pseudocodice :

Algorithm DFS-EA-Tmax-SpazioN

Require: Dizionario Nodo, Dizionario SottoAlberi

  procedure DFS(nodo v)
    if v è Nullo **then**
      return Nodo $= \{\}$
    end if
    if v è foglia **then**
      return Nodo$[v]$: $\{L_v[1], L_v[n]\}$
    end if
    SottoAlberi $= \{\}$
    if $sx(v)$ non è Nullo **then**
      Aggiorna i valori nel dizionario SottoAlberi con i risultati di DFS-EA-Tmax-SpazioN($sx(v)$)
    end if
    if $dx(v)$ non è Nullo **then**
      Aggiorna i valori nel dizionario SottoAlberi con i risultati di DFS-EA-Tmax-SpazioN($dx(v)$)
    end if
    $EA_{sx}, T_{\max,sx}$ =SottoAlberi$[sx(v)]$
    $EA_{dx}, T_{\max,dx}$ =SottoAlberi$[dx(v)]$
    if $EA_{sx} > T_{\max,dx} \vee EA_{dx} > T_{\max,sx}$ **then**
      return $D[v] : \{\infty, \infty\}$
    end if
    $EA = \max(EA_{sx}, EA_{dx})$
    $T_{\max} = \min(T_{\max,sx}, T_{\max,dx})$
    nextEA $=$ BinarySearch(L_v, EA)
    nextTmax $=$ BinarySearch$(L_v, T_{\max})$

$\text{minTime} = \min(\text{nextTmax}, T_{\max})$
$\text{SottoAlberi}[v]$={nextEA,minTime}
return SottoAlberi
end procedure

Versione con spazio $O(N)$

Versione spazio lineare

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

    # Caso base: foglia
    if root.left is None and root.right is None:
        print(f"EA e tempo max visita per il sottoalbero radicato nel nodo {root.value} (foglia): {root.weight[0], root.weight[-1]}")
        return {root.value: (root.weight[0], root.weight[-1])}

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

    # Calcolo ricorsivo per il sottoalbero sinistro
    if root.left is not None:
        sottoalberi.update(dfs_EA_tmax_spazioN(root.left))

    # Calcolo ricorsivo per il sottoalbero destro
    if root.right is not None:
        sottoalberi.update(dfs_EA_tmax_spazioN(root.right))

    # Estrai i valori di EA e Tmax dai figli
    ea_sx, t_max_sx = sottoalberi[root.left.value] if root.left else (float("-inf"), float("inf"))
    ea_dx, t_max_dx = sottoalberi[root.right.value] if root.right else (float("-inf"), float("inf"))

    # Controllo di consistenza tra i sottoalberi
    if ea_sx > t_max_dx and ea_dx > t_max_sx:
        return {root.value: (float("inf"), float("inf"))}

    # Calcolo EA e Tmax per il nodo corrente
    EA = max(ea_sx, ea_dx)
    t_max_visita = min(t_max_sx, t_max_dx)
    print(f"EA e tempo max visita per il sottoalbero radicato nel nodo {root.value} (nodo interno): {EA, t_max_visita}")

    k = binary_search(root.weight,EA)
    nextTimeMax = binary_search_leq(root.weight,t_max_visita) #binary search per trovare il predecessore, quindi il primo tempo t <=
```

```
    t_max_visita

        minTime = min(t_max_visita,nextTimeMax)
        # Aggiornamento del nodo corrente nei risultati
        sottoalberi[root.value] = (k, minTime)

        return sottoalberi
```

Costo computazionale

Analizziamo l'equazione di ricorrenza dell'algoritmo di visita DFS, che è la seguente

$$T(N) = 2T\left(\frac{N}{2}\right) + \log(M)$$

Applicando lo strotolamento, abbiamo che

$$\begin{aligned}
T(N) = & 2T\left(\frac{N}{2}\right) + \log(M) \\
& 2\left(2T\left(\frac{N}{2}\right) + \log(M)\right) + \log(M) \\
& \vdots \\
& 2^i T\left(\frac{N}{2^i}\right) + \sum_{j=0}^{i-1} 2^i \log(M)
\end{aligned}$$

A questo punto, $\frac{N}{2^i} = 1 \iff i = \log_2(N)$
Così facendo, l'equazione diventa

$$\begin{aligned}
& T(N) = 2^{\log_2(N)} + \sum_{j=0}^{\log_2(N)-1} 2^j \log(M) \\
& = \\
& T(N) = N + N\log M \implies T(N) = \Theta(N\log(M))
\end{aligned}$$

Il costo precedente è valido per entrambe le versioni

Correttezza

Definiamo alcune variabili :

- L_v : Lista di timestamp dell'arco che entra in v
- $EA_{\max}$: $\max_{f \text{: f è foglia}} EA$ da $f \in T_v$ fino al padre di v
  - T_v : sottoalbero radicato nel nodo 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

La correttezza di questo algoritmo deriva dal seguente ***lemma***

> ***Lemma***
> L'algoritmo calcola correttamente , per ogni nodo v , i valori di EA e $T_{\max}$ del rispettivo sottoalbero T_v.
> Mentre risale verso la radice, prende i valori appena calcolati e controlla la condizione di connettività temporale tra due sottoalberi diversi, detti $T_{v_i}, T_{v_j}, i \neq j$.
> Quando arriva alla radice, ha correttamente calcolato i valori di EA e $T_{\max}$ dei sottoalberi relativi ai due figli della radice stessa.

Dimostrazione di correttezza

L'algoritmo calcola correttamente, per ogni nodo v, i valori di EA (Earliest Arrival) e $T_{\max}$ (tempo massimo di visita) per il sottoalbero radicato in T_v.
Inoltre, mentre risale verso la radice:

- Usa questi valori per verificare la condizione di connettività temporale tra due sottoalberi T_{v_i} e T_{v_j} ($i \neq j$).
- Al termine, quando risale verso la radice,l'algoritmo ha calcolato correttamente i valori di EA e $T_{\max}$ per i due figli della radice. Di conseguenza, possiamo verificare in tempo costante $O(1)$ se l'albero è temporalmente connesso oppure no

---

Struttura della dimostrazione

La dimostrazione si basa sull'induzione, poiché l'algoritmo risolve il problema tramite una DFS (Depth First Search) che esplora il sottoalbero in maniera ricorsiva.

Base dell'induzione: nodo foglia

Per un nodo foglia v:

1. T_v coincide con il singolo nodo v.
2. I valori EA e $T_{\max}$ del sottoalbero sono esattamente:
   - $EA(v) = L_v[1]$ (tempo di arrivo minimo).
   - $T_{\max}(v) = L_v[n]$ (tempo massimo di visita).

Nell'algoritmo:

- Questo viene calcolato e restituito correttamente nella base del caso:

  ```
  if root.left is None and root.right is None:
      return {root.value: (root.weight[0], root.weight[-1])}
  ```

- Non ci sono figli, quindi la condizione di connettività è automaticamente soddisfatta.
- Il risultato è corretto per il nodo foglia.

Passo induttivo: nodo interno

Supponiamo che l'algoritmo calcoli correttamente EA e $T_{\max}$ per tutti i sottoalberi dei figli di un nodo v. Dimostriamo che calcola correttamente questi valori per il sottoalbero T_v.

1. **Calcolo dei valori dei sottoalberi**:
    - L'algoritmo calcola ricorsivamente EA e $T_{\max}$ per i figli sinistro e destro:

    ```
    sottoalberi.update(dfs_EA_tmax_spazioN(root.left))
    sottoalberi.update(dfs_EA_tmax_spazioN(root.right))
    ```

    - Per ogni figlio v_i (sinistro o destro), EA e $T_{\max}$ sono corretti per il sottoalbero T_{v_i} per ipotesi induttiva.
2. **Verifica della condizione di connettività**:
    - La condizione di connettività temporale tra i sottoalberi T_{v_i} e T_{v_j} ($i \neq j$) è verificata:

    ```
    if ea_sx > t_max_dx and ea_dx > t_max_sx:
        return {root.value: (float("inf"), float("inf"))}
    ```

    - Se questa condizione viene soddisfatta,allora significa che il sottoalbero radicato in v non è temporalmente connesso e vengono restituiti valori non validi (∞).
3. **Calcolo dei valori per il nodo v**:
    - I valori EA e $T_{\max}$ per T_v dipendono dai valori dei figli e dal nodo stesso:

    ```
    EA = max(ea_sx, ea_dx)
    t_max_visita = min(t_max_sx, t_max_dx)
    ```

    - L'algoritmo considera il nodo v:
        - Effettua una ricerca binaria su L_v per determinare il valore k corrispondente a EA.
        - Determina $T_{\max}$ come il minimo tra $t_{\max,visita}$ e il predecessore nell'array dei timestamp.
4. **Conclusione**:
    - Poiché i valori per i figli sono corretti (per ipotesi induttiva) e il calcolo di EA e $T_{\max}$ per v segue le regole definite, anche i valori calcolati per T_v sono corretti.

Conclusione per la radice

Quando l'algoritmo raggiunge la radice:

1. Ha già calcolato EA e $T_{\max}$ per i due figli della radice.
2. Verifica la connettività temporale tra i due sottoalberi:
   - Se soddisfatta, allora l'albero **è temporalmente connesso**
   - Se non soddisfatta, l'albero **non è temporalmente connesso**

Quindi, l'algoritmo calcola correttamente i valori di EA e $T_{\max}$ per ogni sottoalbero, e alla fine risponde correttamente alla richiesta di connettività temporale.

Versione Algoritmo per alberi non binari

Per quanto riguarda gli alberi non binari, abbiamo due casistiche :

1. Usiamo la versione 1 con spazio costante
2. Usiamo la versione 2 con spazio lineare

La correttezza è valida per entrambe le versioni, cambia solo il costo totale finale dell'algoritmo.

Versione 1

Se usiamo questa versione dell'algoritmo, avremmo un costo di $O(N \log(M))$ per ogni sottoalbero partendo dalla radice (quindi per ogni sottoalbero radicato nei figli della radice)

Così facendo, se indentifichiamo con Δ il grado massimo dell'albero T, avremo che il costo di esecuzione dell'algoritmo di visita sarà pari a

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

A questo punto, la condizione di check tra i valori EA e $T_{\max}$ verrà effettuata nel seguente modo :

1. Prendo il primo EA da sinistra, e vedo se vale la seguente condizione

$$EA_1 \leq \min(T_{\max}), \forall T_{\max,i}, i = 0, \ldots, n-1$$

2. Se questa condizione è verificata, significa che EA_1 sarà sempre $\leq$ di ogni $T_{\max}$. Questa ricerca del minimo $T_{\max}$ e check costano $\log(\Delta)$, ne faccio un numero totale pari a Δ, quindi il costo totale del check sarà

$$\Delta \log(\Delta)$$

3. Se anche un solo EA tra tutti non è $\leq \min(T_{\max}), \forall T_{\max,i}, i = 0, \ldots, n-1$, allora posso affermare che l'albero ***NON*** è temporalmente connesso, in quanto esiste almeno un EA che non può collegarsi con gli altri sottoalberi
4. Se invece la condizione di connettività vale per tutti gli EA (che ricordiamo essere un numero pari a Δ) allora posso affermare che l'albero **è** temporalmente connesso

Il costo totale dell'algoritmo in questo caso diventa

$$O(\Delta N \log(M) + \Delta \log(\Delta))$$

Versione 2

La versione 2 è sostianzialmente uguale alla prima versione, cambia solamente il costo.

Infatti in questa versione, paghiamo un pochino meno a livello temporale, ma dobbiamo sfruttare un po di memoria.

Il costo in questa versione è

$$\text{Tempo} = O(N \log(M) + N \cdot \Delta + \Delta \log(\Delta))$$
$$\text{Spazio} = O(N)$$

Il funzionamento dell'algoritmo è lo stesso della prima versione

Correttezza

Le variabili introdotte prima valgono anche qui

Si aggiungono solo due dizionari, che sono

- **Nodo** : Dizionario per le foglie
- **Sottoalberi** : Dizionario per i nodi interni

La correttezza di questo algoritmo deriva dal seguente ***lemma***

> ***Lemma***
> L'algoritmo calcola correttamente , per ogni nodo v , i valori di EA e $T_{\max}$ del rispettivo sottoalbero T_v.
> Mentre risale verso la radice, prende i valori appena calcolati e controlla la condizione di connettività temporale tra tutti i sottoalberi diversi radicati nei figli di v
> Quando arriva alla radice, ha correttamente calcolato i valori di EA e $T_{\max}$ dei sottoalberi relativi a tutti i figli della radice stessa.

Dimostrazione del lemma

L'algoritmo calcola correttamente, per ogni nodo v, i valori di EA (Earliest Arrival) e $T_{\max}$ (tempo massimo di visita) per il sottoalbero T_v. Inoltre:

- Risalendo verso la radice, verifica la condizione di connettività temporale tra tutti i sottoalberi T_{v_i} e T_{v_j}, dove v_i, v_j sono figli diversi di vv.
- Al termine, alla radice, calcola correttamente EA e $T_{\max}$ per il sottoalbero complessivo.

Struttura della dimostrazione

La dimostrazione si basa sull'**induzione** sulla profondità del nodo v nell'albero T.

Base dell'induzione: nodo foglia

Per un nodo foglia vv:

1. Il sottoalbero T_v coincide con il singolo nodo vv.
2. I valori $EA(v)$ e $T_{\max}(v)$ sono determinati direttamente dai pesi del nodo:
    - $EA(v) = L_v[1]$ (tempo di arrivo minimo).
    - $T_{\max}(v) = L_v[n]$ (tempo massimo di visita).

Nell'algoritmo, ciò è implementato nel caso base:

```
if not root.children:
    return {root.value: (root.weight[0], root.weight[-1])}
```

Poiché non ci sono figli, la condizione di connettività temporale è automaticamente soddisfatta. Il risultato è corretto per ogni nodo foglia.

Passo induttivo: nodo interno

Supponiamo che l'algoritmo calcoli correttamente i valori EA e $T_{\max}$ per tutti i figli di un nodo vv. Dimostriamo che li calcola correttamente per vv.

1. **Calcolo dei valori dei sottoalberi figli**:
    - Per ogni figlio v_i, l'algoritmo calcola ricorsivamente $EA(T_{v_i})$ e $T_{\max}(T_{v_i})$, che per ipotesi induttiva sono corretti:

    ```
    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)
    ```

2. **Controllo di consistenza tra i figli**:
    - L'algoritmo verifica la condizione di connettività temporale tra tutti i sottoalberi figli T_{v_i}, T_{v_j}, per $i \neq j$:

    ```
    min_tmax = min(t_max_vals)
    pos_min = t_max_vals.index(min_tmax)
    for i in range(len(ea_vals)):
    ```

```python
        if ea_vals.index(ea_vals[i]) == pos_min:
            continue
        elif ea_vals[i] > min_tmax:
            return {root.value: (float("inf"), float("inf"))}
```

- La condizione richiede che, per ogni coppia (i, j): Se $EA(T_{v_i}) > T_{\max}(T_{v_j})$ allora i sottoalberi non sono connessi temporalmente.
- Il codice verifica questa condizione ottimizzando il confronto:
  - Determina il sottoalbero con il valore $T_{\max}$ minimo.
  - Confronta tutti gli EA dei figli con questo valore minimo.

3. **Calcolo dei valori EA e $T_{\max}$ per il nodo v**:
   - Se la condizione di connettività è soddisfatta, l'algoritmo calcola:

```python
EA = max(ea_vals)
t_max_visita = min(t_max_vals)
```

   - Questi valori rispettano le regole di $EA(T_v)$ e $T_{\max}(T_v)$ per un nodo interno:
     - $EA(T_v) = \max(EA(T_{v_i}))$: il tempo più tardi tra i sottoalberi figli.
     - $T_{\max}(T_v) = \min(T_{\max}(T_{v_i}))$: il tempo più presto tra i sottoalberi figli.

4. **Considerazione del nodo stesso**:
   - L'algoritmo aggiunge il nodo v calcolando EA e $T_{\max}$ nel contesto dei suoi pesi:

```python
k = binary_search(root.weight, EA)
nextTimeMax = binary_search_leq(root.weight, t_max_visita)
minTime = min(t_max_visita, nextTimeMax)
sottoalberi[root.value] = (k, minTime)
```

   - Ciò garantisce che i valori calcolati per T_v tengano conto sia dei figli sia del nodo vv.

Conclusione per la radice

Quando l'algoritmo raggiunge la radice:

1. Ha già calcolato EA e $T_{\max}$ per tutti i figli della radice.
2. Verifica la condizione di connettività temporale tra tutti i sottoalberi figli.
3. Calcola i valori EA e $T_{\max}$ complessivi per il sottoalbero radicato nella radice.

Poiché ogni passo della ricorsione è corretto e la radice è gestita allo stesso modo, l'algoritmo calcola correttamente i valori EA e $T_{\max}$ per tutto l'albero T.

Analisi e Dimostrazione di Correttezza dell'Algoritmo dfs_EA_tmax

L'algoritmo è progettato per trovare, in un albero binario, il massimo **Earliest-Arrival Time (EA)** possibile per un percorso che visiti tutti i nodi, e il corrispondente tempo di visita massimo, in modo tale da poter determinare se l'albero in input è **temporalmente connesso** oppure no.

Funzionamento di base:

1. **Caso base:** Se il nodo è una foglia, l'EA è il peso minimo dell'arco entrante e il tempo di visita massimo è il peso massimo dell'arco entrante.
2. **Caso ricorsivo:**
   - Si calcolano ricorsivamente l'EA massimo e il tempo di visita massimo per i sottoalberi sinistro e destro.
   - Si verifica se i due sottoalberi sono compatibili temporalmente (cioè se esiste un ordine di visita che rispetta i vincoli temporali).
   - Si calcola l'EA massimo del nodo corrente come il massimo tra gli EA massimi dei sottoalberi.
   - Si calcola il tempo di visita massimo del nodo corrente considerando il minimo tra il tempo di visita massimo dei sottoalberi e il peso massimo dell'arco entrante nel nodo corrente.

Ipotesi induttiva: Assumiamo che l'algoritmo funzioni correttamente per tutti i sottoalberi di un nodo.

Passo base: Per le foglie, l'algoritmo calcola correttamente l'EA e il tempo di visita massimo, in quanto non ci sono sottoalberi.

Passo induttivo: Consideriamo un nodo interno. Per ipotesi induttiva, i valori di EA massimo e tempo di visita massimo calcolati per i sottoalberi sinistro e destro sono corretti.

- **Verifica di compatibilità:** La condizione `if not (min_sx<=max_dx or min_dx<=max_sx)` assicura che i due sottoalberi siano compatibili temporalmente. Se questa condizione non fosse verificata, non esisterebbe un ordine di visita valido per l'intero sottoalbero.
- **Calcolo dell'EA massimo:** Il massimo EA del nodo corrente è correttamente calcolato come il massimo dei minimi timestamo di tutti gli archi di un sottoalbero.
- **Calcolo del tempo di visita massimo:** Il tempo di visita massimo è calcolato considerando il minimo tra i massimi di tutti i timestamp di un sottoalbero. Questo è corretto perché il tempo di visita massimo è limitato sia dal tempo necessario per

visitare tutti i nodi del sottoalbero e sia dal tempo necessario per raggiungere il nodo stesso.

Conclusione: L'algoritmo calcola correttamente l'EA massimo e il tempo di visita massimo per ogni nodo dell'albero, e quindi per l'intero albero.